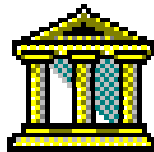


---

# Programació Visual amb Delphi

Tercera edició. Curs 2001-2002

---



**Alejandro Castán Salinas**

Departament de Llenguatges i Sistemes Informàtics  
Escola Universitària d'Enginyeria Tècnica Industrial de Barcelona  
Universitat Politècnica de Catalunya  
Correu electrònic: [alex.castan@upc.es](mailto:alex.castan@upc.es)



**Escola Universitària d'Enginyeria Tècnica  
Industrial de Barcelona**

---



## **Programació Visual amb Delphi - revisió 3**

Copyleft © Alejandro Castán Salinas

Es concedeix el permís per copiar, distribuir i/o modificar aquest document sota els termes de la llicència de documentació lliure GNU, versió 1.1 o qualsevol altre versió posterior publicada per la *Free Software Foundation*.



Podeu consultar dita llicència a <http://www.gnu.org/copyleft/fdl.html>.

El contingut d'aquest document pot canviar degut a ampliacions i a correccions enviades pels lectors. Trobareu sempre la última versió del document a <http://www.lsi.upc.es/~acastan/Docencia/>.



# Índex

|  |           |
|--|-----------|
| <i>Programació Visual amb Delphi</i>               | <i>1</i>  |
| <b>Índex</b>                                       | <b>5</b>  |
| <b>Introducció</b>                                 | <b>8</b>  |
| Presentació  | 8         |
| Conceptes de programació orientada a objectes      | 10        |
| Conceptes de programació orientada a esdeveniments | 17        |
| Elements d'una interfície gràfica                  | 19        |
| Exemples d'interfícies gràfiques                   | 26        |
| Repàs de normes i consells de programació.         | 27        |
| <b>L'entorn de treball de Delphi</b>               | <b>33</b> |
| L'entorn de treball de Delphi                      | 33        |
| Configurant l'entorn                               | 35        |
| Creació d'una aplicació amb Delphi                 | 39        |
| Depuració i tractament d'errors                    | 45        |
| Treballant amb els components                      | 46        |
| Exemple 3-1: cronòmetre                            | 49        |
| Exercici 3-1: calendari                            | 51        |
| <b>El llenguatge Object Pascal</b>                 | <b>53</b> |
| Declaració de variables                            | 53        |
| Declaració de constants                            | 56        |
| Declaració de nous tipus de dades                  | 57        |
| Classes i objectes                                 | 57        |
| Sentències i expressions                           | 58        |
| Funcions i procediments                            | 63        |
| Entrada i sortida                                  | 66        |
| Control d'errors                                   | 67        |
| Mòduls, llibreries d'enllaç dinàmic i paquets      | 67        |
| Exemple 4-1: POO – creació d'una classe            | 69        |
| Exemple 4-2: POO – herència                        | 71        |
| Exemple 4-3: POO – classes i mètodes virtuals      | 73        |
| Exemple 4-4: maneigament d'excepcions              | 76        |
| Exercici 4-1: POO – creació d'una classe           | 77        |
| Exercici 4-2: POO – herència                       | 80        |
| <b>Components VCL</b>                              | <b>81</b> |
| Propietats, esdeveniments i mètodes generals       | 81        |
| Application  | 86        |
| Screen   | 88        |
| Form   | 89        |
| InputBox i MessageDlg                              | 92        |
| Frame  | 93        |
| MainMenu   | 94        |
| PopupMenu  | 96        |
| Label  | 97        |
| Edit   | 98        |
| Memo   | 100       |
| Button, BitBtn i SpeedButton                       | 101       |
| CheckBox   | 102       |
| RadioButton  | 103       |
| RadioGroup   | 104       |
| ListBox  | 105       |
| ComboBox   | 106       |

|   |            |
|---|------------|
| ScrollBar   | 107        |
| GroupBox  | 108        |
| Bevel   | 109        |
| Panel   | 110        |
| Image   | 111        |
| Shape   | 112        |
| Canvas  | 113        |
| Picture   | 115        |
| OLEContainer  | 116        |
| Timer   | 118        |
| Strings i StringList  | 119        |
| Altres Components VCL   | 121        |
| Matrius de components   | 122        |
| Creació de nous components  | 122        |
| Exemple 5-1: arrels de polinomis de segon grau  | 122        |
| Exercici 5-1: càlcul de la mitjana i la desviació estàndard   | 124        |
| Exercici 5-2: instal·lació i ús d'un nou component  | 125        |
| Exercici 5-3: creació de colors   | 126        |
| Exercici 5-4: dissenyador de formularis   | 127        |
| <b>Creació d'una aplicació</b>  | <b>128</b> |
| Els components ImageList  i ActionList  | 129        |
| Una aplicació estàndard   | 134        |
| Quadres de diàleg estàndard   | 137        |
| Ús del portafolis   | 141        |
| Impressió de dades  | 144        |
| Enllaç i inserció d'objectes OLE  | 150        |
| Ús de arrossegar i deixar anar  | 151        |
| Ús del registre   | 153        |
| Ús de varies finestres  | 157        |
| Treballar amb varis documents alhora – Aplicacions MDI  | 163        |
| Creació del fitxer d'ajuda  | 171        |
| Creació de l'assistent d'instal·lació   | 179        |
| Exercici 6-1: creació d'una aplicació estàndard   | 188        |
| <b>Llibreries d'enllaç dinàmic i l'API de Windows</b>   | <b>189</b> |
| El sistema operatiu Windows i les finestres   | 189        |
| Les DLL de Windows  | 190        |
| L'API de Windows  | 190        |
| Exemples basats en preguntes freqüents d'alumnes  | 193        |
| <b>Tècniques de gràfics i so</b>  | <b>198</b> |
| El component MediaPlayer  | 198        |
| Transparències  | 200        |
| Exemple 7-1: explorador multimèdia  | 202        |
| <b>Accés i maneig de bases de dades</b>   | <b>206</b> |
| <b>Programació per Internet</b>   | <b>207</b> |
| <b>Programació multifil</b>   | <b>208</b> |
| <b>Solucions als exercicis</b>  | <b>209</b> |
| Exercici 3-1: calendari   | 209        |
| Exercici 4-1: POO – creació d'una classe  | 211        |
| Exercici 4-2: POO – herència  | 213        |
| Exercici 5-1: càlcul de la mitjana i la desviació estàndard   | 215        |
| Exercici 5-2: instal·lació i ús d'un nou component  | 220        |
| Exercici 5-3: creació de colors   | 225        |
| Exercici 5-4: dissenyador de formularis   | 227        |
| <b>Apèndix I. Excepcions</b>  | <b>231</b> |

|  |            |
|--|------------|
| <b>Apèndix II. Rutines predefinides d'Object Pascal</b>    | <b>235</b> |
| Rutines estàndard  | 235        |
| Rutines de tractament de fitxers                           | 237        |
| Rutines de tractament de cadenes acabades en nul           | 237        |
| <b>Apèndix III. Gramàtica del llenguatge Object Pascal</b> | <b>239</b> |
| <b>Apèndix IV. Taxonomia de classes a Delphi</b>           | <b>245</b> |
| <b>Apèndix V. Delphi a Internet</b>                        | <b>247</b> |

# Introducció

## Presentació

### **Objectius del curs**

La finalitat d'aquest curs és aprendre a dissenyar aplicacions amb interfícies gràfiques que aprofitin els elements dels actuals entorns de finestres (Windows, X-Windows, MacOS, ...), tot fent servir conceptes de programació estructurada, programació orientada a objectes i programació orientada a esdeveniments.

Per treballar les matèries del curs hem escollit, per la seva senzillesa i potència, el llenguatge d'alt nivell Object Pascal i l'entorn de programació Delphi.

Els objectius que ens proposem assolir són:

- Entendre els conceptes d'interfícies gràfiques, de programació orientada a esdeveniments i de programació orientada a objectes. Familiaritzar-nos amb la terminologia.
- Aprendre els fonaments de disseny i implementació d'una aplicació visual dirigida per esdeveniments.
- Saber utilitzar l'entorn de desenvolupament de Delphi per escriure, compilar, executar, depurar i guardar programes.
- Conèixer la sintaxi del llenguatge de programació Object Pascal i escriure programes en aquest llenguatge. Conèixer els tipus de dades d'Object Pascal per a fer-los servir de manera adequada en cada tipus específic de problema.
- Saber utilitzar les eines visuals de Delphi per dissenyar la interfície gràfica de les nostres aplicacions. Conèixer el conjunt de components VCL (Visual Component Library) i CLX (Cross Platform Component Library) de Delphi.
- Aprofundir en el funcionament dels entorns de finestres dels sistemes operatius Windows, MacOS i Linux.

### **Què és Delphi ?**

Delphi és un entorn integrat de desenvolupament (IDE) expressament pensat per desenvolupar de manera ràpida aplicacions (RAD) per a entorns Windows i per a entorns Linux. Delphi porta les eines necessàries per a crear aplicacions orientades a objectes, aplicacions amb arquitectura client/servidor i aplicacions distribuïdes, tres tècniques molt importants en les tendències de programació actuals. Amb Delphi podem crear tant aplicacions de propòsit general com sofisticades bases de dades. El motor de bases de dades de Borland (BDE), construït dintre de Delphi, ens permet desenvolupar aplicacions que poden accedir fàcilment a bases de dades locals com Paradox, dBase i ODBC; i servidors SQL com Oracle, Sybase i Microsoft SQL.

Entorn integrat de desenvolupament vol dir que podem realitzar tots els passos per desenvolupar l'aplicació sense necessitat de sortir de l'entorn. Delphi combina un editor de text per escriure els programes, eines d'ajuda, un compilador amb optimització de codi per 32 bits, eines de correcció d'errors per verificar els programes, eines per a crear dibuixos i gràfics, eines per a crear instal·ladors per a la nostra aplicació, eines per a crear icones, eines per manegar bases de dades, etc.

Emprant Delphi podem crear aplicacions amb interfície gràfica sofisticades d'una manera molt senzilla i amb un mínim de codificació manual, a base de components ja creats que generen el codi automàticament. Es tracta de generar aplicacions d'una manera interactiva seleccionant els components de la nostra aplicació de la Paleta de Components i arrossegant-los sobre un formulari. Llavors Delphi genera el codi automàticament per a cada component visual que hem desplaçat al sobre del nostre formulari.



### Breu història de Delphi

Delphi és, en el seu cor, un compilador de Pascal. Delphi 5 és el nou pas en l'evolució del compilador de Pascal que Borland ha vingut desenvolupant des que Anders Hejlsberg el primer compilador Turbo Pascal fa més de quinze anys.

Delphi està basat en Object Pascal, una varietat de Pascal orientat a objectes de Borland. Pascal fou creat a finals dels anys seixanta per Niklaus Wirth com un llenguatge per aprendre a programar i, per tant, fou dissenyat al voltant del concepte de simplicitat i és fàcil d'aprendre.

Veiem breument la història de la família Delphi:

**Delphi 1** esdevingué la primera aproximació de Borland al desenvolupament d'aplicacions per l'entorn Windows: eines de desenvolupament visual, executables compilats, DLL's, bases de dades. etc.

**Delphi 2** proporcionava, un any més tard, els mateixos beneficis que Delphi 1 però pels nous sistemes operatius de 32 bits Windows 95 i Windows NT. A aquests beneficis cal afegir un compilador de 32 bits que produïa aplicacions més ràpides, una llibreria d'objectes millorada i més extensa, suport de bases de dades millorat, maneigament de cadenes millorat i suport de la tecnologia OLE.

**Delphi 3** aconseguí de fer més senzilla d'utilitzar les tecnologies COM i ActiveX i el desenvolupament d'aplicacions pel World Wide Web.

**Delphi 4** està enfocat en fer el desenvolupament sota Delphi més senzill. El IDE ha estat redissenyat i el depurador ha estat millorat notablement. El nou explorador de mòduls ens permet navegar i editar unitats des d'una interfície gràfica convenient. A més a més, suporta la creació d'aplicacions distribuïdes basades en la tecnologia CORBA.

**Delphi 5** incorpora alguna eina nova, com el dissenyador de mòduls de dades, i millores en gairebé totes les eines: noves opcions de depuració, creació de controls ActiveX més senzilla. També apareixen nous components, propietats i esdeveniments, i la opció d'incorporar marcs (*frames*) a les nostres aplicacions. Els marcs són un tipus especial de formulari que pot estar niat dins un formulari o un altre marc.

**Delphi 6** potencia el desenvolupament multiplataforma, permetent que una aplicació escrita per Delphi funcioni tant a entorns Windows com a entorns Linux. També suporta l'estàndard XML de document d'intercanvi de dades en aplicacions i serveis distribuïts.

### Per què Delphi ?

Per desenvolupar aplicacions amb interfícies gràfiques basades en finestres hi han moltes més opcions possibles. Les més esteses actualment, juntament amb Delphi, serien Visual Basic, C++ amb llibreries X- Windows i Java. Per què hem escollit Delphi ? Observem la següent taula ...

| Llenguatge      | Multiplataforma | Dificultat  | IDE , RAD | Eficiència del codi |
|-----------------|-----------------|-------------|-----------|---------------------|
| Java            | Si              | mitja/alta  | Si        | baixa               |
| C++ i X-Windows | Si              | mitja/alta  | Si        | alta                |
| Visual Basic    | No              | baixa/mitja | Si        | baixa               |
| Delphi          | No              | baixa/mitja | Si        | alta                |

Actualment existeixen entorns de desenvolupament ràpid d'aplicacions (RAD) per a tots els llenguatges de la taula anterior. En les seves últimes versions, les diferències entre ells són insignificants. Tots incorporen gairebé les mateixes eines i components visuals similars. Visual Age 3 de IBM, CodeWarrior de MetroWorks, Delphi 5 i C++ Builder 5.5 de Borland, Visual Studio 6.0 de Microsoft són alguns d'aquests entorns. Menció especial mereix l'entorn Glade (veure figura 1.1), per ser una eina gratuïta i de codi obert. La única gran diferència entre els uns i els altres rau en el llenguatge de programació sobre el que escrivim el codi de l'aplicació.

- Visual Basic és un entorn que treballa amb el llenguatge de programació Basic. La senzillesa d'aquest llenguatge el fa la millor opció per les persones que comencen a programar. Per contra, les aplicacions escrites en Visual Basic funcionen només sobre el sistema operatiu Windows i, a més a més, són interpretades enlloc de compilades.
- Delphi és un entorn que treballa amb una extensió del llenguatge Pascal orientada a objectes. És un llenguatge gairebé tant senzill com Basic, però molt més flexible i eficient. Actualment les

aplicacions creades amb Delphi només treballen sobre el sistema operatiu Windows, però ben aviat o faran també sobre Linux.

- C++ és un potent llenguatge orientat a objectes que ha evolucionat del conegut llenguatge de programació C. És un llenguatge molt estès avui en dia i sobre el que estan escrites gran part de les aplicacions que podem trobar. Existeixen entorns de desenvolupament en C++ per qualsevol plataforma i, si combinem l'ús del llenguatge C++ amb les llibreries gràfiques X-Windows, podem crear aplicacions amb interfície gràfica que funcionen sobre una gran varietat de sistemes operatius.
- Java és un potent llenguatge orientat a objectes molt semblant a C++. Java incorpora llibreries que ens permeten crear sofisticades interfícies gràfiques. Una de les seves principals avantatges és que una aplicació escrita en Java funciona sobre qualsevol plataforma: Windows, Linux, MacOs, ... sempre que aquesta incorpori una 'màquina virtual Java'. Per contra, el fet de compilar els programes a *bytecode* i posteriorment interpretar-los deteriora el rendiment del codi.

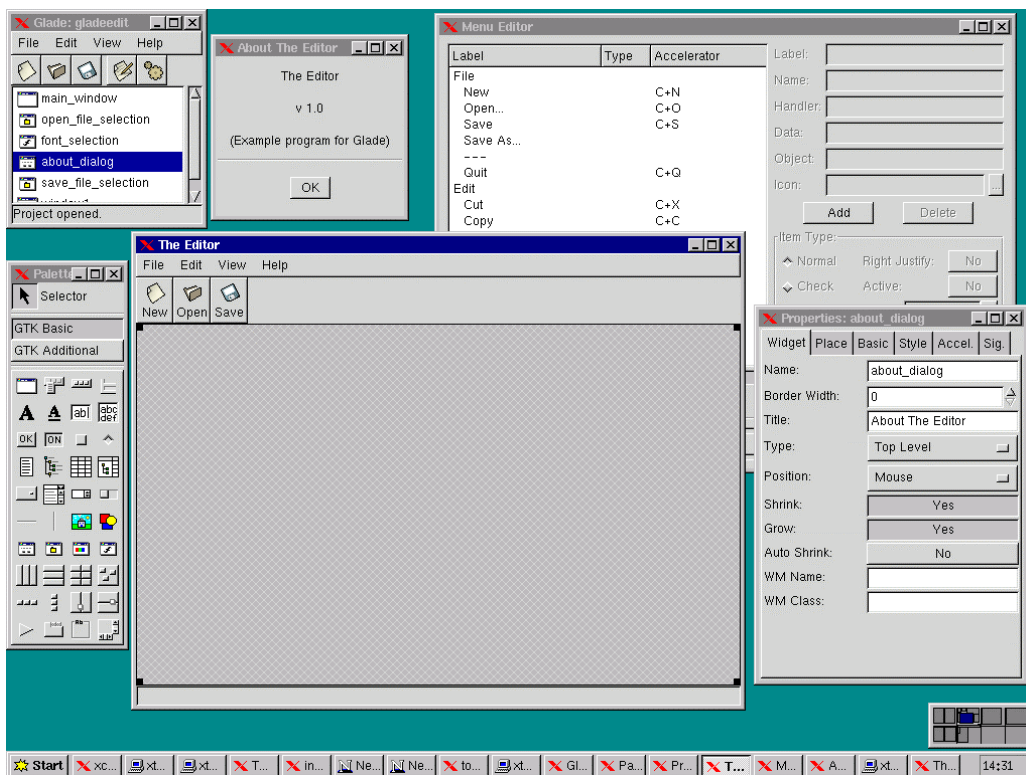


Figura 1.1 Glade és una eina gratuïta i de codi obert que actualment està en fase de desenvolupament. La trobareu acompanyant a qualsevol de les últimes distribucions del sistema operatiu Linux.

## Conceptes de programació orientada a objectes

### Objectes

Un objecte és un tipus de dada complex que conté altres tipus de dada, usualment anomenats atributs o variables, i mòduls de codi que operen sobre aquestes dades, usualment anomenats operacions o mètodes.

Per exemple, podríem considerar una finestra d'una interfície gràfica com un objecte que conté atributs com *alt* i *ample* de la finestra, i operacions com *maximitzar*.

Els atributs i valors associats estan 'amagats' dintre de l'objecte. Qualsevol altre objecte que vulgui obtenir o canviar un valor associat amb el primer objecte haurà de fer-ho enviant un missatge a una de les operacions del primer objecte. (Un missatge és, en termes d'orientació a objectes, l'equivalent d'una crida a una funció.) Cada operació d'un objecte manega els seus propis atributs. És el que anomenem encapsulament.

## Encapsulament

Encapsulament significa que els objectes no saben ni es preocupen de com els altres objectes emmagatzemen i processen les dades. Podem considerar els objectes com caixes negres que contenen tota la informació relacionada amb aquest objecte. Això ens permet manipular els objectes com unitats bàsiques, romanent oculta la seva estructura bàsica.

Un objecte envia un missatge a una operació associada amb un segon objecte. Una de les operacions del segon objecte, en conseqüència, contesta o realitza una acció. Sovint diem que el primer objecte demana que el segon objecte realitzi alguna tasca o servei per a ell. La figura 1.2 ens proporciona una visió general d'objectes i pas de missatges.

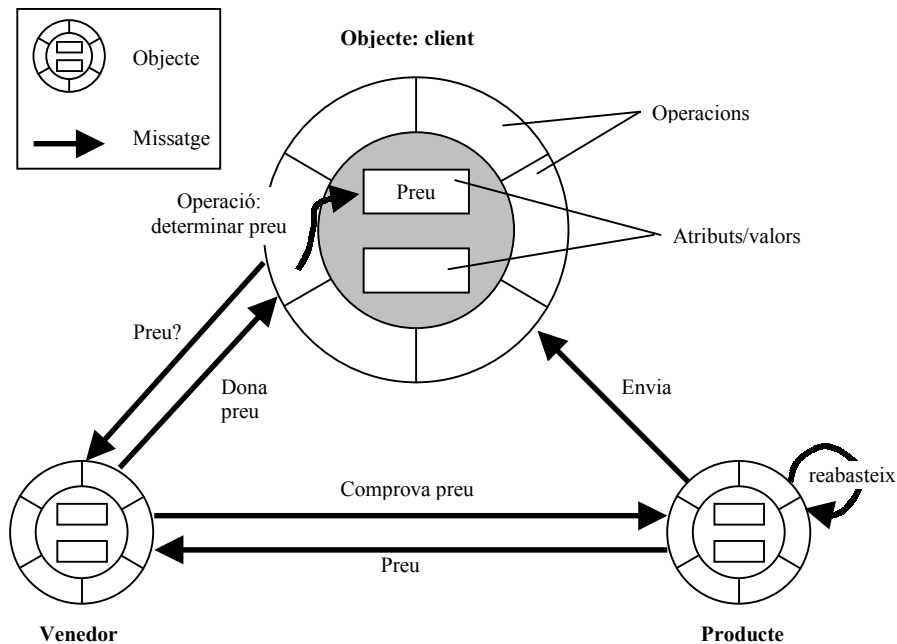
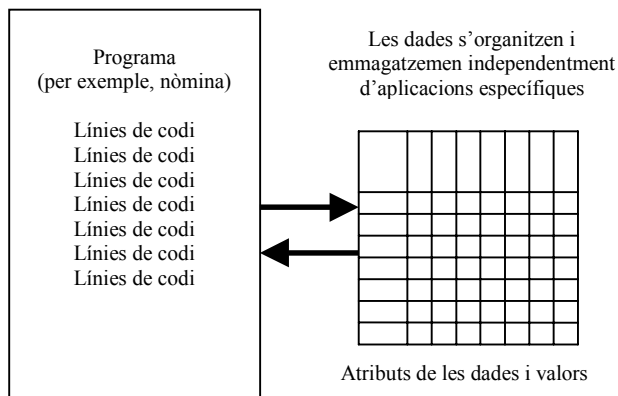


Figura 1.2 Objectes i pas de missatges.

Per apreciar una aproximació orientada a objectes a la programació, l'hauríem de contrastar amb la programació convencional (veure figura 1.3). Un sistema de software convencional està compost bàsicament de (1) un bloc de codi, el programa, i (2) dades, que són independents del codi i s'emmagatzemen en fitxers o taules. En una aplicació convencional tot el codi es guarda com un únic bloc, i qualsevol canvi en el programa s'ha de fer d'una manera acurada per assegurar que els canvis en un punt del codi no tenen efectes laterals no desitjats en alguna altre part. De manera similar, com que el codi crida o referència les dades, qualsevol canvi en el programa s'ha de fer d'una manera acurada per assegurar que un procediment no modifica una dada de manera inadequada abans que un altre procediment pugui emprar la dada. És possible modularitzar un programa convencional, que és exactament el que les metodologies estructurades dels últims anys setanta han intentat fer, però la pròpia natura del mètode procedural garanteix que la modularització normalment serà limitada i que les modificacions següents seran difícils.

### Una aplicació procedural



### Una aplicació orientada a objectes

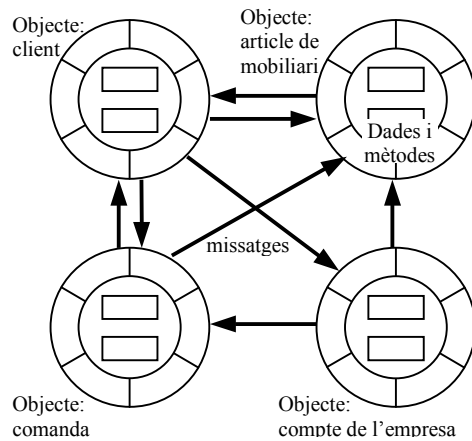


Figura 1.3 Una aplicació procedural en comparació amb una aplicació orientada a objectes.

Una aplicació orientada a objectes està modularitzada, en el sentit de que cada objecte és independent de qualsevol altre objecte. A més, el codi procedimental contingut dins cada objecte està modularitzat en varies operacions independents. Un missatge usualment només desencadena una única operació. Si es necessita realitzar un canvi, el desenvolupador sovint només necessita canviar una operació. Si es realitzen canvis sobre les estructures de dades internes associades amb un objecte, un desenvolupador pot confiar en que tots els canvis que necessitarà fer sobre el codi només involucraran operacions associades amb el mateix objecte.

Els objectes també poden encapsular grups més complexos d'atributs i mètodes o inclòs altres objectes. Així, podem usar objectes per capturar els diferents elements d'una interfície gràfica d'usuari (GUI). Així, icones, finestres, barres de desplaçament i altres eines, totes esdevenen objectes.

A un nivell encara més alt, un objecte pot encapsular tots els atributs i mètodes que constitueixen conceptes de negocis. Aquests conceptes de negocis inclouen coses com empleat, màquina, producte, pla, client, etc. Es poden desenvolupar models d'una companyia i les seves activitats emprant models d'aquest tipus. A aquest nivell, la metàfora orientada a objectes s'apropa molt a la realitat, i parlem d'empleats enviant missatges a clients, o enviant peticions per actualitzar plans empresarials, o fent que les màquines generin productes. Cadascun d'aquests objectes d'alt nivell, sovint anomenats objectes compostos, són, per suposat, fets d'objectes més bàsics.

### Abstracció

Mitjançant l'abstracció aconseguim no capficar-nos en els detalls concrets de les coses que no interessen a cada moment, sinó generalitzar i centrar-nos en els aspectes que ens permetin tenir una visió global del tema. Precisament la clau de la programació orientada a objectes rau en abstraure els mètodes i les dades comuns a un conjunt d'objectes i emmagatzemar-los en una classe. Amb aquest nivell d'abstracció, la introducció o eliminació d'un objecte en una determinada aplicació suposarà un treball mínim o nul.

### Classes

Les classes són plantilles que contenen mètodes, noms d'atributs i informació de tipus, però no valors reals. Els objectes són generats per les classes i sí contenen valors reals (veure figura 1.4).

Per posar un símil, les classes són a la programació orientada a objectes el que els tipus de dades són a la programació convencional, mentre que els objectes són a la programació orientada a objectes el que les variables són a la programació convencional. Alguns desenvolupadors utilitzen el terme *instància* com sinònim d'objecte.

Quan inicialment creem un sistema orientat a objectes, ens centrem primordialment en les classes que l'aplicació necessitarà.

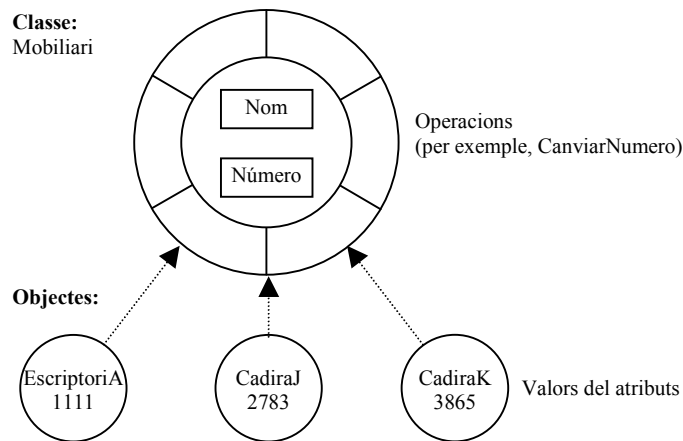


Figura 1.4 Classes i objectes.

### Missatges

Els missatges són el mitjà pel qual els objectes es comuniquen els uns amb els altres. Quan un objecte crida un mètode d'un altre objecte o accedeix a alguna de les propietats d'altre objecte, direm que li ha enviat un missatge.

Per exemple, quan fem clic en el botó de maximitzar una finestra, la finestra rep un missatge de notificació de que té que maximitzar-se. Quan un objecte rep un missatge, ha de conèixer perfectament el que ha de fer. Quan un objecte envia un missatge, simplement ha de conèixer que és el que fa el missatge, però no s'ha de preocupar de com l'executa internament l'objecte que el rep.

### Polimorfisme

Polimorfisme significa que la mateixa operació es comportarà de manera diferent quan sigui aplicada a objectes de classes diferents. També significa que diferents operacions associades amb diferents classes poden interpretar el mateix missatge de manera diferent. Així, per exemple, un objecte pot enviar un missatge `ESCRIURE` a varis objectes, i cada objecte usará la seva pròpia operació `ESCRIURE` per executar el missatge.

### Constructors i destructors

Un constructor és una funció membre especial d'una classe que es crida automàticament sempre que es declara un objecte d'aquesta classe. La seva funció es crear i inicialitzar un objecte de la seva classe. Donat que els constructors són funcions membre, admeten arguments igual que aquestes.

Un destructor és una funció membre especial d'una classe que s'utilitza per eliminar un objecte d'aquesta classe, alliberant-se la memòria que ocupa.

### Jerarquia de classes i herència

Parlant d'herència de classes, quan indiquem que una classe *B* és la filla d'una altre classe *A*, la classe filla automàticament adquireix tots els atributs i operacions que són implementats a la classe pare *A*. De manera similar, si encara creem una altre classe *C*, i indiquem que la classe *C* és filla de la classe *B*, la classe *C* adquireix tots els atributs i operacions de totes dues classes *A* i *B* (veure figura 1.5).

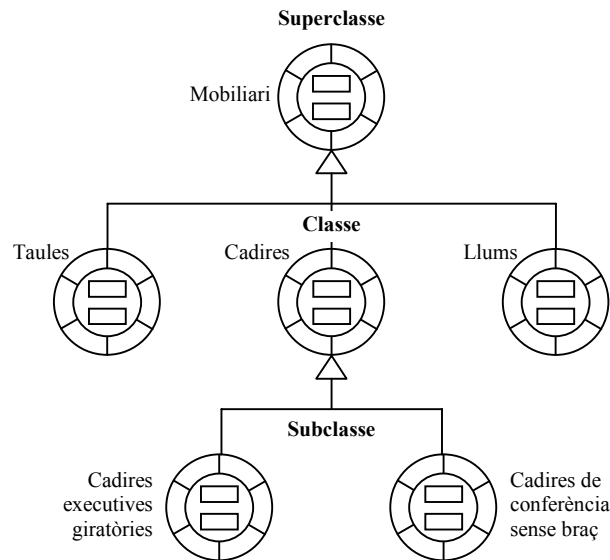


Figura 1.5 Classes i herència.

Emprant l'herència de classes, la majoria de classes de llenguatges orientat a objectes poden ser organitzades en jerarquies tal que les classes més concretes hereten atributs i operacions de les classes més abstractes.

Quan observem una classe específica en una jerarquia, les classes que estan per sobre d'aquesta classe a la jerarquia (les classes de les quals hereta) les anomenem pares o superclasses, i les classes per sota de la classe específica les anomenem filles o subclasses.

Es poden crear classes més concretes afegint nous atributs i mètodes o canviant el codi associat als mètodes heretats (veure figura 1.6). Així, podem començar amb una classe que descriu empleats i llavors crear dues subclasses més especialitzades tal que una representa empleats per hores i l'altre representa empleats a temps complet. Com que reutilitzem la majoria del codi associat a la classe empleat i només necessitem afegir uns pocs refinaments a les subclasses empleats per hores i a temps complet, el desenvolupament orientat a objectes tendeix a ser més ràpid que en programació convencional, un cop un desenvolupador ha acumulat una llibreria de les classes utilitzades més freqüentment.

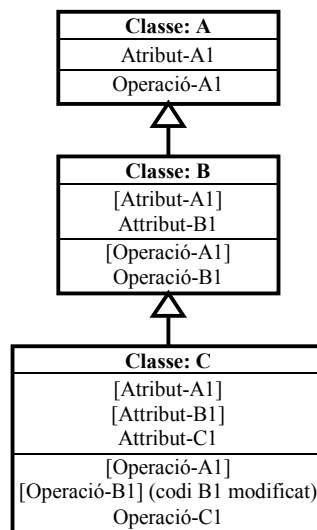
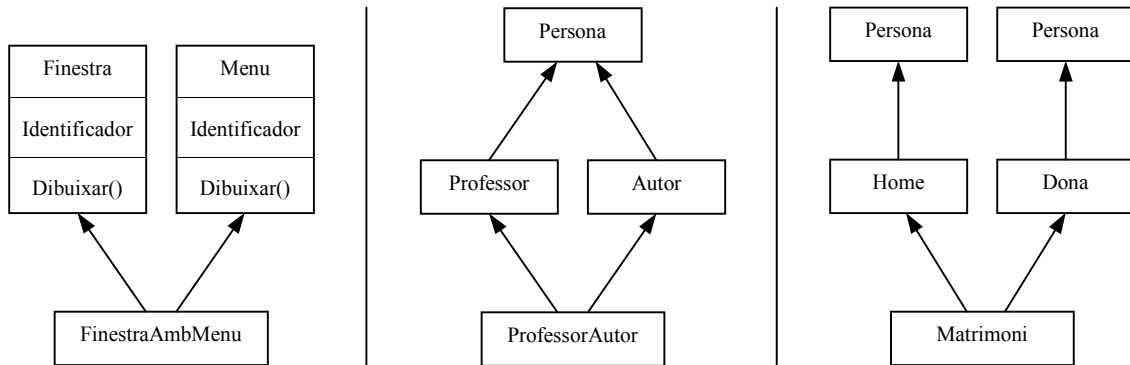


Figura 1.6 Herència de classes i especialització.

Alguns llenguatges orientats a objectes, com CLOS o C++, permeten a una classe heretar de més d'una superclasse. Quan això ocorre, parlem d'herència múltiple. En aquest tipus d'herència es poden presentar

dos problemes: les col·lisions de noms i l'herència repetida. Una col·lisió de noms es produeix quan una classe hereta de dues o més superclasses diferents que tenen el mateix nom a algun dels seus mètodes o atributs. L'herència repetida es produeix quan una classe és ascendent d'una altre classe per més d'un camí, és a dir, quan una classe hereta de dues o més superclasses que al seu temps hereten de la mateixa superclasse. (Veure figura 1.7).



**Figura 1.7**

- A l'esquerra, col·lisió de nom d'atributs i de mètodes: `FinestraAmbMenu` hereta el camp `Identificador` i el mètode `Dibuixar()`, que poden tenir implementacions diferents a cadascuna de les seves classes pare.
- Al centre, herència repetida amb una còpia de la superclasse: `ProfessorAutor` només ha d'heretar una còpia dels atributs de les seves classes pare.
- A la dreta, herència repetida amb dues còpies de la superclasse: `Matrimoni` ha d'heretar dues còpies dels atributs, una per cada classe pare.

### ***Públic, privat i protegit***

En la majoria de llenguatges orientats a objectes, el desenvolupador pot especificar si un atribut pot ser accedit per altres objectes o només pot ser accedit pels mètodes i operacions associats a la classe en qüestió (i les seves subclasses i objectes). Un atribut que només pot ser accedit pels seus propis mètodes s'anomena atribut privat. Aquesta és la situació normal i és la que hem assumit en tot el que hem explicat anteriorment. El cas contrari és un atribut públic, que és un atribut que pot ser modificat per operacions associades a qualsevol classe. Això viola l'encapsulació i no s'hauria d'emprar.

Si enlloc d'atributs parlem d'operacions, aquestes poden ser de tres tipus: públiques, privades i protegides. Una operació pública és tal que el seu nom està exposat als altres objectes, és a dir, una operació pública és un mètode que pot rebre missatges d'altres objectes.

Una operació privada és un mètode que no pot ser accedit per altres objectes. Només pot ser usada per altres operacions associades al mateix objecte per algun propòsit intern.

Les operacions protegides són un cas especial. Només les subclasses que descendeixen directament de la classe que conté una operació protegida coneixen la seva existència i poden usar-la.

### ***Herència d'interfície***

L'herència d'interfície no és tant simple ni tant potent com l'herència de classes, però ofereix algunes avantatges evitant problemes de disseny que poden ocórrer quan els programadors utilitzen herència de classes.

Una classe interfície defineix un conjunt d'atributs i mètodes que una altre classe pot voler emprar. Si un desenvolupador crea una nova classe i vol que aquesta hereti una interfície, haurà de crear un atribut que referenciï la classe interfície. També haurà de crear noms d'operacions que apuntin a les operacions de la classe interfície. Un cop fet això, la nova classe té accés als atributs i operacions de la classe interfície. Una classe pot heretar de qualsevol nombre de classes interfície (veure figura 1.8).

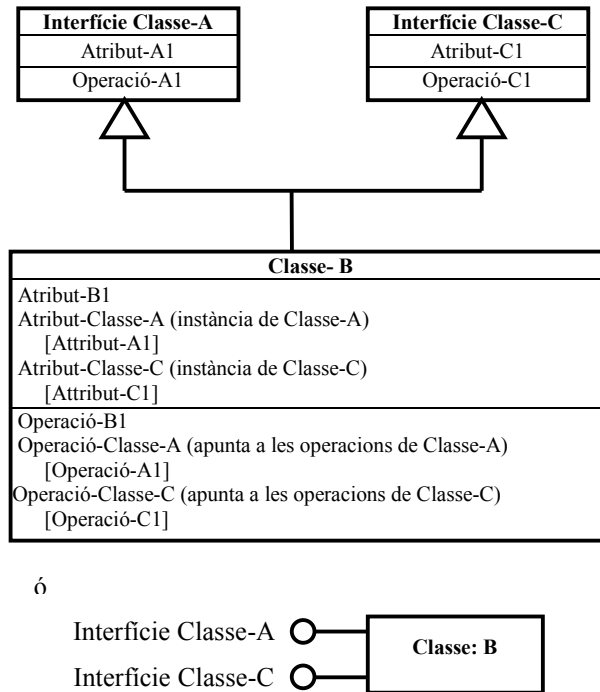


Figura 1.8 Herència de classes interfície.

### Components

Normalment, la gent utilitza la paraula component amb molts significats diferents. A les següents línies veurem el que utilitzarem nosaltres.

Un objecte és un concepte orientat a llenguatge. En general, l'objecte s'escriu en codi font d'un llenguatge específic i hereta d'una jerarquia de classes que ve amb el llenguatge. El objectes es comuniquen mitjançant missatges escrits en el mateix llenguatge, i l'entorn del llenguatge proporciona un context en el qual s'executen els objectes.

D'altra banda, els components són un concepte d'ordre més alt. Un component pot estar compost d'una o més classes, pot incorporar una aplicació heretada escrita en codi procedural. Les classes i el codi contingut dins els components pot estar escrit en diferents llenguatges. Un component ve definit per una o més interfícies escrites en un llenguatge d'interfície de component, no pel codi intern del component. El components s'envien missatges els uns als altres mitjançant un entorn de pas de missatges de llenguatge neutre, o recipient (veure figura 1.9).



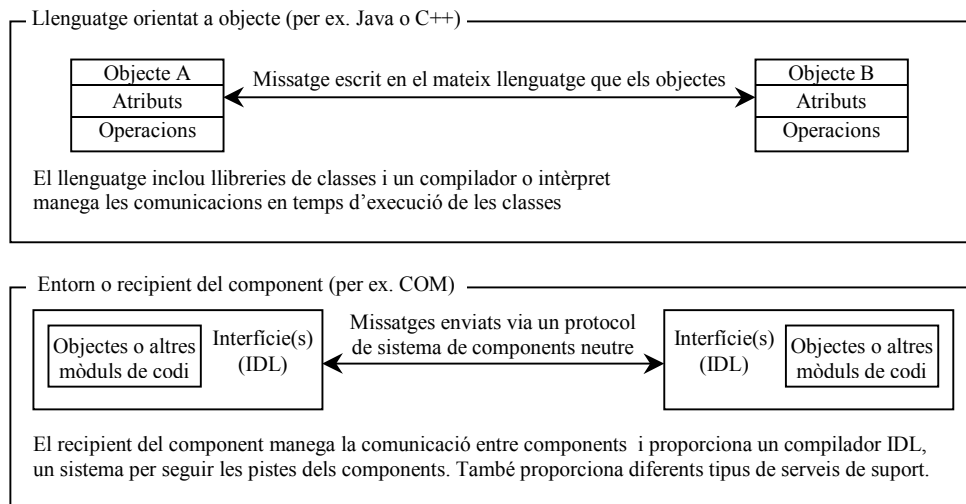


Figura 1.9 Un llenguatge orientat a objectes en comparació a un sistema de components.

### Avantatges de la programació orientada a objectes

- Reusabilitat del codi.
- Disseny més ràpid i de millor qualitat.
- Facilita el manteniment, l'evolució i l'extensió dels models.
- Proporciona una representació consistent per a l'anàlisi i el disseny. Analistes i dissenyadors usen els mateixos models.
- Pensar en termes d'objectes és més natural.

## Conceptes de programació orientada a esdeveniments

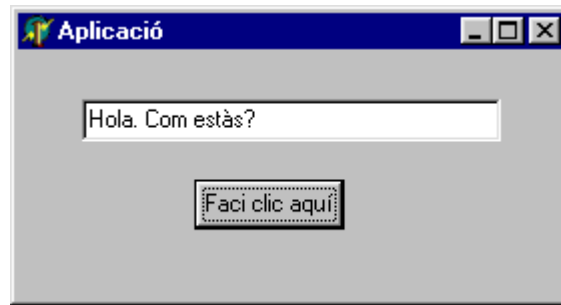
Centrem-nos ara a la programació orientada a esdeveniments contra la programació descendent. Ens hem de fer a la idea de que ja no som propietaris de la màquina i amos del seu comportament, sinó que hem d'observar el seu comportament i respondre a ell.

La programació que realitzàvem sota el sistema operatiu MS-DOS, per exemple amb Pascal, difereix bastant de com es programa una aplicació en el sistema operatiu Windows, per exemple amb Delphi. Un programa escrit per MS-DOS és un conjunt de sentències que s'executen en l'ordre que el programador a dissenyat. Per exemple:

```
WRITELN('Premi ENTER per continuar');
READLN;
WRITELN('Hola. Com estàs?');
```

En aquest exemple, la funció READLN senzillament espera fins que l'usuari prem la tecla ENTER, moment en el qual es passa a la següent instrucció i apareix el missatge *Hola. Com estàs?*. Si a continuació hagués més sentències, l'execució continuaria seqüencialment.

Una aplicació per Windows presenta totes les opcions possibles en una o més finestres perquè l'usuari esculli una d'elles. Això dona lloc a una nova manera de pensar i programar. Per exemple, a la figura següent, quan l'usuari faci clic sobre el botó *Faci clic aquí*, a la caixa de text apareixerà el missatge *Hola. Com estàs?*.



Quan desenvolupem una aplicació sota aquest tipus de programació, la seqüència en que s'executaran les sentències no pot ser prevista pel programador. Per exemple, si en lloc d'un botó hi haguessin dos o més botons, clarament es veu que el programador no pot escriure el programa pensant que l'usuari els premerà en una seqüència determinada.

Quan un sistema operatiu està dirigit per esdeveniments, vol dir que el codi de les aplicacions que s'executen roman inactiu fins que és cridat en resposta a algun esdeveniment com, per exemple, prémer un botó o seleccionar una opció d'un menú. Un sistema operatiu d'aquest tipus està governat per un processador d'esdeveniments (veure figura 1.10). Res no succeeix fins que es detecta un esdeveniment. Un cop es detecta un esdeveniment, el sistema operatiu li comunica a l'aplicació corresponent i el codi associat a aquest esdeveniment s'executa. El control del programa es retorna llavors de nou al processador d'esdeveniments.

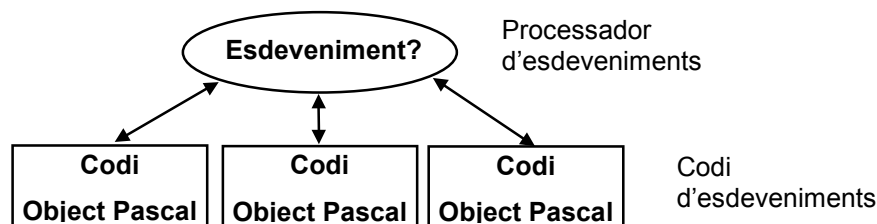


Figura 1.10 Motor d'una interfície gràfica orientada a esdeveniments.

Així doncs, per programar una aplicació orientada a esdeveniments s'ha d'escriure codi separat per a cada objecte, quedant l'aplicació dividida en petites funcions, conduïda cadascuna d'elles per un esdeveniment. Per exemple:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Text := 'Hola. Com estàs?';
end;
```

Per a que es produeixi el resultat que mostra la figura anterior, la funció `Button1Click` ha d'estar lligada al botó titulat *Faci clic aquí* i ha d'executar-se quan es produeixi l'esdeveniment clic sobre ell. Això vol dir que quan un usuari faci clic en aquest botó, s'executarà la funció `Button1Click`. Per això, aquesta manera de programar s'anomena programació orientada, o conduïda, per esdeveniments. Aquesta nova forma de desenvolupar una aplicació s'aparta bastant de la programació descendent, on un programa és un únic bloc d'instruccions que s'executen seqüencialment.

Un esdeveniment, que és una acció reconeguda per un objecte (finestra o control), pot estar causat per l'usuari (per exemple, quan prem una tecla), pel sistema (per exemple, quan transcorre un temps determinat), o indirectament pel codi (per exemple, quan el codi carrega una finestra). A una interfície gràfica, cada finestra i cada control poden respondre a un conjunt d'esdeveniments predefinits. Quan ocorre algun d'aquests esdeveniments, el sistema operatiu envia un missatge a l'objecte per identificar i executar la funció associada amb l'objecte per aquest esdeveniment.

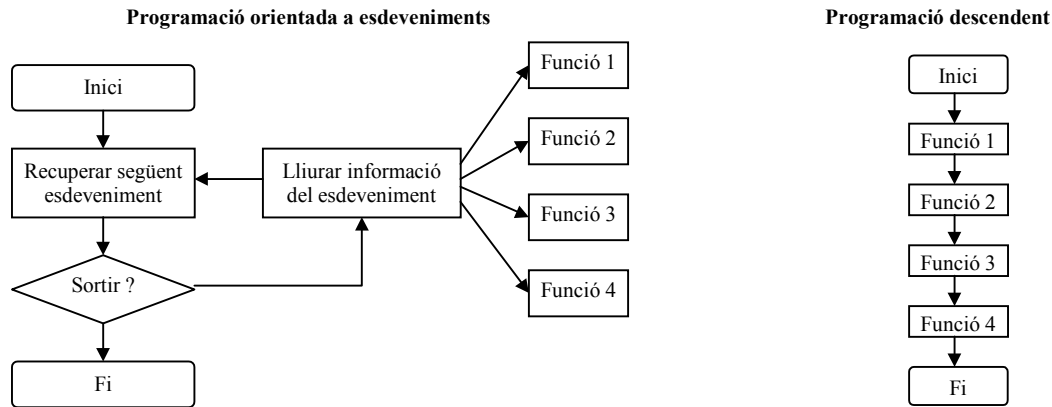


Figura 1.11 Programació orientada a esdeveniments contra programació descendent.

El sistema de missatges d'alguns sistemes operatius fa possible que diverses tasques comparteixin el microprocessador, característica que defineix un sistema multitasca. Això és, el sistema operatiu forma una cua amb els missatges que es produeixen i els distribueix a les aplicacions corresponents. Cada aplicació que rep un missatge, només ha d'executar el procés associat. Això vol dir que la CPU canviarà d'una aplicació a una altre en espais de temps molt curts, el que es coneix com a temps compartit o multiprogramació.

Per que el sistema operatiu mantingui el control de la multitasca, aquest ha d'estar entre l'aplicació i el hardware. Només així pot interceptar qualsevol entrada de l'usuari i enviar el missatge corresponent a l'aplicació apropiada. Això vol dir que tot contacte que la nostre aplicació tingui amb el hardware ha de ser a través del sistema operatiu. Per exemple, una aplicació de Windows no escriu directament sobre la impressora, sinó que utilitza les rutines apropiades del *Kit de Desenvolupament de Software* (SDK) per fer-ho.

## Elements d'una interfície gràfica

### Icones

Les icones són petites imatges que representen tant accions que podem realitzar com elements que formen el nostre sistema, ja siguin dispositius de hardware, aplicacions de software, el punter del ratolí, etc. Aquests elements ajuden molt a fer intuïtives les interfícies gràfiques.



Figura 1.12 D'esquerra a dreta, icones representant una unitat de disquet, una unitat de disc dur i una unitat de cdrom.

### Finestres

Una de les avantatges més grans de treballar amb un entorn de finestres, com per exemple Windows, MacOS o X-Windows, és que totes les finestres es comporten de la mateixa manera i totes les aplicacions utilitzen els mateixos mètodes bàsics (menús descendents, botons, etc.) per introduir ordres.

Una finestra típica d'una aplicació que funciona sota un entorn de finestres consta de les següents parts (veure figura 1.13):

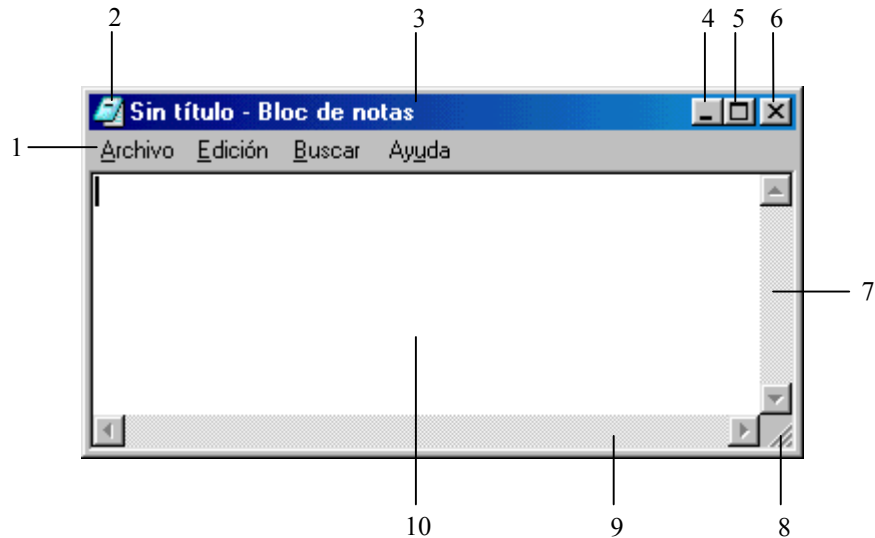


Figura 1.13 Finestra típica de Windows.

1. **Barra de menús.** Visualitza el conjunt dels menús disponibles per aquesta aplicació. Quan algun dels menús s'activa fent clic amb el ratolí sobre el seu títol, es visualitza el conjunt d'ordres que el formen.
2. **Icona de l'aplicació i menú de control.** El menú de control proporciona ordres per restaurar la mida, moure, dimensionar, minimitzar, maximitzar i tancar la finestra.
3. **Barra de títol.** Conté el nom de la finestra i del document. Per moure la finestra a un altre lloc, apuntem amb el ratolí a aquesta barra, fem clic utilitzant el botó esquerre del ratolí i, mantenint premut el botó, arrossegam en la direcció desitjada. Un doble clic maximitza o retorna la mida normal la finestra, depenent això del seu estat natural.
4. **Botó per minimitzar la finestra.** Quan es prem aquest botó, la finestra es redueix a la seva forma mínima. Aquesta és la millor manera de mantenir les aplicacions quan tenim varies d'elles activades i no s'estan fent servir en aquell instant.
5. **Botó per maximitzar la finestra.** Quan es prem aquest botó, la finestra s'amplia al màxim i el botó es transforma en . Si es torna a prémer, la finestra es redueix a la mida anterior.
6. **Botó per tancar la finestra.** Quan es prem aquest botó, es tanca la finestra, i també l'aplicació si la finestra és la principal.
7. **Barra de desplaçament vertical.** Quan la informació no hi cap verticalment en una finestra, s'afegeix una barra de desplaçament vertical a la dreta de la finestra.
8. **Marc de la finestra.** Permet modificar la mida de la finestra. Per canviar la mida hem d'apuntar amb el ratolí a la cantonada o a un costat del marc i, quan el punter canviï a una fletxa doble, amb el botó del ratolí premut, arrosseguem en el sentit adequat per aconseguir la mida desitjada.
9. **Barra de desplaçament horitzontal.** Quan la informació no hi cap horitzontalment en una finestra, s'afegeix una barra de desplaçament horitzontal a sota de la finestra.
10. **Àrea de treball.** És la part de la finestra en la que l'usuari porta a terme la seva tasca, per exemple col·locant text i gràfics.

Moltes altres finestres, com veurem posteriorment a les nostres aplicacions, inclouen també una barra d'eines i una barra d'estat.

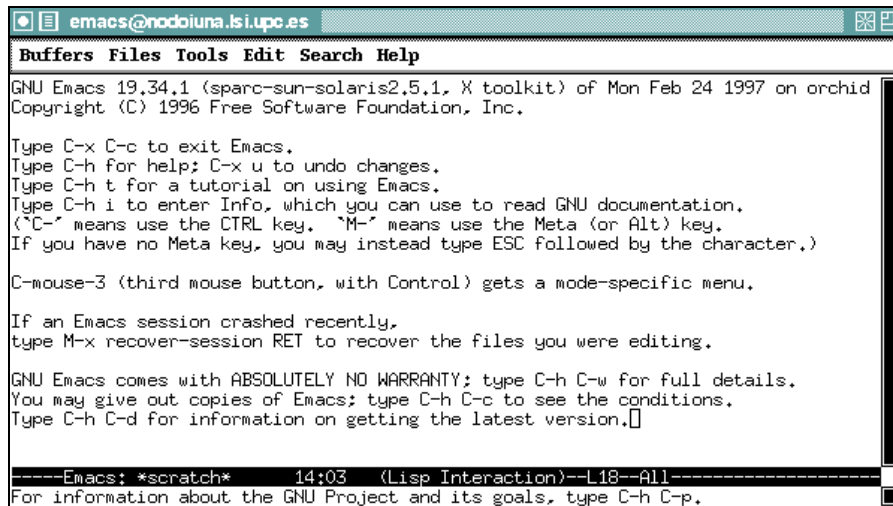


Figura 1.14 Finestra típica de X-Windows.

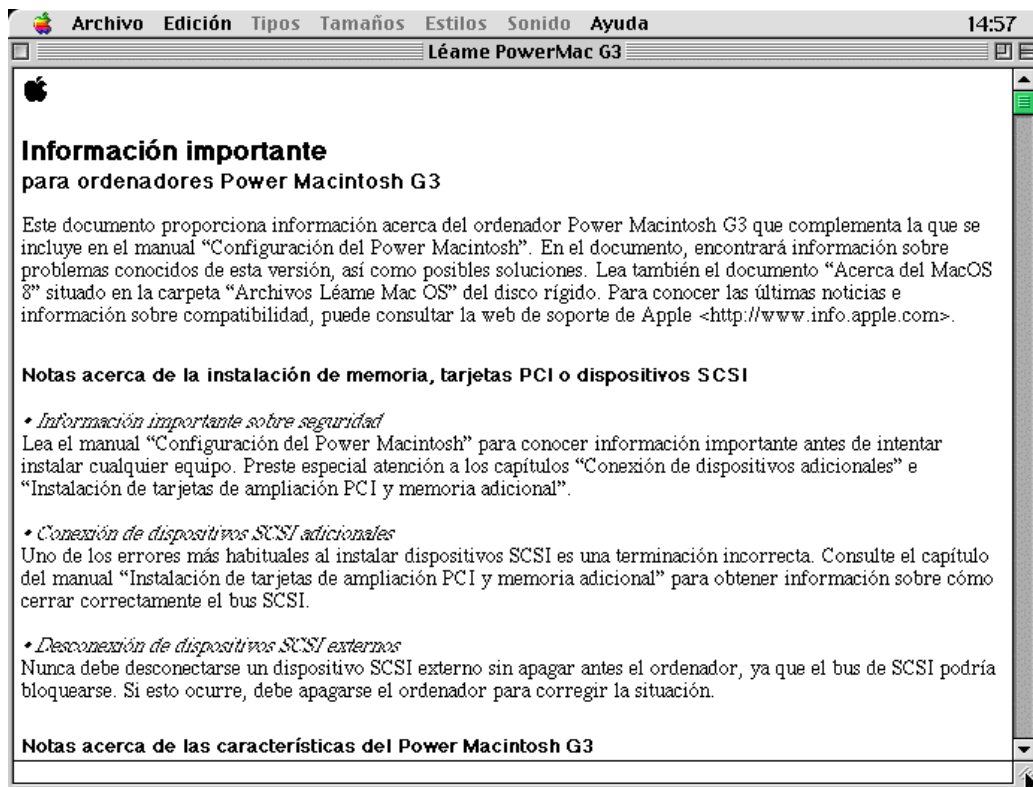


Figura 1.15 Finestra típica de MacOS.

## Elements d'entrada de dades

### Menús



Un menú és una llista desplegable d'opcions. Quan escollim una opció d'un menú pot ser que es realitzi una acció associada a l'opció escollida, o bé que l'opció escollida desplegui de nou un altre menú d'opcions per concretar encara més l'acció a realitzar, formant el que s'anomena un menú multinivell. Als

entorns de finestres bàsicament hi ha dos tipus de menú: el menú principal de l'aplicació i el menú contextual. Aquest últim apareix fent clic amb el botó dret del ratolí sobre algun element, i ens mostra les operacions que sobre aquest element podem realitzar.

### Barres de desplaçament



Les barres de desplaçament són uns elements que proporcionen una manera intuïtiva de moure's a través d'una llista d'informació. Cada barra de conté un quadre de desplaçament que es mou al llarg de la barra per indicar en quina posició ens trobem amb respecte el principi i al final de la informació tractada, i dos fletxes de desplaçament. Per desplaçar-se:

- Una distància petita, com una línia verticalment o un caràcter horitzontalment, utilitzem les fletxes de desplaçament de les barres.
- Una distància més gran, com una pantalla completa, fem clic sobre la barra de desplaçament. Per pujar fem clic a sobre del quadre de desplaçament de la barra vertical, i per baixar fem clic a sota del quadre. Per moure'ns a l'esquerra, fem clic a l'esquerra del quadre de desplaçament de la barra horitzontal, i per moure'ns a la dreta fem clic a la dreta del quadre.
- A un lloc específic, fem clic sobre el quadre de desplaçament i, mantenint el botó del ratolí premut, arrosseguem el quadre.

### Botons



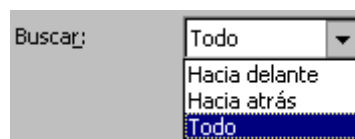
Són els elements més emprats d'una interfície gràfica. S'utilitzen per iniciar, interrompre o finalitzar un procés en particular.

### Caixes de text



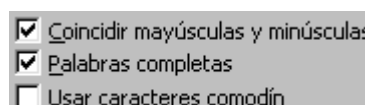
Permeten a l'usuari introduir text. Les caixes de text poden ser d'una línia o de vàries línies. Hi han unes caixes especials de text que visualitzen caràcters asterisc \* enlloc dels caràcters teclejats per l'usuari, i s'utilitzen per demanar contrasenyes.

### Caixes de llista



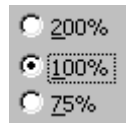
Una caixa de llista visualitza una llista d'elements de la qual l'usuari pot escollir un o, de vegades, més. Si el nombre d'elements excedeix el nombre que pot ser visualitzat, llavors la llista apareix amb una barra de desplaçament.

### Caixes de selecció



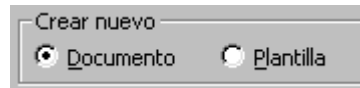
Les caixes de selecció proporcionen una manera d'escollir opcions d'una llista d'opcions candidates a ser seleccionades. Es pot seleccionar algunes, totes o cap de les opcions.

### Botons de radi.



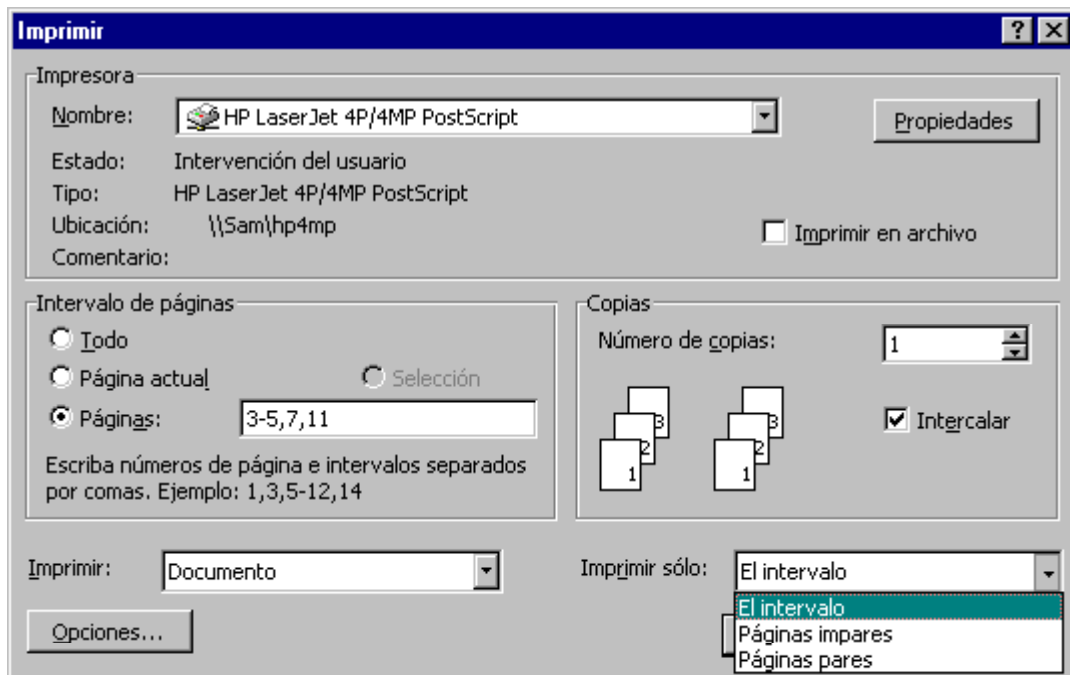
Els botons de radi proporcionen la capacitat de realitzar eleccions mútuament exclusives entre un grup d'opcions candidates a ser seleccionades. Una i només una opció del grup ha d'estar seleccionada.

### Caixes d'agrupació



Les caixes d'agrupació són línies dibuixades al voltant d'altres elements de la interfície gràfica. Milloren la presentació agrupant elements relacionats.

### Quadres de diàleg



Són finestres on apareixen combinacions dels elements d'entrades de dades que acabem de veure (botons, caixes de text, etc.) per sol·licitar informació a l'usuari.

### Elements de sortida de dades

Una interfície gràfica proporciona la informació de sortida a l'usuari en forma de text i tot tipus de gràfics, organitzada en llistes, arbres, etc.. Amb l'adveniment de les tecnologies multimèdia, s'ha convertit en quelcom quotidià proporcionar la informació també en forma de vídeo i de so.



- 1er trim.
- 2on trim.
- 3er trim.
- 4art trim.



### **Altres elements 'no estàndards'**

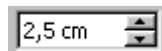
Existeixen multitud d'altres elements, a part dels que hem vist, per introduir i rebre dades de les aplicacions. Aquests elements no estan disponibles a totes les interfícies gràfiques, però sí a la gran majoria. Aquí veurem només uns poc exemples:

#### **Pestanyes**



Permeten organitzar la informació a pantalla agrupant-la en diversos panells. La informació o el quadre de diàleg d'un panell determinat es mostra quan es fa clic amb el ratolí sobre la seva pestanya.

#### **PujaBaixa**



Permet seleccionar una valor numèric, bé introduint-lo per teclat, bé fent clic sobre els botons amb fletxa cap a dalt o cap a baix, que incrementaran o decrementaran el valor actual, respectivament.

#### **Barres de rang**



Permet introduir informació numèrica dins un rang o interval de valors, lliscant la icona d'un element apuntador dins els límits d'una barra.

#### **Barres de progrés**



Quan una aplicació està realitzant un procés lent, aquest element informa l'usuari en quin punt del procés es troba, és a dir, quin percentatge del procés s'ha realitzat i quant falta per acabar.

### **Escriptori**

Els escriptoris no són ben bé elements d'una interfície gràfica, però estan molt relacionats amb aquesta. Els escriptoris són entorns amigables que permeten als usuaris usar i configurar fàcilment els seus ordinadors. Els escriptoris acostumen a incloure un panell per iniciar aplicacions i veure el seu estat, un escriptori on es col·loquen les finestres de les aplicacions, un conjunt d'aplicacions i d'eines de l'escriptori, i un conjunt de convencions que faciliten la cooperació entre aplicacions i la seva consistència amb les altres aplicacions. Un exemple de convenció seria que un objecte en general pot ser mogut a un altre lloc fent clic a sobre d'ell i arrossegant-lo mantenint premut un botó del ratolí.

Normalment els escriptoris són molt configurables, permetent-nos ajustar la manera en que ens agrada que aparegui i es comporti.



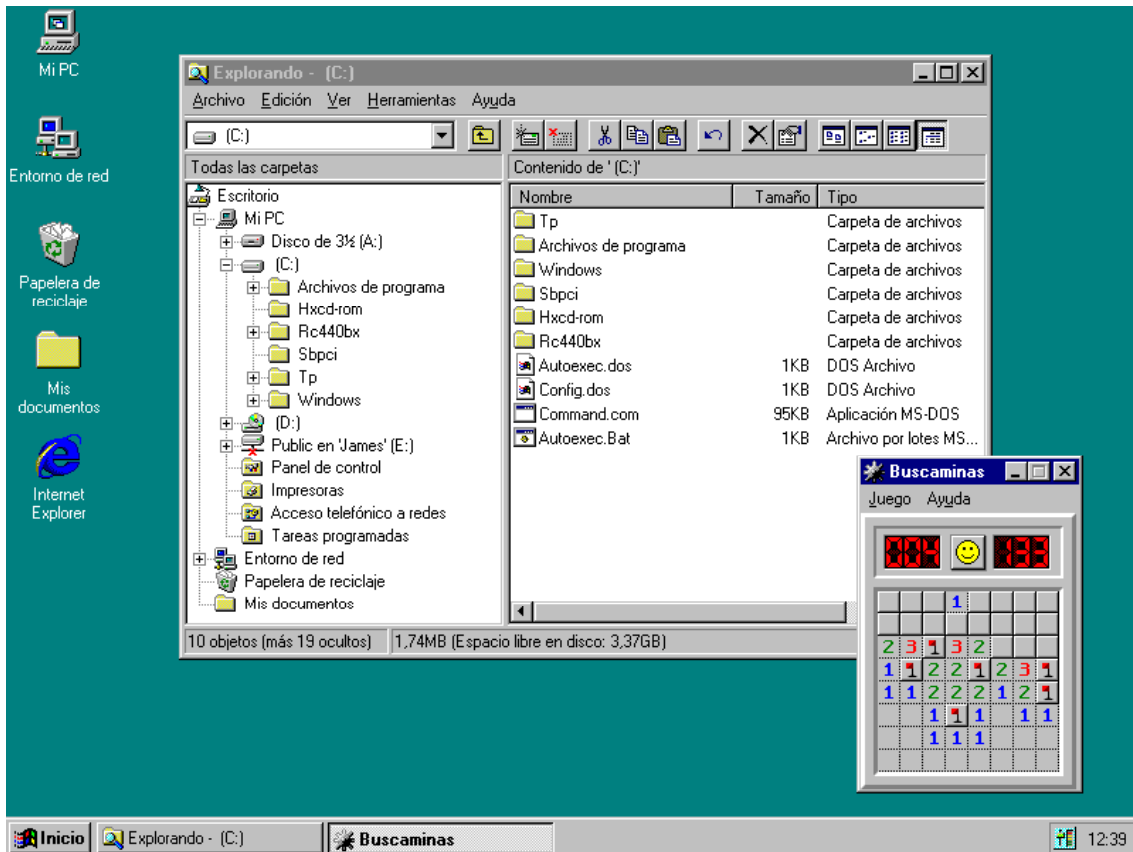


Figura 1.16 Escritori de Windows 95.

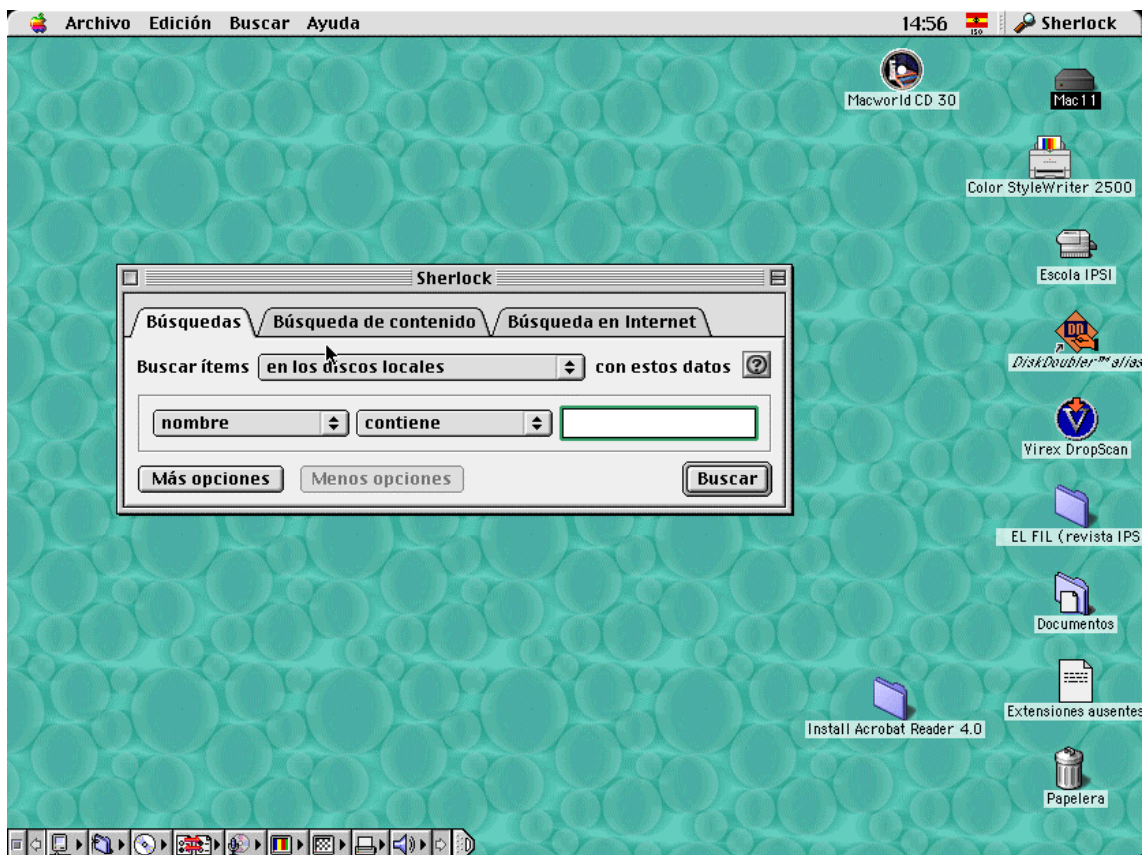


Figura 1.17 Escritori de MacOS 8.1.

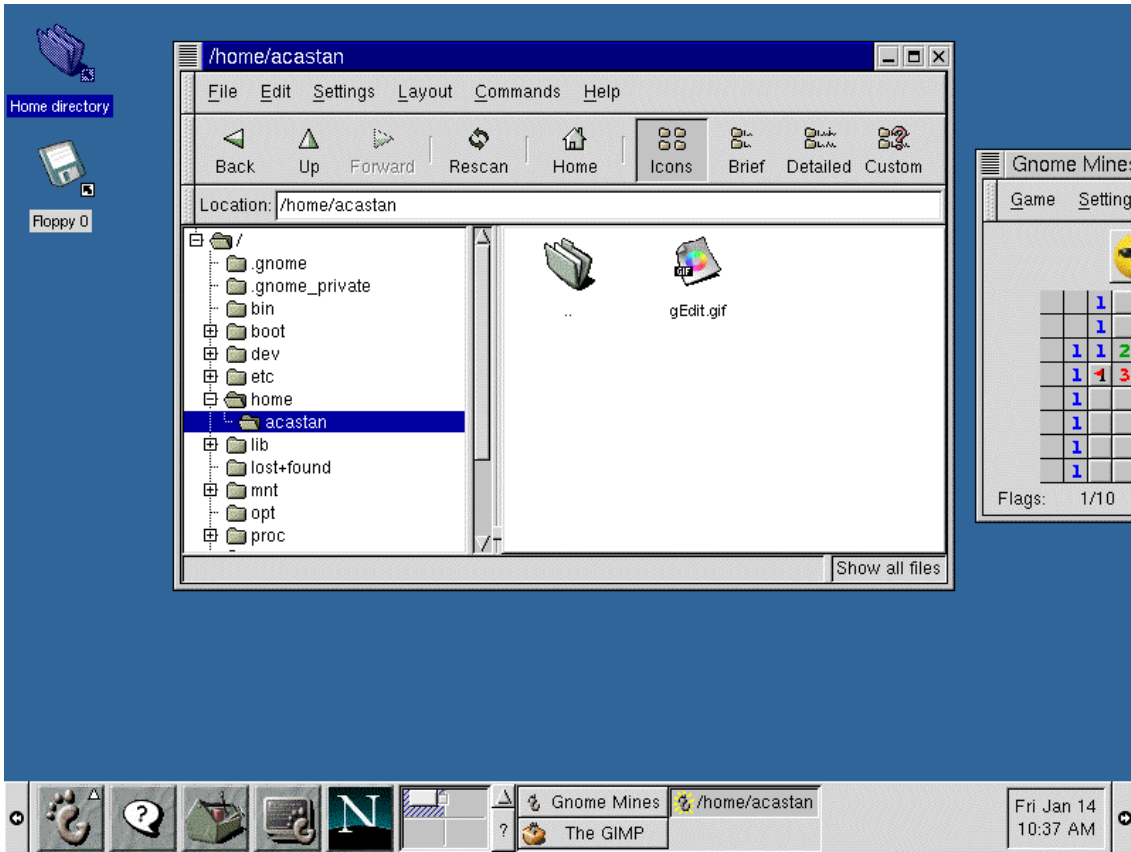


Figura 1.18 Escritori de Gnome 1.0.

## Exemples d'interfícies gràfiques

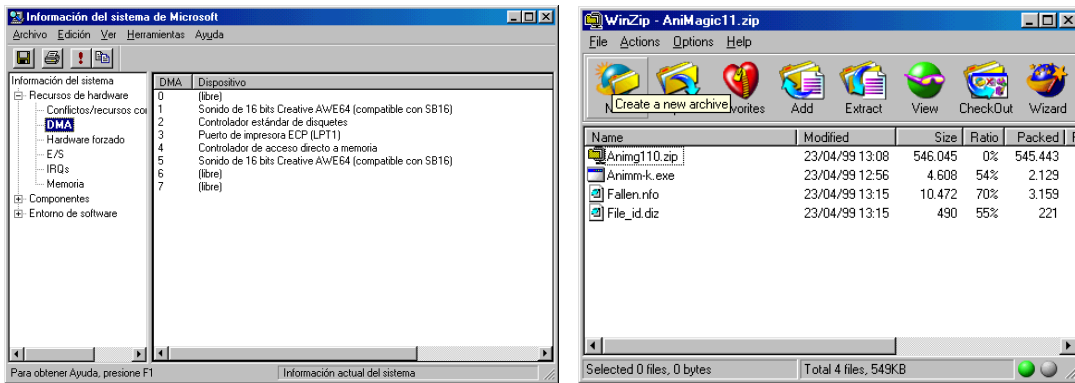


Figura 1.19 Interfícies gràfiques 'estàndard' de Windows: marc estàndard, menú, barra d'eines amb icones, barra d'estat, espai de treball amb barres de desplaçament, etc.





**Figura 1.20** Interfícies gràfiques 'no estàndard' de Windows. Les icones de tots els components han estat modificades. No obstant, les interfícies són intuïtives i fàcils d'utilitzar degut a la seva senzillesa i a la seva similitud amb les interfícies de dispositius de la vida diària amb els quals ja estem familiaritzats (barres de desplaçament imitant equalitzadors d'un equip de música, etc.).

## Repàs de normes i consells de programació.

Quan es crea un programa d'ordinador, és important agafar un enfocament estructurat. S'han de complir certs passos en un cert ordre i ens hauríem d'acostumar a fer les coses correctament. Sempre es temptador començar a dissenyar i codificar el programa immediatament, però el fet d'emprar algun temps en la planificació ens estalviarà molts maldecaps posteriors.

Els passos clau en la creació d'un programa d'ordinador són els següents:

1. Planificar les tasques a fer pel programa, és a dir, com ha de funcionar.
2. Dissenyar la interfície d'usuari, és a dir, l'aspecte que ha de tenir.
3. Escriure el codi del programa. Es tracta de la implementació dels passos 1 i 2.
4. Provar i depurar el programa. Això inclou la prova en beta amb usuaris que estan fora de l'equip de desenvolupament.
5. Documentar i distribuir el programa.

Aquests passos són molt generals. Ara veurem més detingudament cadascun d'aquests.

### **Disseny de l'aplicació**

El procés de disseny hauria de produir els següents resultats:

- Una llista concisa de les tasques a realitzar pel programa.
- Termini d'acabament de tasques determinades.
- Clarificació de la dependència entre les diferents parts del programa.
- Els criteris de prova del programa.

Com a consells o guies més importants a donar en el procés de disseny tenim les següents:

### **Entendre el problema**

Començar a treballar en un projecte pot ser una tasca intimidant, especialment si és un programa llarg i complex. Necessitarem decidir el propòsit pel qual serà feta l'aplicació, quines característiques suportarà, com haurà de semblar i comportar-se la interfície d'usuari, com haurà d'estar estructurat el programa, i moltes coses més.

### **Descompondre el problema**

Si descomponem un programa llarg en components individuals més petits, podem tractar cada component o part del programa com una entitat independent, fent la tasca de programació més senzilla i l'aplicació més flexible.

Si manipulem cada component de manera independent, podem modificar un component sense necessitat de redissenyar els altres.

### Integrar la solució

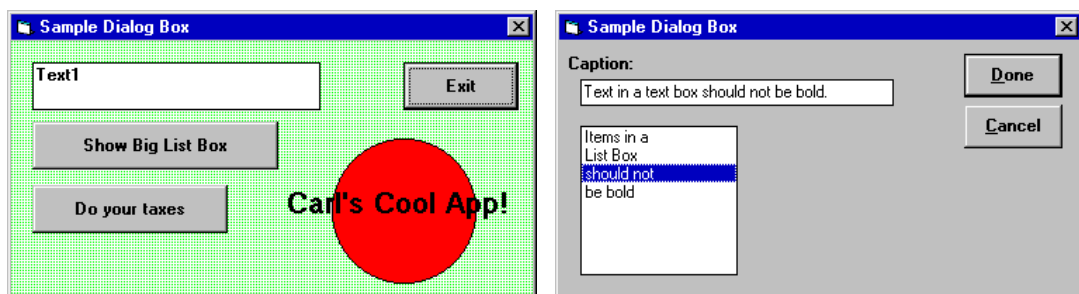
Un cop tinguem plenament definit el problema i descompost en els seus components independents, començarem a definir les interfícies entre els components. Quan hi ha interfícies estables i ben definides entre els components, podem modificar el funcionament intern de qualsevol component sense haver de redissenyar l'aplicació.

### Disseny de la interfície gràfica

L'objectiu últim d'una interfície d'usuari és que l'ús dels objectes es faci evident senzillament mirant-los, bé sigui perquè representen objectes coneguts per l'usuari (metàfores), bé sigui perquè l'usuari ja ha fet servir objectes similars a altres aplicacions.

Part de l'èxit del sistema operatiu Windows i de les aplicacions de Microsoft en particular és degut a les similituds entre les interfícies d'usuari d'aquestes aplicacions. L'usuari gaudeix d'un gran avantatge si té idea de com fer anar bé l'aplicació des de el primer moment. Això només pot passar si el disseny de la interfície té l'aspecte, la resposta i la funcionalitat de les aplicacions que ja s'estan usant. Per tal d'aconseguir això podem:

- Seguir les convencions establertes per la disposició dels elements de la interfície. Això donarà a la nostra aplicació aparença professional i a l'usuari un grau extra de familiaritat amb l'aplicació. La clau rau en la consistència. No dissenyem tres finestres diferents, cadascuna amb un color de fons cridaner. No tinguem un botó 'OK' a una finestra, un botó 'Sortir' a l'altre, i un botó 'Anul·lar' a l'altre. Una de les idees més potents que hi ha darrera dels entorns basats en finestres és l'anomenat *Accés Comú d'Usuari*: que cada finestra pugui canviar-se de mida, obrir-se, tancar-se, minimitzar-se i maximitzar-se amb el mateix mètode; que els menús estiguin al mateix lloc; etc. Així l'usuari no ha de gastar un munt de temps aprenent com controlar una aplicació. Un cop s'ha après una aplicació, aprendre noves aplicacions pren menys temps.



**Figura 1.21** A l'esquerre, exemple de quadre de diàleg mal dissenyat: fons cridaner, massa botons (per aquestes ocasions tenim els menús), caixa de text amb lletra massa gran i en negreta, ... A la dreta, quadre de diàleg ben dissenyat. Aquest quadre de diàleg podria servir d'estàndard en el sistema operatiu Windows: estil de cantonada doble fixa, color de fons gris clar, caixes de text i llistes amb caràcters sense negreta, el botó 'Done' seleccionat per defecte i s'activa si l'usuari prem Enter, el botó 'Cancel' s'activa si l'usuari prem Escape.

- Emprar color per ajudar a transmetre informació, no per amagar-la. Hi ha gent que considera molt maques les finestres amb molts colors i dibuixos cridaners, però recordeu que si la vostra aplicació ha de definir la feina d'algú, si algú seurà per utilitzar la vostra aplicació durant tot el dia, això pot ser dur pels ulls. El millor lloc per un logotip no és la finestra principal de l'aplicació, sinó una finestra prèvia que aparegui quan es carrega l'aplicació.
- Intentar no saturar l'usuari amb dotzenes d'icones i menús multicapa. Encara que la nostra aplicació sigui molt complexa i presenti moltes opcions, hem d'intentar que la interfície amb l'usuari romangui el més clara i senzilla possible. Pel mateix motiu, és millor no emprar moltes finestres. Les aplicacions més elegants només utilitzen una finestra i es comuniquen amb l'usuari mitjançant quadres de diàleg estàndard.

- Posar el control a les mans de l'usuari. Els entorns de finestres dels actuals sistemes operatius proveeixen l'usuari d'una gran llista d'opcions de personalitzar el seu entorn. Aprofitem-les a les nostres aplicacions emprant les personalitzacions que els usuaris ja han definit, com colors, format de la data i hora, etc. Així tindrem una interfície més consistent, a més a més d'estalviar-nos el problema de construir una interfície per personalitzar les propietats a la nostra aplicació.
- Emprar les eines del sistema mentre estiguin disponibles, com per exemple els quadres de diàleg per obrir un arxiu o seleccionar un color. Això donarà als usuaris l'avantatge de saber com utilitzar parts de la nostra aplicació abans d'haver-les fet servir.

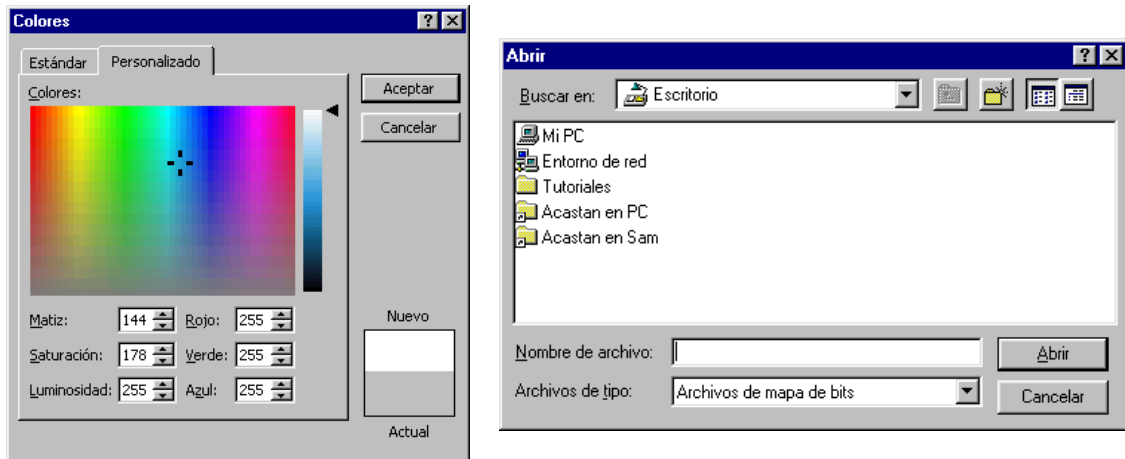


Figura 1.22 Eines del sistema operatiu Windows per seleccionar un color i per obrir un fitxer.

- No assumir que la nostra aplicació és l'únic programa en ús. Així que no intentem controlar propietats que normalment s'encarrega l'usuari de manipular, com la resolució de pantalla, l'estat de la finestra de l'aplicació, etc.
- Integrar la nostra aplicació amb el sistema operatiu. La nostra aplicació hauria de suportar les extensions comuns del sistema operatiu, com registrar i associar tipus de fitxer apropiadament amb la nostra aplicació, suportant arrossegar i llençar si fos aplicable, i suportant característiques de correu electrònic.

També cal tenir en compte a l'hora de dissenyar una interfície gràfica:

- Informar l'usuari de tot el que està succeint i les accions que aquest pot realitzar mitjançant missatges precisos i aclaridors. Emprar barres de progrés o transformar la icona del ratolí en un rellotge quan l'aplicació està realitzant càlculs llargs.
- Assegurar que totes les persones són capaces d'utilitzar l'aplicació. A Europa, al voltant del 20% de la població (80 milions de persones) té més de 65 anys o algun tipus de disminució: poca o cap visió, poca o cap oïda, mobilitat limitada, problemes cognitius, desconeixement de l'idioma, etc.

### **Estàndards de codificació**

És un punt clau a recordar el que, més tard o més d'hora, algú diferent de nosaltres haurà de llegir i entendre el nostre codi. Com a consells o guies més importants a donar en el procés de disseny tenim les següents:

### **Composició i estil**

La composició del nostre codi tindrà un impacte significatiu en l'habilitat de llegir-lo posteriorment. Les guies bàsiques de composició a fer servir són les següents:

- Escriure un comentari d'una o varies línies a l'inici de cada mòdul del programa i procediment. Aquest comentari hauria d'indicar el propòsit de la següent secció de codi, més que no pas la implementació.
- Sagnar d'una manera apropiada el codi corresponent als procediments, instruccions condicionals i bucles. Amb dos o tres espais n'hi haurà prou. Això farà substancialment més fàcil de seguir la lògica del codi.
- Afegir línies en blanc entre seccions de codi, per separar grups de declaracions de dades i blocs de codi, fa més senzill de llegir el programa i més senzill de seguir la lògica.
- Afegir comentaris d'una línia per descriure les declaracions de dades i els blocs de codi. A més a més, s'haurien d'afegir comentaris a qualsevol punt del programa on el propòsit d'una sentència o grup de sentències no és obvi.

### Mòduls

Quan escrivim aplicacions sèries que utilitzen un gran nombre de variables globals, probablement perdem bastant temps eliminant errors causats per accessos no apropiats a les dades globals. Fent servir dades a nivell de mòduls i rutines d'accés per les dades de naturalesa global, podem controlar al màxim els mitjans emprats per llegir i escriure les dades i coordinar procediments i dades relacionats.

Hi han dues maneres bàsiques d'organitzar els procediments en un mòdul:

1. Agrupant els procediments que utilitzen unes mateixes dades. Això permet protegir l'accés a les dades compartides per varis procediments. A més a més, ja que alguns llenguatges de programació visual carreguen els mòduls sota demanda, podem utilitzar la memòria d'una manera més eficient agrupant el codi que utilitza les dades compartides.
2. Agrupant els procediments que realitzen tasques relacionades.

Un altre benefici d'un mòdul ben organitzat és que sovint el podem fer servir a una nova aplicació amb poques o cap modificacions, estalviant-nos un munt d'esforç de codificació en nous projectes.

### Procediments

Les guies bàsiques de composició a fer servir són les següents:

- Cada procediment només hauria de realitzar una tasca.
- Donar al procediment un nom senzill i clar que indiqui el que fa. Si això no es pot realitzar és que probablement necessitem reescriure el codi.
- Separar el maneig de dades de la interfície d'usuari. Escriure procediments separats per manejar les dades que cridem des del codi de la interfície d'usuari, ens permet redissenyar la interfície (que és el canvi més comú) sense redissenyar el codi que manega les dades.

### Codificar per la eficiència

Quan mirem de millorar l'eficiència d'una aplicació, hi han tres camps a considerar:

- L'eficiència a l'inici, que és el temps que tarda una aplicació en carregar i començar a executar-se.
- L'eficiència real, que és la velocitat actual a la qual el codi s'executa.
- L'eficiència percebuda, que és l'eficiència tal com és percebuda per l'usuari.

A continuació veurem uns quants consells per millorar aquesta eficiència. Tingueu en ment, però, que una bona arquitectura i disseny de l'aplicació poden fer més per millorar el rendiment de l'aplicació que el codi més precís i acurat.

L'eficiència a l'inici és crítica en el moment que s'inicia l'aplicació, però també s'aplica a la carrega i inici de tot formulari. Si en començar la nostra aplicació, aquesta és ràpidament visible i preparada per l'entrada de dades per part de l'usuari, presentarà l'aparença de ser molt ràpida. Hi han varis factors que poden afectar a l'eficiència de l'aplicació en general i a la dels formularis individuals en particular:

- Reduir la quantitat de treball a fer al programa principal i als esdeveniments d'inicialització i càrrega dels formularis. Tot codi que corre abans de que el primer formulari sigui visible significa temps que l'usuari a d'esperar per veure una resposta de la nostra aplicació.
- Reduir el nombre de controls dels formularis. Els controls consumeixen memòria i recursos, i els controls a mida generalment consumeixen més que els controls estàndard de l'entorn de finestres.
- Fer servir vectors de controls. Els vectors de controls ajuden a reduir el consum de recursos i també poden reduir l'esforç de codificació consolidant els procediments dels esdeveniments en un únic procediment per a tot el vector.
- Reduir els gràfics al mínim. La riquesa de gràfics a les interfícies, encara que atractiva, consumeix un munt de memòria i pren un munt de cicles de computador per dibuixar-se.
- Mostrar una pantalla de presentació. Qualsevol aplicació no trivial necessita fer feina de posta a punt en el temps d'inici. Millor que deixar l'usuari assegut davant l'ordinador preguntant-se si el programa està treballant, és mostrar una pantalla senzilla i maca per donar-li quelcom que mirar mentre el codi d'inicialització acaba d'executar-se. Encara que una pantalla d'inici no faci res per aconseguir que l'aplicació comenci més ràpid, millora l'aparença de l'aplicació i fa saber a l'usuari que l'aplicació està treballant.
- Carregar únicament les dades que calguin. Tot i el rendiment dels sistemes de discos d'avui en dia, l'entrada i sortida a fitxers és uns quants ordres de magnitud més lenta que l'accés directe a memòria. Recuperar dades del disc, ja sigui mitjançant fitxers d'inicialització, fitxers de text, fitxers d'accés directe, fitxers binaris a mida, o una base de dades local o remota, és un procés que requereix temps. Carregant únicament les dades que necessitem podem reduir substancialment la quantitat de sobrecàrrega d'entrada i sortida.

A la gran majoria dels casos l'eficiència real és irrellevant, i sovint es sacrifica la velocitat per aconseguir més flexibilitat. Però quan necessitem que el codi de l'aplicació treballi el més ràpid possible, podem aplicar una varietat de tècniques per ajustar el codi que incrementaran el rendiment:

- No fer ús del tipus variant. Les variables de tipus variant, encara que flexibles, són notablement lentes. A menys que la situació requereixi l'ús de variants per manipular Nulls i tipus de dades desconeguts en temps de disseny i compilació, treballem amb els tipus de dades bàsics.
- Fem servir enters quan sigui possible. Els enters han estat el tipus de dada natiu de gairebé tots els microprocessadors, així que el hardware sempre realitzarà les operacions sobre enters més ràpid que les operacions amb punt flotant i variant.
- Fem servir la instrucció `with` quan sigui possible. Tota referència a un objecte pren uns quants cicles de computació per resoldre's, però emprant `with` només cal resoldre les referències a l'objecte un cop. Sense `with` les referències a l'objecte s'han de resoldre per a cada declaració d'aquest.
- Guardar a variables locals 'caché' els valors de les propietats. Com que les operacions de lectura i escriptura de propietats són operacions costoses, si necessitem accedir més d'un cop en un procediment al valor d'una propietat, és millor guardar temporalment el valor d'aquesta propietat a una variable local i fer servir la variable en lloc de llegir la propietat directament.
- Treure tan codi com sigui possible de les iteracions. Si les iteracions estan niuades, intenteu moure tan codi com sigui possible cap al nivell més extern de niat. També, sempre que sigui possible, evitar realitzar referències a objectes o lectures i escriptures en el cos de la iteració.
- Guardar a memòria 'cache' els fitxers. Les dades que s'usen freqüentment i que estan emmagatzemades a disc, haurien d'estar a memòria. No cal abusar d'aquesta tècnica o forçarem al sistema operatiu a intercanviar moltes dades de la memòria RAM al fitxer o disc de paginació (memòria virtual), perdent els beneficis que havíem obtingut prèviament.
- Guardar localment les dades remotes. Si hem de buscar les nostres dades a un disc de xarxa o un servidor remot de bases de dades, podem intentar guardar localment les dades més freqüentment usades. Avui en dia encara és més ràpid accedir a les dades en memòria o disc que a través de les últimes i més ràpides xarxes Ethernet.
- No buscar dades que no necessitem. Si, per exemple, utilitzem la instrucció `SELECT *` rutinàriament a les nostres consultes a bases de dades, segurament estem rebent dades que no necessitem. Com que

cada byte de dades mogut del disc a la nostra aplicació consumeix temps i memòria, és un malbaratament demanar dades que després no fem servir.

- Depurar el rendiment brut emprant eines per mesurar el rendiment i ajustant posteriorment el codi de l'aplicació que ho necessiti.

L'eficiència percebuda és la més important, ja que es refereix a la percepció que té l'usuari de la velocitat de la nostra aplicació. Hi han dues maneres bàsiques d'incrementar l'eficiència percebuda d'una aplicació:

1. Distraient a l'usuari durant els processos llargs, mitjançant pantalles impactants, imatges maques i mesuradors de progrés del procés.
2. Donant als usuaris alguna feina a fer durant els processos llargs. Quan sigui possible, s'hauria de moure les tasques que consumeixen temps a processos en segon pla i permetre l'usuari que continuï realitzant una altra tasca mentre s'executa aquest procés.

#### **Els nou pecats capitals del programador**

1. Usar el tipus de dada `variant`. No declarar explícitament el tipus de dada de totes les variables i funcions.
2. Escriure codi il·legible.  
Escriure procediments més llargs que dos o tres pantalles de codi.  
No seguir les convencions de nomenament de variables, objectes i procediments.  
Utilitzar els noms d'objectes proveïts per defecte pels entorns de desenvolupament visual.
3. Codificar valors que haurien de ser calculats o definits per l'usuari: nombres màgics, noms de fitxers i camins, etc.
4. Manca d'atenció cap a les necessitats de l'usuari final.  
Ús de llenguatge i argot inadequats.  
Missatges d'error poc amistosos.  
Manca d'atenció cap a l'ergonomia.  
Alineació pobre dels controls.  
Desviar-se sense cap bona raó de les guies de la interfície estàndard d'usuari amb finestres.
5. Escriure codi fràgil.  
No utilitzar els manipuladors d'errors apropiats.  
No anticipar o recuperar-se amb elegància dels errors més comuns.
6. No escriure codi reutilitzable.  
Utilitzar controls a mida per fer senzilles tasques de programació.  
Posar massa codi en els procediments dels esdeveniments.  
No escriure procediments dedicats per a les tasques més comunes.
7. Documentació pobre.  
No escriure comentaris a les capçaleres i al codi font.  
No escriure documentació de suport.
8. Ús no apropiat de dades.  
No limitar l'abast de les variables.  
No validar les entrades i sortides.  
Usar tipus de dades no apropiats.
9. Ús no apropiat de l'assistència dels companys programadors.  
Demanar ajut sense fer l'esforç d'aprendre.  
No demanar ajut en els problemes complexos



# L'entorn de treball de Delphi

## L'entorn de treball de Delphi

### Modes de treball

Delphi opera en tres modes de treball:

- Mode de disseny, utilitzat per construir l'aplicació. Per ara ens centrarem en aquest mode.
- Mode d'execució, utilitzat per executar l'aplicació.
- Mode de depuració, on l'aplicació està parada i el depurador de codi està disponible.

### Finestres

L'entorn de treball de Delphi 5 està dividit en quatre seccions: la finestra principal, el dissenyador de formularis, l'inspector d'objectes i l'editor de codi (veure figura 3.1).

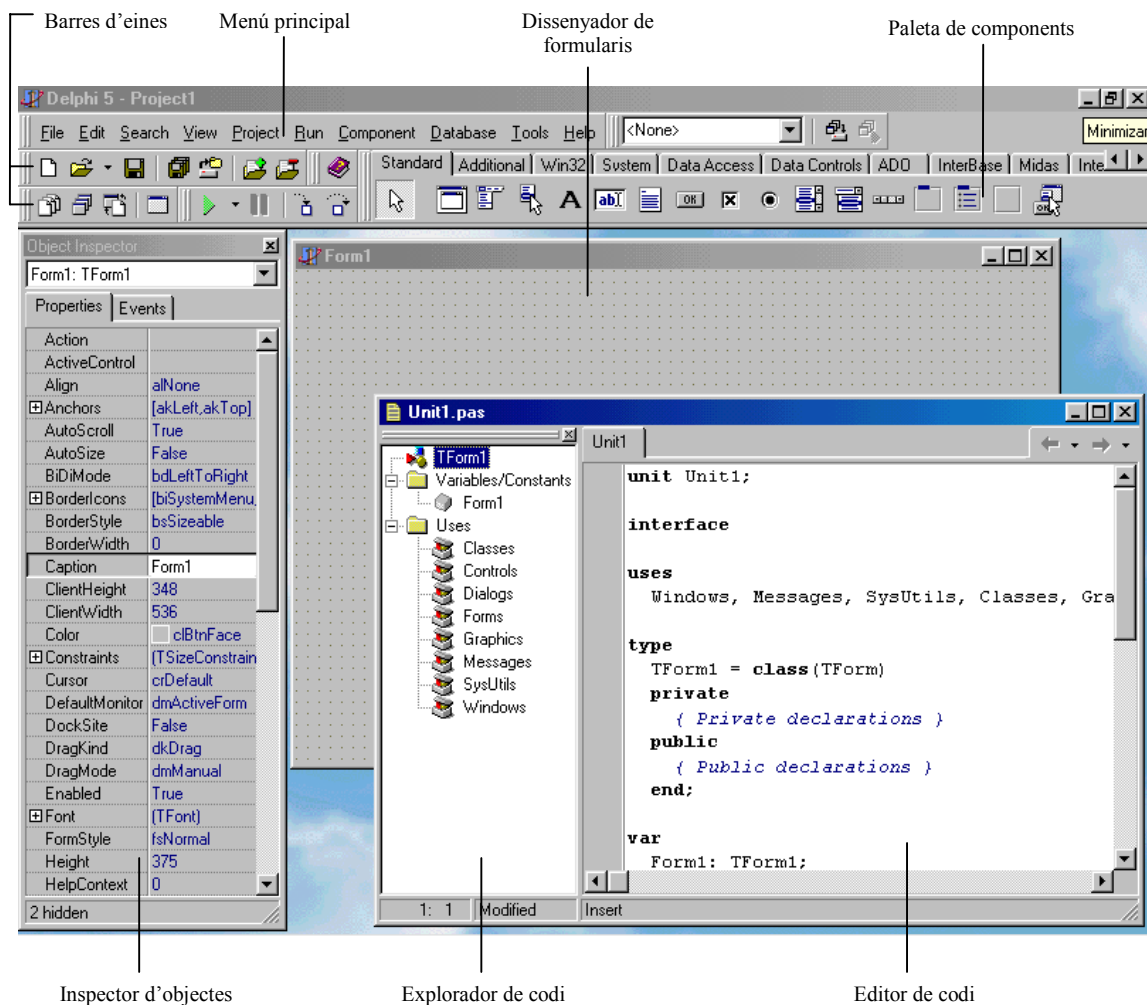


Figura 3.1 Entorn de treball de Delphi 5.

La finestra principal és el centre de control de l'entorn de treball de Delphi 5. Consisteix en un menú principal, les barres d'eines i una paleta de components. Cada botó d'una barra d'eines correspon a alguna


funció de l'entorn de treball, com obrir un fitxer o construir un projecte. Hi han quatre barres d'eines separades a la finestra principal: la barra estàndard, la barra de vista, la barra de depuració i la barra feta a mida.

El dissenyador de formularis es pot considerar com la tela on l'artista crea les seves aplicacions basades en finestres. Aquí és on es determina com les aplicacions seran presentades visualment als usuaris. Es tracta d'interactuar amb el dissenyador de formularis seleccionant components de la paleta de components i deixant-los caure a sobre el formulari. Un cop tenim un determinat component sobre el formulari, podem fer servir el punter del ratolí per ajustar la posició i mida del component. Podem controlar l'aparença i el comportament d'aquests components fent servir l'inspector d'objectes i l'editor de codi.

Amb l'inspector d'objectes podem modificar les propietats d'un component o formulari, o permetre al nostre component o formulari respondre a diferents esdeveniments. Les propietats són dades com alçada, color i font, i determinen com un objecte apareixerà sobre la pantalla. Els esdeveniments són porcions de codi que s'executen en resposta a ocurrències dins l'aplicació. Dos exemples d'esdeveniments podrien ser un clic del ratolí o un missatge dirigit a una finestra perquè es redibuixi.

L'editor de codi és on es realitza la programació (en el sentit estricte) de les aplicacions. És on escrivim el codi que dictamina com és comportarà el nostre programa, i on Delphi insereix el codi que genera basat en els components de la nostre aplicació. Juntament amb l'editor de codi trobem l'explorador de codi, que ens permet navegar fàcilment entre unitats i afegir nous elements o tornar a anomenar els elements existents dins una unitat.

### Ajuda

Dins l'entorn de treball de Delphi es pot obtenir ajuda sobre qualsevol aspecte del llenguatge Object Pascal, dels components VCL o del mateix entorn, a la icona  de la barra d'eines i a l'opció *Help* → *Delphi Help* del menú principal. A més a més, es pot trobar una amplia informació sobre les diferents eines de desenvolupament que acompanyen Delphi a la opció *Help* → *Delphi Tools* del mateix menú.

També disposem d'ajuda sensible al context quan premem la tecla **F1**:

- Si estem al menú principal, rebrem ajuda sobre l'opció del menú apuntada pel ratolí.
- Si estem a un quadre de diàleg, rebrem ajuda sobre el seu funcionament i les diferents opcions que conté.
- Si estem al dissenyador de formularis, rebrem ajuda sobre el component que en aquell moment tenim seleccionat.
- Si estem treballant amb l'inspector d'objectes, rebrem ajuda sobre la propietat o esdeveniment que tinguem seleccionat.
- Si estem treballant amb l'editor de codi, rebrem ajuda sobre la paraula del llenguatge Object Pascal sobre la que tenim el cursor.
- Si estem compilant o linkant el programa i apareixen missatges d'error a una finestra especial sota l'editor de codi, rebrem ajuda detallada de l'error que tinguem seleccionat.

Delphi també inclou diverses ajudes a l'escriptura de codi, que veurem a continuació. Les cinc primeres eines són les anomenades *Code Insight*, i presenten finestres emergents sensibles al context a l'editor de codi, i la última és la eina anomenada *Class Completion*:

- *Code Completion*. Escriviu el nom d'una classe seguit d'un punt per veure la llista de propietats, mètodes i successos apropiats per la classe. Escriviu l'inici d'una sentència d'assignació i premeu **Ctrl** + **espai**; això mostrarà una llista de valors vàlids per a la variable.
- *Code Parameters*. Per veure la sintaxi dels arguments d'un mètode, escriviu el nom del mètode i un signe d'obrir parèntesi.
- *Code Templates*. Prement **Ctrl** + **J** apareixerà una llista de les sentències de programació més comuns que poden inserir-se al codi. Poden crear-se plantilles addicionals a les subministrades amb Delphi.
- *Tooltip expression evaluation*. Quan el programa està en pausa durant la depuració, assenyalant qualsevol variable es mostra el seu valor actual.

- *Tooltip Symbol Insight*. Durant l'edició de codi, assenyalant qualsevol identificador es mostra la seva declaració.
- *Class Completion*. Aquesta eina genera l'esquelet de codi de les classes. Si situeu el cursor a qualsevol posició dins una declaració de classe, i a continuació premeu **Ctrl** + **Majús** + **C**, o feu clic amb el botó dret i seleccioneu al menú contextual *Complete Class at Cursor*, Delphi afegeix automàticament tots els especificadors privats `read` i `write` a les declaracions per a totes les propietats que ho necessitin i a continuació crea el codi estructural per a tots els mètodes de la classe. Per exemple, si teníem escrita la següent porció de codi a la secció *interface* d'una unitat:

```
type TMyButton = class(TButton)
  property Size: Integer;
  procedure DoSomething;
end;
```

el completat de classes afegirà especificadors de lectura i escriptura a la declaració d'interfície:

```
type TMyButton = class(TButton)
  property Size: Integer read FSize write SetSize;
private
  FSize: Integer;
  procedure SetSize(const Value: Integer);
```

i afegirà el següent codi a la secció d'implementació de la unitat:

```
{ TMyButton }
procedure TMyButton.DoSomething;
begin
end;

procedure TMyButton.SetSize(const Value: Integer);
begin
  FSize := Value;
end;
```

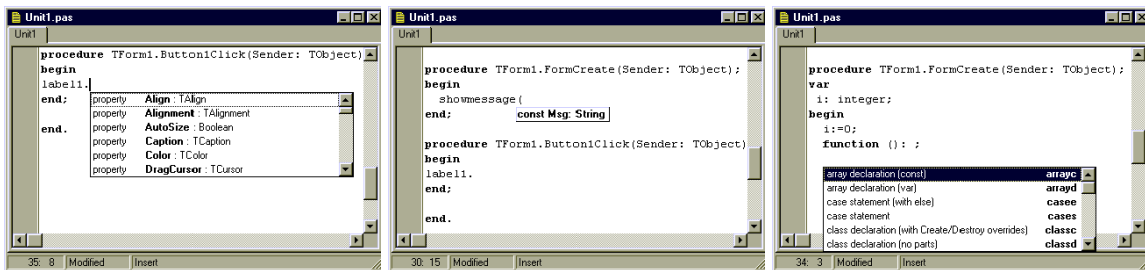


Figura 3.2 De esquerra a dreta: *code completion*, *code parameters* i *code templates*.

## Comandes

## Configurant l'entorn

Delphi ens permet configurar l'entorn de treball de tal manera que l'adaptem a les nostres necessitats i augmentem la nostre productivitat. Podem afegir i treure icones de les barres d'eines i components de les paletes, podem augmentar i disminuir l'espai de treball, etc.

### Personalitzar la finestra principal

Podem canviar de lloc les barres d'eines i la paleta de components fent clic sobre el seu extrem esquerre i arrossegant-les a la posició desitjada (veure figura 3.3).

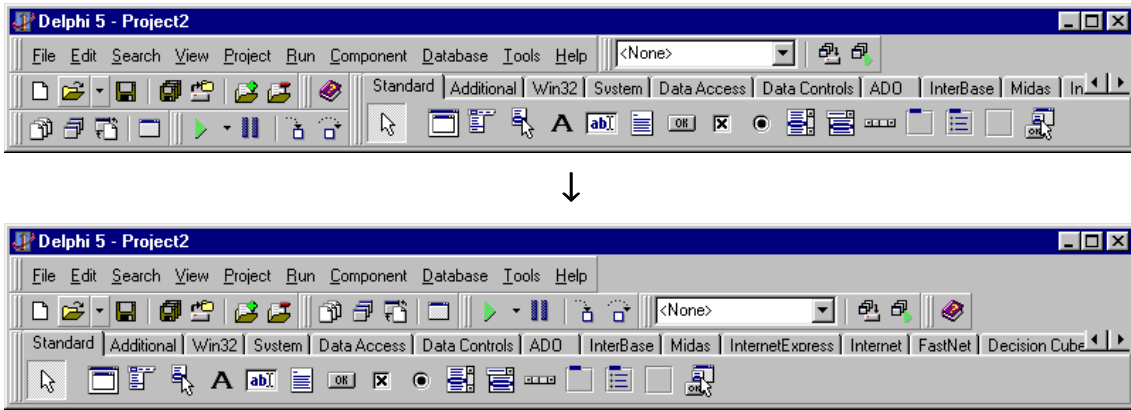


Figura 3.3 Canviant l'ordre de les barres d'eines a la finestra principal.

Podem treure i afegir barres d'eines si obrim el seu menú contextual (fent clic sobre el menú o les barres amb el botó dret del ratolí) i seleccionem quines barres volem. Així mateix, podem treure i afegir icones de les barres d'eines, si obrim el seu menú contextual (fent clic sobre el menú o les barres amb el botó dret del ratolí) i seleccionem l'opció *customize* (veure figura 3.4). A continuació, si arrosseguem una icona de la barra d'eines fora del menú principal, aquesta desapareixerà; i si arrosseguem una icona de la pestanya *commands* fins a una barra d'eines, obtindrem aquesta nova icona a la barra i posició on l'hem arrossegat.

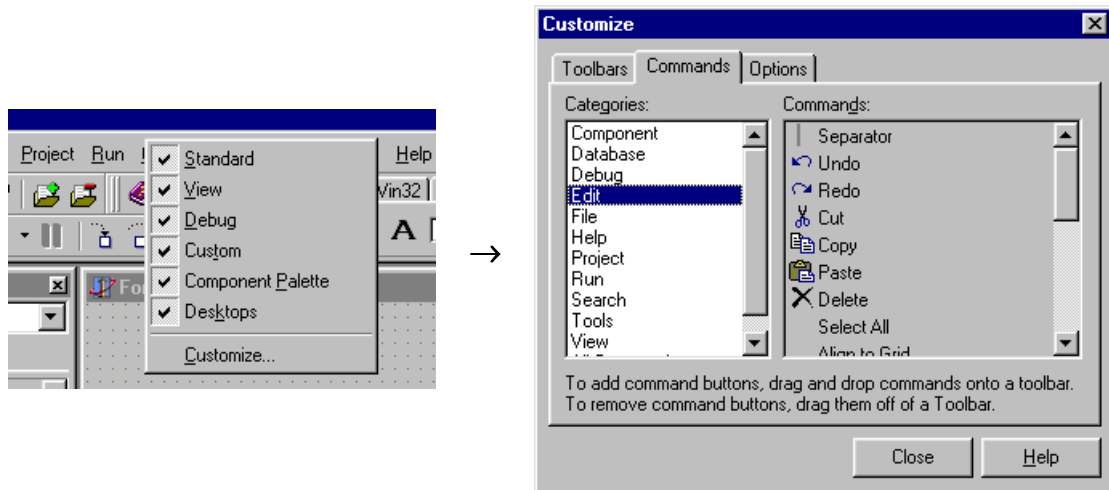


Figura 3.4 Afegint i traient icones de la finestra principal.

D'una manera molt semblant podem afegir, treure i reordenar components de la paleta de components si obrim el seu menú contextual (fent clic sobre la paleta de components amb el botó dret del ratolí) i seleccionem l'opció *properties* (veure figura 3.5).

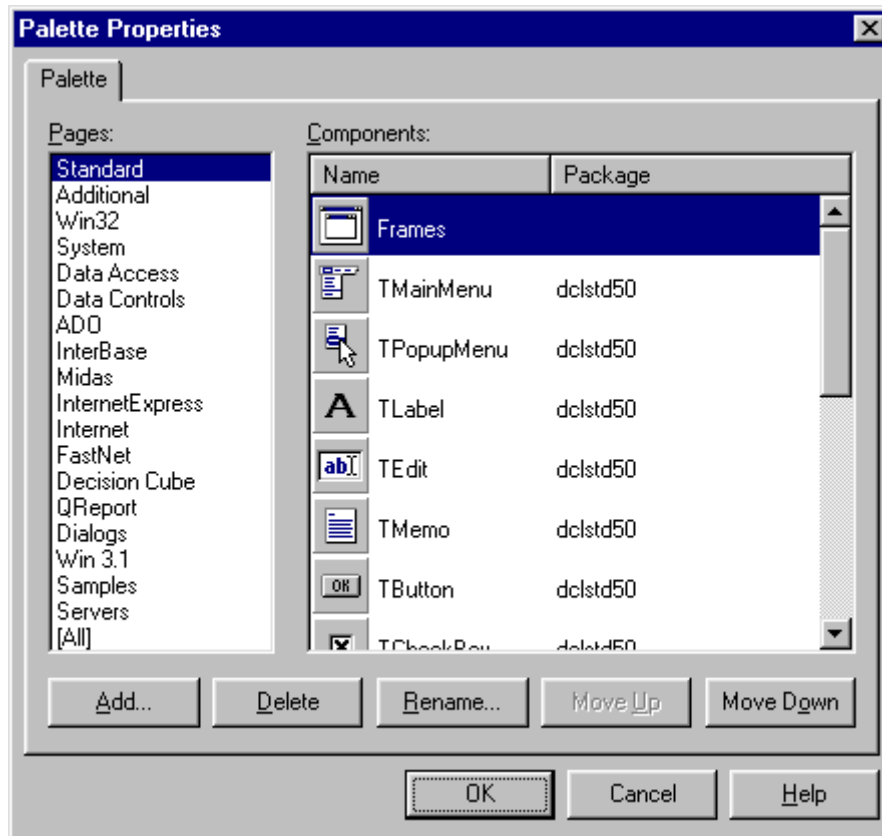


Figura 3.5 Afegint i traient components de la paleta de components.

### Personalitzar el dissenyador de formularis

El dissenyador de formularis també disposa d'un menú contextual que ens permet, entre d'altres coses, afegir el formulari que hem dissenyat a un magatzem de formularis, o veure el formulari i els component inserits en ell com a text (veure figura 3.6).

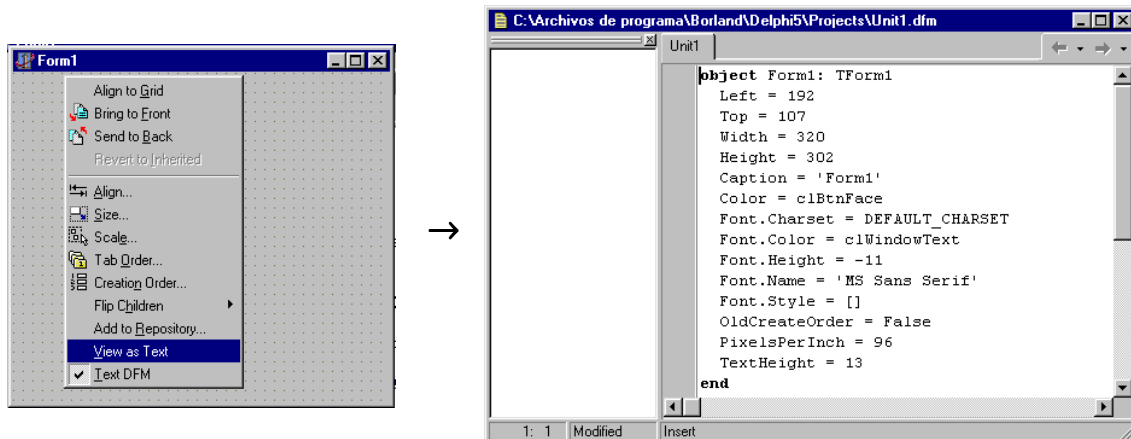


Figura 3.6 El dissenyador de components ens permet veure i editar l'arxiu .dfm corresponent al nostre formulari.

### Personalitzar l'inspector d'objectes

L'opció de personalització més interessant de l'inspector d'objectes és que el seu menú contextual ens permet seleccionar si volem veure les propietats i esdeveniments associats a un objecte endreçats alfabèticament o per categories (veure figura 3.7). En el cas que inspeccionem components, que tenen moltes propietats i esdeveniments associats, és molt més còmode visualitzar aquests endreçats per categories.

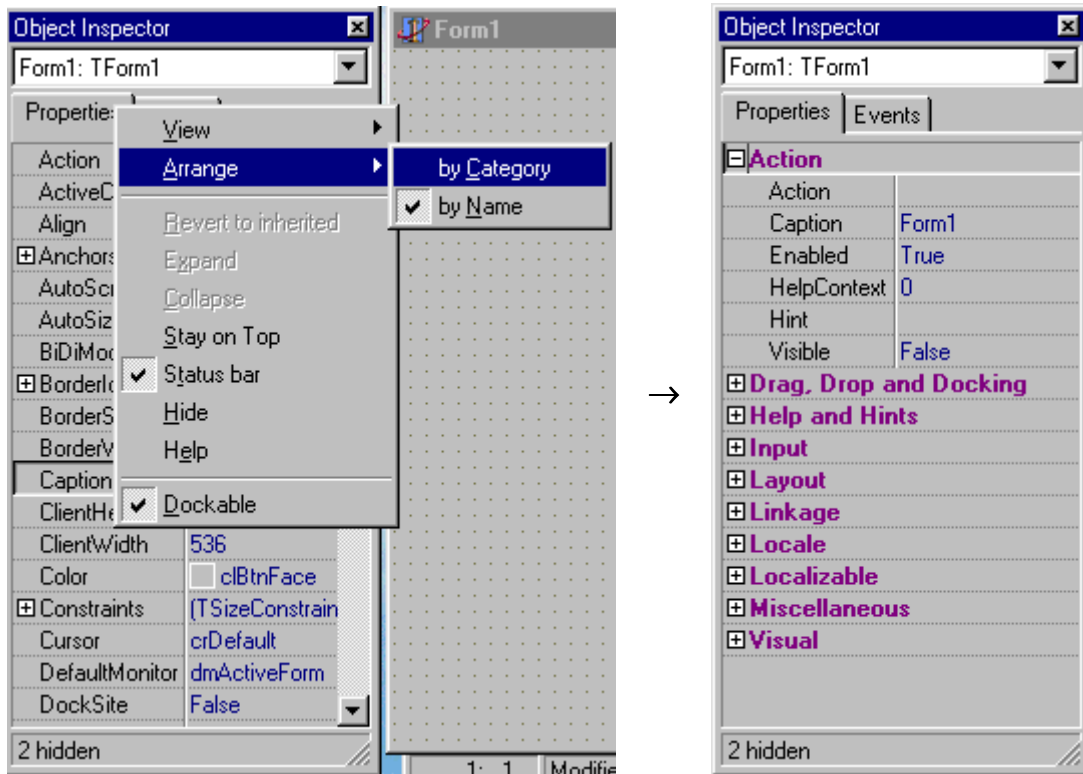


Figura 3.7 L'inspector d'objectes ens permet veure les propietats i esdeveniments endreçats per categories.

### Personalitzar l'editor de codi

L'editor de codi i l'explorador de codi disposen de moltes propietats configurables. Podem accedir a aquestes propietats mitjançant l'opció *properties* del seu menú contextual (veure figura 3.8).

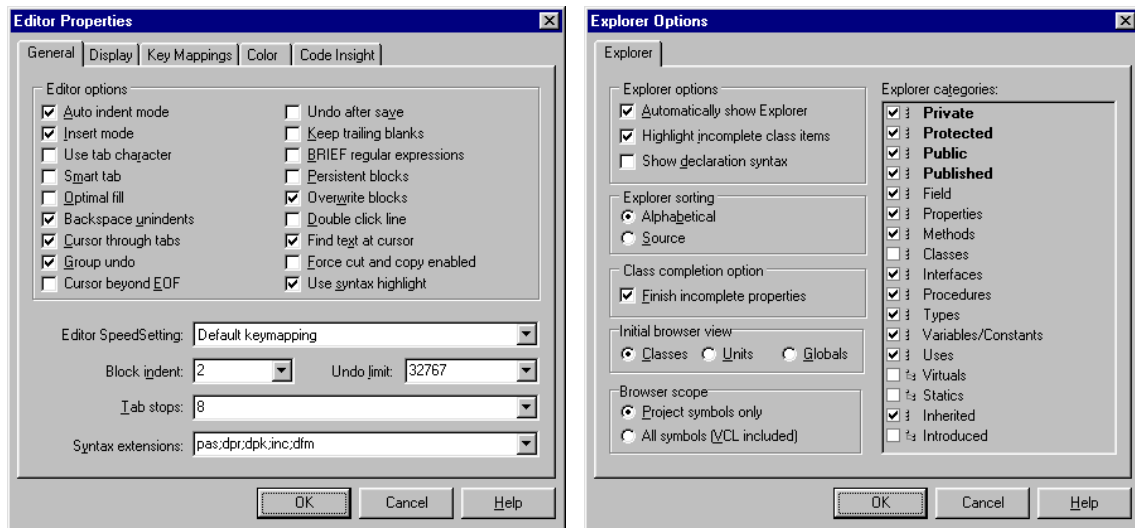


Figura 3.8 Propietats de l'editor de codi (figura de l'esquerra) i de l'explorador de codi (figura de la dreta).

### Ancorant finestres

Moltes finestres poden 'ancorar-se' a altres finestres. Per ancoratge s'entén tant connectar finestres de manera que es moguin al mateix temps, com combinar varies d'elles formant una espècie de llibreta amb pestanyes. Aquestes disposicions permeten utilitzar de manera eficient la pantalla i accedir ràpidament a les eines.

Per ancorar una finestra és precis arrossegar-la sobre una altre fins que el contorn rectangular de la primera passi a ser estret i vertical, i llavors deixar anar el botó del ratolí. Fent clic sobre la barra de títol d'una finestra ancorada i arrossegant-la, s'alliberarà.

Per exemple, a la configuració per defecte de Delphi l'explorador de codi es troba ancorat a l'esquerra de l'editor de codi. Es possible afegir l'administrador de projectes a aquestes dues finestres; amb això obtindrem tres finestres ancorades (veure figura 3.9).

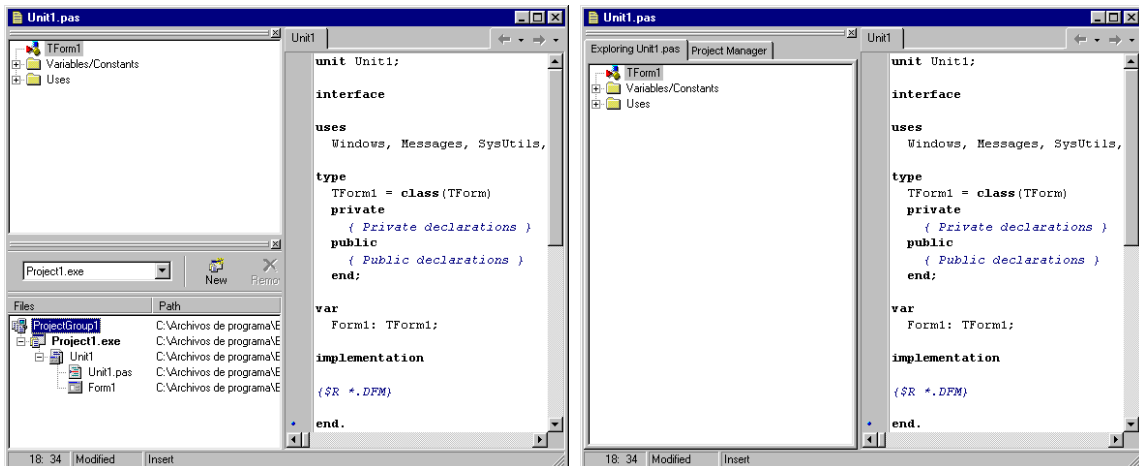



Figura 3.9 Administrador de projectes ancorat a l'editor i l'inspector de codi. A la figura de l'esquerra està ancorat formant una tercera finestra. A la figura de la dreta està ancorat en forma de fulls amb pestanyes.

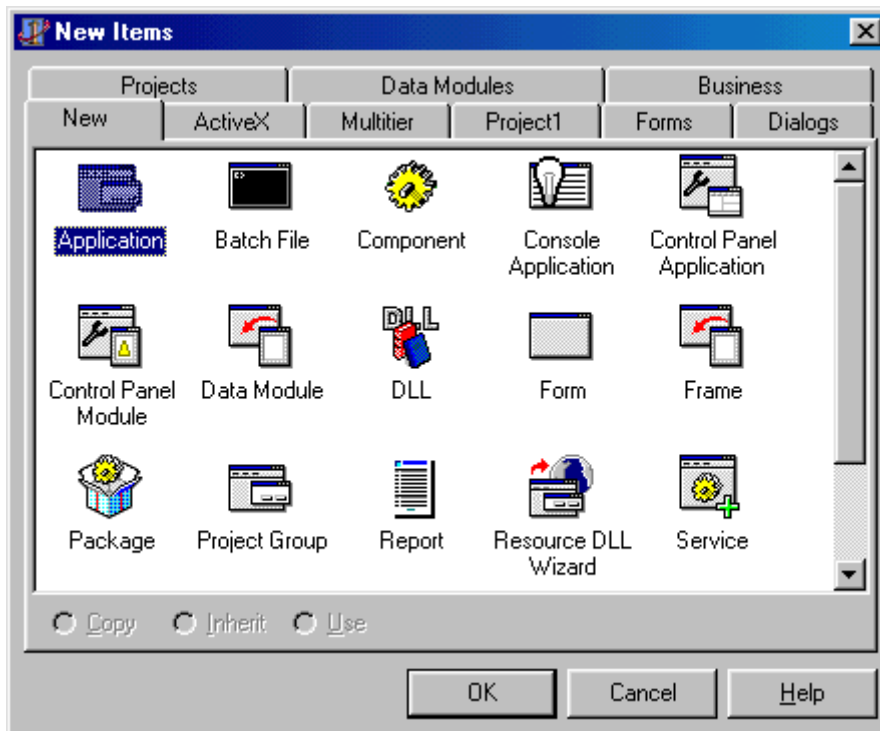
## Creació d'una aplicació amb Delphi

### Passos a l'hora de desenvolupar una aplicació

1. Escollir el tipus d'aplicació a crear.
2. Inserir dins el formulari els components que utilitzarem a la interfície gràfica.
3. Editar les propietats d'aquests components.
4. Editar el codi associat als nostres components, és a dir, gestionar els esdeveniments.
5. Guardar i executar el projecte.

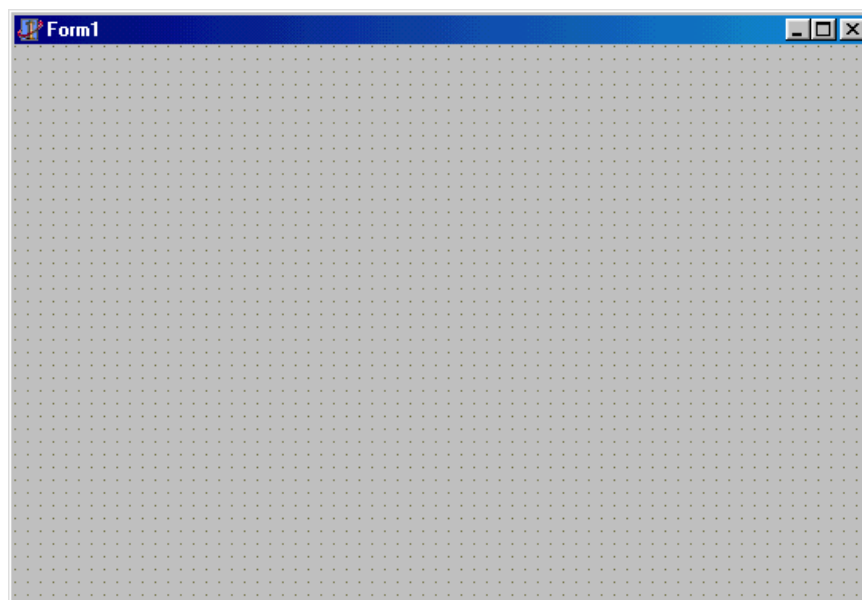
### La nostra primera aplicació

L'entorn de treball de Delphi genera codi font en Object Pascal tal com treballem amb els components visuals del dissenyador de formularis. L'exemple més senzill d'aquesta capacitat és començar un nou projecte. A la barra d'eines fem clic sobre la icona *new* . A continuació apareixerà una finestra on haurèm de seleccionar el tipus d'aplicació que volem (veure figura 3.10). Escollim *Application*. També podríem aconseguir el mateix d'una altre manera: seleccionant al menú l'opció *File* → *New Application*.



**Figura 3.10** El quadre de diàleg *New Items* ens permet escollir un formulari, una plantilla de projecte o un assistent que podem fer servir com a punt d'inici de la nostre aplicació.

A continuació veurem aparèixer un nou formulari (veure figura 3.11) en el dissenyador de formularis i el codi font esquelet d'aquest formulari en l'editor de codi. El codi font de la unitat corresponent al nou formulari el podeu veure en el llistat 3.1.



**Figura 3.11** Formulari buit.

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
```



```
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

**Llistat 3.1** L'esquelet d'un programa.

Per guardar el projecte seleccionem l'opció del menú *File* → *Save Project As...*. El nom d'un projecte en Delphi acaba en *.dpr*, que significa *Delphi project*. El fitxer del projecte és on resideix la porció principal del nostre programa. A diferència d'altres versions de Pascal que ens són familiars, la major part de la 'feina' del nostre programa es realitza en unitats enlloc de en el mòdul principal. Aquí tenim el fitxer de projecte de l'aplicació anterior.

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

**Llistat 3.2** Fitxer de projecte.

Tal com afegim més formularis i unitats a l'aplicació, aquests apareixen a la clàusula *uses* del fitxer del projecte. A més a més, el simple fet de afegir un component com un botó sobre un formulari (veure figura 3.12) fa que el codi per aquest element sigui generat i afegit a l'objecte formulari:

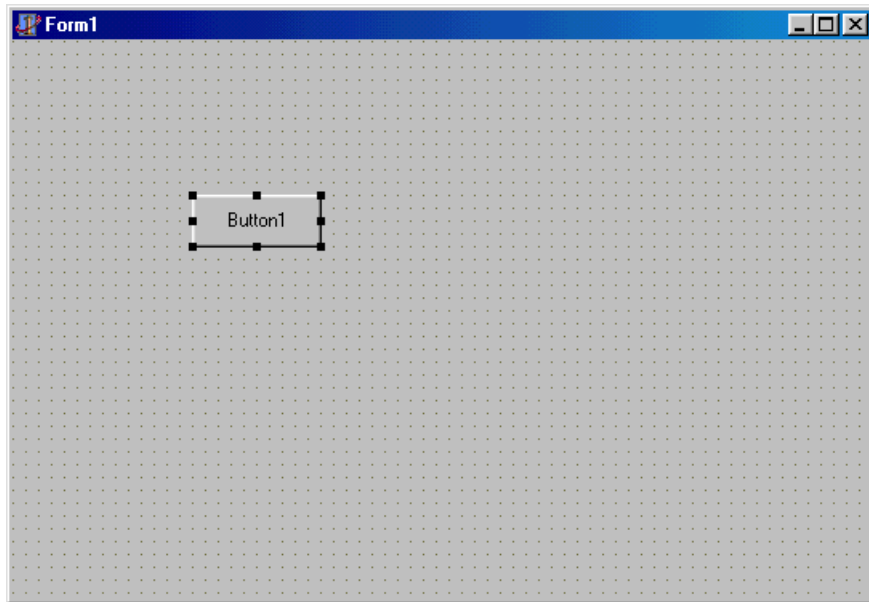


Figura 3.12 Formulari amb un botó.

```
type
  TForm1 = class(TForm)
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Llistat 3.3 Codi resultant d'afegir un component de tipus botó.

Com podeu veure, el botó és una variable de la classe `TForm1`. Quan més tard en el codi font ens referim al botó fora del context de `TForm1`, haurem de recordar la seva adreça com part de l'àmbit de `TForm1` referint-nos a `TForm1.Button1`.

Quan seleccionem aquest objecte botó en el dissenyador de formularis, podem canviar el seu comportament a través de l'inspector d'objectes. Suposem que, en temps de disseny, volem canviar l'amplada del botó a 100 píxels, i en temps d'execució, volem que el botó respongui a una pulsació doblant la seva alçada\*. Per canviar l'amplada del botó ens dirigim a l'inspector d'objectes, seleccionem l'objecte `Button1`, cerquem la propietat `Width`, i canviem el valor associat a la propietat `Width` a 100 (veure figura 3.13). Noteu que aquest canvi no té efecte en el dissenyador de formularis fins que no premem `Enter` o ens movem fora de la propietat `Width`. Per fer que el botó respongui a un clic del ratolí, seleccionem la pàgina d'esdeveniments de l'inspector d'objectes per veure la llista d'esdeveniments a la que pot respondre el botó. Fem doble clic a la casella propera a l'esdeveniment `OnClick` (veure figura 3.14), i Delphi generarà l'esquelet del procediment de resposta a un clic del ratolí i ens mourà ràpidament al lloc corresponent del codi font – en aquest cas, un procediment anomenat `TForm1.Button1Click()`. Tot el que ens resta per fer és inserir el codi que dobla l'alçada del botó entre el `begin` i `end` del mètode de resposta a l'esdeveniment. Aquest codi és `Button1.Height := Button1.Height * 2;`

---

\* A partir d'ara, quan parlem de modificar un atribut en temps de disseny ens referim a modificar-lo a través de l'inspector d'objectes, mentre que quan parlem de modificar un atribut en temps d'execució ens referim a modificar-lo mitjançant línies de codi al programa.

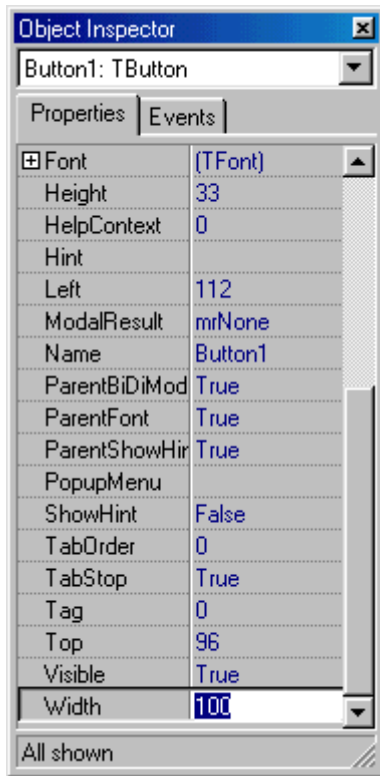


Figura 3.13 Modificant la propietat Width.



Figura 3.14 Creant resposta a l'esdeveniment OnClick.

Fixeu-vos que quan escrivim el punt darrere de Button1, automàticament s'obre una llista amb les propietats d'aquest objecte (veure figura 3.15).

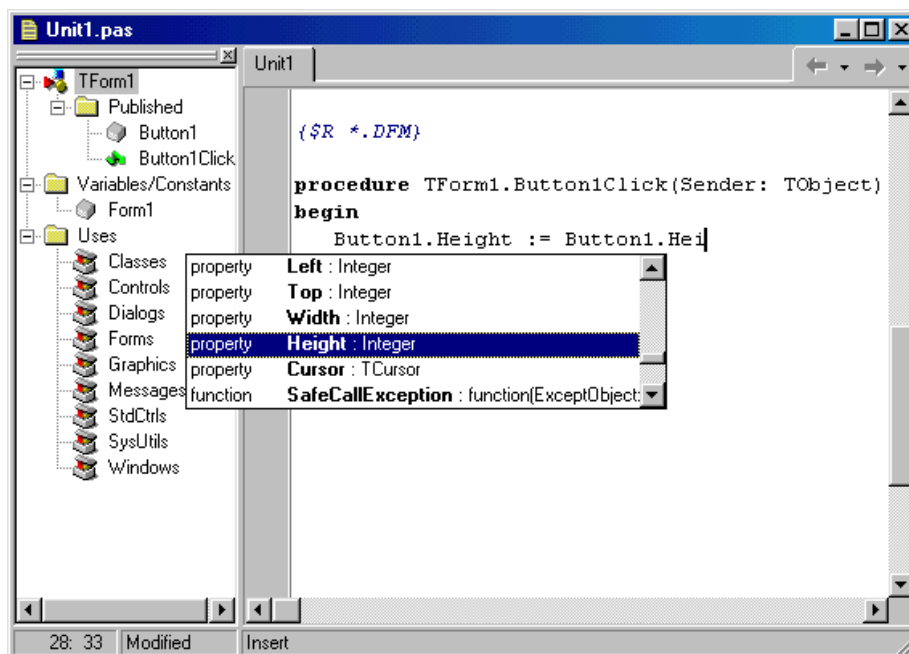


Figura 3.15 L'editor de codi facilita la nostra feina, obrint una llista en la que és possible escollir la propietat a la que volem accedir.

El codi font de la unitat corresponent al formulari ara queda:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Height := Button1.Height * 2;
end;

end.
```

**Llistat 3.4** Codi final de la nostre aplicació.

Per verificar que la nostre aplicació compila\* i s'executa, premem la tecla **F9** del nostre teclat.

### **Estructura d'una aplicació Delphi**

Una aplicació o projecte està formada per:

- **Formularis o fitxes.** Són les finestres que creem per la interfície d'usuari.
- **Components o controls.** Són entitats gràfiques dibuixades sobre els formularis que permeten interacció amb l'usuari. Per exemple: caixes de text, etiquetes, barres de desplaçament, botons, etc. Tant els formularis com els components són objectes.
- **Propietats.** Cada característica d'un formulari o component ve especificada per una propietat. Exemples de propietats són els noms, anotacions, mida color, posició i continguts. Delphi dona valors per defecte a les propietats per defecte. Es pot canviar el valor d'aquestes propietats tant en temps de disseny, mitjançant l'inspector d'objectes, com en temps d'execució, mitjançant codi escrit a l'aplicació.
- **Procediments d'esdeveniments.** És el codi associat a algun objecte i que s'executa quan ocorre cert esdeveniment.
- **Procediments generals.** És el codi que no està associat a objectes i que és invocat per l'aplicació.
- **Mètodes.** Són els procediments que poden ser invocats per realitzar alguna acció sobre un objecte en particular.

- 
- El procés de compilació consisteix en agafar un programa escrit en un llenguatge de programació (en el nostre cas Object Pascal) i generar un nou programa escrit en llenguatge màquina, tot verificant que el programa no contingui errors sintàctics.
  - El procés de linkatge consisteix en agafar tots els programes que formen part d'una aplicació i unir-los per crear un sol fitxer executable. El linkador pren com a entrada un conjunt de programes en llenguatge màquina i genera un únic programa final, també en llenguatge màquina. El linkador permet que una aplicació es pugui desglossar en un conjunt de mòduls.

- **Unitats o mòduls.** És la col·lecció de procediments generals, declaracions de variables i definicions de constants i de tipus de dades emprades per l'aplicació.

Projecte (.DPR, .RES)

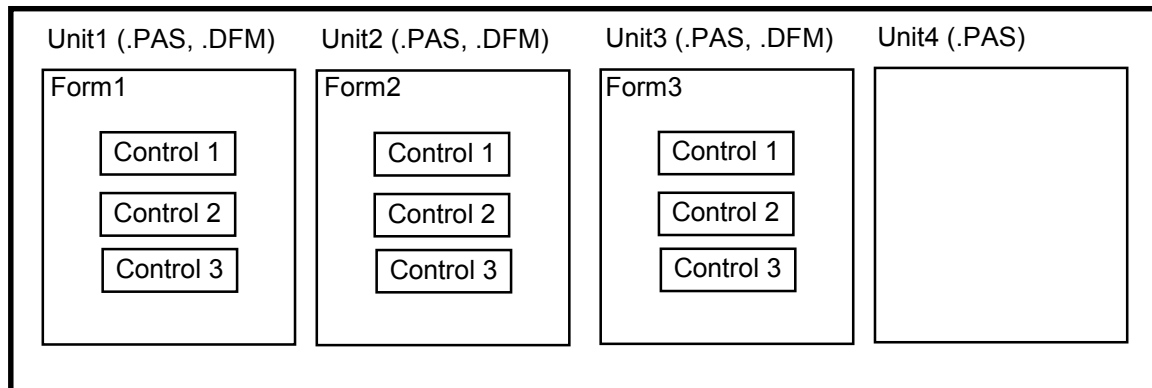


Figura 3.16 Estructura d'una aplicació de Delphi.

## Depuració i tractament d'errors

Si quan compilem una aplicació apareixen missatges d'error:

- En prémer la tecla **F1** apareixerà una pàgina d'ajuda amb una explicació detallada de l'error, possibles causes i possibles solucions.
- En fer doble clic sobre el missatge d'error, el cursor es posicionarà a la línia del programa on s'ha produït aquest error.

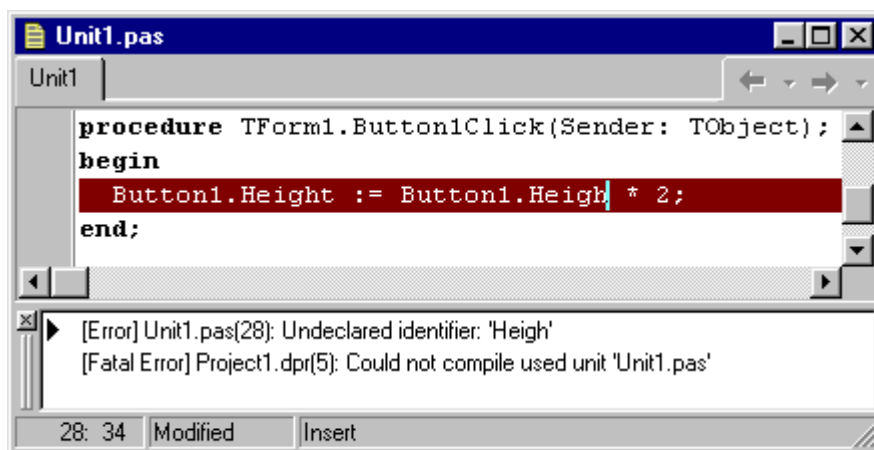


Figura 3.17 Missatges d'error en temps de compilació.

Si quan executem una aplicació apareixen missatges d'error, o no apareixen però el programa dona resultats erronis, cal fer una execució pas per pas amb traça de variables per analitzar en quin punt del programa hi ha un comportament que no és l'esperat. Disposem de les següents eines:

- Executar una línia de codi del programa, cada cop que es premen les tecles **F7** i **F8**, o les opcions del menú *Run* → *Trace into* i *Run* → *Step over*.
- Executar el programa fins a la línia de codi on es troba el cursor, prement la tecla **F4**, o l'opció del menú *Run* → *Run to cursor*.

- Parar l'execució i inicialitzar de nou el programa, prement les tecles **[Ctrl] + [F2]**, o l'opció del menú *Run → Program reset*.
- Visualitzar i modificar en un moment donat el valor d'una variable o l'atribut d'un objecte, prement les tecles **[Ctrl] + [F7]**, o l'opció del menú *Run → Evaluate/Modify*.
- Visualitzar en tot moment de l'execució pas per pas el valor d'una variable o l'atribut d'un objecte, prement les tecles **[Ctrl] + [F4]**, o l'opció del menú *Run → Add watch*.

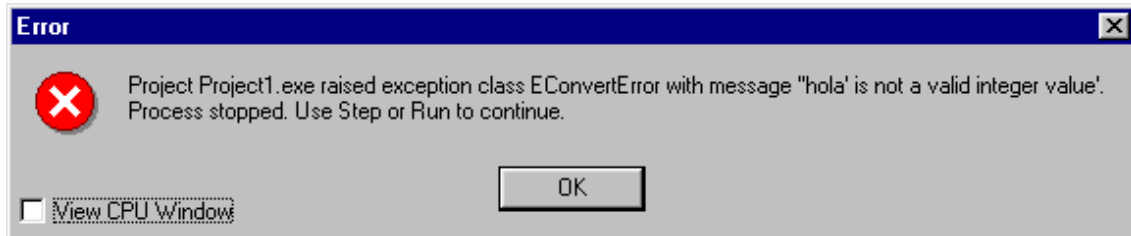


Figura 3.18 Missatges d'error en temps d'execució.

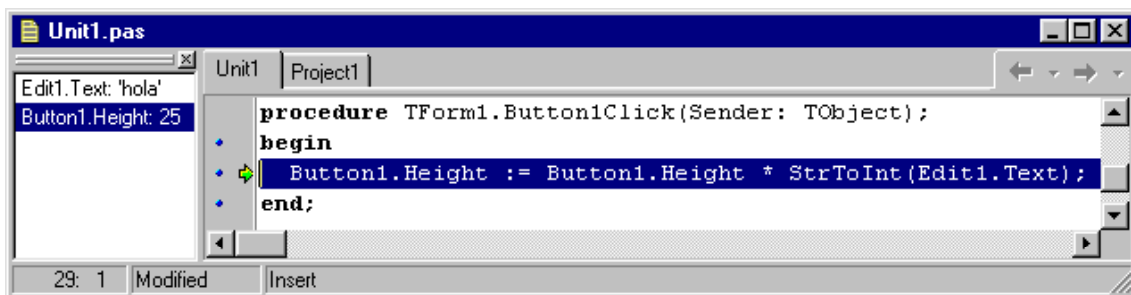


Figura 3.19 Execució pas per pas amb traça de variables.

## Treballant amb els components

### ***Inserir un nou component al formulari***

Per inserir un component en un formulari poden emprar-se diferents tècniques. La més efectiva és escollir el component a la paleta de components, fent un sol clic sobre ell amb el ratolí. Seguidament prémer sobre el punt del formulari on volem inserir el control i, sense deixar anar el botó del ratolí, moure's a un segon punt, establint així la mida del component.

### ***Eliminar un component del formulari***

Per eliminar un component del formulari cal seleccionar-lo fent un clic amb el ratolí. A continuació només cal prémer la tecla **[Supr]** i el component quedarà eliminat. Si haviem afegit a la aplicació codi associat al component (resposta a esdeveniments, etc.) caldrà eliminar aquest codi a mà.

### ***Seleccionar més d'un component alhora***

Si volem aplicar canvis a més d'un component alhora, només cal seleccionar-los i els canvis que a continuació realitzem s'aplicaran a tots els components seleccionats. Hi han dues maneres de seleccionar més d'un component alhora. La primera és fer clic amb el ratolí a sobre d'un component i, a continuació, fer clic sobre la resta de components a seleccionar mantenint premuda la tecla **[Majús]**. La segona manera és fer clic sobre un punt del formulari i, sense deixar anar el botó del ratolí, arrossegar aquest fins que formi un rectangle que englobi tots els components a seleccionar.

### Modificar les propietats dels components

Per canviar les propietats dels components o del formulari hem de seleccionar el component i a continuació posicionar-nos a l'inspector d'objectes sobre la propietat que volem modificar. En introduir un nou valor al camp corresponent de la propietat, haurem de prémer `Enter` o fer clic amb el ratolí fora de la propietat per que els canvis tinguin efecte.

### Canviar les dimensions d'un component

Per canviar les dimensions d'un component del formulari només cal seleccionar-lo fent un clic amb el ratolí. A continuació apareixeran vuit petits quadres negres al voltant component: un a cada cantonada i un al mig de cada aresta que delimita el component. Si arrosseguem qualsevol d'aquest quadres amb el ratolí, canviarem les dimensions del component.

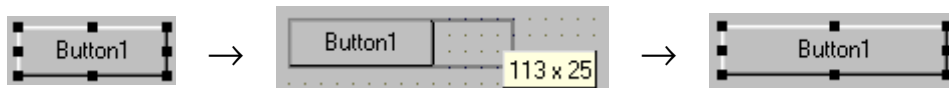


Figura 3.20 Redimensionant un component.

Per canviar les dimensions del formulari podem arrossegar les arestes i cantonades d'aquest, tal com fèiem amb els components, però sense veure els vuit petits quadres negres.

Una altre manera de canviar les mides dels components o del formulari és modificar el valor de les seves propietats `Height` i `Width` a l'inspector d'objectes.

### Canviar la posició d'un component

Per canviar la posició dins el formulari de qualsevol component només cal seleccionar el component que volem moure fent un clic amb el ratolí, i arrossegar-lo a la posició desitjada.

Una altre manera de canviar la posició dels components dins el formulari o del formulari dins la pantalla és modificar el valor de les seves propietats `Top` i `Left` a l'inspector d'objectes. També podem canviar la posició del formulari dins la pantalla modificant la seva propietat `Position` a l'inspector d'objectes.

### Alinear

Per alinear un o més components cal seleccionar-los i, a continuació, obrir el menú contextual, fent clic amb el botó dret del ratolí, i seleccionar l'opció `align...`. Apareixerà un quadre de diàleg per escollir les diferents opcions d'alineació.

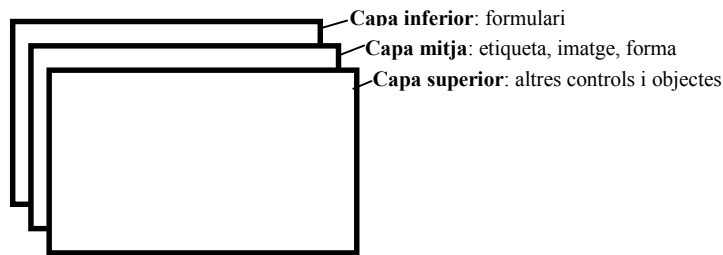
### Portar davant / Portar darrere. Capes de visualització

Els elements que un formulari ens mostra no estan necessàriament sobre la mateixa capa de visualització. La visualització d'un formulari consta de tres capes tal com s'esbossa a la figura 3.21. Tota la informació visualitzada directament al formulari, impresa o dibuixada amb mètodes gràfics, apareix a la capa inferior. La informació d'etiquetes de text (component `Label`), imatges (component `Image`), i formes (component `Shape`), apareixen a la capa mitja. Tots els altres objectes són visualitzats a la capa superior.

Això significa que hem d'anar amb compte d'on col·loquem els objectes a un formulari per tal de no cobrir altres de manera no desitjada. Per exemple: el text imprès sobre el formulari pot ser amagat per un botó situat a sobre. Els objectes dibuixats amb `Shape` són coberts per tots els controls excepte pel component `Image`.



La següent qüestió és què estableix la posició relativa d'objectes a la mateixa capa. És a dir, si dos botons són a la mateixa àrea d'un formulari, quin tancarà a quin? L'ordre en el qual els objectes d'una mateixa capa es superposen els uns sobre els altres s'anomena l'ordre *z*. Aquest ordre s'estableix per primer cop quan creem el formulari. Els últims elements que hem incorporat al formulari reposen sobre els que hem incorporat primer. No obstant, un cop establert l'ordre *z* aquest pot ser modificat seleccionant l'objecte desitjat i escollint l'opció `Bring to Front` del seu menú contextual. La opció `Send to Back` del mateix menú té l'efecte oposat. Noteu que aquestes dues comandes només treballen sobre una capa; els objectes

de la capa mitja sempre apareixeran darrere dels de la capa superior i els objectes de la capa inferior sempre apareixeran darrere dels de la capa mitja.



**Figura 3.21** Capes de visualització.

### **Copiar / Enganxar**

Podem crear còpies de components que tenim sobre el formulari seleccionant el component i, a continuació, fent clic a la icona  o prement la combinació de tecles **Alt** + **C**. Aquesta acció crea una còpia del component a memòria. A partir d'aquest moment, quan fem clic a la icona  o premem la combinació de tecles **Alt** + **V**, obtindrem una còpia del component sobre el formulari, amb les mateixes propietats que el component original però amb un nom o identificador diferent.

### **Propietats més importants**

- **Name** És el nom que rebrà dins el programa el component.
- **Caption** Especifica una cadena de caràcters que etiqueta el control de cara a l'usuari. En el text que escrivim dins aquesta propietat podem incloure el caràcter **&**. Aquest caràcter denota que el següent caràcter serà la tecla d'accés al component. És a dir, si el component és un botó o una opció del menú, en addició a fer clic sobre el component per invocar un esdeveniment, també podem prémer la seva tecla d'accés en conjunció amb la tecla **Alt**. Noteu que als components la tecla d'accés apareixerà subratllada.
- **Left i Top** Indiquen la posició d'un component dins d'un altre.
- **Width i Height** Indiquen l'alt i l'ample, respectivament, d'un component.
- **Align** Facilita l'alineació d'un component a un dels marges del component que el conté, o bé ocupant tot l'espai disponible.
- **Enabled** Indica si el component es troba actiu o inactiu, segons que l'usuari podrà o no interactuar amb ell i, a més, apareixerà amb una aparença o una altre en pantalla.
- **Visible** Controla la visibilitat d'un component, és a dir, indica si el component es visualitza per pantalla o no.

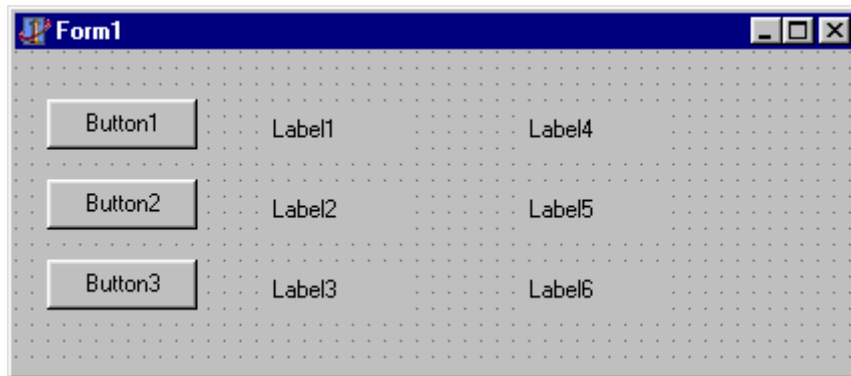
### **Esdeveniments més importants**

- **OnClick, OnDbClick, OnMouseDown, OnMouseUp i OnMouseMove** El primer d'ells, que és l'esdeveniment més emprat, es produeix en prémer sobre un control, generalment fent servir el ratolí. Els altres quatre es produeixen en efectuar un doble clic, prémer un botó, alliberar-lo i moure el ratolí, respectivament. Mitjançant aquests esdeveniments podem saber, per exemple, quan el punter del ratolí s'està desplaçant sobre un `Panel` o si s'ha premut a un `CheckBox` canviant el seu estat.
- **OnKeyPress, OnKeyDown i OnKeyUp** Són similars a `OnClick`, `OnMouseDown` i `OnMouseUp`, respectivament, però en aquest cas indiquen si s'ha premut una tecla del teclat i no un botó del ratolí. Aprofitant l'esdeveniment `OnKeyPress` és possible, per exemple, controlar la introducció de dades a un control `Edit`, limitant els caràcters, realitzant conversions, etc.



## Exemple 3-1: cronòmetre

1. Iniciem un nou projecte. El que volem fer amb aquest projecte és iniciar un cronòmetre, llavors parar el cronòmetre i calcular el temps transcorregut en segons.
2. Col·loquem tres botons i sis etiquetes sobre el formulari. Moveu i canvieu la mida del formulari i dels components tal que sembli quelcom així:



3. Modifiquem les propietats del formulari, dels botons i de les etiquetes:

**Form1:**

|             |                |
|-------------|----------------|
| BorderStyle | bsDialog       |
| Caption     | Cronòmetre     |
| Name        | FormCronometre |

**Button1:**

|         |             |
|---------|-------------|
| Caption | &Iniciar    |
| Name    | ButtonInici |

**Button2:**

|         |              |
|---------|--------------|
| Caption | &Parar       |
| Name    | ButtonParada |

**Button3:**

|         |               |
|---------|---------------|
| Caption | &Sortir       |
| Name    | ButtonSortida |

**Label1:**

|         |             |
|---------|-------------|
| Caption | Temps inici |
|---------|-------------|

**Label2:**

|         |             |
|---------|-------------|
| Caption | Temps final |
|---------|-------------|

**Label3:**

|         |                     |
|---------|---------------------|
| Caption | Temps transcorregut |
|---------|---------------------|

**Label4:**

|                      |            |
|----------------------|------------|
| Alignment            | taCenter   |
| Caption              | [en blanc] |
| Color                | cl3DLight  |
| Constraints.MinWidth | 125        |
| Name                 | LabelInici |

**Label5:**

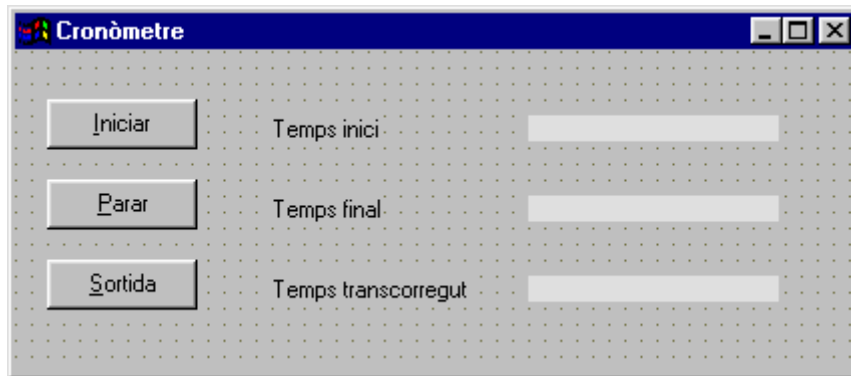
|           |            |
|-----------|------------|
| Alignment | taCenter   |
| Caption   | [en blanc] |
| Color     | cl3DLight  |

|                      |            |
|----------------------|------------|
| Constraints.MinWidth | 125        |
| Name                 | LabelFinal |

**Label6:**

|                      |                    |
|----------------------|--------------------|
| Alignment            | taCenter           |
| Caption              | [en blanc]         |
| Color                | cl3DLight          |
| Constraints.MinWidth | 125                |
| Name                 | LabelTranscorregut |

4. En aquests moments el nostre formulari hauria de tenir el següent aspecte:



5. Posicionem-nos sobre la finestra de l'editor de codi, o seleccionem l'opció *View* → *Code Explorer* del menú principal, que fa el mateix.
6. Declarem tres noves variables de tipus `TDateTime`, que emmagatzemaran instants de temps:

```
var  
    TempsInici: TDateTime;  
    TempsFinal: TDateTime;  
    TempsTranscorregut: TDateTime;
```

7. A l'inspector d'objectes seleccioneu el botó `ButtonInici`, posicioneu-vos sobre el seu esdeveniment `OnClick` i feu doble clic a la caixa de la seva dreta. A l'editor de codi apareixerà la plantilla o esquelet del codi associat a aquest esdeveniment. Escriviu al seu interior el següent codi:

```
procedure TFormCronometre.ButtonIniciClick(Sender: TObject);  
begin  
    (* Estableix i escriu el temps inicial *)  
    TempsInici := Time();  
    LabelInici.Caption := TimeToStr(TempsInici);  
    LabelFinal.Caption := '';  
    LabelTranscorregut.Caption := '';  
end;
```

En aquest procediment, un cop premut el botó 'Iniciar', llegim el temps actual i l'imprimim a una etiqueta. També buidem el contingut de les altres etiquetes

8. A l'inspector d'objectes seleccioneu el botó `ButtonParada`, posicioneu-vos sobre el seu esdeveniment `OnClick` i feu doble clic a la caixa de la seva dreta. A l'editor de codi apareixerà la plantilla o esquelet del codi associat a aquest esdeveniment. Escriviu al seu interior el següent codi:

```
procedure TFormCronometre.ButtonParadaClick(Sender: TObject);
```


```
begin  
  (* Troba el temps final, calcula el temps transcorregut i *)  
  (* escriu tots dos valors a les corresponents etiquetes *)  
  TempsFinal := Time();  
  TempsTranscorregut := TempsFinal - TempsInici;  
  LabelFinal.Caption := TimeToStr(TempsFinal);  
  LabelTranscorregut.Caption := TimeToStr(TempsTranscorregut);  
end;
```

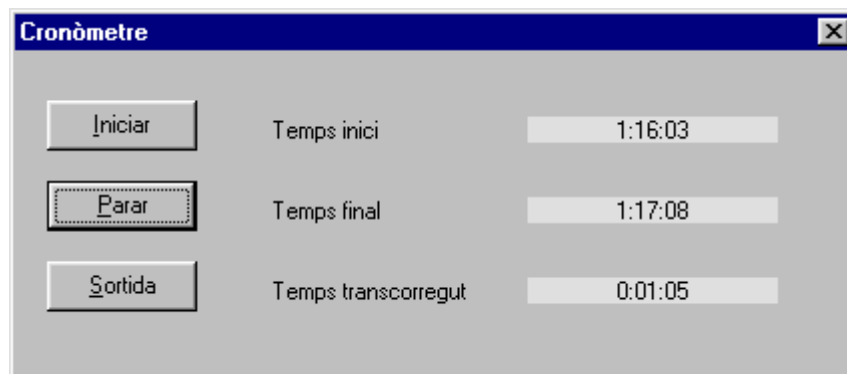
En aquest procediment, quan es prem el botó Parar, llegim el temps actual, calculem el temps transcorregut, i posem tots dos a les seves corresponents etiquetes.

9. A l'inspector d'objectes seleccioneu el botó ButtonSortida, posicioneu-vos sobre el seu esdeveniment OnClic i feu doble clic a la caixa de la seva dreta. A l'editor de codi apareixerà la plantilla o esquelet del codi associat a aquest esdeveniment. Escriviu al seu interior el següent codi:

```
procedure TFormCronometre.ButtonSortidaClick(Sender: TObject);  
begin  
  Close;      (* Tanca el formulari i acaba el programa *)  
end;
```

Aquest procediment senzillament finalitza l'aplicació un cop premut el botó Sortida.

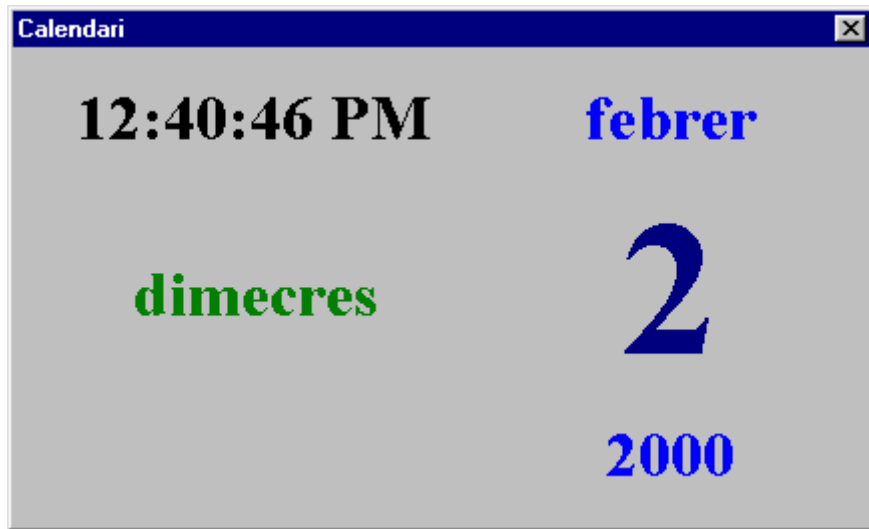
10. Graveu la vostra aplicació seleccionant l'opció *File* → *Save Project As...* del menú principal.
11. Executeu l'aplicació prement el botó *Run*  a la barra d'eines, o prement la tecla **[F9]**. Força maco, no?



12. Si teniu temps, intenteu ampliar aquesta aplicació fent el següent:
- Intenteu canviar el color del formulari i les fonts emprades a les etiquetes i els botons.
  - Noteu que podeu prémer el botó Parar abans que el botó Iniciar. Això no hauria de ser així. Canvieu l'aplicació de tal manera que no es pugui realitzar aquesta acció. A més a més, feu que un cop iniciat el cronometratge, no es pugui prémer de nou el botó Iniciar fins que el botó Parar hagi estat premut. Pista: treballeu amb la propietat Enabled del component botó.
  - Penseu com es pot visualitzar per pantalla d'una manera continua el temps final i el temps transcorregut. Pista: treballeu amb el component Timer.

## Exercici 3-1: calendari

Dissenyeu una finestra semblant a la pàgina d'un calendari. Volem que la finestra visualitzi el dia, mes i any actuals. També volem que visualitzi l'hora, actualitzant-la a cada segon (feu servir el component Timer).



# El llenguatge Object Pascal

El llenguatge sobre el qual reposa Delphi és una versió orientada a objectes de Pascal, que Borland anomena Object Pascal. Object Pascal proporciona nombroses característiques afegides al llenguatge Pascal, per tal de millorar la productivitat. Entre aquestes s'inclouen:

- Orientació a objectes: encapsulament, herència simple i polimorfisme.
- Manejament d'excepcions, que ens permet detectar i recuperar-nos d'una manera elegant dels errors en temps d'execució.
- Informació de tipus en temps d'execució, que ens permet determinar el tipus d'un objecte en temps d'execució, enlloc d'en temps de compilació.
- Suport d'interfícies, que fa el desenvolupament COM més senzill i ens permet construir fonaments robustos d'aplicacions.
- Cadenes (*strings*) sense límit de mida, que ens permet escriure codi de maneig de cadenes sense haver de preocupar-nos de limitacions de mida i d'assignació d'espai.
- El tipus de dada *currency*, per càlculs monetaris més acurats.
- El tipus de dada *variant*, que ens permet impulsar tecnologies com OLE que involucra dades sense tipus i correspondència de tipus tardana.

En aquest capítol donarem per suposat que tots coneixem ja el llenguatge de programació Pascal i ens centrarem en les noves característiques que aporta el llenguatge Object Pascal sobre Pascal.

## Declaració de variables

Les variables són identificadors el valor dels quals pot canviar durant l'execució del programa. Amb altres paraules, una variable és el nom d'una o més posicions de memòria, i pot emprar-se per llegir o escriure en aquestes posicions. Les variables tenen un tipus que informa al compilador de com ha d'interpretar les dades que contenen.

### Declaració de variables

La sintaxi bàsica d'una declaració de variables és:

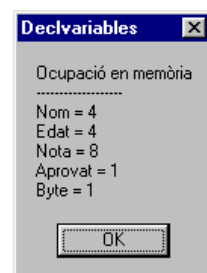
```
var Llista_noms_variables : tipus;
```

O bé podem declarar una variable assignant-li un valor inicial:

```
var nom_variable : tipus = expressió_constant;
```

Aquí tenim un petit exemple de declaració de variables:

```
program DeclVariables;  
  
uses  
  Dialogs, SysUtils;  
  
var  
  Nom : string = 'Josepet';    // Una variable cadena  
  Edat, i : integer;          // Dues variables senceres  
  Nota : real;                // Una variable de punt flotant  
  Aprovat : boolean;         // Una variable booleana  
  
begin  
  // Preparem una cadena i la mostrem  
  ShowMessage(  
    'Ocupació en memòria' + #13 +  
    '-----' + #13 +
```



```
'Nom = ' + IntToStr(SizeOf(Nom)) + #13 +  
'Edat = ' + IntToStr(SizeOf(Edat)) + #13 +  
'Nota = ' + IntToStr(SizeOf(Nota)) + #13 +  
'Aprovat = ' + IntToStr(SizeOf(Aprovat)) + #13 +  
'Byte = ' + IntToStr(SizeOf(Byte));  
end.
```

### Noms de components, objectes i variables

En molts llenguatges d'avui en dia, a l'hora d'escriure el codi d'una forma llegible i que pugui ser entesa ràpidament per altres programadors (o per nosaltres mateixos quan passi el temps), es segueix una notació especial per anomenar les variables i els objectes utilitzats en el codi. L'important no és seguir una notació concreta sinó alguna que sigui coherent i homogènia. La més utilitzada i que quasi que qualsevol programador entendreà és l'anomenada 'notació hongaresa'. Aquesta notació consisteix el nom de qualsevol variable i objecte amb un identificador que ens indiqui de quin tipus és. A més a més, el conveni indica que si el nom d'una variable està format per diverses paraules totes juntes, cadascuna d'elles ha de començar per una lletra majúscula per indicar-ho. D'aquest mode, si al codi d'algú llegim `ButtonSortir`, sabrem automàticament que es tracta d'un botó que serveix per abandonar l'aplicació, mentre que `MenuSortir` seria un menú per sortir.

Existeixen moltes variants de la notació hongaresa en quant als prefixos utilitzats i realment no és important quin d'ells fem servir. L'important és que fem servir una encara que ens inventem nosaltres els prefixos.

64 caràcters, primer caràcter lletra, lletres + nombres + `_`, no paraules reservades, indiferent majúscules i minúscules.

### Abast i resolució de conflictes (variables amb noms iguals)

Punt per elaborar.

### Tipus de dades

| Tipus           | Rang de valors           | Ocupació |
|-----------------|--------------------------|----------|
| <i>ShortInt</i> | -128 a 127               | 8 bits   |
| <i>Byte</i>     | 0 a 255                  | 8 bits   |
| <i>SmallInt</i> | -32768 a 32767           | 16 bits  |
| <i>Word</i>     | 0 a 65535                | 16 bits  |
| <i>Integer</i>  | -2147483648 a 2147483647 | 32 bits  |
| <i>LongInt</i>  | -2147483648 a 2147483647 | 32 bits  |
| <i>LongWord</i> | 0 a 4294967295           | 32 bits  |
| <i>Cardinal</i> | 0 a 4294967295           | 32 bits  |
| <i>Int64</i>    | $-2^{63}$ a $2^{63}$     | 64 bits  |
| <i>Single</i>   | 7/8 dígits               | 32 bits  |
| <i>Real48</i>   | 11/12 dígits             | 48 bits  |
| <i>Real</i>     | 15/16 dígits             | 64 bits  |
| <i>Double</i>   | 15/16 dígits             | 64 bits  |
| <i>Extended</i> | 19/20 dígits             | 80 bits  |
| <i>Comp</i>     | 19/20 dígits             | 64 bits  |
| <i>Currency</i> | 19/20 dígits             | 64 bits  |

Taula 4.1 Tipus de dades numèriques.

| Tipus           | Ocupació |
|-----------------|----------|
| <i>Boolean</i>  | 8 bits   |
| <i>ByteBool</i> | 8 bits   |
| <i>WordBool</i> | 16 bits  |
| <i>LongBool</i> | 32 bits  |

Taula 4.2 Tipus de dades booleans.

| Tipus           | Pot contenir                                   |
|-----------------|--|
| <i>Char</i>     | Un caràcter qualsevol, igual a <i>AnsiChar</i> |
| <i>AnsiChar</i> | Un caràcter qualsevol del conjunt ASCII        |

|                    |   |
|--------------------|---|
| <i>WideChar</i>    | Un caràcter qualsevol del conjunt UNICODE                 |
| <i>String</i>      | Una cadena de qualsevol longitud                          |
| <i>ShortString</i> | Una cadena de fins 255 caràcters                          |
| <i>AnsiString</i>  | Una cadena de qualsevol longitud                          |
| <i>WideString</i>  | Una cadena de caràcters UNICODE de qualsevol longitud     |
| <i>PChar</i>       | Apuntador a cadena de caràcters terminada amb nul         |
| <i>PAnsiChar</i>   | Igual que PChar   |
| <i>PWideChar</i>   | Apuntador a cadena de caràcters UNICODE terminada amb nul |

**Taula 4.3** Tipus de dades de caràcters.

| Tipus          | Pot contenir                               |
|----------------|--|
| <i>Pointer</i> | Adreces de memòria. Apuntador a una dada   |
| <i>Variant</i> | Valors de tipus dinàmic, que poden canviar |

**Taula 4.4** Altres tipus de dades.

La taxonomia dels tipus de dades a Object Pascal és la següent:

|                 |
|-----------------|
| simple          |
| ordinal         |
| integer         |
| character       |
| boolean         |
| enumerated      |
| subrange        |
| real            |
| string          |
| structured      |
| set             |
| array           |
| record          |
| file            |
| class           |
| class reference |
| interface       |
| pointer         |
| procedural      |
| variant         |

### Conversió de tipus

En ocasions, una dada emmagatzemada a una variable d'un cert tipus podem necessitar-la com una dada de diferent tipus. L'operació de conversió es realitza mitjançant un operador compost pel nom del tipus de dada al qual es vol convertir seguit d'uns parèntesis, entre els quals va la dada a convertir. Per realitzar la conversió cal que els tipus siguin compatibles. Per exemple:

```

program Moldejat;
var Dies : (dilluns, dimarts, dimecres, dijous, divendres, dissabte, diumenge);
begin
  Dies := dimecres;
  ShowMessage('El número de dia és ' + IntToStr( Byte(Dies) ));
end.

```

### Tipus variant

En ocasions és necessari treballar amb dades el tipus de les quals varia o no pot determinar-se durant la compilació. En aquests casos podem optar per fer servir variables i paràmetres del tipus *variant*, que representen valors el tipus de les quals pot canviar durant l'execució del programa. Els variants ofereixen més flexibilitat, però consumeixen més memòria que les variables normals i les operacions amb ells són més lentes.

Els variants ocupen 16 bytes de memòria i consten d'un codi de tipus i un valor o un punter a un valor del tipus especificat pel codi. En el moment de la seva creació, tots els variants s'inicialitzen amb el valor

especial Unassigned. El valor especial Null representa dades desconegudes o no trobades. La funció VarType retorna el codi de tipus d'un variant.

Veiem un petit exemple:

```
var
  V1, V2: variant;
  I: integer;
  S: string;

begin
  V1 := 1000;           { valor enter }
  V2 := '1000';        { valor string }
  I := V1 + V2;        { I = 2000 (valor enter) }
  V1 := 'Hola!';       { valor string }
  V2 := 1000;          { valor enter }
  S := V2;             { S = '1000' (valor string) }
end.
```

### Tipus matriu dinàmica

La implementació d'Object Pascal a Delphi 5 permet declarar matrius dinàmiques, és a dir, matrius amb varies dimensions els límits de les quals no estan determinats en temps de compilació. La declaració d'una matriu dinàmica es caracteritza perquè en aquesta no s'especifica el nombre d'elements que té cadascuna de les dimensions, pel que darrera la paraula ARRAY no es posen els habituals claudàtors amb el subrang que indiquen els límits. Per exemple:

```
Imatge : ARRAY OF ARRAY OF INTEGER;
```

L'identificador Imatge és nul en un principi, ja que la matriu no disposa inicialment de cap element. Cal establir el nombre d'elements que ha de tenir cadascuna de les dimensions amb la funció SetLength. Per exemple:

```
SetLength(Imatges, 800, 600);
```

A partir d'aquest moment podem accedir als elements de la matriu fent servir la notació habitual, tenint en compte que, a diferència de les matrius estàtiques, el límit inferior de cada dimensió d'una matriu dinàmica és zero. Podem fer servir les funcions Length, Low i High per conèixer el nombre d'elements i els límits de la matriu. Per obtenir una còpia d'una matriu dinàmica hem de fer servir la funció Copy.

|   |  |  |
|---|--|--|
| <pre>var a, b : array [0..1] of integer;  begin   a[0] := 1;   a[1] := 4;   b := a;   b[0] := 2; end.</pre> | <pre>var a, b : array of integer;  begin   SetLength(a,2);   a[0] := 1;   a[1] := 4;   b := a;   b[0] := 2; end.</pre> | <pre>var a, b : array of integer;  begin   SetLength(a,2);   a[0] := 1;   a[1] := 4;   b := Copy(a);   b[0] := 2; end.</pre> |
| a) Còpia de matrius estàtiques. a i b són matrius diferents. Al final a=(1,4) i b=(2,4).                    | b) Còpia de matrius dinàmiques sense Copy. a i b són referències a la mateixa matriu. Al final a=(2,4) i b=(2,4).      | c) Còpia de matrius dinàmiques amb Copy. a i b són referències a matrius diferents. Al final a=(1,4) i b=(2,4).              |

## Declaració de constants

Les constants són identificadors els valors dels quals no pot canviar durant el programa.

La sintaxi bàsica d'una declaració de constant és:

```
const nom_constant = expressió_constant;
```

O bé podem declarar una constant designant-li un tipus concret:

```
const nom_constant : tipus = expressió_constant;
```

Aquí tenim un petit exemple de declaració de constants:

```
program DeclConstants;
```



```
uses
  Dialogs, SysUtils;

const
  RETURN = #13;
  PI : single = 3.1416;
  RADIS : array [1..4] of integer = (1, 2, 4, 10);

var
  i : integer;

begin
  for i := 1 to 4 do
    ShowMessage('L'àrea del cercle de radi ' + IntToStr(RADIS[i]) +
      ' és ' + FloatToStr(RADIS[i]*PI*PI) + RETURN);
  end.
```

## Declaració de nous tipus de dades

Les declaracions de tipus especifiquen un identificador que denota un tipus concret.

La sintaxi bàsica d'una declaració de tipus és:

```
type nom_nou_tipus = tipus;
```

O bé, si volem definir un tipus nou per utilitzar informació de tipus en temps d'execució, la següent sintaxi:

```
type nom_nou_tipus = type tipus;
```

Aquí tenim un petit exemple de declaració de tipus:

```
program DeclTipus;

uses SysUtils;

type
  TPersona = record
    Nom : string[15];
    edat : 0..100;
  end;
  TAlumne = TPersona;           // El compilador no crea un nou tipus
  TProfessor = type TPersona; // El compilador sí crea un nou tipus

var
  pe : TPersona;           // pe és del tipus TPersona
  al : TAlumne;           // al és del tipus TPersona
  pr : TProfessor;       // al és del tipus TProfessor

begin
  pe.Nom := 'Alex';
  pe.Edat := 30;
  al := pe;               // Correcte. Es consideren del mateix tipus.
  pr := pe;               // Incorrecte. Es consideren de tipus diferents.
  pr := TProfessor(pe); // Correcte. Realitzem conversió de tipus.
end.
```

## Classes i objectes

Veure fotocòpies adjuntes.

### Declaració

Declaració, herència i TObject.

És un estàndard de notació a Object Pascal el que les classes s'anomenin precedint el nom de la classe amb la lletra T, que significa tipus. Així, un objecte botó és de la classe TButton.

Tots els objectes disponibles en Delphi s'hereten de la classe base `TObject`. Això fa que Delphi suporti correctament el polimorfisme.

### **Accessibilitat dels membres d'una classe**

Private, Protected, Public, Published i Automated.

### **Atributs**

A Object Pascal s'accedeix a les propietats dels objectes escrivint el nom de l'objecte, a continuació un punt i a continuació el nom de la propietat. Per exemple: `window.size := 100;`

### **Mètodes**

A Object Pascal es crida als mètodes dels objectes escrivint el nom de l'objecte, a continuació un punt i a continuació el nom del mètode amb els paràmetres que li calguin entre parèntesi. Per exemple: `window.resize(100);`

Self.

Inherited.

Overload.

Static, Virtual i Dinamic, Abstract, Override.

Constructors i destructors.

Mètodes de classe

Manejadors de missatges.

### **Propietats**

#### **Operadors de classe**

IS, AS.

ClassType, ClassParent i InheritsFrom.

### **Interfícies d'objectes**

## **Sentències i expressions**

### **Comentaris**

A part dels comentaris estàndard de Pascal (`* comentari *`) i dels comentaris de Turbo Pascal `{comentari}` que poden ocupar diverses línies, ara tenim un nou tipus de comentaris que poden ocupar només una única línia i que són molt còmodes quan aquests van a final d'una sentència. Consisteix en escriure davant el comentari dues barres inclinades consecutives, la seqüència `//`. Als llistats 4.1 i 4.2 anteriors trobeu exemples de la utilització d'aquest nou tipus de comentari.

### **Més d'una instrucció per línia**

Com sempre, una instrucció pot ocupar més d'una línia. Per exemple:

```
Writeln('La teva edat és ',  
        edat);
```

### **Més d'una línia per instrucció**

Com sempre, pot haver més d'una instrucció a cada línia de programa. Per exemple:

```
Write('Com et dius? ');    Readln(nom);
```

## Operadors

| Operador | Operació                      |
|----------|-------------------------------|
| +        | Addició                       |
| -        | Substracció / Canvi de signe  |
| *        | Multiplicació                 |
| /        | Divisió real                  |
| Div      | Quocient de la divisió entera |
| Mod      | Residu de la divisió entera   |

Taula 4.5 Operadors aritmètics.

| Operador | Relació                  |
|----------|--------------------------|
| =        | A és igual a B           |
| <>       | A és diferent de B       |
| <        | A és menor que B         |
| <=       | A és menor o igual que B |
| >        | A és major que B         |
| >=       | A és major o igual que B |

Taula 4.6 Operadors de comparació.

| Operador | Operació            | Torna TRUE si ...                           |
|----------|---------------------|---|
| AND      | Conjunció           | A = TRUE i B = TRUE                         |
| OR       | Disjunció           | A = TRUE o B = TRUE                         |
| XOR      | Disjunció exclusiva | A = TRUE i B = FALSE o A = FALSE i B = TRUE |
| NOT      | Negació             | A = FALSE                                   |

Taula 4.7 Operadors lògics.

| Operador | Operació   |
|----------|--|
| AND      | Conjunció bit a bit (torna 1 si A = 1 i B = 1)                           |
| OR       | Disjunció bit a bit (torna 1 si A = 1 o B = 1)                           |
| XOR      | Disjunció exclusiva bit a bit (torna 1 si A = 1 i B = 0 o A = 0 i B = 1) |
| NOT      | Negació bit a bit (torna 1 si A = 0)                                     |
| SHL      | Desplaça els bits de l'operand cap a l'esquerra.                         |
| SHR      | Desplaça els bits de l'operand cap a la dreta.                           |

Taula 4.8 Operadors de manipulació de bits.

| Operador  | Funció   |
|-----------|--|
| +         | Concatena les dues cadenes                                     |
| =         | Comprova si S <sub>1</sub> és igual a S <sub>2</sub>           |
| <>        | Comprova si S <sub>1</sub> és diferent de S <sub>2</sub>       |
| <         | Comprova si S <sub>1</sub> és menor que S <sub>2</sub>         |
| <=        | Comprova si S <sub>1</sub> és menor o igual que S <sub>2</sub> |
| >         | Comprova si S <sub>1</sub> és major que S <sub>2</sub>         |
| >=        | Comprova si S <sub>1</sub> és major o igual que S <sub>2</sub> |
| Length    | Retorna la longitud d'una cadena                               |
| SetLength | Estableix la longitud d'una cadena                             |

Taula 4.9 Operadors de manipulació de cadenes.

| Operador | Funció  |
|----------|---|
| +        | Unió de dos conjunts                                      |
| *        | Intersecció de dos conjunts                               |
| -        | Resta al conjunt C <sub>1</sub> el conjunt C <sub>2</sub> |
| =        | El conjunt C <sub>1</sub> és igual a C <sub>2</sub>       |
| <>       | El conjunt C <sub>1</sub> és diferent de C <sub>2</sub>   |
| <=       | El conjunt C <sub>1</sub> és subconjunt de C <sub>2</sub> |
| >=       | El conjunt C <sub>2</sub> és diferent de C <sub>1</sub>   |

IN Pertinença a un conjunt

**Taula 4.10** Operadors de manipulació de conjunts.

| Operador | Funció  |
|----------|---|
| +        | Addició de punters  |
| -        | Subtracció de punters                                       |
| ^        | Dades apuntades pel punter                                  |
| @        | Adreça d'una variable o rutina                              |
| =        | P <sub>1</sub> i P <sub>2</sub> apunten a la mateixa adreça |
| <>       | P <sub>1</sub> i P <sub>2</sub> apunten a diferent adreça   |

**Taula 4.11** Operadors de manipulació de punters.

| Operador | Funció   |
|----------|--|
| AS       | Enllaç dinàmic a la interfície                               |
| IS       | Verifica la classe d'un objecte                              |
| =        | O <sub>1</sub> i O <sub>2</sub> apunten al mateix objecte    |
| <>       | O <sub>1</sub> i O <sub>2</sub> apunten a objectes diferents |

**Taula 4.12** Operadors de manipulació de classes.

Quan a una mateixa expressió s'utilitzen varis operadors, s'ha de tenir en compte que a l'avaluació de l'expressió uns operands afectats per un determinat operador són tractats abans que d'altres. L'ordre en que es realitzen les operacions ve determinat per les regles de precedència que podeu veure a la taula 4.13. Quan dos operadors tenen la mateixa precedència, les operacions es realitzen d'esquerre a dreta.

| Operadors                         | Precedència   |
|-----------------------------------|---------------|
| @, not                            | primer (alta) |
| *, /, div, mod, and, shl, shr, as | segon         |
| +, -, or, xor                     | tercer        |
| =, <>, <, >, <=, >=, in, is       | quart (baixa) |

**Taula 4.13** Regles de precedència.

En cas necessari, aquest ordre de prioritat pot ser alterat mitjançant l'ús de parèntesi, mitjançant els quals podem delimitar subexpressions que seran avaluades amb una prioritat superior a qualsevol operador. Per exemple, considereu l'expressió  $X = Y \text{ or } X = Z$ . La interpretació desitjada d'aquesta expressió, òbviament, és  $(X = Y) \text{ or } (X = Z)$ . No obstant, sense els parèntesi el compilador segueix les regles de precedència d'operadors i la interpreta com  $(X = (Y \text{ or } X)) = Z$ , que donarà un error de compilació a menys que Z sigui booleà.

### Sentències

De les següents sentències de Pascal, les úniques realment noves són les sentències de tractament d'errors: `try ... except`, `try ... finally` i `raise`. Tanmateix, refrescarem la memòria recordant la sintaxi i veient un exemple de les diferents sentències.

#### Begin ... End

Delimiten una seqüència d'instruccions a ser executades en l'ordre en que han estat escrites. Les instruccions van separades per punt i coma.

##### Sintaxi

```
begin
  instrucció 1;
  instrucció 2;
  instrucció 3;
  ...
end
```

##### Exemple

```
begin
  i := 5;
  while i > 0 do
  begin
    Writeln(i);
    i := i - 1;
  end;
```

`end;`

### Assignació

Assigna a una variable el resultat d'avaluar una expressió.

#### Sintaxi

`variable := expressió`

#### Exemple

`i := i + 1;`

### With ... do ...

Permet referenciar d'una manera curta els camps d'un registre, o els atributs i mètodes d'un objecte, estalviant-nos escriure el nom d'aquest registre o objecte.

#### Sintaxi

`with objecte do instruccions`

#### Exemple

```
type TData = record
  dia, mes, any: integer;
end;
var Data: TData;
...
with Data do
begin
  mes := 1;
  any := any + 1;
end;
```

### Asm

Permet inserir instruccions en llenguatge ensamblador dins el nostre programa.

#### Sintaxi

```
asm
  instruccions ensamblador
end
```

#### Exemple

```
var x,y,z:integer;
...
asm
  mov eax, x
  add eax, y
  mov z, eax
end;
```

### If ... then ... else ...

Permet seleccionar el conjunt d'instruccions a executar segons s'acompleixi o no una condició.

#### Sintaxi

```
if condició then
  instruccions
[else
  instruccions]
```

#### Exemple

```
if a <> 0 then
  Writeln('Solució: ', -b/a)
else
  if b <> 0 then
    Writeln('No existeix cap solució')
  else
    Writeln('Infinites solucions');
```

### Case ... of ...

Permet seleccionar el conjunt d'instruccions a executar segons el valor d'una variable o expressió.

#### Sintaxi

```
case expressió of
  valors: instruccions;
  ...
  valors: instruccions;
[else
  instruccions;]
end
```

#### Exemple

```
case Ch of
  'A'..'Z','a'..'z': Writeln('Lletra');
  '0'..'9':          Writeln('Dígit');
  '+','-','*','/':  Writeln('Oper. ');
else
  Writeln('Caràcter especial');
end;
```

### While ... do ...

Repeteix un conjunt d'instruccions mentre s'acompleixi una condició.

### Sintaxi

**while** condició **do** instruccions

### Exemple

```
while not Eof(FitxerEntrada) do  
begin  
  Readln(FitxerEntrada, Linia);  
  Writeln(Linia);  
end;
```

### **Repeat ... until ...**

Repeteix un conjunt d'instruccions fins que s'acompleixi una condició.

### Sintaxi

**repeat** instruccions **until** condició

### Exemple

```
repeat  
  Write('Introdueix la nota final');  
  Readln(nota);  
until (nota >= 0) and (nota <= 10);
```

### **For**

Repeteix un conjunt d'instruccions un nombre determinat de vegades.

### Sintaxi

```
for comptador := inici to final do  
  instruccions  
  
for comptador := final downto inici do  
  instruccions
```

### Exemple

```
SumaVect := 0;  
for i := low(Vect) to high(Vect) do  
  SumaVect := SumaVect + Vect[i];
```

### **Crida subrutina**

Consisteix en el nom d'una funció o procediment seguit d'una llista de paràmetres entre parèntesi. La crida a una funció pot estar inserida dins una expressió, mentre que la crida a un procediment és una instrucció de per sí.

### Sintaxi

subrutina(arg1, arg2, ...)

### Exemple

```
area := a*b*sin(angle)/2; (* func. *)  
Writeln(area); (* proc. *)
```

### **GoTo**

Transfereix l'execució del programa a una instrucció marcada amb una determinada etiqueta.

### Sintaxi

```
label etiq1, etiq2, ...;  
...  
goto etiqn;  
...  
etiqn:instrucció;  
...
```

### Exemple

```
label EtiqXiulet;  
...  
EtiqXiulet: Beep;  
goto EtiqXiulet;  
...
```

### **Break / Continue / Exit / Halt**

### Sintaxi

```
break; (* surt d'una iteració *)  
exit; (* surt del bloc actual *)  
halt; (* aborta el programa *)  
continue; (* continua una iteració *)
```

### Exemple

### **Try ... except**

Indica el codi a executar en cas que es produeixi una excepció.

### Sintaxi

```
try  
  instruccions;  
except  
  manipular excepcions;
```

### Exemple

```
try  
  X := Y/Z;  
  ...  
except
```

```
end
|
|   on EZeroDivide do ...;
|   on EOverflow do ...;
|   on EMathError do ...;
| else
|   ...;
| end;
```

### Try ... finally

Assegura que part del codi sigui executat, encara que es produeixi una excepció.

#### Sintaxi

```
try
  instruccions;
finally
  instruccions;
end
```

#### Exemple

```
Reset(Fitxer);
try
  ... // processar fitxer
finally
  CloseFile(Fitxer);
end;
```

### Raise

Genera una excepció.

#### Sintaxi

```
raise excepció
```

#### Exemple

```
type ETrigError = class(EMathError);

function Tan(X: extended): extended;
begin
  try
    Result := Sin(X) / Cos(X);
  except
    on EMathError do
      raise ETrigError.Create('Tangent
                                de 0');
  end;
end;
```

## Funcions i procediments

### Declaració

Per declarar un procediment o funció és necessari especificar el seu nom, el nombre i tipus dels paràmetres que accepta i, en el cas de les funcions, el tipus de valor retornat. Aquesta part de la declaració s'anomena prototipus o capçalera. A continuació, s'escriu un bloc de codi que s'executa quan es crida al procediment o a la funció. Aquesta part s'anomena secció principal o bloc de la rutina.

La declaració de procediments presenta la forma:

```
procedure NomProcediment(LlistaParàmetres); directives;
  DeclaracionsLocals;
begin
  Sentències
end;
```

I la declaració de funcions presenta la forma:

```
function NomFunció(LlistaParàmetres) : TipusRetornat; directives;
  DeclaracionsLocals;
begin
  Sentències
end;
```

Les directives són: cdecl, register, dynamic, virtual, export, external, far, forward, message, override, overload, pascal, reintroduce, safecall, stdcall.

### **Pas de paràmetres per valor, per referència, constants, de sortida i per defecte**

Per impedir que el valor d'una variable corresponent a un paràmetre, ja estigui passat per valor o per referència, sigui modificat dins el procediment o funció, l'únic que hem de fer es disposar la paraula CONST a la llista de paràmetres davant l'identificador, de tal manera que el compilador generi un avís en cas de que per error intentem modificar el seu contingut. El següent exemple donaria error de compilació:

```
procedure ProvaConst(const Num : integer);  
begin  
    Num := 5;  
end;
```

Denotar variables que no hem de modificar com a constants, no és únicament una manera d'evitar errors, sinó que, a més a més, fem que es generi un codi més eficient en la compilació quan aquestes variables són de tipus ARRAY o RECORD.

Per indicar que un paràmetre no aporta informació de entrada però que emmagatzemarà informació de sortida, l'únic que hem de fer es disposar la paraula OUT a la llista de paràmetres davant el seu identificador, de tal manera que el compilador no tingui en compte el valor inicial del paràmetre. Els paràmetres de sortida es passen per referència, i la crida a la rutina allibera immediatament la memòria utilitzada per la variable de sortida, abans que el control del programa passi a la rutina. Per exemple:

```
procedure ProvaOut(out Num : integer);  
begin  
    Num := 5;  
end;
```

Els paràmetres de sortida s'empren freqüentment amb models d'objectes distribuïts com COM i CORBA, així com per passar variables no inicialitzades a funcions i procediments.

L'Object Pascal de Delphi permet que els identificadors de la llista de paràmetres d'un procediment o funció comptin amb un valor per defecte. Aquest valor s'utilitzarà quan, en realitzar una crida al procediment o funció, no es faciliti el paràmetre esmentat. Un mateix procediment pot comptar amb varis paràmetres amb valor per defecte; la única condició és que aquests paràmetres siguin al final de la llista.

Per exemple, la següent funció rep dos paràmetres per defecte, Longitud i Caracter:

```
function Cadena(Longitud : integer = 10; Caracter : char = ' ') : string;  
begin  
    Result := StringOfChar(Caracter, longitud);  
end;
```

i la podem cridar de diverses maneres:

```
Cadena()  
Cadena(20)  
Cadena(15, '*')
```

### **Sobrecàrrega**

Un procediment o funció sobrecarregat és aquell que compta amb vàries implementacions que comparteixen un mateix nom i es diferencien entre si per la llista de paràmetres. A objecte Pascal indiquem que un procediment o funció estan sobrecarregats afegint en terme OVERLOAD al final de la seva capçalera.

Per exemple, les següents declaracions de funcions serien correctes a un programa d'Object Pascal. Segons cridem a la funció Divideix amb uns paràmetres o uns altres, s'executarà el codi de la funció corresponent.

```
function Divideix(X, Y: real): real; overload;  
begin  
    Result := X/Y;  
end;  
  
function Divideix(X, Y: integer): integer; overload;  
begin  
    Result := X div Y;  
end;
```



## Crida

Quan es crida a un procediment o una funció, el control del programa passa del punt on es realitza la crida a la secció principal de la rutina. La crida pot efectuar-se mitjançant el nom declarat de la rutina o mitjançant una variable de procediment que apunti a aquesta rutina. En qualsevol cas, si la rutina es declara amb paràmetres, la crida ha de passar-li els paràmetres corresponents, en l'ordre i amb el tipus especificat en la llista de paràmetres de la rutina. Els paràmetres que es passen a les rutines es denominen paràmetres reals, i els de les declaracions de les rutines, paràmetres formals.

## Variables de procediment

Les variables, a més de poder contenir valors numèrics, caràcters, etc., també són capaces d'emmagatzemar referències a procediments i funcions. Per aconseguir això declarem la variable indicant que contindrà una referència a un procediment o funció i especificant els paràmetres i tipus de dades que aquest retorna. Un cop fet això, podem assignar a la variable l'adreça de qualsevol procediment o funció que s'ajusti a la definició i, a més a més, podem passar com paràmetres referències a procediments i funcions.

Com a exemple suposem que tenim un procediment que aplica a una matriu de valors una determinada funció, obtenint uns resultats que es mostraran per pantalla. Cada cop que haguem d'aplicar una funció distinta a la matriu de valors haurem de retocar el procediment modificant la crida. Per evitar això dissenyem aquest procediment de tal forma que rep com a paràmetre la referència a la funció que ha de cridar i així convertir-se en un procediment més genèric, que cridarà a qualsevol funció facilitada com paràmetre passant cadascun dels valors de la matriu.

```

program VariableProcediment;

uses Dialogs, SysUtils;

type TFuncio = function(N : integer) : longInt;

var X : TFuncio;      // Aquesta variable es del tipus TFuncio
    Cadena : string;

// Una funció que retorna el quadrat d'un nombre
function Quadrat(N : integer) : longInt;
begin
    result := N*N;
end;

// Una funció que retorna el cub d'un nombre
function Cub(N : integer) : longInt;
begin
    result := N*N*N;
end;

// Aquest procediment rep com paràmetres una funció i una matriu
// d'enters, als que aplicarà la funció, mostrant els resultats obtinguts
procedure Calcula(F : TFuncion; M : array of integer);
var i : integer;
begin
    // Recórrer tota la matriu passant cada valor a la funció
    for i := Low(M) to High(M) do
        Cadena := Cadena + IntToStr(M[i]) + '->' + IntToStr(F(M[i])) + #13;
    ShowMessage(Cadena);
end;

begin
    X := Quadrat;          // X conté una referència a la funció Quadrat

    ShowMessage(IntToStr((X(2)))); // Fem servir X per cridar a la funció.

    Cadena := 'Quadrats' + #13;    // Usem el procediment Calcula passant
    Calcula(Quadrat, [2, 4, 6]); // com a paràmetre la funció Quadrat.

    Cadena := 'Cubs' + #13;      // Usem el procediment Calcula passant com
    Calcula(Cub, [2, 4, 6]);    // a paràmetre la funció Cub.

```

| end.

## Entrada i sortida

### Fitxers

Les instruccions típiques Reset, Rewrite, Read i Write es mantenen, mentre que Assign i Close canvien de nom i ara s'anomenen AssignFile i CloseFile.

A més a més, apareixen noves instruccions:

- Per accedir al contingut de fitxers: FileOpen, FileClose, FileRead, FileWrite, FileSeek, ...
- Per accedir als atributs de fitxers: DeleteFile, FileExists, FileSearch, FileGetAttr, FileSetAttr, FileGetDate, FileSetDate, RenameFile, FindFirst, FindNext, FindClose, ...
- Per treballar amb directoris i discos: ChDir, CreateDir, RemoveDir, DirectoryExists, ForceDirectories, GetCurrentDir, GetDir, SetCurrentDir, ...
- Per treballar amb discos: DiskFree, DiskSize, ...

He aquí un petit exemple:

```
// Procediment que processa tots els fitxers d'un directori
// i, recursivament, també tots els seus subdirectoris.
procedure ProcessarDirectorio(const Directorio:string; const Mascara:string);
var Fitx, Dir: TSearchRec;
begin

    (* Primer processem els fitxers *)
    if FindFirst(Directorio+'\' +Mascara, faAnyFile-faDirectory, Fitx) = 0 then
    begin
        ProcessarFitxer(Directorio+'\' +Fitx.name); // Primer fitxer
        while FindNext(Fitx) = 0 do // Següents fitxers
            ProcessarFitxer(Directorio+'\' +Fitx.name);
        FindClose(Fitx);
    end;

    (* Després processem recursivament els subdirectoris *)
    FindFirst(Directorio+'\' *.*', faDirectory, Dir); // Directorio .
    FindNext(Dir); // Directorio ..
    while FindNext(Dir) = 0 do // Següents subdirectoris
        ProcessarDirectorio(Directorio+'\' +Dir.name);
    FindClose(Dir);

end;
```

### InputDialog

La funció InputBox és l'equivalent de la instrucció Readln treballant amb finestres. La seva sintaxi és:

```
function InputBox(const Capçalera, Missatge, ValorDefecte: string): string;
```

A continuació podeu veure un exemple del seu ús:

```
Edat := StrToInt(InputBox('Edat', 'Quants anys tens? ', '0'));
```

### MessageBox i ShowMessage

La instrucció MessageBox i la funció ShowMessage són els equivalents de la instrucció Writeln treballant amb finestres. La seva sintaxi és:

```
procedure ShowMessage(const Missatge: string);

function MessageDlg(const Missatge: string; TipusDialog: TMsgDlgType;
    Botons: TMsgDlgButtons; AjudaContext: Longint): Word;
```

A continuació podeu veure uns exemples del seu ús:

```

if FitxerATrobar = '' then
    ShowMessage('No puc trobar ' + Edit1.Text + '.')
else
    ShowMessage('Trobat ' + FileToFind + '.');

if MessageDlg('Sortir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
        MessageDlg('Sortint!', mtInformation, [mbOk], 0);
        Close;
    end;

```

## Control d'errors

Veure fotocòpies adjuntes.

## Mòduls, llibreries d'enllaç dinàmic i paquets

### Mòduls

Habitualment, una aplicació estarà composta per tants procediments i funcions que la seva estructura en un únic arxiu de codi serà complexa, ja que aquest serà massa extens com perquè la seva manipulació sigui còmoda. És aquí on sorgeix la necessitat de repartir el codi de l'aplicació en varis arxius separats (mòduls o unitats), contenint cadascun aquells tipus (incloent classes), constants, variables i rutines (funcions i procediments), que guarden una certa relació entre sí. Això facilita la divisió d'una aplicació complexa en múltiples unitats més fàcils de mantenir.

Una altra aplicació dels mòduls consisteix en contenir codi d'ús comú, estructurat en procediments, funcions i objectes. Un mòdul, un cop compilat, pot ser utilitzat des de qualsevol programa que el necessiti sense necessitat de disposar del codi font. Basta conèixer la interfície pública d'aquest mòdul, formada pels identificadors de procediments i el coneixement dels paràmetres necessaris.

Cada mòdul o unitat està definit en el seu propi fitxer .PAS. El fitxer d'un mòdul comença amb la capçalera `unit`, seguida per les seccions d'interfície, implementació, inicialització i finalització. Aquestes dues últimes seccions, inicialització i finalització, són opcionals. El mòdul conclou amb la paraula `end` seguida d'un punt i final. L'estructura del fitxer d'un mòdul és la següent:

```

unit NomUnitat;

interface

uses { La llista d'unitats va aquí }
    { La secció d'interfície va aquí }

implementation

uses { La llista d'unitats va aquí }
    { La secció d'implementació va aquí }

initialization
    { La secció d'inicialització va aquí }

finalization
    { La secció de finalització va aquí }

end.

```

Com podeu comprovar, l'estructura d'un mòdul s'assembla en gran mesura a la de qualsevol programa d'Object Pascal, contenint una capçalera, unes declaracions i els corresponents blocs de codi. A diferència d'un programa, però, els identificadors d'un mòdul poden ser locals o públics, cas aquest últim en el que poden ser emprats des d'altre programes o unitats alienes a aquella en que s'ha declarat. Tots els

identificadors globals del mòdul més les capçaleres de les funcions i procediments que més endavant es definiran formen part de la interfície, nom amb que es coneix la porció del mòdul posada a disposició d'altres programes, mentre que els identificadors locals o privats no accessibles a altres programes més la definició dels procediments i funcions prèviament declarats formen part de la implementació.

A més de la capçalera, la part interfície i la part d'implementació, un mòdul també pot tenir, opcionalment, una part d'inicialització, consistent en un bloc de codi similar al bloc principal de qualsevol programa i que s'executa quan es carrega el mòdul, i una altre part de finalització que s'executa en descarregar-se el mòdul.

### **Llibreries d'enllaç dinàmic**

Independentment del nombre de mòduls en que dividim el codi de la nostre aplicació, un cop aquesta sigui compilada tot el codi objecte s'allotjarà en un únic arxiu executable. Quan una aplicació és de certa complexitat, l'arxiu executable pot ser bastant gran, el que dificulta la seva distribució i actualització. Delphi ens permet dividir qualsevol aplicació en un programa executable, amb extensió .EXE, i un o més fitxers, amb extensió .DLL, emmagatzemant diferents mòduls de programa. Aquest esquema de funcionament, a més de facilitar les tasques de distribució i actualització, que poden fer-se a base a paquets o mòduls enlloc d'afectar a tot el programa sencer, permet que dues o més aplicacions comparteixin el codi d'una única DLL, reduint així la càrrega del sistema.

Les llibreries d'enllaç dinàmic (*Dinamic Link Library*) són col·leccions de rutines que poden ser cridades en temps d'execució tant per altres aplicacions com per altres DLL. Els programes d'Object Pascal poden cridar a DLL escrites en altres llenguatges i, per la seva part, les aplicacions per Windows escrites en altres llenguatges poden cridar a DLL desenvolupades en Object Pascal.

Per cridar des d'un programa escrit en Object Pascal a una rutina definida a una DLL, abans cal 'importar' aquesta rutina. Hi ha dues maneres de fer això: la càrrega estàtica i la càrrega dinàmica.

La càrrega estàtica utilitza la directiva `external` per carregar a inici del programa les DLL que contenen rutines que el nostre codi utilitza. Durant l'execució del nostre programa, els identificadors d'aquestes rutines sempre fan referència al mateix punt d'entrada de les mateixes DLL. La sintaxi seria la següent:

```
procedure NomRutina; external 'NOMLLIBRERIA.DLL';
```

La càrrega dinàmica fa ús de les funcions de gestió de biblioteques de Windows. D'entre d'altres: `LoadLibrary`, `FreeLibrary` i `GetProcAddress`. En aquest cas, la DLL no es carrega fins que no s'executa el codi que conté la crida a `LoadLibrary`. Posteriorment, la DLL es descarrega mitjançant la crida a `FreeLibrary`. Això permet economitzar memòria i executar el programa encara que algunes de les DLL no hi siguin presents.

Si el que volem és crear una DLL, l'estructura del fitxer és la mateixa que la d'un programa normal, amb l'excepció de que comença per la paraula `library` i conté una clàusula `export` que indica quines rutines del fitxer s'exporten:

```
library NomLlibreria;  
  
uses { La llista d'unitats va aquí }  
  
declaració de funcions i procediments (amb stdcall)  
  
exports { La llista de rutines a exportar va aquí }  
  NomRutina1 [index NumIndex1] [name NouNomRutina1];  
  NomRutina2 [index NumIndex2] [name NouNomRutina2];  
  ...  
  
begin  
  { El codi d'inicialització de la llibreria va aquí }  
  { S'executa sempre quan es carrega una llibreria. }  
end.
```

## Paquets

Els paquets són llibreries d'enllaços dinàmics, compilades de forma especial, que fan servir les aplicacions Delphi. Per crear un paquet hem d'escriure un arxiu de projecte que contindrà bàsicament quatre dades: el nom del paquet, una descripció, referències a d'altres paquets i la llista de mòduls que contindrà el paquet resultant. El resultat de la compilació d'aquest projecte serà una llibreria d'enllaç dinàmic especial, amb extensió `.BPL`, preparada per ser usada des de qualsevol programa Delphi. L'estructura del fitxer d'un paquet és la següent:

```

package NomPaquet;

{$DESCRIPTION 'descripció del paquet'}

requires { La llista de paquets va aquí }

contains { La llista de mòduls va aquí }

end.
```

Abans de compilar el paquet hem de guardar l'arxiu amb extensió `.DPK`. A continuació tanquem la corresponent finestra de l'editor de codi i la tornem a obrir, fent aparèixer la finestra del l'editor de paquets. El procés de compilació generarà un nou arxiu amb el mateix nom però amb extensió `.BPL`. Aquest fitxer és una DLL de Windows amb característiques especials específiques de Delphi. Per fer servir el paquet dins un programa només cal carregar el fitxer projecte del programa i seleccionar al menú de Delphi les opcions *Project* → *Options* → *Packages* → *Add* → *Runtime packages* i cercant l'esmentada llibreria.

## Exemple 4-1: POO – creació d'una classe

Creu una classe `TPila` amb la següent estructura:

|             |                      |  |
|-------------|----------------------|--|
| Atribut     | <code>Pila</code>    | Vector d'enters que conté els elements de la pila.         |
| Atribut     | <code>Cim</code>     | Enter que indica el nombre d'elements de la pila.          |
| Constructor | <code>Create</code>  | Crea una pila amb la capacitat especificada.               |
| Destructor  | <code>Destroy</code> | Destruïx la pila i allibera l'espai a memòria.             |
| Mètode      | <code>Ficar</code>   | Introdueix l'element especificat al final de la pila.      |
| Mètode      | <code>Treure</code>  | Elimina l'últim element de la pila i retorna el seu valor. |

Aquesta classe ha de detectar desbordaments positius i negatius de la pila, llençant les excepcions `EPilaBuida` i `EPilaPlena` quan s'intenti treure un element d'una pila buida o ficar un element a una pila plena.

Solució:

```

unit Pila;

interface

uses SysUtils;

type
  TElement = integer;
  EPilaBuida = class(Exception);
  EPilaPlena = class(Exception);

type TPila = class
  private
    Pila : array of TElement;
    Cim : integer;
```

```
public
  constructor Create(Capacitat : integer);
  procedure Ficar(Element : TElement);
  procedure Treure(var Element : TElement);
  destructor Destroy; override;
end;

implementation

constructor TPila.Create(Capacitat : integer);
begin
  inherited Create;
  SetLength(Pila, Capacitat);
  Cim := 0;
end;

procedure TPila.Ficar(Element : TElement);
begin
  if Cim = High(Pila)+1 then
    Raise EPilaPlena.Create('pila plena')
  else
    begin
      Pila[Cim] := Element;
      Cim := Cim + 1;
    end;
end;

procedure TPila.Treure(var Element : TElement);
begin
  if Cim = 0 then
    Raise EPilaBuida.Create('pila buida')
  else
    begin
      Cim := Cim - 1;
      Element := Pila[Cim];
    end;
end;

destructor TPila.Destroy;
begin
  Pila := nil;
  inherited Destroy;
end;

end.
```

```
program Exemple4_1; //Programa de prova per a la classe TPila

{$APPTYPE CONSOLE}

uses SysUtils, Pila in 'pila.pas';

var
  a : TPila;
  x : TElement;
  i : integer;

begin
  a := TPila.Create(10);

  try
    for i := 1 to 11 do
      a.Ficar(i);
    except
      on E: Exception do Writeln(E.Message);
    end;
  end;
```

```

try
  for i := 1 to 11 do
    begin
      a.Treure(x);
      writeln(x);
    end
  except
    on E: Exception do Writeln(E.Message);
  end;

  a.Free;
end.

```

## Exemple 4-2: POO – herència

Disposem de la classe TobjecteGeom:

```

type TobjecteGeom = class
  private
    X, Y : real;
  public
    constructor Create(X : real = 0; Y : real = 0);
    procedure VeureCentre();
  end;

constructor TobjecteGeom.Create(X : real = 0; Y : real = 0);
begin
  inherited Create;
  Self.X := X;
  Self.Y := Y;
end;

procedure TobjecteGeom.VeureCentre();
begin
  Writeln('Centre : ( ', X:3:2, ' , ', Y:3:2, ' )');
end;

```

Dissenyeu les classe TCercle i TQuadrat derivades de TobjecteGeom que permetin calcular les seves àrees. Un cop dissenyades totes les classes, escriviu un programa que creï un objecte de cada classe, visualitzi els centres de cada figura i a continuació calculi i visualitzi les àrees de cada figura.

Solució:

```

unit Geometria;

interface

type TobjecteGeom = class
  private
    X, Y : real;
  public
    constructor Create(X : real = 0; Y : real = 0);
    procedure VeureCentre();
  end;

type TCercle = class(TobjecteGeom)
  private
    Radi : real;
  public
    constructor Create(Radi : real; X : real = 0; Y : real = 0);
    procedure VeureRadi();

```

```
    procedure VeureArea();
end;

type TQuadrat = class(TObjecteGeom)
private
    Costat : real;
public
    constructor Create(Costat : real; X : real = 0; Y : real = 0);
    procedure VeureCostat();
    procedure VeureArea();
end;

implementation

constructor TObjecteGeom.Create(X : real = 0; Y : real = 0);
begin
    inherited Create;
    Self.X := X;
    Self.Y := Y;
end;

procedure TObjecteGeom.VeureCentre();
begin
    Writeln('Centre : ( ', X:3:2, ' , ', Y:3:2, ' )');
end;

constructor TCercle.Create(Radi : real; X : real = 0; Y : real = 0);
begin
    inherited Create(X, Y);
    Self.Radi := Radi;
end;

procedure TCercle.VeureRadi();
begin
    Writeln('Radi : ', Radi:3:2);
end;

procedure TCercle.VeureArea();
begin
    Writeln('Area : ', PI*Sqr(Radi):3:2);
end;

constructor TQuadrat.Create(Costat : real; X : real = 0; Y : real = 0);
begin
    inherited Create(X, Y);
    Self.Costat := Costat;
end;

procedure TQuadrat.VeureCostat();
begin
    Writeln('Costat : ', Costat:3:2);
end;

procedure TQuadrat.VeureArea();
begin
    Writeln('Area : ', Sqr(Costat):3:2);
end;

end.

program Exemple4_2; //Programa de prova de la unitat Geometria
{$APPTYPE CONSOLE}
uses Geometria in 'geometria.pas';
```



```

var
  a : TObjecteGeom;
  b : TCercle;
  c : TQuadrat;

begin
  a := TObjecteGeom.Create(3, -2.1);
  b := TCercle.Create(10, 1, 1);
  c := TQuadrat.Create(5);

  Writeln('OBJECTE GEOMETRIC');
  Writeln('-----');
  a.VeureCentre();
  Writeln;

  Writeln('CERCLE');
  Writeln('-----');
  b.VeureCentre();
  b.VeureRadi();
  b.VeureArea();
  Writeln;

  Writeln('QUADRAT');
  Writeln('-----');
  c.VeureCentre();
  c.VeureCostat();
  c.VeureArea();
  Writeln;

  a.free();
  b.free();
  c.free();
end.

```

## Exemple 4-3: POO – classes i mètodes virtuals

Un magatzem de components elèctrics desitja crear fitxes que emmagatzemin la descripció, les existències i el preu de cada component. Crear la classe TComponent que implementi les dades anteriors. A partir d'aquesta classe, dissenyar tres classes derivades: TResistencia, amb el valor en ohms de la resistència; TCondensador, amb el valor en faradis de la capacitat; i TBobina amb el valor en henris de la inductància. Cadascuna de les tres classes tindrà un mètode Llegir per introduir les dades i una funció Mostrar per visualitzar les dades. A més a més, crearem una nova classe anomenada TCateg que contindrà un vector de components (resistències, condensadors i bobines) amb les existències del magatzem. Aquesta classe també disposarà dels mètodes Llegir i Mostrar.

### Solució:

```

program Exemple4_1c;

{$APPTYPE CONSOLE}

uses SysUtils;

type TComponent = class
  private
    Descripcio : string[40];
    Existències : integer;
    Preu : currency;
  public
    procedure Llegir(); virtual;

```

```
    procedure Mostrar(); virtual;
end;

type TResistencia = class(TComponent)
private
    Resistencia : real;
public
    procedure Llegir(); override;
    procedure Mostrar(); override;
end;

type TCondensador = class(TComponent)
private
    Capacitancia : real;
public
    procedure Llegir(); override;
    procedure Mostrar(); override;
end;

type TBobina = class(TComponent)
private
    Inductancia : real;
public
    procedure Llegir(); override;
    procedure Mostrar(); override;
end;

type TCataleg = class
private
    Cataleg : array of TComponent;
public
    constructor Create(Capacitat : integer);
    procedure Llegir();
    procedure Mostrar();
    destructor Destroy; override;
end;

procedure TComponent.Llegir();
begin
    Write('Descripció ? ');
    Readln(Descripció);
    Write('Existències ? ');
    Readln(Existències);
    Write('Preu ? ');
    Readln(Preu);
end;

procedure TComponent.Mostrar();
begin
    Writeln('Descripció : ', Descripció);
    Writeln('Existències : ', Existències, ' unitats');
    Writeln('Preu : ', Preu:4:2, ' pessetes');
end;

procedure TResistencia.Llegir();
begin
    inherited Llegir();
    Write('Resistència en ohms ? ');
    Readln(Resistència);
end;

procedure TResistencia.Mostrar();
begin
    inherited Mostrar();
    Writeln('Resistència : ', Resistència:4:2, ' ohms');
end;

procedure TCondensador.Llegir();
```

```

begin
  inherited Llegir();
  Write('Capacitancia en faradis ? ');
  Readln(Capacitancia);
end;

procedure TCondensador.Mostrar();
begin
  inherited Mostrar();
  Writeln('Capacitancia : ', Capacitancia:4:2, ' faradis');
end;

procedure TBobina.Llegir();
begin
  inherited Llegir();
  Write('Inductancia en henris ? ');
  Readln(Inductancia);
end;

procedure TBobina.Mostrar();
begin
  inherited Mostrar();
  Writeln('Inductancia : ', Inductancia:4:2, ' henris');
end;

constructor TCataleg.Create(Capacitat : integer);
begin
  inherited Create;
  SetLength(Cataleg, Capacitat);
end;

procedure TCataleg.Llegir();
var
  i : integer;
  Tipus : char;
begin
  for i := 0 to high(Cataleg) do
    begin
      WriteLn('Component ', i+1);
      WriteLn('-----');
      repeat
        Write('Tipus de component ([R]esistencia/[C]ondensador/[B]obina) ? ');
        Readln(Tipus);
        case Tipus of
          'R', 'r' : Cataleg[i] := TResistencia.Create;
          'C', 'c' : Cataleg[i] := TCondensador.Create;
          'B', 'b' : Cataleg[i] := TBobina.Create;
          else WriteLn('Component desconegut. Torní-ho a provar. ');
        end;
      until Tipus in ['R', 'r', 'C', 'c', 'B', 'b'];
      Cataleg[i].Llegir();
      WriteLn;
    end;
end;

procedure TCataleg.Mostrar();
var i : integer;
begin
  for i := 0 to high(Cataleg) do
    begin
      WriteLn('Component ', i+1);
      WriteLn('-----');
      Cataleg[i].Mostrar();
      WriteLn;
    end;
end;

destructor TCataleg.Destroy();

```

```
var i : integer;
begin
  for i := 0 to high(Cataleg) do Cataleg[i].Free;
  Cataleg := nil;
  inherited destroy;
end;

var
  Components : TCataleg;
  NumComponents : integer;

// Programa principal
begin
  Writeln('MAGATZEM DE COMPONENTS ELECTRICS');
  Writeln('-----');
  Write('Quants components tenim en cataleg ? ');
  Readln(NumComponents);
  Writeln;

  Components := TCataleg.Create(NumComponents);
  Components.Llegir();
  Components.Mostrar();
  Components.Free;
end.
```

## Exemple 4-4: maneigament d'excepcions

Considerem el procediment `Arrels` que calcula les arrels quadrades d'una equació de segon grau. Dissenyem el procediment de manera que llenci excepcions si no existeixen arrels reals o si el primer coeficient és zero. Les excepcions seran `ENoArrelsReals` i `EPrimerCoeficientZero`, descendents de la classe d'excepció `EMathError`.

Dissenyem també el programa principal que llegeix els coeficients de l'equació, fa una crida al procediment `Arrels` i escriu el resultat per pantalla, gestionant en tot moment les excepcions que es puguin produir.

### Solució:

```
program Exemple4_4;

uses SysUtils, Dialogs;

type
  ENoArrelsReals = class(EMathError);
  EPrimerCoeficientZero = class(EMathError);

var
  a, b, c, sol1, sol2: single;

(* Procediment que calcula les arrels reals d'una equació de segon grau. *)
procedure Arrels(const coef1, coef2, coef3: single; var arrel1, arrel2: single);
var discr: single;
begin
  (* Comprovem que la equació sigui de segon grau *)
  if coef1 = 0 then
    raise EPrimerCoeficientZero.Create('L'equació no és de segon grau');

  (* Calculem el discriminant i comprovem que les arrels siguin reals *)
  discr := b*b - 4*a*c;
  if discr < 0 then
```

```

    raise ENoArrelsReals.Create('Les arrels són imaginàries');

(* Calculem les arrels reals *)
arrel1 := (-b - Sqrt(discr))/(2*a);
arrel2 := (-b + Sqrt(discr))/(2*a);

// Una altra manera de fer el mateix hagués estat:
//
// try
//   discr := b*b - 4*a*c;
//   arrel1 := (-b - Sqrt(discr))/(2*a);
//   arrel2 := (-b + Sqrt(discr))/(2*a);
// except
//   on EZeroDivide do
//     raise EPrimerCoeficientZero.Create('L'equació no és de segon grau');
//   on EMathError do
//     raise ENoArrelsReals.Create('Les arrels són imaginàries');
//   end;
end;

begin
  try

    (* Llegim els coeficients de l'equació *)
    a := StrToFloat(InputBox('Equació quadràtica', 'Coeficient grau 2', '0'));
    b := StrToFloat(InputBox('Equació quadràtica', 'Coeficient grau 1', '0'));
    c := StrToFloat(InputBox('Equació quadràtica', 'Coeficient grau 0', '0'));

    (* Calculem les arrels *)
    Arrels(a, b, c, sol1, sol2);

    (* Visualitzem les arrels *)
    ShowMessageFmt('Arrel 1 : %4.4f' + #13 + 'Arrel 2 : %4.4f', [sol1, sol2]);

  except

    (* Si s'ha produït error, tant de càlcul com de conversió, *)
    (* mostrem el missatge d'error a l'usuari. *)
    on E: Exception do MessageDlg(E.Message, mtError, [mbOK], 0);

    // També hauriem pogut gestionar les excepcions individualment, una a una:
    //
    //   on EPrimerCoeficientZero do ...
    //   on ENoArrelsReals do ...
    //   on EConvertError do ...

  end;
end.

```

## Exercici 4-1: POO – creació d'una classe

Dissenyem una classe que guarda a una cua les dades que anem obtenim a un experiment, i és capaç de realitzar alguna operació sobre aquestes dades, per exemple l'obtenció de la mitjana i la desviació estàndard segons les següents fórmules:

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N} \quad \text{i} \quad \sigma^2 = \frac{\sum_{i=1}^N x_i^2 - N \left( \frac{\sum_{i=1}^N x_i}{N} \right)^2}{N-1}$$

on  $N$  és el nombre de valors introduïts.

La classe TExperiment pot tenir la següent estructura:

|             |                    |  |
|-------------|--------------------|--|
| Atribut     | Dades : TCua       | Cua de reals que conté les dades de l'experiment.            |
| Constructor | Create (integer)   | Crea un experiment amb la capacitat de dades especificada.   |
| Destructor  | Destroy            | Destruïx l'experiment i allibera l'espai de les dades.       |
| Mètode      | Obtenir()          | Obté una nova dada de l'experiment i la fica a la cua.       |
| Mètode      | Mitjana() : real   | Calcula la mitja de les dades de l'experiment.               |
| Mètode      | Desviacio() : real | Calcula la desviació estàndard de les dades de l'experiment. |

Si un objecte de la classe TExperiment està ple de dades, quan entra una nova dada desapareix la més antiga.

Per a emmagatzemar les dades de l'experiment feu servir la classe TCua, que a continuació us proporciona. La seva interfície ve donada per:

|             |                              |  |
|-------------|------------------------------|--|
| Constructor | Create (integer)             | Crea una cua de capacitat especificada.    |
| Destructor  | Destroy                      | Destruïx la cua i allibera el seu espai.   |
| Mètode      | Ficar (TElement)             | Introdueix una dada a la cua.              |
| Mètode      | Treure (var TElement)        | Treu i retorna una dada de la cua.         |
| Mètode      | Capacitat() : integer        | Retorna la capacitat de la cua.            |
| Mètode      | NumElements() : integer      | Retorna el nombre de dades a la cua.       |
| Mètode      | Element (integer) : TElement | Retorna la dada a la posició especificada. |
| Mètode      | Buida() : boolean            | Retorna si la cua és buida o no.           |
| Mètode      | Plena() : boolean            | Retorna si la cua és plena o no.           |

Un objecte de la classe TCua generarà una excepció a qualsevol dels tres següents casos: quan la cua està plena i s'intenta ficar un nou element (excepció ECuaPlena), quan la cua està buida i s'intenta treure un element (excepció ECuaBuida), i quan s'intenta accedir a un element que no existeix (excepció EForaRang).

```
unit Cua;  
  
interface  
  
uses SysUtils;  
  
type  
  TElement = integer;  
  ECuaBuida = class(Exception);  
  ECuaPlena = class(Exception);  
  EForaRang = class(Exception);  
  
type TCua = class  
  private  
    Cua : array of TElement;  
    Primer, Ultim, Num : integer;  
  public  
    constructor Create(Capacitat : integer);  
    procedure Ficar(Element : TElement);  
    procedure Treure(var Element : TElement);  
    function Capacitat() : integer;  
    function NumElements() : integer;  
    function Element(n:integer) : TElement;  
    function Buida() : boolean;  
    function Plena() : boolean;  
    destructor Destroy; override;  
  end;  
  
implementation  
  
constructor TCua.Create(Capacitat : integer);  
begin  
  inherited Create;  
  SetLength(Cua, Capacitat);
```

```

    Primer := 0;
    Ultim := -1;
    Num := 0;
end;

procedure TCua.Ficar(Element : TElement);
begin
    if Num = High(Cua) + 1 then
        Raise ECuaPlena.Create('cua plena')
    else
        begin
            Ultim := (Ultim + 1) mod (High(Cua) + 1);
            Cua[Ultim] := Element;
            Num := Num + 1;
        end;
end;

procedure TCua.Treure(var Element : TElement);
begin
    if Num = 0 then
        Raise ECuaBuida.Create('cua buida')
    else
        begin
            Element := Cua[Primer];
            Primer := (Primer + 1) mod (High(Cua) + 1);
            Num := Num - 1;
        end;
end;

function TCua.Capacitat():integer;
begin
    Result := High(Cua) + 1;
end;

function TCua.NumElements():integer;
begin
    Result := Num;
end;

function TCua.Element(n:integer):TElement;
begin
    if (n < 1) or (n > Num) then
        Raise EForaRang.Create('element ' + IntToStr(n) + ' no existeix')
    else
        begin
            Result := Cua[(Primer + n - 1) mod (High(Cua) + 1)];
        end;
end;

function TCua.Buida():boolean;
begin
    Result := (Num = 0);
end;

function TCua.Plena():boolean;
begin
    Result := (Num = High(Cua) + 1);
end;

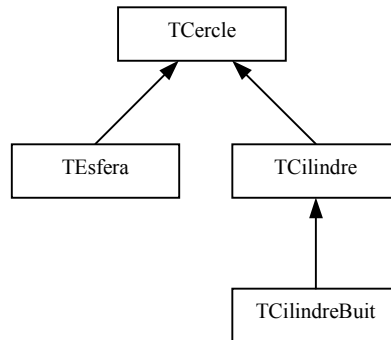
destructor TCua.Destroy;
begin
    Cua := nil;
    inherited Destroy;
end;

end.

```

## Exercici 4-2: POO – herència

Dissenyar una jerarquia de classes: TCercle, TEsfera, TCilindre i TCilindreBuit.



- La classe TCercle té un atribut Radi i els mètodes LlegirRadi, VeureArea i VeurePerimetre.
- La classe TEsfera té un atribut Radi i els mètodes LlegirRadi, VeureArea i VeureVolum.
- La classe TCilindre té els atributs Radi i Alçada i els mètodes LlegirRadi, LlegirAlçada, VeureArea i VeureVolum.
- La classe TCilindreBuit té els atributs Radi, RadiIntern i Alçada i els mètodes LlegirRadi, LlegirRadiIntern, LlegirAlçada, VeureArea i VeureVolum.

Escriu un petit programa que provi les classes que heu dissenyat.

Fòrmules:

|               |   |  |
|---------------|---|--|
| Cercle        | $\text{Àrea} = \pi r^2$   | $\text{Perímetre} = 2\pi r$                        |
| Esfera        | $\text{Àrea} = 4\pi r^2$  | $\text{Volum} = 4\pi r^3 / 3$                      |
| Cilindre      | $\text{Àrea} = 2\pi r h + 2\pi r^2$   | $\text{Volum} = \pi r^2 h$                         |
| Cilindre buit | $\text{Àrea} = 2\pi (r + r_{\text{intern}}) h + 2\pi (r^2 - r_{\text{intern}}^2)$ | $\text{Volum} = \pi (r^2 - r_{\text{intern}}^2) h$ |



# Components VCL

En aquest capítol veurem una descripció dels components més importants de la llibreria VCL de Delphi, amb les seves propietats, esdeveniments i mètodes. Primer descriurem les propietats, esdeveniments i mètodes comuns a la majoria de components, i a continuació els particulars de cada component en concret.

Abans de començar recordem dos conceptes:

- Anomenem control a un component visual, com per exemple un botó. El component `Timer`, en canvi, no és un control, ja que no té una representació visual quan s'executa l'aplicació.
- Anomenem contenidor a un component que pot agrupar dins seu altres components. Exemples de contenidors serien `Form`, `Panel` i `GroupBox`. Els components agrupats dins un contenidor hereten d'aquest el color de fons i el tipus de lletra (és a dir, que si canviem el color o el tipus de lletra del contenidor, per defecte també canvien els dels components que conté).

## Propietats, esdeveniments i mètodes generals

### Propietats

- **Name:** És l'identificador del component dins el programa. Aquesta propietat no es pot canviar en temps d'execució. En temps de disseny, quan canviem el nom d'un component amb l'inspector d'objectes, automàticament queda canviat el nom a tot el codi del programa.
- **Caption:** Especifica una cadena de caràcters que etiqueta el control de cara a l'usuari. En el text que escrivim dins aquesta propietat podem incloure el caràcter `&`. Aquest caràcter denota que el següent caràcter serà la tecla d'accés ràpid al component. És a dir, si el component és un botó o una opció del menú, en addició a fer clic sobre el component per invocar un esdeveniment, també podem prémer la seva tecla d'accés en conjunció amb la tecla `Alt`. Noteu que als components la tecla d'accés apareixerà subratllada.
- **Left** i **Top:** Indiquen la posició d'un control respecte a l'interior del seu contenidor. En el cas especial del control formulari, indiquen la posició absoluta del formulari a pantalla. Són les coordenades horitzontal i vertical, respectivament, de la cantonada superior esquerra del control. Aquestes dues propietats es mesuren en píxels.
- **Width** i **Height:** Indiquen l'alçada i l'amplada, respectivament, d'un control. Aquestes dues propietats es mesuren en píxels.
- **BoundsRect:** Especifica el límit rectangular del control, expressat en el sistema de coordenades del seu contenidor. S'empra per obtenir a la vegada la localització dels quatre píxels que formen les cantonades del control.
- **Align:** Facilita l'alineació d'un component a un dels marges del seu contenidor, o bé ocupant tot l'espai disponible.
- **Anchor:** Especifica com queda ancorat el control dins el seu contenidor. S'empra per assegurar que un control manté la seva posició relativa als marges del seu contenidor, encara que aquest últim sigui redimensionat.
- **Constraints:** Especifica l'amplada i alçada màximes i mínimes del control. Quan el control sigui redimensionat no podrà violar aquestes cotes.
- **Enabled:** Indica si el component es troba actiu o inactiu, és a dir, si l'usuari pot o no interactuar amb ell i, a més, apareixerà amb una aparença o una altre en pantalla.

- **Visible:** Controla la visibilitat d'un control, és a dir, indica si el control es visualitza per pantalla o no. Cridar el mètode `Show` del control establirà el valor de la propietat `Visible` a `True`. Cridar el mètode `Hide` establirà el seu valor a `False`.
- **Hint i ShowHint:** La propietat `Hint` conté un text d'ajuda que apareix a una petita caixa quan l'usuari mou el punter del ratolí sobre un control i espera uns instants. Per tal que aparegui cal que la propietat `ShowHint` tingui el valor `True`.
- **PopupMenu:** Identifica el menú contextual associat a un control. Si assignem un valor a `PopupMenu` farem que aparegui un menú contextual quan l'usuari faci clic sobre el control amb el botó dret del ratolí.
- **HelpContext:** Proporciona un nombre que determina que pantalla d'ajuda apareix quan l'usuari crida a l'ajut contextual (apareix quan l'usuari prem la tecla `F1`). Cada pantalla en el sistema d'ajuda té un nombre de context únic. Si `HelpContext` val zero, el control hereta l'ajuda contextual del seu contenidor.
- **Ct13D:** Determina si un control té una aparença tridimensional (valor `True`) o bidimensional (valor `False`).
- **BevelInner i BevelOuter:** Especifiquen, respectivament, el tall interior i exterior del bisell d'un control, proporcionant a aquest un aspecte elevat, enfonsat o pla.
- **BorderWidth:** Especifica, en píxels, la distància dels marges d'un contenidor fins a l'àrea dins la qual es poden col·locar controls.
- **Font:** Especifica els atribut del text escrit sobre o dins el control. Dintre seu conté altres propietats:
  - `Name` i `CharSet`: la propietat `Name` especifica el nom de la font (Arial, Courier, Times New Roman, Symbol, etc.), mentre que la propietat `CharSet` especifica el joc de caràcters dins la font escollida (ANSI, rus, hebreu, àrab, etc.).
  - `Pitch`: especifica si volem la font equiespaiada o no.
  - `Size` i `Height`: mida en punts de la font i alçada en píxels de la font, respectivament.
  - `Style`: especifica l'estil (negreta, cursiva, subratllada, ttxada).
  - `Color`: color de la font.
- **Color:** La propietat `Color` especifica el color de fons del control.
- **Canvas:** Proporciona accés a l'àrea de dibuix del control, juntament amb totes les eines i mètodes necessaris per dibuixar i pintar. Per la seva importància, les propietats i mètodes del component `Canvas` les comentarem apart més endavant en aquest mateix capítol.
- **Cursor:** Especifica la imatge emprada per representar el punter del ratolí quan passa sobre la regió coberta pel control. El valor contingut a la propietat `Cursor` és l'índex del cursor a la llista de cursors continguda a la variable global `Screen`. A més a més, podem afegir nous cursors fets a mida a les nostres aplicacions, tal com veiem al següent exemple:

```
const crMyCursor = 5;  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    Screen.Cursors[crMyCursor] := LoadCursor(HInstance, 'NewCursor');  
    Cursor := crMyCursor;  
end;
```

Aquest codi assumeix que hem afegit un cursor a mida amb el nom `NewCursor` al fitxer de recursos (`.res`) de l'aplicació. Podem afegir el cursor mitjançant l'editor d'imatges (*Tools* → *Image Editor*). El codi fa el cursor disponible a l'aplicació mitjançant la constant `crMyCursor`, i l'activa com a cursor global de l'aplicació.

- **AutoScroll, HorzScrollBar i VertScrollBar:** La propietat `AutoScroll` d'alguns controls indica si apareix una barra de desplaçament automàticament quan el control no és suficientment gran per visualitzar tot el que conté. Si el valor de la propietat `AutoScroll` és `True`, les barres de desplaçament apareixen automàticament quan són necessàries (per exemple, quan

redimensionem el control tal que la informació que conté queda parcialment amagada). Si el valor de la propietat `AutoScroll` és `Fals`, les barres de desplaçament no apareixen automàticament. En aquest cas podem fer aparèixer les barres de desplaçament amb les propietats `HorzScrollBar` i `VertScrollBar`, que respectivament ens permeten amagar, visualitzar i manipular les barres de desplaçament horitzontal i vertical del control.

- **Action:** Especifica l'acció associada al control. L'acció determina alguns dels esdeveniments i propietats del control, com per exemple: `Caption`, `Hint`, `Visible`, `Enabled` i `OnClick` (veure components `Action` i `ActionList`).
- **TabOrder** i **TabStop:** La propietat `TabOrder` indica la posició del control en l'ordre de tabul·lació del seu contenidor, és a dir, és l'ordre en que el control serà visitat quan l'usuari premi la tecla de tabul·lació. El control amb el valor de `TabOrder` a 0 serà el control que tindrà el focus quan el formulari aparegui per primer cop. La propietat `TabStop` determina si l'usuari pot accedir al control mitjançant la tecla de tabul·lació.
- **Tag:** Ens permet guardar qualsevol valor enter que necessitem associar a l'objecte. Aquesta propietat no té cap significat ni es emprada per cap dels objectes que compten amb ella, però com veurem més endavant, en ocasions ens serà molt útil.
- **ComponentCount**, **ControlCount**, **Components** i **Controls:** Aquestes propietats només són accessibles en temps d'execució. `ComponentCount` i `ControlCount` indiquen, respectivament, quants components i controls hi han dins un contenidor, facilitant l'accés individual a cadascun d'ells mitjançant un índex numèric. D'aquesta manera es possible realitzar en temps d'execució operacions que afecten a múltiples elements de la interfície sense necessitat de fer referència a cadascun d'ells. `Component` i `Control` són els vectors que ens permeten accedir, respectivament, als controls i components mitjançant l'índex. La seva base (l'índex del primer element) és zero.
- **Owner:** És una referència al component propietari del component. Quan un component és propietari d'un altre, la memòria del component posseït s'allibera quan s'allibera la memòria de component propietari. Això significa, per exemple, que quan es destrueix un formulari també es destrueixen tots els components sobre el formulari.
- **Parent:** És una referència al contenidor del control. Per exemple, si una aplicació inclou tres `RadioButtons` dins una `GroupBox`, la `GroupBox` és accessible a través de la propietat `Parent` dels `RadioButtons`, i els `RadioButtons` són accessibles a través de la propietat `Controls` de la `GroupBox`.
- **ParentColor**, **ParentFont** i **ParentShowHint:** Si el valor d'alguna d'aquestes propietats és `True`, llavors en canviar la propietat `Color`, `Font` o `ShowHint` del contenidor del control, automàticament canvia la propietat corresponent del control.
- **Handle:** Proporciona accés al *handle* del control, necessari en moltes crides a les funcions de l'API de Windows. Si no existeix el *handle* del control, el crea. Al següent exemple podem veure una crida a la funció `ShowWindow` de l'API de Windows per visualitzar el control `Form2` com una icona, però sense activar-lo:

```
ShowWindow(Form2.Handle, SW_SHOWMINNOACTIVE);
```

### Esdeveniments

- **OnClick:** Es produeix en prémer sobre un control, generalment fent servir el ratolí, encara que també pot ser mitjançant tecles d'accés ràpid, la tecla `Enter` quan hi ha un botó seleccionat per defecte, la tecla `Espai` quan una `CheckBox` rep el focus, etc. Com a paràmetre aquest esdeveniment rep una referència al control sobre el qual s'ha produït.
- **OnDblClick:** Es produeix en efectuar un doble clic amb el ratolí sobre un control. Com a paràmetre aquest esdeveniment rep una referència al control sobre el qual s'ha produït.
- **OnMouseDown** i **OnMouseUp:** Es produeixen, respectivament, en prémer un botó del ratolí i en deixar anar aquest botó sobre un control. Com a paràmetres, aquests esdeveniments reben una referència al control sobre el qual s'ha produït, quin dels tres botons del ratolí l'ha originat, si estaven premudes les tecles `Control`, `Alt` i `Maj` i les coordenades *x* i *y* en píxels respecte la cantonada esquerra superior del control.

- **OnMouseMove**: Es produeix en desplaçar el ratolí sobre un control. Com a paràmetres, aquest esdeveniment rep una referència al control sobre el qual s'ha produït, si estaven premudes les tecles `Control`, `Alt` i `Maj` i les coordenades  $x$  i  $y$  en píxels respecte la cantonada esquerre superior del control.
- **OnMouseWheel**: Es produeix en moure el botó-roda del ratolí sobre un control. Com a paràmetres, aquest esdeveniment rep una referència al control que el gestiona, si estaven premudes les tecles `Control`, `Alt` i `Maj`, el nombre de cops que s'ha rotat el botó-roda, la posició del punter i un paràmetre booleà per referència per retornar si el control gestionarà l'esdeveniment o ho farà el seu contenidor.
- **OnMouseWheelDown** i **OnMouseWheelUp**: Es produeixen, respectivament, en moure cap avall o cap amunt el botó-roda del ratolí sobre un control. Com a paràmetres, aquests esdeveniments reben una referència al control que el gestiona, si estaven premudes les tecles `Control`, `Alt` i `Maj`, la posició del punter i un paràmetre booleà per referència per retornar si el control gestionarà l'esdeveniment o ho farà el seu contenidor.
- **OnKeyPress**: Es produeix quan s'ha premut una tecla del teclat. Com a paràmetres aquest esdeveniment rep una referència al control sobre el qual s'ha produït i la tecla premuda (codi ASCII).
- **OnKeyDown** i **OnKeyUp**: Es produeixen, respectivament, quan es prem una tecla del teclat i quan es deixa anar. Com a paràmetres aquests esdeveniments reben una referència al control sobre el qual s'ha produït, la tecla premuda (codi alfanumèric) i si en combinació estaven premudes les tecles `Control`, `Alt` i `Maj`.
- **OnEnter** i **OnExit**: Es produeixen, respectivament, quan un control rep el focus i quan el perd. Com a paràmetre aquests esdeveniments reben una referència al control sobre el qual s'ha produït.
- **OnDragOver** i **OnDragDrop**: Es produeixen, respectivament, quan s'arrossega un control per sobre d'un altre i quan es deixa anar. Com a paràmetres aquests esdeveniments reben una referència al control sobre el qual s'ha produït, una altre referència al control arrossegat, les coordenades  $x$  i  $y$  del control arrossegat i, en al cas de `OnDragOver`, l'estat de l'objecte arrossegat (si està entrant, sortint o sobre el control) i un paràmetre booleà per referència per retornar si el control accepta el control arrossegat o no.
- **OnStartDrag** i **OnEndDrag**: Es produeixen, respectivament, quan es comença a arrossegar un control per sobre d'un altre i quan s'acaba d'arrossegar. Com a paràmetres aquests esdeveniments reben una referència al control sobre el qual s'ha produït, una altre referència al control arrossegat i, en al cas de `OnEndDrag`, les coordenades en píxels on s'ha deixat anar.
- **OnStartDock** i **OnEndDock**: Es produeixen, respectivament, quan s'arrossega un control per sobre d'un altre per ancorar-lo i quan es deixa anar. Com a paràmetres aquests esdeveniments reben una referència al control sobre el qual s'ha produït, una altre referència al control arrossegat i, en al cas de `OnEndDock`, les coordenades en píxels on s'ha deixat anar.
- **OnContextPopup**: Es produeix quan s'invoca el menú contextual d'un control. Com a paràmetres, aquest esdeveniment rep una referència al control sobre el qual s'ha produït, la posició del punter i un paràmetre booleà per referència per retornar si el control mostrarà el menú contextual després de gestionar l'esdeveniment o no.

## Mètodes

- **Create** i **Destroy**: S'utilitzen, respectivament, per crear i destruir un component. És millor no cridar al mètode `Destroy` directament, sinó cridar al mètode `Free`, que verifica que el component no sigui `nil` i llavors crida a `Destroy`. El mètode `Create` rep com a paràmetre una referència del component propietari del nou component.
- **Show** i **Hide**: S'utilitzen, respectivament, per fer visible i amagar un control.
- **BringToFront** i **SendToBack**: S'utilitzen, respectivament, per posar un control a sobre o a darrera d'altres controls que es sobreposen.
- **SetFocus**: S'utilitza per proporcionar el focus al control, és a dir, seleccionar-lo per tal que rebí els esdeveniments del teclat.

- **SetBounds:** S'utilitza per especificar alhora a un control les seves propietats `Left`, `Top`, `Width` i `Height`, que rep com a paràmetres.
- **ClientToScreen:** S'utilitza per convertir un punt, passat com a paràmetre, de l'àrea del control a coordenades de pantalla.
- **FindComponent:** S'utilitza per trobar una referència a un component fill del component, passat com a paràmetre el nom d'aquest component fill.
- **Broadcast i NotifyControls:** S'utilitzen per enviar un missatge a tots els controls continguts dins un contenidor.

## Application

### Definició

La classe `TApplication` forma la base d'una aplicació, proporcionant mètodes i propietats que encapsulen el comportament estàndard d'una aplicació al sistema operatiu Windows (creació, execució, manteniment i destrucció), simplificant així la interfície entre el programador i l'entorn de Windows. Exemples de comportaments encapsulats per aquest component són: processament de missatges de Windows, manegament d'excepcions, ajuda sensible al context, processament del teclat, manegament de la finestra principal, etc.

### Propietats

- **ExeName**: Conté el nom i camí del fitxer executable de l'aplicació.
- **HelpFile** i **CurrentHelpFile**: La propietat `HelpFile` especifica el fitxer d'ajuda de l'aplicació que serà emprat pel sistema d'ajuda de Windows, mentre que la propietat `CurrentHelpFile` especifica el fitxer d'ajuda que està emprant actualment l'aplicació, que pot ser diferent si el formulari actiu té un fitxer d'ajuda associat diferent al de l'aplicació.
- **Icon** i **Title**: Contenen, respectivament, la icona i el títol que apareixen a la barra de tasques de Windows quan es minimitza l'aplicació. Les dues propietats es poden establir tant en temps de disseny com en temps d'execució. Per defecte el títol és el nom del fitxer executable.
- **MainForm** i **ShowMainForm**: La propietat `MainForm` indica quin és el formulari principal de l'aplicació. Aquest és el primer formulari creat al programa principal de l'aplicació, i quan aquest es tanca l'aplicació termina. La propietat `ShowMainForm` determina si l'aplicació mostra o no a l'inici el seu formulari principal.
- **ShowHint**, **HintPause**, **HintHidePause** i **HintColor**: La propietat `ShowHint` determina per l'aplicació sencera si les petites caixes d'ajuda són actives o no. La propietat `HintPause` determina el temps que passa d'ençà que el punter de ratolí es posiciona sobre un control o opció d'un menú fins que apareix la seva caixa d'ajuda. La propietat `HintHidePause` determina el temps que tarda en aparèixer una nova caixa d'ajuda quan ja s'estava mostrant una altre. Per últim, la propietat `HintColor` especifica el color de les caixetes d'ajuda.

### Esdeveniments

- **OnActivate** i **OnDeactivate**: Es produeixen, respectivament, quan una aplicació és torna activa (quan s'inicia o quan l'usuari canvia d'una altre aplicació a aquesta) i quan es desactiva (quan l'usuari canvia d'aquesta aplicació a una altre) .
- **OnMinimize** i **OnRestore**: Es produeixen, respectivament, quan una aplicació es minimitza i quan deixa d'estar minimitzada. Cal no confondre minimitzar i restaurar l'aplicació amb minimitzar o restaurar un formulari.
- **OnIdle**: Es produeix quan una aplicació deixa de processar codi, per exemple per què està esperant que l'usuari introdueixi dades.
- **OnException**: Es produeix quan ocorre una excepció que no a estat gestionada pel codi de l'aplicació. Es pot fer servir, per exemple, per tancar l'aplicació d'una manera elegant.
- **OnMessage**: Es produeix quan l'aplicació rep un missatge del sistema. Aquest esdeveniment permet a l'aplicació respondre a altres missatges que els especificats als esdeveniments de l'aplicació. Quan l'aplicació no té un manipulador específic per a un missatge, aquest es passa a la finestra corresponent i el sistema manipula el missatge.
- **OnHelp**: Es produeix quan hi ha una sol·licitud d'ajuda a l'aplicació.
- **OnHint**: Es produeix quan el punter del ratolí es mou sobre un control que pot mostrar una caixa d'ajuda.

## Mètodes

- **CreateForm**: S'utilitza per crear un formulari dinàmicament (en temps d'execució).
- **MessageBox**: S'utilitza per visualitzar un quadre de diàleg genèric amb un missatge i un o varis botons. Retorna un valor segons quin sigui el botó premut per l'usuari.
- **ShowException**: S'utilitza per visualitzar un quadre de diàleg genèric amb un missatge per les excepcions que no són gestionades pel codi de l'aplicació. Es crida per defecte si no s'ha declarat codi a l'esdeveniment `OnException` de l'aplicació.
- **ProcessMessages**: S'utilitza per interrompre l'execució d'una aplicació per tal que el sistema pugui processar la seva cua de missatges. Aquest mètode és especialment útil quan a una rutina de càlcul intensiu d'una aplicació, aquesta deixa de respondre als missatges del sistema.
- **HelpCommand**: S'utilitza per accedir a les comandes de WinHelp. Per exemple:
 

```
Application.HelpFile := 'MYHELP.HLP';
Application.HelpCommand(HELP_FINDER, 0);
```
- **Minimize i Restore**: S'utilitzen, respectivament, per minimitzar l'aplicació a la barra de tasques i per restaurar l'aplicació a la seva mida normal.
- **Terminate**: S'utilitza per finalitzar l'execució de l'aplicació.

## Exemple

```
program Pizza;

uses
  Forms, Unit5 in 'Unit5.pas' {FormProgres},
  Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3}, Unit4 in 'Unit4.pas' {Form4};

{$R *.RES}

begin
  Application.Initialize;
  Application.HelpFile := 'pizza.hlp';

  // mostra un formulari amb una barra de progres mentres
  // inicialitza l'aplicació i crea els quatre formularis
  with TFormProgres.Create(nil) do
    try
      ProgressBar1.Max := 100;
      Show;
      Update;
      Application.CreateForm(TForm1, Form1);
      ProgressBar1.StepBy(25);
      Application.CreateForm(TForm2, Form2);
      ProgressBar1.StepBy(25);
      Application.CreateForm(TForm3, Form3);
      ProgressBar1.StepBy(25);
      Application.CreateForm(TForm4, Form4);
      ProgressBar1.StepBy(25);
    finally
      Free;
    end;

    Application.MessageBox('Prem el botó per començar', 'Pizza', 0);
    Application.Run;
end.
```

## Screen

### Definició

La classe `TScreen` representa l'estat de la pantalla en la que s'executa l'aplicació. S'utilitza en temps d'execució pel seguiment dels formularis i per obtenir informació específica del sistema, com la resolució de la pantalla i quines fonts estan disponibles.

### Propietats

- **Width** i **Height**: Indiquen, respectivament, l'amplada i l'alçada de la pantalla en píxels.
- **ActiveControl** i **ActiveForm**: Indiquen, respectivament, quin control i quin formulari són els actius. Com són propietats de només lectura, per canviar-les cal cridar, respectivament, els mètodes `SetFocusedControl` i `SetFocus` del formulari corresponent.
- **FormCount** i **Forms**: Aquestes propietats només són accessibles en temps d'execució. `FormCount` indica quants formularis d'una aplicació s'estan visualitzant actualment a pantalla. Aquest formularis poden ser accedits a través de la propietat `Forms`, facilitant l'accés individual a cadascun d'ells mitjançant un índex numèric que està entre 0 i `FormCount-1`. D'aquesta manera es possible iterar en temps d'execució sobre tots els formularis oberts d'una aplicació sense necessitat de fer referència a cadascun d'ells.
- **Fonts**: Retorna una llista amb els noms de les fonts tipogràfiques actualment instal·lades al sistema. D'aquesta manera una aplicació pot saber si utilitza alguna font inexistente, cas aquest en que Windows substituiria la font per alguna altre potser no apropiada.
- **Cursor** i **Cursors**: La propietat `Cursor` controla l'aspecte del cursor a una aplicació. Quan el seu valor és `crDefault`, l'aspecte del cursor és controlat per la propietat `Cursor` dels controls individuals, mentre que si té assignat qualsevol altre valor el seu aspecte serà el mateix per a totes les finestres de l'aplicació. La propietat `Cursors` permet, mitjançant un índex numèric, accedir a qualsevol cursor en ús per l'aplicació. Això ens pot permetre, per exemple, canviar la imatge associada a un cursor predefinit del sistema.

### Esdeveniments

- **OnActiveControlChange**: Es produeix quan el focus passa d'un control a un altre, ja siguin controls del mateix formulari o de formularis diferents.
- **OnActiveFormChange**: Es produeix quan el focus passa d'un formulari a un altre de la mateixa aplicació.

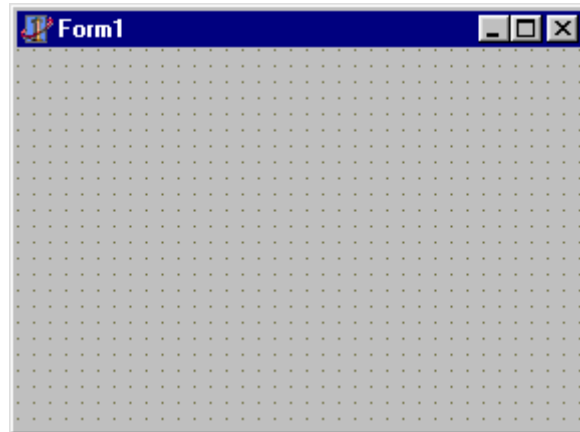
### Mètodes

#### Exemple

```
var
  SaveCursor:TCursor;
  i:integer;
begin
  SaveCursor := Screen.Cursor;
  Screen.Cursor := crHourglass;
  for i := 0 to Screen.FormCount-1 do
    begin
      Screen.Forms[I].Height := Screen.Height div 2;
      Screen.Forms[I].Width := Screen.Width div 2;
      Screen.Forms[I].Top := (Screen.Height div 4) + (15 * i);
      Screen.Forms[I].Left := (Screen.Width div 4) + (15 * i);
    end;
  Screen.Cursor := SaveCursor;
end;
```



## Form



### Definició

Els formularis són els components principals d'una aplicació estàndard. Un formulari pot representar la finestra principal d'una aplicació, quadres de diàleg o fills MDI. Un formulari pot contenir altres objectes, com per exemple botons, quadres de text i imatges.

### Propietats

- **Caption:** Títol que apareixerà en la part superior del formulari.
- **Left i Top:** Indiquen la posició absoluta del formulari a pantalla. Són les coordenades horitzontal i vertical, respectivament, de la cantonada superior esquerra del formulari. Aquestes dues propietats es mesuren en píxels.
- **Width, Height, ClientWidth i ClientHeight:** Les dues primeres propietats indiquen, respectivament, l'alçada i l'amplada del formulari mesurada en píxels. Les dues següents propietats indiquen l'espai interior del formulari dedicat a contenir altres components, és a dir, l'espai que realment queda usable al formulari un cop hem descomptat l'espai de la barra de títol, les barres de desplaçament, els marges, etc.
- **Position:** Representa la mida i posició amb que el formulari apareix per pantalla. Com a valors més importants que pot agafar trobem:
  - `poDesigned`: posició i mida que el formulari tenia en temps de disseny.
  - `poDefault`: posició i mida que determina el sistema operatiu.
  - `poDefaultPosOnly`: mida que el formulari tenia en temps de disseny i posició determinada pel sistema operatiu.
  - `poDefaultSizeOnly`: posició que el formulari tenia en temps de disseny i mida determinada pel sistema operatiu.
  - `poScreenCenter`: mida que el formulari tenia en temps de disseny i posició en el centre de pantalla.
- **Scaled i PixelsPerInch:** Si la propietat `Scaled` està activada al seu valor `True`, el formulari s'escalarà a una mida especificada per la propietat `PixelsPerInch`. Això ens permet conservar les proporcions del formulari encara que es produeixi un canvi en la resolució de la pantalla.
- **PrintScale:** Especifica l'escala a la que volem que es realitzi la impressió d'un formulari. Els valors que pot agafar són:
  - `poNone`: no s'escala i pot aparèixer al full amb diferents mides que a pantalla.
  - `poProportional`: s'escala tal que aparegui al full amb les mateixes mides que a pantalla.

- `poPrintToFit`: s'escala tal que ocupi tota la superfície imprimible del full.
- **Icon**: Especifica la icona que apareix quan es minimitza el formulari. Podem especificar el fitxer de la icona en temps de disseny, o utilitzar el mètode `LoadFromFile` de l'objecte icona per carregar-la en temps d'execució, com mostra el següent exemple:

```
// Aquest codi assigna una icona a un formulari quan aquest es crea:
procedure TForm1.FormCreate(Sender: TObject);
begin
  Icon.LoadFromFile('MDIChild1.ICO');
end;
```
- **BorderIcons** i **BorderStyle**: La propietat `BorderIcons` ens permet seleccionar quines icones (minimitzar, maximitzar, ajuda i menú principal) hi seran presents a la barra de títol del formulari. La propietat `BorderStyle` ens permet seleccionar l'aspecte dels marges del formulari (amb marges o sense, amb barra de títol normal o petita, amb botons per redimensionar o no). Encara que modifiquem els valors d'aquestes propietats, el formulari no apareix amb l'aparença final fins executar el programa.
- **WindowState**: Representa l'estat en el que el formulari apareix per pantalla. Els valors que pot agafar són:
  - `wsNormal`: estat normal (ni minimitzat ni maximitzat).
  - `wsMinimized`: estat minimitzat.
  - `wsMaximized`: estat maximitzat.
- **DefaultMonitor**: En configuracions multimonitor, especifica en quin monitor per defecte apareix el formulari quan es fa visible per primer cop. Els valors que pot agafar són:
  - `dmDesktop`: No s'intenta posicionar el formulari en cap monitor específic.
  - `dmPrimary`: en el primer monitor llistat a la propietat `Monitor`.
  - `dmMainForm`: en el mateix monitor que el formulari principal de l'aplicació.
  - `dmActiveForm`: en el mateix monitor que el formulari actiu actualment.
- **FormStyle**: Determina l'estil del formulari. Els valors que pot agafar són:
  - `fsNormal`: normal.
  - `fsMDIChild`: el formulari és una finestra MDI filla.
  - `fsMDIForm`: el formulari és una finestra MDI pare.
  - `fsStayOnTop`: romandrà sobre els altres formularis, que no el podran ocultar quan s'activin.
- **FormState**: Informa sobre l'estat transicional de la finestra. Pot agafar valors del següent conjunt: `fsCreating`, `fsVisible`, `fsShowing`, `fsModal`, `fsCreatedMDIChild` i `fsActivated`.
- **Menu**: Especifica quin és el menú principal del formulari.
- **Active**: Indica si el formulari rep el focus o no.
- **ActiveControl**: Especifica quin control del formulari és el inicialment actiu, és a dir, quin control rep actualment el focus.
- **MDIChildCount**, **MDIChildren** i **ActiveMDIChild**: Aquestes propietats només són accessibles en temps d'execució. `MDIChildCount` indica, quan el formulari és un contenidor MDI (és a dir, si el valor de la seva propietat `FormStyle` és `fsMDIForm`), quantes finestres filla hi han obertes dintre seu, facilitant l'accés individual a cadascuna d'elles mitjançant un índex numèric. D'aquesta manera es possible realitzar en temps d'execució operacions que afecten a múltiples finestres MDI filles sense necessitat de fer referència directa a cadascuna d'elles. `MDIChildren` és el vector que ens permet accedir a les finestres MDI filles mitjançant l'índex. La seva base és zero. Si volem saber quina és la finestra MDI filla activa en aquell moment, aquest ve indicat per la propietat `ActiveMDIChild`.

- **Floating:** Determina si el formulari és una finestra lliure o si està ancorat a una altre finestra.

### Esdeveniments

- **OnCreate, OnClose i OnDestroy:** Es produeixen, respectivament, quan es crea, es tanca i es destrueix un formulari. Tots els objectes creats dins l'esdeveniment `OnCreate` haurien de ser alliberats a l'esdeveniment `OnDestroy`. Quan es crea un formulari amb el valor `True` a la propietat `Visible`, es produeixen per ordre els següents esdeveniments: `OnCreate`, `OnShow`, `OnActivate` i `OnPaint`. Quan es sol·licita tancar un formulari, poden especificar-se una de les següents quatre accions a realitzar: res, minimitzar-lo, amagar-lo i alliberar-lo de memòria.
- **OnActivate i OnDeactivate:** Es produeixen, respectivament, quan el formulari rep el focus (per exemple, perquè l'usuari fa clic a sobre) i quan el perd (per exemple, perquè passa a un altre formulari de la mateixa aplicació).
- **OnShow i OnHide:** Es produeixen, respectivament, quan es mostra i s'amaga el formulari (quan la propietat `Visible` agafa els valors `True` o `False`, respectivament).
- **OnPaint:** Es produeix cada cop que es redibuixa el formulari, abans que els controls del formulari siguin dibuixats. Es pot fer servir, per exemple, per realitzar dibuixos especials sobre un formulari.
- **OnResize:** Es produeix quan s'intenta redimensionar el formulari. Es pot fer servir, per exemple, per afinar els canvis a l'hora d'ajustar la mida.
- **OnHelp:** Es produeix quan hi ha una sol·licitud d'ajuda al formulari.

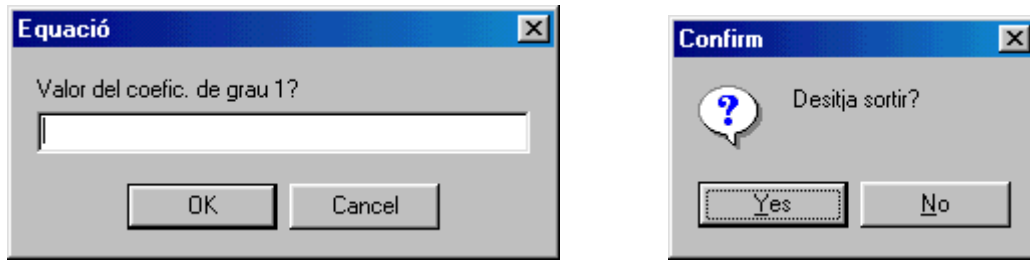
### Mètodes

- **ShowModal:** S'utilitza per mostrar un formulari de tal manera que no pugui continuar l'execució de l'aplicació fins que aquest es tanqui.
- **Print:** S'utilitza per imprimir el formulari per impressora.
- **GetFormImage:** S'utilitza per obtenir una imatge "bitmap" del formulari.
- **SetFocus:** S'utilitza per assignar el focus al formulari.
- **FocusControl:** S'utilitza per assignar el focus a un control del formulari.

### Exemple

```
// Crea en temps d'execució 20 controls Edit dins el formulari.
procedure TForm1.FormCreate(Sender: TObject);
var i: Integer;
begin
  for i := 1 to 20 do
    with TEdit.Create(Self) do
      begin
        Name := 'Edit' + IntToStr(i);
        Left := 10;
        Top := i * 20;
        Parent := Self;
      end;
  end;
end;
```

## InputDialog i MessageDlg



### Definició

InputDialog i MessageDlg no són realment components VCL de Delphi, sinó que són petits i senzills quadres de diàleg dedicats a entrada i sortida de dades, respectivament, originats per crides a l'API de Windows.

### InputDialog

InputDialog es tracta d'una funció que obre un quadre de diàleg que permet a l'usuari introduir una cadena de text. Els paràmetres de la crida a InputBox són, per ordre, el títol del quadre de diàleg, el text del quadre de diàleg i el valor inicial que per defecte apareix al quadre. Si l'usuari selecciona el botó Cancel, InputBox retorna la cadena per defecte, mentre que si l'usuari selecciona el botó OK, InputBox retorna la cadena escrita per l'usuari.

Un exemple de crida a InputBox seria:

```
a := StrToFloat( InputBox('Equació', 'Valor del coefic. de grau 1?', '' ) );  
b := StrToFloat( InputBox('Equació', 'Valor del terme independent?', '0' ) );
```

### MessageDlg

MessageDlg es tracta d'una funció que obre un quadre de diàleg per visualitzar una cadena de text i obtenir la resposta de l'usuari. Els paràmetres de la crida a MessageDlg són, per ordre, el missatge per l'usuari, el tipus de quadre o icona que apareixerà (mtWarning, mtError, mtInformation, mtConfirmation, mtCustom), els botons que apareixeran (mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll, mnNoToAll, mbYesToAll, mbHelp) i l'identificador del context de la ajuda a aparèixer en cas que l'usuari premi la tecla **F1** o el botó d'ajuda. El valor de retorn de MessageDlg serà el botó premut per l'usuari (mrNone, mrAbort, mrYes, mrOk, mrRetry, mrNo, mrCancel, mrIgnore, mrAll).

Un exemple de crida a MessageDlg seria:

```
if MessageDlg('Desitja sortir?', mtConfirmation, [mbYes, mbNo], 0) = mrYes  
then MessageDlg('Adéu...', mtInformation, [mbOk], 0);
```

## Frame



### **Definició**

El component `TFrame` és un contenidor d'altres components. Un “frame” o marc pot estar niuat dintre d'un formulari o un altre marc. Els marcs poden emmagatzemar-se a la paleta de components per reutilitzar-los com a plantilles de components.

### **Propietats**

Veure “propietats generals”.

### **Esdeveniments**

Veure “esdeveniments generals”.

### **Mètodes**

Veure “mètodes generals”.

### **Exemple**

## MainMenu



### Definició

El component `TMainMenu` permet afegir un menú principal a un formulari. Aquest menú principal consta de la barra de menú i les seves opcions de menú en cascada.

Per tal d'editar el menú principal, cal fer doble clic sobre el component `TMainMenu` un cop s'ha afegit aquest al formulari. Ens apareixerà un editor de menús multinivell que ens permetrà canviar les propietats de cada opció del menú, així com inserir noves opcions (tecla `Insert`), nous submenús (tecles `Ctrl` + `→`) i esborrar opcions existents (tecla `Supr`). El text que apareixerà escrit a cada opció del menú s'escriu a la corresponent propietat `Caption`. Si volem una línia de separació enlloc d'una opció de menú haurem d'escriure un guió '-' a l'esmentada propietat.

Els menús s'organitzen en forma d'arbre. A la propietat `Items` de cada opció del menú, podem afegir noves opcions del menú (submenús). I així successivament, formant el que s'anomena menús en cascada.

### Propietats

- **Images:** Especifica la llista d'imatges que apareixeran al costat de les opcions del menú. Cada opció del menú indica mitjançant la seva propietat `ImageIndex` quina imatge de la llista es veurà a l'esquerra del seu text.
- **Items:** Es tracta d'un objecte de la classe `TMenuItem` que permet accedir a informació sobre les opcions del menú. Podem accedir a les opcions del menú en temps de disseny fent clic sobre aquesta propietat i ens apareixerà l'editor de menús. També podem accedir a les opcions del menú en temps d'execució mitjançant un índex (per exemple: `Opcio1 := Menu1.Items[0]`). Les diferents propietats de les opcions d'un menú les veurem a continuació.
- **TMenuItem.Caption:** Especifica el text que apareixerà escrit a l'opció del menú. Si volem una línia de separació enlloc d'una opció de menú haurem d'escriure un guió '-' a aquesta propietat.
- **TMenuItem.SubMenuImages:** Especifica la llista d'imatges que apareixeran al costat de les opcions del submenú associat a l'opció del menú. Cada opció del submenú indica mitjançant la seva propietat `ImageIndex` quina imatge de la llista es veurà a l'esquerra del seu text.
- **TMenuItem.Bitmap** i **TMenuItem.ImageIndex:** Especificuen la imatge associada a l'opció del menú. En el cas de `TMenuItem.ImageIndex` es tracta del número d'imatge de la propietat `Images` del menú principal o de la propietat `SubMenuImages` de la opció de menú pare.
- **TMenuItem.Break:** S'utilitza per trencar un menú llarg en varies columnes, on la següent columna comença a partir de la opció del menú.
- **TMenuItem.Checked:** Indica si apareix una marca de selecció al costat de l'opció del menú.
- **TMenuItem.Enabled:** Indica si l'opció del menú està habilitada o inhabilitada. En aquest últim cas l'opció al menú apareix més clara i l'usuari no pot seleccionar-la.
- **TMenuItem.Visible:** Indica si l'opció del menú apareix o no.
- **TMenuItem.Default:** Especifica que l'opció del menú es l'opció per defecte a un submenú, és a dir, que serà executada quan l'usuari faci doble clic sobre l'opció que obre el submenú que conté l'opció per defecte. Un submenú només pot tenir una opció per defecte, que apareix en negreta.
- **TMenuItem.Action:** Indica l'acció (objecte de la classe `TAction`) associada a l'opció del menú.
- **TMenuItem.GroupIndex:** S'utilitza per fusionar múltiples menús que són a diferents formularis, especialment a aplicacions MDI. Quan es fusionen menús de formularis diferents, les opcions dels menús apareixen ordenades segons el valor numèric de la seva propietat `GroupIndex`.

- **TMenuItem.RadioItem**: Especifica si l'opció del menú exclou a d'altres opcions del menú al seu grup (opcions amb el mateix valor a la propietat `GroupIndex`), comportant-se com si fos un `RadioItem`.
- **TMenuItem.Count**: Indica el nombre d'opcions del submenú associat a l'opció del menú.
- **TMenuItem.Items**: Llista les d'opcions del submenú associat a l'opció del menú, com si d'un vector es tractés. El valor de l'índex és la posició de cada opció del submenú, on la primera posició té índex 0. Per exemple, si dintre del menú Fitxer apareixen les opcions Nou, Obrir, Guardar i Sortir, llavors `Fitxer.Items[2]` fa referència a l'opció Guardar.
- **TMenuItem.MenuIndex**: Indica la posició de l'opció del menú dins el menú o submenú, on la primera posició té valor 0.

### Esdeveniments

- **TMenuItem.OnClick**: Es produeix quan l'usuari prem sobre l'opció del menú.

### Mètodes

- **TMenuItem.Add** i **TMenuItem.Insert**: S'utilitzen per afegir en temps d'execució una o més opcions de menú noves a l'opció del menú. En el segon mètode es pot especificar la posició dins el submenú on s'inseriran.
- **TMenuItem.Clear**: S'utilitza per esborrar i alliberar de memòria les opcions de menú llistades a la propietat `Items`.
- **TMenuItem.Click**: S'utilitza per simular un clic de ratolí sobre l'opció del menú, generant un esdeveniment `OnClick`.
- **TMenuItem.Delete** i **TMenuItem.Remove**: S'utilitzen per esborrar en temps d'execució opcions del menú. Al primer mètode cal especificar l'índex de l'opció del menú, mentre que al segon cal especificar una referència a l'opció del menú. Si el que volem és únicament amagar temporalment la opció del menú, millor utilitzar la seva propietat `Visible`.

### Exemple

```
(* En temps d'execució afegeix a l'opció Window del menú principal
una subopció per a cada formulari de l'aplicació *)
var
  NouItem: TMenuItem;
  i : integer;
begin
  (* Crea i afegeix el separador *)
  NouItem := TMenuItem.Create(Self);
  NouItem.Caption := '-';
  Windows.Add(NouItem);

  (* Crea i afegeix una opció per cada formulari *)
  for i := 0 to Screen.FormCount-1 do
  begin
    NouItem := TMenuItem.Create(Self);
    NouItem.Caption := Screen.Forms[i].Name;
    NouItem.Checked := Screen.Forms[i].Active;
    Windows.Add(NouItem);
  end;
end;
```

## PopupMenu



### Definició

El component TPopupMenu permet crear menús sensitius al context, que apareixen quan l'usuari prem el botó dret del ratolí sobre un control. Per associar un menú contextual a un control cal editar la propietat PopupMenu del control.

Per tal d'editar el menú principal, cal fer doble clic sobre el component TPopupMenu un cop s'ha afegit aquest al formulari. Ens apareixerà un editor de menús multinivell que ens permetrà canviar les propietats de cada opció del menú, així com inserir noves opcions (tecla **Insert**), nous submenús (tecles **Ctrl** + **→**) i esborrar opcions existents (tecla **Supr**). El text que apareixerà escrit a cada opció del menú s'escriu a la corresponent propietat *Caption*. Si volem una línia de separació enlloc d'una opció de menú haurem d'escriure un guió '-' a l'esmentada propietat.

Els menús s'organitzen en forma d'arbre. A la propietat *Items* de cada opció del menú, podem afegir noves opcions del menú (submenús). I així successivament, formant el que s'anomena menús en cascada.

### Propietats

- **Alignment**: Especifica si el menú contextual apareix a la dreta, a l'esquerra o al centre del ratolí quan aquest menú és cridat per l'usuari.
- **AutoPopup**: Especifica si ha d'aparèixer o no el menú contextual quan aquest menú és cridat per l'usuari.
- **MenuAnimation**: Especifica, a Windows 98, Windows NT 5.0 i posteriors, com a d'aparèixer el menú a pantalla.

Veure "propietats de MainMenu".

### Esdeveniments

- **OnPopup**: Es produeix just abans d'aparèixer el menú contextual.

Veure "esdeveniments de MainMenu".

### Mètodes

- **Popup**: Visualitza el menú contextual a la pantalla a les coordenades especificades.

Veure "mètodes de MainMenu".

### Exemple

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  PopupMenu1.AutoPopup := False;
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
  PopupMenu1.Popup(X, Y);
end;

procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
  Pastel.Enabled := Clipboard.HasFormat(CF_TEXT);
end;
```



## Label

### A

#### Definició

El component TLabel o “etiqueta” permet mostrar text sobre un formulari. Aquest text no pot ser editat directament per l’usuari del programa.

#### Propietats

- **Caption:** És el text escrit dins l’etiqueta. Veure “propietat general Caption”.
- **Font:** Són les característiques de la font del text. Veure “propietat general Font”.
- **Alignment:** Especifica si el text escrit dins l’espai de l’etiqueta queda justificat a la dreta, a l’esquerra o bé centrat horitzontalment.
- **Layout:** Especifica si el text escrit dins l’espai de l’etiqueta queda justificat a dalt, a baix o bé centrat verticalment.
- **Transparent:** Indica si l’etiqueta tapa o no un altre control que estigui a sota.
- **AutoSize:** Indica si l’etiqueta ajusta la seva mida automàticament per acomodar-se a l’alçada i amplada del text escrit al seu interior.
- **WordWrap:** Indica si l’etiqueta trenca el text en vàries línies, quan aquest text és massa llarg per la seva amplada.
- **FocusControl** i **ShowAccelChar:** En el cas que el text escrit a l’etiqueta tingui una tecla d’accés ràpid (precedència de &), la propietat FocusControl especifica quin control rebrà el focus quan l’usuari premi dita tecla. La propietat ShowAccelChar indica si la tecla d’accés ràpid apareix subratllada o no.

#### Esdeveniments

Veure “esdeveniments generals”.

#### Mètodes

Veure “mètodes generals”.

#### Exemple

```
begin
  (* Prova 1 *)
  Label1.AutoSize := False;
  Label1.WordWrap := False;
  Label1.Caption := 'Segurament aquesta frase és massa llarga per al control';

  (* Prova 2 *)
  Label2.AutoSize := True;
  Label2.WordWrap := False;
  Label2.Caption := 'Segurament aquesta frase és massa llarga per al control';

  (* Prova 3 *)
  Label3.AutoSize := True;
  Label3.WordWrap := True;
  Label3.Caption := 'Segurament aquesta frase és massa llarga per al control';
end;
```

## Edit



### Definició

El component TEdit o “línia d’edició” permet a l’usuari introduir una línia de text. També pot ser emprat en algunes ocasions per mostrar text a l’usuari.

### Propietats

- **Text:** És el text escrit dins el control. Es tracta d’una cadena de caràcters, així que per convertir aquesta cadena a d’altres formats haurem de fer servir les funcions `StrToInt`, `StrToFloat`, `TimeToStr`, etc.
- **MaxLength:** Especifica el nombre màxim de caràcters que l’usuari podrà escriure dins el control. El valor zero indica que no hi ha límit.
- **Modified:** Indica si l’usuari ha canviat el text del control.
- **PasswordChar:** Especifica quin caràcter es visualitzarà en lloc dels caràcters escrits per l’usuari. Si el valor és nul (caràcter ANSI zero), el text es visualitzarà amb normalitat.
- **ReadOnly:** Indica si l’usuari pot canviar el text escrit al control.
- **BorderStyle:** Indica si el control queda delimitat per un marge d’una línia o no.
- **CharCase:** Especifica si el text escrit dins el control apareixerà en minúscules, majúscules o normal.
- **Font:** Són les característiques de la font del text. Veure “propietat general Font”.
- **AutoSelect** i **HideSelection:** La propietat `AutoSelect` indica si el text escrit queda seleccionat o no quan el control rep el focus, i la propietat `HideSelection` indica si l’indicació visual de text seleccionat continua o no quan el control perd el focus.
- **SelLength**, **SelStart** i **SelText:** Especifiquen, respectivament, la longitud del text seleccionat, la posició del primer caràcter seleccionat i el text seleccionat.
- **AutoSize:** Indica si el control es redimensiona automàticament per adaptar-se a l’alçada del text.
- **CanUndo:** Indica si l’usuari ha fet canvis que es poden desfer cridant al mètode `Undo`. Es pot utilitzar, per exemple, per activar o desactivar les opcions de menú que corresponen a desfer.
- **OEMConvert:** Indica si els caràcters teclejats són convertits d’ANSI a OEM i llavors de nou a ANSI, per tal de comprovar que poden ser convertits correctament al joc de caràcters OEM.

### Esdeveniments

- **OnChange:** Es produeix quan el text dins el control canvia.
- **OnKeyPress:** Es produeix quan s’ha premut una tecla del teclat, rebent com a paràmetre el codi ASCII de la tecla premuda. L’esdeveniment `OnKeyPress` és especialment important en aquest tipus de control, ja que s’utilitza per filtrar les tecles premudes per l’usuari abans que aquestes es visualitzin. Per exemple:

```
// Controla cada pulsació de tecla en Edit1, filtrant els caràcters '-'
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
    if Key = '-' then Key := #0;
end;
```

### Mètodes

- **SelectAll**: S'utilitza per seleccionar tot el text del control. Per seleccionar només una part cal emprar les propietats `SelStart` i `SelLength`.
- **Clear**: S'utilitza per esborrar tot el text del control.
- **ClearSelection**: S'utilitza per esborrar el text seleccionat del control.
- **CopyToClipboard**: S'utilitza per copiar el text seleccionat al portafolis.
- **CutToClipboard**: S'utilitza per copiar el text seleccionat al portafolis, esborrant la selecció.
- **PasteFromClipboard**: S'utilitza per enganxar el text del portafolis al control, reemplaçant el text seleccionat.
- **GetSelTextBuf**: S'utilitza per copiar el text seleccionat a memòria.
- **SetSelTextBuf**: S'utilitza per reemplaçar el text seleccionat amb un text a memòria.
- **Undo**: S'utilitza per desfer tots els canvis que s'han realitzat sobre el text.
- **ClearUndo**: S'utilitza per esborrar la llista de canvis, per tal que aquests canvis no es puguin desfer.

### Exemple

```
// En el text que introdueix l'usuari, filtra les xifres,
// la coma decimal, el signe de restar i la tecla d'esborrar.
procedure TFormEqu.EditCoefKeyPress(Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9', ',', '-', #8]) then Key := #0;
end;

// Calcula la solució de l'equació de primer grau
procedure TFormEqu.EditCoefChange(Sender: TObject);
  var a, b, sol : single;
begin
  // Deixem a 'a' el coeficient de grau 1
  if EditCoef1.text = '' then
    a := 0
  else
    a := StrToFloat(EditCoef1.text);

  // Deixem a 'b' el coeficient de grau 0
  if EditCoef0.text = '' then
    b := 0
  else
    b := StrToFloat(EditCoef1.text);

  if a <> 0 then
    LabelSolucio.caption := FormatFloat('0.####', -b/a)
  else
    if b <> 0 then
      LabelSolucio.caption := 'No existeix solució'
    else
      LabelSolucio.caption := 'Existeixen infinites solucions';
end;
```

## Memo



### Definició

El component TMemo o “quadre d’edició” permet a l’usuari introduir diverses línies de text. També pot ser emprat en algunes ocasions per mostrar text a l’usuari.

### Propietats

- **Lines:** Conté les línies de text dins el control. Per treballar amb tot el text alhora s’utilitza la propietat `Text`, mentre que per treballar amb les línies de text individualment s’utilitza aquesta propietat. Veure “component `Strings`”.
- **Alignment:** Especifica si el text dins el control queda alineat a la dreta, a l’esquerra o al centre.
- **ScrollBars:** Especifica si al control apareixen les barres de desplaçament vertical i/o horitzontal.
- **WantReturns** i **WantTabs:** Indiquen, respectivament, si l’usuari pot introduir caràcters de retorn i de tabulació dins el control. En cas contrari, prémer aquestes tecles provocarà, respectivament, l’activació d’un control o la selecció d’un altre control .
- **WordWrap:** Indica si les línies de text es trenquen (sense afegir caràcters de retorn) quan aquestes no hi caben dins el control.

Veure “propietats de `Edit`”.

### Esdeveniments

Veure “esdeveniments de `Edit`”.

### Mètodes

Veure “mètodes de `Edit`”.

### Exemple

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Memo1.Lines.LoadFromFile('C:\AUTOEXEC.BAT');
  Label1.Caption := 'La tercera línia de AUTOEXEC.BAT és: ' + Memo1.Lines[2];
end;
```

## Button, BitBtn i SpeedButton



### Definició

El component TButton és l'arxiconegut control botó de qualsevol interfície gràfica.

El component TBitBtn no és més que un botó que conté una imatge dins el seu espai, i les seves propietats són les mateixes que les del component TButton.

El component TSpeedButton és un botó amb imatge però sense text al seu interior. Normalment s'agrupa amb d'altres del seu tipus en pannels, formant paletes o barres d'eines especialitzades.

### Propietats

- **Caption:** És el text que apareix escrit al botó.
- **Font:** Són les característiques de la font del text. Veure "propietat general Font".
- **Cancel i Default:** Indiquen si l'esdeveniment OnClick del botó es produeix quan es premen les tecles d'escape **[Esc]** i retorn **[↵]**, respectivament.
- **ModalResult:** Especifica com el botó tanca el formulari (normalment un quadre de diàleg) i amb quin valor de retorn (mrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo o mrAll). Quan es prem el botó, el valor de la propietat ModalResult del formulari esdevé el mateix que la del botó.
- **Glyph i NumGlyph:** En un botó amb dibuix la propietat Glyph especifica el dibuix que apareix sobre el botó. El gràfic que creem per a un botó pot arribar a contenir fins a quatre imatges de les mateixes dimensions, disposades horitzontalment una al costat de l'altre. La primera d'elles serà emprada quan el botó estigui en estat normal, la segona quan estigui desactivat, la tercera quan sigui premut i la quarta quan romangui en aquest últim estat. Amb la finalitat de que el control conegui quantes imatges haguem inclòs en el gràfic, assignarem aquest nombre a la propietat NumGlyphs.
- **Kind:** En un botó amb dibuix, aquesta propietat especifica el dibuix per defecte per algun dels botons més comuns (bkCustom, bkOK, bkCancel, bkHelp, bkYes, bkNo, bkClose, bkAbort, bkRetry, bkIgnore, bkAll).
- **Flat:** En un botó amb dibuix, aquesta propietat indica si el botó té un aspecte tridimensional o no.

### Esdeveniments

- **OnClick:** Es produeix en prémer sobre un botó, generalment fent servir el ratolí, encara que també pot ser mitjançant tecles d'accés ràpid, la tecla **[Enter]** quan hi ha un botó seleccionat per defecte, etc.

### Mètodes

Veure "mètodes generals".

### Exemple

```
procedure TForm1.Button1MouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  Button1.top := Button1.top + 1;
  Button1.left := Button1.left + 1;
end;
```

## CheckBox



### **Definició**

El component TCheckBox o “caixa de selecció” presenta una opció a l’usuari, que aquest pot activar o desactivar a partir d’un petit quadre.

### **Propietats**

- **Caption:** És el text associat a la caixa de selecció.
- **Alignment:** Indica si el text apareix a la dreta o esquerra de la caixa de selecció.
- **AllowGrayed:** Indica si tenim un tercer estat o no a la caixa de selecció. Aquest tercer estat és opció seleccionada però gris (no seleccionat → [seleccionat i gris] → seleccionat → no seleccionat).
- **Checked:** Indica si l’opció està seleccionada o no seleccionada.
- **State:** Indica si l’opció està seleccionada, seleccionada i gris, o no seleccionada.

### **Esdeveniments**

Veure “esdeveniments generals”.

### **Mètodes**

Veure “mètodes generals”.

### **Exemple**

```
procedure TForm1.CheckBoxAnxovesClick(Sender: TObject);  
begin  
    if CheckBoxAnxoves.checked then  
        PreuPizza := PreuPizza + 150;  
end;
```

## RadioButton



### Definició

El component `TRadioButton` o “caixa de selecció exclusiva” presenta un conjunt d’opcions mútuament exclusives a l’usuari, que aquest pot activar o desactivar a partir d’un petit cercle. Només una opció pot estar activa alhora: quan l’usuari selecciona una opció, la opció seleccionada anteriorment esdevé inactiva.

Les caixes de selecció exclusiva s’agrupen mitjançant contenidors. El contenidor més freqüent és el component `TGroupBox` o “caixa d’agrupament”. Primer inserim la caixa d’agrupament sobre el formulari i a continuació afegim dintre les caixes de selecció exclusiva que ens calguin. Dos caixes de selecció exclusiva només poden estar seleccionades al mateix temps si estan contingudes dins contenidors diferents, com dues caixes d’agrupament o dos panells diferents.

### Propietats

- **Caption:** És el text associat a la caixa de selecció exclusiva.
- **Alignment:** Indica si el text apareix a la dreta o esquerra de la caixa de selecció exclusiva.
- **Checked:** Indica si l’opció està seleccionada o no seleccionada.

### Esdeveniments

Veure “esdeveniments generals”.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
procedure TForm1.RadioButtonPetitaClick(Sender: TObject);
begin
    PreuPizza := 500;
end;

procedure TForm1.RadioButtonMitjanaClick(Sender: TObject);
begin
    PreuPizza := 1000;
end;

procedure TForm1.RadioButtonGranClick(Sender: TObject);
begin
    PreuPizza := 1500;
end;
```

## RadioGroup



### Definició

El component TRadioGroup o “grup de caixes de selecció exclusiva” presenta un conjunt d’opcions mútuament exclusives a l’usuari, que aquest pot activar o desactivar a partir d’un petit cercle. Només una opció pot estar activa alhora: quan l’usuari selecciona una opció, la opció seleccionada anteriorment esdevé inactiva.

Aquest component equival a la combinació de varis components TRadioButton dins un component TGroupBox, però el seu ús és una mica més senzill i potent. Per afegir caixes de selecció exclusiva dins aquest component haurem d’editar la seva propietat Items a l’inspector d’objectes. Cada cadena de caràcters dins aquesta propietat serà el text d’una nova opció. El valor de la propietat ItemIndex determina quina opció està seleccionada.

### Propietats

- **Caption:** És el text que apareix com a títol del grup de caixes.
- **Columns:** Especifica el nombre de columnes en que repartirem les opcions.
- **Items:** Conté el text de les caixes de selecció exclusives. Veure “component Strings”.
- **ItemIndex:** Conté el número d’opció seleccionada. Cal tenir en compte que la primera és la 0. Si no volem cap opció seleccionada haurem de posar el seu valor a -1.

### Esdeveniments

Veure “esdeveniments generals”.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
procedure TForm1.RadioGroupMidaPizzaClick(Sender: TObject);
begin
  case RadioGroupMidaPizza.ItemIndex of
    0 : PreuPizza := 500; // Petita
    1 : PreuPizza := 1000; // Mitjana
    2 : PreuPizza := 1500; // Gran
  end;
end;
```



## ListBox



### Definició

El component `TListBox` o “llista d’opcions” visualitza una llista d’opcions de les quals l’usuari pot seleccionar una o varies.

### Propietats

- **Items:** Conté el text de les opcions. Veure “component Strings”.
- **ItemIndex:** Conté el número d’opció seleccionada. Cal tenir en compte que la primera és la 0. Si no hi ha cap opció seleccionada el seu valor és -1. Si hi han varies opcions seleccionades, conté el número de l’opció que rep el focus.
- **SelCount:** Conté quantes opcions hi han seleccionades.
- **Selected:** Indica quines opcions estan seleccionades i quines no, fent servir el número de l’opció com a paràmetre.
- **MultiSelect:** Indica si l’usuari pot seleccionar només una opció o bé varies (fent servir les tecles de majúscules o control en combinació amb el ratolí).
- **ExtendedSelect:** Indica si l’usuari pot seleccionar un conjunt seqüencial d’opcions fent servir la tecla de majúscules en combinació amb el ratolí.
- **BorderStyle:** Indica si el control queda delimitat per un marge d’una línia o no.
- **Columns:** Especifica el nombre de columnes visibles sense haver d’emprar la barra de desplaçament horitzontal.
- **ItemHeight:** Especifica l’alçada en píxels de les opcions de la llista.
- **IntegralHeight:** Indica si es visualitzen o no les opcions que queden parcialment tapades pels marges verticals del control.
- **Sorted:** Indica si les opcions de la llista estan ordenades alfabèticament.
- **TopIndex:** Especifica el número d’opció que serà visualitzada a dalt de la llista.

### Esdeveniments

Veure “esdeveniments generals”.

### Mètodes

- **Clear:** S’utilitza per esborrar totes les opcions de la llista d’opcions.

### Exemple

```
(* Copiem les opcions seleccionades d'una llista a l'altre *)
procedure TForm1.ButtonSeleccClick(Sender: TObject);
var i: Integer;
begin
  ListBoxSelecc.Clear;
  for i := 0 to (ListBoxFitxers.Items.Count - 1) do
    if ListBoxFitxers.Selected[i] then
      ListBoxSelecc.Items.Add(ListBoxFitxers.Items[i]);
end;
```

## ComboBox



### Definició

El component TComboBox o “llista d’opcions desplegable” visualitza una llista d’opcions de les quals l’usuari pot seleccionar o escriure una. Combina una llista d’opcions amb una línia d’edició.

### Propietats

- **Text:** És el text escrit dins la línia d’edició de la llista. Es tracta d’una cadena de caràcters.
- **Items:** Conté el text de les opcions. Veure “component Strings”.
- **ItemIndex:** Conté el número d’opció seleccionada. Cal tenir en compte que la primera és la 0. Si no hi ha cap opció seleccionada el seu valor és -1. El text de l’opció seleccionada és el que apareix a la línia d’edició.
- **SelLength, SelStart i SelText:** Especifiquen, respectivament, la longitud del text seleccionat, la posició del primer caràcter seleccionat i el text seleccionat dins la línia d’edició de la llista desplegable.
- **CharCase:** Especifica si el text escrit dins el control apareixerà en minúscules, majúscules o normal.
- **DropDownCount:** Especifica el nombre màxim d’opcions visibles quan es desplega la llista.
- **DroppedDown:** Indica si la llista està desplegada.
- **ItemHeight:** Especifica l’alçada en píxels de les opcions de la llista.
- **MaxLength:** Especifica el nombre màxim de caràcters que l’usuari podrà escriure dins la línia d’edició. El valor zero indica que no hi ha límit.
- **Sorted:** Indica si les opcions de la llista estan ordenades alfabèticament.
- **Style:** Especifica el tipus de llista desplegable (si l’usuari la pot editar, si es desplegable, si les opcions són de mida fixa, etc.).

### Esdeveniments

- **OnChange:** Es produeix quan l’usuari canvia el text de la línia d’edició.
- **OnDropDown:** Es produeix quan l’usuari desplega la llista.

### Mètodes

- **Clear:** S’utilitza per esborrar totes les opcions de la llista d’opcions.
- **SelectAll:** S’utilitza per seleccionar tot el text de la línia d’edició.

### Exemple

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Style := csDropDownList;
  ComboBox1.Items := Screen.Fonts;
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
  RichEdit1.Font.Name := ComboBox1.Items[ComboBox1.ItemIndex];
end;
```

## ScrollBar



### Definició

El component TScrollBar o “barra de desplaçament” s'utilitza per desplaçar el contingut d'una finestra, formulari o control.

Cal tenir en compte que hi han controls que ja integren barres de desplaçament, com per exemple, el component TMemo.

### Propietats

- **Kind:** Indica si l'orientació de la barra de desplaçament és vertical o horitzontal.
- **Min i Max:** Especifiquen, respectivament, els valors enters mínim i màxim representats per la barra de desplaçament.
- **Position:** Especifica la posició de la barra de desplaçament, tenint en compte que en la seva posició inferior o esquerra pren el valor de la propietat Min i en el seu valor superior o dreta pren el valor de la propietat Max.
- **SmallChange i LargeChange:** Especifiquen, respectivament, quant canvia el valor de la propietat Position quan l'usuari prem les fletxes de la barra de desplaçament o les fletxes del teclat amb el ratolí (canvi petit) o quan l'usuari prem la part buida de la barra de desplaçament o les tecles d'avenç i retrocés de pàgina (canvi gran).
- **PageSize:** Especifica la distància de la barra de desplaçament que correspon a una vista. En teoria, especifica quant canvia el valor de la propietat Position quan l'usuari prem les tecles d'avenç i retrocés de pàgina. A la pràctica, canvia la mida de la pestanya de la barra de desplaçament.

### Esdeveniments

- **OnChange:** Es produeix quan canvia el valor de la propietat Position.
- **OnScroll:** Es produeix quan l'usuari mou la barra de desplaçament amb el teclat o el ratolí. El tercer paràmetre conté el tipus d'acció que ha realitzat l'usuari: scLineUp, scLineDown, scPageUp, scPageDown, scPosition, scTrack, scTop, scBottom, scEndScroll. El segon paràmetre retorna el nou valor que s'assignarà a la propietat Position.

### Mètodes

- **SetParams:** S'utilitza per canviar alhora el valor de les propietats Position, Min i Max.

### Exemple

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Volum := (VOLUM_MAX - VOLUM_MIN + 1) div 10;
    ScrollBar1.Min := VOLUM_MIN;
    ScrollBar1.Max := VOLUM_MAX;
    ScrollBar1.SmallChange := 1;
    ScrollBar1.LargeChange := (VOLUM_MAX - VOLUM_MIN + 1) div 10;
    ScrollBar1.Position := Volum;
end;

procedure TForm1.ScrollBar1Change(Sender: TObject);
begin
    Volum := Scrollbar1.Position;
end;
```

## GroupBox



### Definició

El component TGroupBox o “caixa d’agrupament” s’utilitza per agrupar dins el formulari controls relacionats.

Quan es col·loca un control dins una caixa d’agrupament, aquesta esdevé el pare i contenidor del control.

### Propietats

- **Caption:** És el text que apareix com a títol de la caixa d’agrupament.

### Esdeveniments

Veure “esdeveniments generals”.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
procedure TForm1.GroupBoxMidaPizzaClick(Sender: TObject);
begin
  if RadioButtonPetita.Checked then
    PreuPizza := 500
  else
    if RadioButtonMitjana.Checked then
      PreuPizza := 1000
    else
      if RadioButtonGran.Checked then
        PreuPizza := 1500
end;
```

## Bevel



### **Definició**

El component `TBevel` o “bisell” és un element decoratiu que representa un marge, i s'utilitza per crear quadres i línies amb relleu. Aquest relleu pot ser d'enfonsament o d'elevació.

### **Propietats**

- **Shape**: Especifica la forma del bisell. Aquest pot ser una línia, un quadre, un marc o un espai.
- **Style**: Especifica si el bisell apareix enfonsat o elevat.

### **Esdeveniments**

cap esdeveniment.

### **Mètodes**

Veure “mètodes generals”.

### **Exemple**

## Panel



### Definició

El component TPanel o “pannell” és un contenidor amb marge bisellat. Tot pannel té dos bisells que li proporcionen una aparença tridimensional. El bisell exterior està dibuixat prop el marge del control, i el bisell interior està dibuixat dins el bisell exterior.

Com a contenidor que és permet manegar la posició dels controls incrustats dins seu.

### Propietats

- **Alignment**: Especifica si el text escrit dins el pannel queda justificat a la dreta, a l'esquerra o bé centrat horitzontalment.
- **BevelInner** i **BevelOuter**: Especifiquen, respectivament, el tipus de bisell interior i exterior. El tipus de bisell pot ser elevat, enfonsat, espai o cap.
- **BevelWidth** i **BorderWidth**: Especifiquen, respectivament, el gruix en píxels dels bisells i la distància en píxels entre els bisells interior i exterior.
- **BorderStyle**: Indica si apareix una línia addicional o no envoltant el pannel.

### Esdeveniments

cap esdeveniment.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
// Mou el panell a baix del formulari i li
// proporciona l'aparença d'una línia d'estat.
procedure TForm1.Button1Click(Sender: TObject);
begin
  with Panell do
  begin
    Align := alBottom;
    BevelInner := bvLowered;
    BevelOuter := bvRaised;
    BorderWidth := 1;
    BevelWidth := 1;
  end;
end;
```

## Image



### Definició

El component TImage o “imatge” permet visualitzar imatges dins un formulari.

### Propietats

- **Canvas:** Proporciona accés a l'àrea de dibuix de la imatge, juntament amb totes les eines i mètodes necessaris per dibuixar i pintar (veure component Canvas).
- **Picture:** Especifica la imatge que apareix dins el control (veure component Picture). Fer doble clic sobre aquesta propietat en temps de disseny fa aparèixer l'editor d'imatges.
- **AutoSize:** Indica si el control s'adapta a la mida de la imatge que conté.
- **Stretch:** Indica si la imatge continguda s'adapta a la mida del control.
- **Center:** Indica si la imatge queda centrada dins el control o no, quan la imatge carregada és més petita que el control i les propietats AutoSize i Stretch contenen el valor False.
- **Transparent:** Indica si el fons de la imatge tapa o no els controls sota la imatge.

### Esdeveniments

- **OnProgress:** Es produeix periòdicament mentre es realitzen operacions lentes sobre la imatge (com per exemple carregar una gran imatge comprimida), i s'utilitza per mostrar informació a l'usuari d'aquestes operacions. Un dels seus paràmetres indica si l'operació està començant, continuant o acabant, i un altre paràmetre aproxima el percentatge de la feina realitzada. Així, si s'utilitza aquest esdeveniment per mostrar una barra de progrés, aquesta barra es pot crear quan l'operació està començant, actualitzar-se proporcionalment al percentatge de feina realitzada mentre aquesta està continuant, i desaparèixer quan la feina està acabant.

Veure “esdeveniments generals”.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
procedure TForm1.ButtonConvertirIconaABitmapClick(Sender: TObject);
var Fitxer: string;
    Icon: TIcon;
begin
  OpenFileDialog1.DefaultExt := '.ICO';
  OpenFileDialog1.Filter := 'icones (*.ico)|*.ICO';
  OpenFileDialog1.Options := [ofOverwritePrompt, ofFileMustExist, ofHideReadOnly];
  if OpenFileDialog1.Execute then begin
    Icon := TIcon.Create;
    try
      Icon.LoadFromFile(OpenDialog1.FileName);
      Fitxer := ChangeFileExt(OpenDialog1.FileName, '.BMP');
      Image1.Width := Icon.Width;
      Image1.Height := Icon.Height;
      Image1.Canvas.Draw(0, 0, Icon);
      Image1.Picture.SaveToFile(Fitxer);
      ShowMessage(OpenDialog1.FileName + ' gravat a ' + Fitxer);
    finally
      Icon.Free;
    end;
  end;
end;
```

## Shape



### Definició

El component TShape o “forma” representa un objecte geomètric que es pot dibuixar sobre un formulari.

### Propietats

- **Shape:** Especifica la forma de l’objecte geomètric a dibuixar: quadrat, rectangle, el·lipse, cercle, quadrat amb cantonades arrodonides i rectangle amb cantonades arrodonides.
- **Pen:** Especifica els atributs de la línia que dibuixa l’objecte (contorn). Dintre seu conté altres propietats:
  - Width: amplada en píxels de la línia.
  - Color: color de la línia.
  - Style: estil de la línia (seguida, ratllada, puntejada, sense línia, etc.).
  - Mode: especifica com interacciona la línia amb el que ja hi ha dibuixat al llenç.
- **Brush:** Especifica els atributs del color i patró que omplen l’objecte (fons). Dintre seu conté altres propietats:
  - Color: color del fons.
  - Style: patró del fons (sòlid, ratllat, puntejat, transparent, etc.).
  - Bitmap: especifica una imatge externa que forma el patró del fons.

### Esdeveniments

Veure “esdeveniments generals”.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Timer1.Interval := 10;
  with Shape1 do
    begin
      Shape := stCircle;
      Pen.Color := clBlack;
      Brush.Color := clRed;
      Top := Self.Height div 2;
      Left := Self.Width div 2;
      Height := 10;
      Width := 10;
    end;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Shape1.SetBounds(Shape1.Left+1, Shape1.Top+1, Shape1.Width, Shape1.Height);
end;
```



## Canvas

### Definició

El component TCanvas o “llenç” és la superfície de dibuix dels controls que dibuixen una imatge d’ells mateixos. Els controls estàndard de Windows no empen llenç, ja que són dibuixats per Windows.

El component llenç permet especificar el tipus de línies, fons, objectes geomètrics i lletres a dibuixar, accedir als píxels del dibuix, etc.

### Propietats

- **Pixels:** Conté el color del píxel a les coordenades especificades. Si les coordenades cauen fora de la regió de dibuix del llenç, el valor retornat és -1.
- **PenPos:** Especifica les coordenades actuals del llapis del llenç.
- **ClipRect:** Especifica els límits de la regió de dibuix dins el llenç. Tot dibuix fora d’aquests límits no apareixerà. Per exemple: per dibuixar una porció de cercle tallada per un rectangle, primer dibuixem el rectangle, després situem la regió de dibuix a l’àrea dins el rectangle dibuixat, i a continuació dibuixem el cercle.
- **Pen:** Especifica els atributs de la línia que s’utilitza per dibuixar el contorn dels objectes. Dintre seu conté altres propietats:
  - **Width:** amplada en píxels de la línia.
  - **Color:** color de la línia.
  - **Style:** estil de la línia (seguida, ratllada, puntejada, sense línia, etc.).
  - **Mode:** especifica com interacciona la línia amb el que ja hi ha dibuixat al llenç.
- **Brush:** Especifica els atributs del color i patró que omplen el fons dels objectes dibuixats. Dintre seu conté altres propietats:
  - **Color:** color del fons.
  - **Style:** patró del fons (sòlid, ratllat, puntejat, transparent, etc.).
  - **Bitmap:** especifica una imatge externa que forma el patró del fons.
- **Font:** Especifica els atribut del text escrit sobre o dins el control. Veure “propietat general Font”.
- **CopyMode:** Especifica com interacciona una imatge que copiem al llenç mitjançant el mètode CopyRect amb el que ja hi ha dibuixat al llenç.

### Esdeveniments

- **OnChange:** Es produeix quan s’ha realitzat un canvi a la imatge.
- **OnChangeing:** Es produeix just abans de realitzar-se un canvi a la imatge.

### Mètodes

- **MoveTo:** S’utilitza per canviar la posició de dibuix a les coordenades especificades. És equivalent a modificar la propietat PenPos.
- **LineTo:** S’utilitza per dibuixar una línia des de les coordenades actuals a les coordenades especificades.
- **PolyLine:** S’utilitza per dibuixar una sèrie de línies des de les coordenades actuals seguint les coordenades especificades.

- **Polygon**: S'utilitza per dibuixar un polígon des de les coordenades actuals seguint les coordenades especificades. És similar al mètode `PolyLine`, però a més a més dibuixa la línia que va des de les coordenades finals a les coordenades inicials, tancant el polígon.
- **PolyBezier** i **PolyBezierTo**: S'utilitzen per dibuixar una sèrie de corbes de Bézier cúbiques des de les coordenades actuals fins a les coordenades finals fent servir les coordenades especificades com a punts de control. El mètode `PolyBezierTo`, a més a més actualitza el valor de la propietat `PenPos`.
- **Rectangle**, **RoundRect** i **FillRect**: S'utilitzen, respectivament, per dibuixar a les coordenades especificades un rectangle, un rectangle amb cantonades arrodonides i un rectangle ple.
- **Ellipse**: S'utilitza per dibuixar una el·lipse a les coordenades especificades.
- **Arc**, **Pie** i **Chord**: S'utilitzen, respectivament, per dibuixar a les coordenades especificades un arc d'el·lipse, un "tros de pastís" d'el·lipse i un arc "amb corda" d'el·lipse.
- **FloodFill**: S'utilitza per omplir una regió del llenç.
- **TextOut** i **TextRect**: S'utilitzen per dibuixar un text al llenç a les coordenades especificades. La diferència és que el segon mètode limita el text dins un rectangle especificat.
- **TextWidth** i **TextHeight**: Retornen, respectivament, l'amplada i alçada en píxels del text passat com a paràmetre.
- **Draw** i **StretchDraw**: S'utilitzen per dibuixar un gràfic passat com a paràmetre al llenç. El primer mètode el dibuixa a les coordenades especificades i el segon mètode a l'àrea especificada.
- **CopyRect**: S'utilitza per copiar part d'una imatge d'un altre llenç al llenç actual, dins l'àrea especificada.

### Exemple

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
                               Shift: TShiftState; X, Y: Integer);
var
  Bitmap: TBitmap;
  i: Integer;
begin
  Bitmap := TBitmap.Create;
  try
    Bitmap.LoadFromFile('C:\Windows\Aros.bmp');
    Canvas.Brush.Bitmap := Bitmap;
    Canvas.FillRect(Rect(X, Y, X+100, Y+100));
    Canvas.Pen.Color := clWhite;
    Canvas.Polyline([Point(X+50, Y+10), Point(X+30, Y+60), Point(X+80, Y+30),
                    Point(X+20, Y+30), Point(X+70, Y+60), Point(X+50, Y+10)]);
    for i := 10 to 90 do
      Canvas.Pixels[X+i, Y+5] := clRed;
  finally
    Canvas.Brush.Bitmap := nil;
    Bitmap.Free;
  end;
end;
```

## Picture

### Definició

El component `TPicture` o “dibuix” emmagatzema un gràfic, el tipus de qual (icona, mapa de punts, metagràfic o definit per l'usuari) queda especificat a la propietat `Graphic`.

### Propietats

- **Graphic:** Conté el gràfic del dibuix. Aquest pot ser una icona, un mapa de punts, un metagràfic o un gràfic de tipus definit per l'usuari.
- **Bitmap:** Conté el gràfic del dibuix com si fos un mapa de punts (tipus “bitmap”).
- **Icon:** Conté el gràfic del dibuix com si fos una icona.
- **Metafile:** Conté el gràfic del dibuix com si fos un metagràfic (tipus “enhanced Windows metafile graphic - EMF”).
- **Height** i **Width:** Especificuen, respectivament, l'alçada i l'amplada del gràfic contingut al dibuix.

### Esdeveniments

- **OnChange:** Es produeix quan s'ha realitzat un canvi a la imatge.
- **OnProgress:** Es produeix periòdicament mentre es realitzen operacions lentes sobre la imatge (com per exemple carregar una gran imatge comprimida), i s'utilitza per mostrar informació a l'usuari d'aquestes operacions. Un dels seus paràmetres indica si l'operació està començant, continuant o acabant, i un altre paràmetre aproxima el percentatge de la feina realitzada. Així, si s'utilitza aquest esdeveniment per mostrar una barra de progrés, aquesta barra es pot crear quan l'operació està començant, actualitzar-se proporcionalment al percentatge de feina realitzada mentre aquesta està continuant, i desaparèixer quan la feina està acabant.

### Mètodes

- **Assign:** S'utilitza per copiar el contingut d'un objecte dins d'un altre.
- **LoadFromFile** i **SaveToFile:** S'utilitzen, respectivament, per llegir i per guardar una imatge a disc.

### Exemple

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Shape1.Shape := stEllipse;
  Shape1.Brush.Color := clLime;
  Image1.Stretch := True;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  FormImage: TBitmap;
begin
  FormImage := GetFormImage;
  try
    Clipboard.Assign(FormImage);
    Image1.Picture.Assign(Clipboard);
    PaintBox1.Canvas.Draw(0,0, Image1.Picture.Graphic);
  finally
    FormImage.Free;
  end;
end;
```

## OLEContainer



### Definició

El component `TOleContainer` o “contenedor d’objectes OLE” permet a la nostra aplicació inserir o enllaçar objectes OLE, creats i editats per una altre aplicació. Exemples d’objectes OLE són: documents de Word, equacions de l’editor d’equacions, arxius de so, diapositives de PowerPoint, etc.

Quan l’usuari activa un objecte OLE a la nostra aplicació, es transfereix el control a l’aplicació que va crear aquest objecte.

Quan l’objecte OLE forma part de l’aplicació diem que està inserit, mentre que quan està contingut en un fitxer apart diem que està enllaçat. En aquest últim cas és possible editar i modificar l’objecte OLE.

### Propietats

- **OLEObject**: Representa l’objecte OLE allotjat pel contenidor, i permet accedir a ell.
- **ObjectVerbs**: Conté la llista d’accions que l’objecte OLE suporta.
- **Linked**: Indica si l’objecte OLE està enllaçat o inserit dins el contenidor.
- **SourceDoc**: És el nom del document que conté l’objecte OLE, quan aquest està enllaçat.
- **AutoActivate**: Especifica com s’activa l’objecte OLE: mitjançant un clic, doble clic o cridant al mètode `DoVerb`.
- **CanPaste**: Indica si el portafolis de Windows conté un objecte OLE que pot ser enganxat al contenidor OLE.
- **Iconic**: Indica si el contenidor OLE mostra una imatge de l’objecte o una icona de l’aplicació associada a l’objecte.
- **Modified**: Indica si l’objecte OLE ha estat modificat.
- **State**: Descriu l’estat de l’objecte OLE: no hi ha cap objecte, hi ha però no està actiu, està actiu, etc.

### Esdeveniments

- **OnActivate**: Es produeix just quan s’activa l’objecte OLE.
- **OnDeactivate**: Es produeix just quan es desactiva l’objecte OLE.
- **OnObjectMove**: Es produeix quan es mou o canvia la mida de l’objecte OLE.

### Mètodes

- **Copy** i **Paste**: El primer còpia l’objecte OLE al portafolis, i el segon enganxa l’objecte al contenidor des del portafolis.
- **CreateLinkToFile** i **CreateObjectFromFile**: Creen, respectivament, un objecte OLE enllaçat o inserit a l’aplicació.
- **DoVerb**: S’utilitza per sol·licitar a l’objecte OLE que realitzi alguna acció de les llistades per la propietat `ObjectVerbs`.
- **InsertObjectDialog**, **ObjectPropertiesDialog** i **PasteSpecialDialog**: Obren, respectivament, els quadres de diàleg estàndard de Windows per inserir un objecte OLE, per veure les seves propietats i per enganxar els continguts del portafolis al contenidor.

- **LoadFromFile** i **SaveToFile**: Carreguen i guarden, respectivament, un objecte OLE dins un fitxer.
- **UpdateObject** i **UpdateVerbs**: Actualitzen, respectivament, l'objecte OLE i la llista d'accions que aquest pot realitzar.

### Exemple

```

procedure TForm1.Archivo1Click(Sender: TObject);
begin
  Salvar1.Enabled := (ContenedorOle.State = osLoaded);
end;

procedure TForm1.Cargar1Click(Sender: TObject);
begin
  EstaModificado;
  if CDCargarDialog.Execute Then
    ContenedorOle.LoadFromFile(CDCargar.FileName);
  ContenedorOle.Refresh;
end;

procedure TForm1.Salvar1Click(Sender: TObject);
begin
  if CDSalvarDialog.Execute then
    ContenedorOle.SaveToFile(CDSalvar.FileName);
end;

procedure TForm1.Insertar1Click(Sender: TObject);
begin
  EstaModificado;
  ContenedorOLE.InsertObjectDialog;
end;

procedure TForm1.Editar1Click(Sender: TObject);
begin
  Copiar1.Enabled := (ContenedorOle.State = osLoaded);
  Cerrar1.Enabled := (ContenedorOle.State = osUIActive);
end;

procedure TForm1.Copiar1Click(Sender: TObject);
begin
  ContenedorOle.Copy;
end;

procedure TForm1.Cerrar1Click(Sender: TObject);
begin
  ContenedorOle.Close;
end;

procedure TForm1.Salir1Click(Sender: TObject);
begin
  EstaModificado;
  Close;
end;

Procedure TForm1.EstaModificado;
begin
  if (ContenedorOle.State = osLoaded) and (ContenedorOle.Modified) then
    if MessageDlg('El objeto ha sido modificado, ¿desea guardarlo?',
      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
      Salvar1Click(Self);
end;

```

## Timer



### Definició

El component `TTimer` o “cronòmetre” s'utilitza per llençar un esdeveniment, ja sigui un cop o repetidament, transcorregut un determinat interval de temps.

### Propietats

- **Interval**: Especifica l'interval de temps en milisegons que tarda en produir-se l'esdeveniment.
- **Enabled**: Activa o desactiva el cronòmetre.

### Esdeveniments

- **OnTimer**: Es produeix cíclicament cada cop que transcorre l'interval.

### Mètodes

Veure “mètodes generals”.

### Exemple

```
(* Mostra una pantalla de presentació quan comença l'aplicació *)
procedure TForm1.FormCreate(Sender: TObject);
begin
    SplashScreen.Show;
end;

(* SplashScreen conté un component Timer d'interval 3000 *)
procedure TForm2.Timer1Timer(Sender: TObject);
begin
    Close;
end;

var TempsRepr: int;

(* Si fan clic al formulari obrim el lector de CDs *)
procedure TForm1.FormClick(Sender: TObject);
begin
    MediaPlayer1.DeviceType := dtCDAudio;
    MediaPlayer1.Open;
    MediaPlayer1.Play;
    TempsRepr := 15;
    Timer1.Enabled := True;
end;

(* Reproduim un temps determinat i expulsem el CD *)
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    if TempsRepr = 0 then
        begin
            MediaPlayer1.Eject;
            MediaPlayer1.Close;
            Timer1.Enabled := False;
        end
    else
        Dec(TempsRepr);
end;
```

## Strings i StringList

### Definició

Els components `TStrings` i `TStringList` o “cadena de caràcters” ens permet operar amb llistes de cadenes de caràcters que en diverses ocasions apareixen com a propietats dels nostres components. Entre aquestes ocasions estan:

- Les opcions d’una llista d’opcions (`TListBox`) i d’una llista d’opcions desplegable (`TComboBox`).
- Les línies de text d’un quadre d’edició (`TMemo`).
- La llista de fonts suportades pel sistema (`TScreen`).
- Una línia o columna d’entrades a una quadrícula de cadenes de caràcters (`TStringGrid`).

Quan fem doble clic a l’inspector d’objectes sobre alguna d’aquestes propietats apareix l’editor de cadenes de caràcters, que ens permet afegir, modificar i esborrar línies.

### Propietats

- **Count:** Conté el nombre de línies de cadenes de caràcters a la llista.
- **Strings:** Permet accedir a les línies de cadenes de caràcters a partir d’un índex. La primera línia és la 0. Aquesta propietat pot no mostrar-se si no es vol (per exemple `MiStringList.Strings[0]` equival a `MiStringList[0]`).
- **Text:** Permet accedir a les línies de cadenes de caràcters com si fossin una única cadena de caràcters que les englobes a totes. Les línies individuals de cadenes de caràcters queden llavors delimitades per retorns de carro i salts de línia.
- **Name i Value:** Contenen, respectivament, la part del nom i la part del valor de la línia especificada per un índex, quan aquesta línia és de la forma `nom=valor` (per exemple: `DisplayGrid=1`).
- **Sorted:** Indica si les cadenes de caràcters s’ordenen automàticament o no.

### Esdeveniments

- **OnChange:** Es produeix just després de realitzar un canvi a la llista de cadenes de caràcters.
- **OnChangeing:** Es produeix just abans de realitzar un canvi a la llista de cadenes de caràcters.

### Mètodes

- **Add, Append i Insert:** S’utilitzen per afegir una cadena de caràcters a la llista. `Add` i `Append` l’afegeixen al final de la llista o a la posició corresponent si la llista està ordenada, mentre que `Insert` l’afegeix a la posició especificada per un índex. `Add` retorna la posició on ha inserit la cadena.
- **Clear:** S’utilitza per esborrar totes les cadenes de caràcters de la llista.
- **Delete:** S’utilitza per esborra la cadena de caràcters especificada per un índex.
- **Exchange:** S’utilitza per intercanviar a la llista dos cadenes de caràcters especificades per dos índex.
- **IndexOf:** S’utilitza per trobar la posició d’una cadena de caràcters dins la llista.
- **Move:** S’utilitza per canviar la posició d’una cadena de caràcters dins la llista.
- **SaveToFile i LoadFromFile:** S’utilitzen, respectivament, per guardar i recuperar una llista de cadenes de caràcters dins un fitxer.
- **Sort:** S’utilitza per ordenar les cadenes de caràcters dins la llista.

**Exemple**

```
(* Primer exemple *)
if OpenDialog1.Execute then
begin
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
  Memo1.Lines.Add('Hi han ' + IntToStr(Memo1.Lines.Count) + ' línies ');
  Memo1.Lines.Add('Alex és la línia ' +
    IntToStr(Memo1.Lines.IndexOf('Alex')));
end;

(* Segon exemple *)
MiStringList := TStringList.Create;
try
  Session.GetAliasNames(MiStringList);
  MiStringList.Sort;
  for i := 0 to MiStringList.Count - 1 do
    ListBox1.Items.Add(MiStringList[i]);
  finally
    MiStringList.Free;
  end;
```



## Altres Components VCL

### **Animate**



S'utilitza per visualitzar dibuixos i imatges animades (en format AVI) al formulari.

### **Chart**



S'utilitza per visualitzar sèries mitjançant tot tipus de gràfiques.

### **DateTimePicker i MonthCalendar**



S'utilitzen per introduir temps, dates i rangs de dates.

### **Dialogs**



Són els quadres de diàleg estàndard de Windows per obrir un fitxer, configurar la impressora, etc.

### **DrawGrid i StringGrid**



S'utilitzen per representar informació a una quadrícula, mitjançant files i columnes.

### **ListView i TreeView**



S'utilitzen per visualitzar llistes d'objectes en diferents formats.

### **MaskEdit**



S'utilitza per introduir dades en un format prèviament especificat (màscara).

### **MediaPlayer**



S'utilitza per reproduir i controlar dispositius multimèdia: CD-ROM, escàner, vídeo, etc.

### **PageControl i TabControl**



S'utilitzen per aconseguir quadres de diàleg de diverses pàgines i també per separar pàgines d'informació mitjançant pestanyes.

### **ProgressBar**



S'utilitza per donar informació visual a l'usuari del progrés d'una acció en al temps.

### **RichEdit**



S'utilitza per introduir text amb atributs de font i paràgraf (format RTF).

### **Splitter**



S'utilitza per separar dos controls i permet que l'usuari pugui canviar la seva mida durant l'execució del programa.

### TrackBar



Similar a les barres de desplaçament, s'utilitza per introduir i visualitzar un valor numèric dins un rang.

### UpDown



S'utilitza per incrementar o decrementar un valor numèric.

## Matrius de components

### Creació de nous components

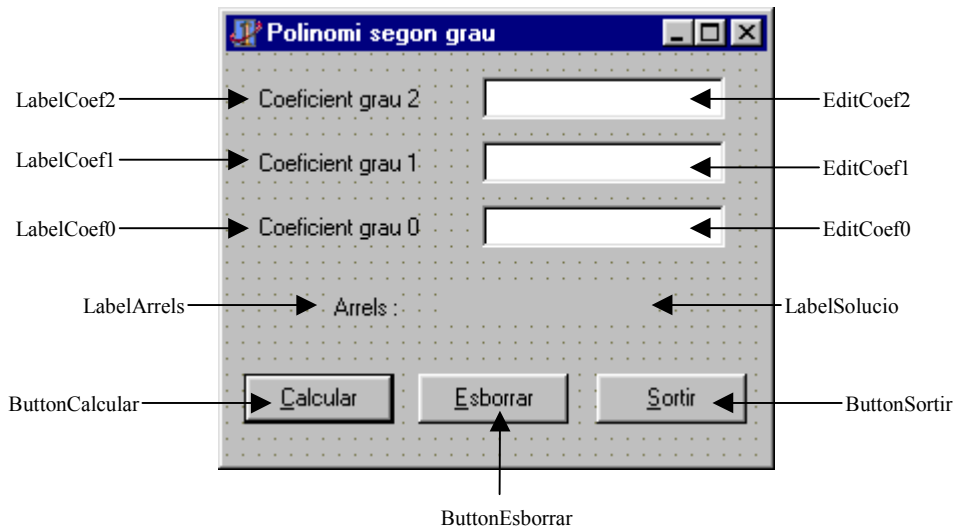
#### Exemple 5-1: arrels de polinomis de segon grau

Anem a desenvolupar una aplicació que permeti a l'usuari trobar les arrels de polinomis de segon grau. Aquestes es calculen mitjançant la fórmula:

$$P_2(x) = ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Hem de tenir en compte que les arrels poden ser tant reals com imaginàries. També hem de tenir en compte que si el primer coeficient  $a$  és zero, llavors el polinomi no és de segon grau i, per tant, no podem aplicar la fórmula.

Un possible quadre de diàleg seria el següent:



#### Propietats:

##### Form **FormArrels**:

|             |                            |
|-------------|----------------------------|
| Caption     | Polinomi segon grau        |
| BorderStyle | bsSingle                   |
| BorderIcons | [biSystemMenu, biMinimize] |
| Height      | 278                        |
| Width       | 231                        |

##### Label **LabelCoef2**, **LabelCoef1** i **LabelCoef0**:

|         |  |
|---------|--|
| Caption | Coeficient grau 2, Coef. grau 1 i Coef. grau 0, respectivament |
|---------|--|

|        |                             |
|--------|-----------------------------|
| Left   | 16                          |
| Top    | 16, 48 i 80, respectivament |
| Width  | 80                          |
| Height | 13                          |

Edit **EditCoef2**, **EditCoef1** i **EditCoef0**:

|        |                             |
|--------|-----------------------------|
| Text   |                             |
| Left   | 128                         |
| Top    | 12, 44 i 76, respectivament |
| Width  | 121                         |
| Height | 21                          |

Label **LabelArrels**:

|         |          |
|---------|----------|
| Caption | Arrels : |
| Left    | 50       |
| Top     | 120      |
| Width   | 32       |
| Height  | 13       |

Label **LabelSolucio**:

|            |        |
|------------|--------|
| Caption    |        |
| Left       | 101    |
| Top        | 120    |
| Width      | 118    |
| Height     | 13     |
| Font.Style | fsBold |

Button **ButtonCalcular**, **ButtonEsborrar** i **ButtonSortir**:

|         |  |
|---------|--|
| Caption | &Calcular, &Esborrar i &Sortir, respectivament |
| Left    | 9, 96 i 185, respectivament                    |
| Top     | 160  |
| Width   | 175  |
| Height  | 25   |
| Default | True només a ButtonCalcular                    |
| Cancel  | True només a ButtonEsborrar                    |

### Codi:

```
// Filtra el text que introdueix l'usuari. Només deixa passar les xifres,
// la coma decimal, el signe de restar i la tecla d'esborrar.
procedure TFormArrels.EditCoefKeyPress(Sender: TObject; var Key: Char);
begin
    if not (Key in ['0'..'9', ',', '-', #8]) then Key := #0;
end;

// Calcula les arrels del polinomi de segon grau.
procedure TFormArrels.ButtonCalcularClick(Sender: TObject);
var a, b, c, discr : single;
begin
    try
        // Agafem els coeficients dels quadres de text
        a := StrToFloat(EditCoef2.text);
        b := StrToFloat(EditCoef1.text);
        c := StrToFloat(EditCoef0.text);

        // Calculem el discriminant i les arrels
        discr := b*b - 4*a*c;
        if discr = 0 then // Una única arrel real
            LabelSolucio.caption := FormatFloat('0.####', -b/(2*a))
        else
            if discr > 0 then // Dues arrels reals
                LabelSolucio.caption := FormatFloat('0.####', (-b+Sqrt(discr))/(2*a))
                    + ' ' + FormatFloat('0.####', (-b-Sqrt(discr))/(2*a))
            else // Dues arrels complexes conjugades
                LabelSolucio.caption := FormatFloat('0.####', -b/(2*a)) + ' ± ' +
```

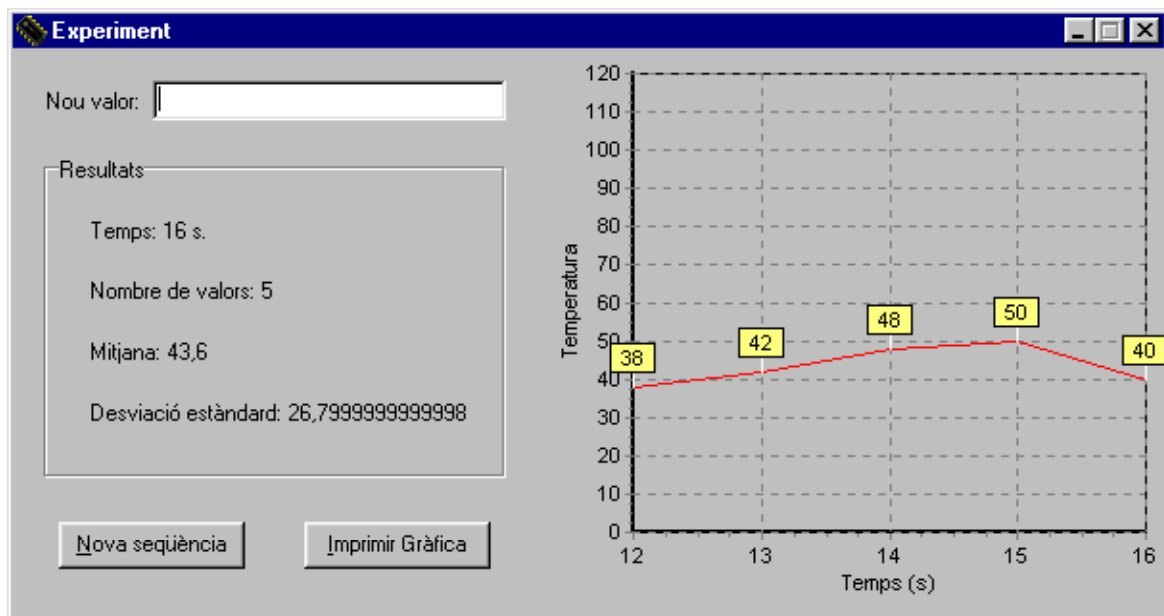
```
FormatFloat('0.####', Sqrt(-discr)/(2*a)) + ' i';  
except  
  on EInvalidOp do ShowMessage('El polinomi no és de segon grau!');  
  on EConvertError do ShowMessage('Coeficient invàlid!');  
end;  
end;  
  
// Esborra el text dels coeficients i les arrels del polinomi.  
procedure TFormArrels.ButtonEsborrarClick(Sender: TObject);  
begin  
  EditCoef2.text := '';  
  EditCoef1.text := '';  
  EditCoef0.text := '';  
  LabelSolucio.caption := '';  
end;  
  
// Tanca el formulari i finalitza el programa.  
procedure TFormArrels.ButtonSortirClick(Sender: TObject);  
begin  
  close;  
end;
```

## Exercici 5-1: càlcul de la mitjana i la desviació estàndard

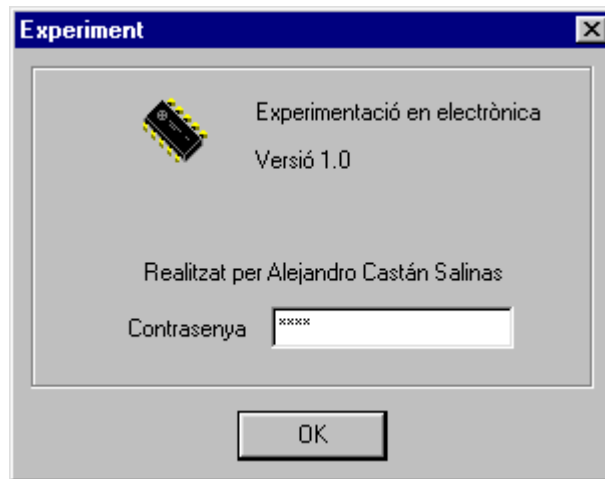
Emprant la classe TExperiment desenvolupada a l'exercici 4-1, dissenyeu una aplicació amb interfície gràfica que permeti a l'usuari introduir una seqüència de nombres. Mentre es realitza la introducció dels nombres, el programa hauria de dibuixar-los a una gràfica així com calcular la mitjana i la desviació estàndard d'aquests, segons les següents fórmules:

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N} \quad \text{i} \quad \sigma^2 = \frac{\sum_{i=1}^N x_i^2 - N \left( \frac{\sum_{i=1}^N x_i}{N} \right)^2}{N-1}$$

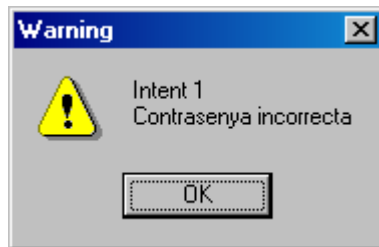
on  $N$  és el nombre de valors introduïts.




Opcional: Si us atreviu, feu que en iniciar l'aplicació aparegui un AboutBox que demani una paraula de pas.



Si no encertem la paraula no podrem accedir al formulari principal de l'aplicació. Tindrem però tres oportunitats per encertar. Per cada oportunitat fallada apareixerà un missatge d'error. Si fallem les tres sortirem del programa.



Pautes i pistes:

- Aproveu la classe TExperiment desenvolupada a l'exercici 4-1. Amplieu-la al vostre gust si fos necessari.
- Per a la gràfica feu servir el component TChart  que apareix a la paleta de components addicional. Per editar les seves propietats en temps de disseny feu doble clic sobre el component inserit al formulari. Apareixerà un quadre de diàleg que us permetrà retocar el tipus de gràfica, les etiquetes dels eixos, els colors, les sèries de dades, el nombre màxim de punts a visualitzar, etc. Per inserir i esborrar dades en temps d'execució empreu els mètodes de la sèrie AddXY (X, Y:Double; Etiq:String; Color:TColor) i Delete (Index:LongInt). Per imprimir la gràfica empreu el seu mètode Print ().
- Per tal que l'aparença del formulari en fer-lo més gran o més petit sigui impecable, jugueu amb les propietats Constraints del formulari i Anchors dels controls.
- En cas que us atreviu a implementar la finestra prèvia que demana contrasenya, pareu a pensar on és millor inserir el codi per controlar l'aparició d'aquesta finestra i de la finestra principal.

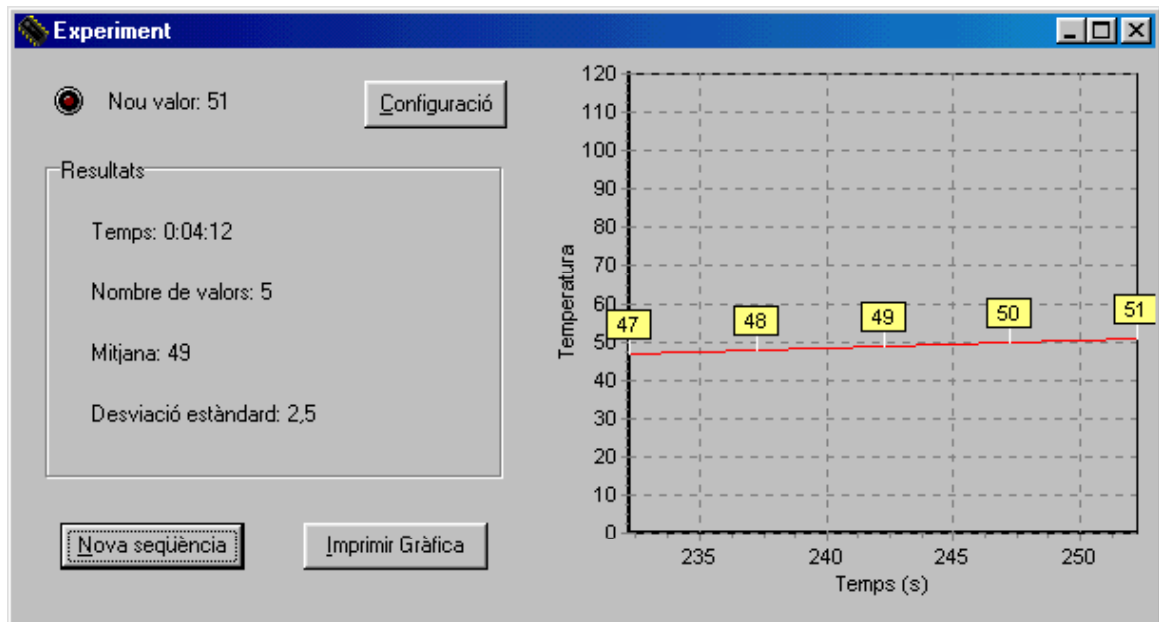
## Exercici 5-2: instal·lació i ús d'un nou component

Cerqueu a Internet un nou component per a Delphi que permeti adquirir i enviar dades pel port sèrie de l'ordinador. Seguint les instruccions que acompanyen aquest component, instal·leu-lo a la paleta de components. Consulteu al seu fitxer d'ajuda:

- Quines són les seves propietats més importants i en què consisteixen.

- Quins són els seus esdeveniments més importants i en què consisteixen.
- Quins són els mètodes específics per aquest component i quines accions realitzen.

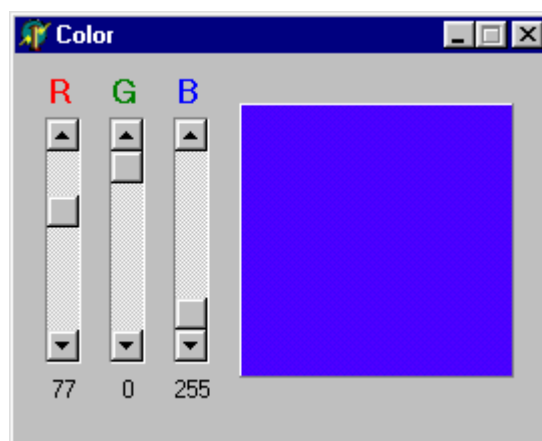
A continuació, fent ús d'aquest nou component, modifiqueu l'exercici 5-1 per tal que rebí les dades pel port sèrie enlloc de ser introduïdes per l'usuari.



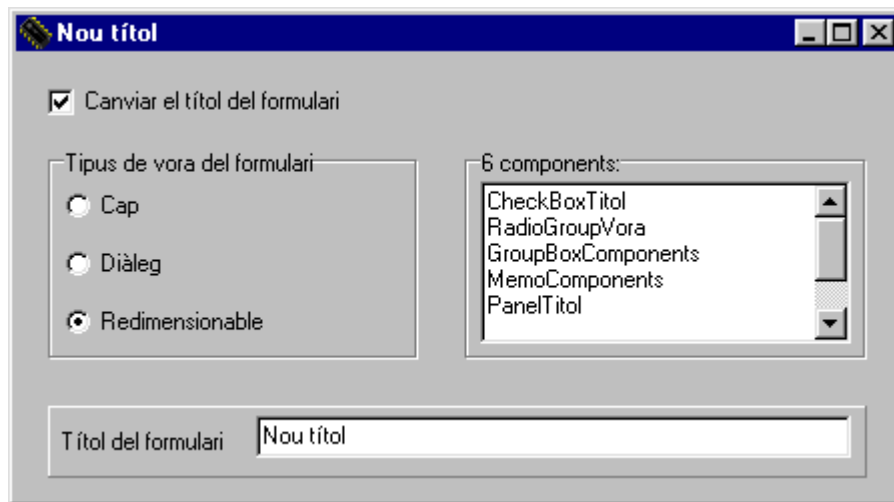
### Exercici 5-3: creació de colors

Dissenyeu una aplicació amb un formulari que permeti visualitzar un color determinat a partir de la barreja d'una tonalitat de vermell, una tonalitat de verd i una tonalitat de blau (color RGB). Aquest formulari està format pels següents components:

- Tres `ScrollBar` que ens permeten canviar el nivell de vermell, de verd i de blau, respectivament. Aquest nivell és un nombre enter entre 0 i 255. Quan canvia la posició de qualsevol de les barres de desplaçament, automàticament canvia el color.
- Tres `Label`, una per cada `ScrollBar`, que indiquen numèricament la tonalitat de vermell, de verd i de blau, respectivament.
- Un `Panel` quadrat, on visualitzem el color indicat per la posició de les barres de desplaçament. La funció `RGB` de l'Object Pascal ens retorna un color a partir dels nivells de vermell, verd i blau que el formen: `RGB(vermell, verd, blau)`.



## Exercici 5-4: dissenyador de formularis



Dissenyeu una aplicació amb un formulari que permeti canviar les característiques d'ell mateix. Aquest formulari està format pels següents components:

- Un `CheckBox` que ens permet canviar el títol del formulari. Quan el `CheckBox` està seleccionat, a la part inferior del formulari apareix un component `Edit` que ens permet canviar el títol del formulari. El cursor es posiciona automàticament al component `Edit`. Quan el `CheckBox` no està seleccionat, aquest component desapareix.
- Un `RadioGroup` amb que podem seleccionar la vora que tindrà el formulari. A més a més, si l'opció 'Cap' està seleccionada, no podem accedir als components `CheckBox` i `Edit` per canviar el títol. L'opció per defecte al `RadioGroup` és 'Diàleg'.
- Un `Memo` i un `GroupBox` envoltant-lo. En crear-se el formulari, al `Memo` apareix automàticament el nombre de components del formulari i al `GroupBox` apareixen automàticament els noms dels components del formulari. (Pista: utilitzeu les propietats `ComponentCount` i `Components` del formulari.)
- Un `Edit` a sobre d'un `Panel`. El text escrit al component `Edit` serà el títol del formulari. Aquest `Edit` serà una mica 'murri': no ens deixarà introduir xifres i canviarà les lletres *a* per *e* i viceversa.
- El formulari tindrà una icona escollida per vosaltres.

I una funcionalitat addicional pensada per vosaltres. Feu servir la imaginació. Per que us feu una idea, exposo algunes que se m'acaben d'ocórrer, però prefereixo que us les inventeu vosaltres:

- Fer que cada dos segons desaparegui un control del formulari i aparegui un altre que estava desaparegut. Repetir-ho cíclicament per tots els controls del formulari.
- Canviar gradualment el color de fons del formulari quan ens desplaçem amb el ratolí per sobre.

L'aparença del formulari en fer-lo més gran o més petit ha de ser impecable. Pista: jugueu amb les propietats `Constraints` del formulari i `Anchors` dels controls.

## Creació d'una aplicació

Als capítols anteriors hem fet una petita introducció a l'entorn de desenvolupament de Delphi, i també hem conegut alguns dels components bàsics que conformen una aplicació i la seva interfície gràfica. Al següent capítol veurem com desenvolupar una aplicació estàndard de Windows. És a dir, una aplicació formada per un menú principal, una o més barres d'icones, una barra d'estat, quadres de diàleg estàndard, amb ús del portaretalls i d'impressió, amb fitxer d'ajuda, i capaç de treballar amb més d'un document alhora. Tot això ho farem seguint un petit tutorial on crearem la nostra aplicació: un petit editor de text.

### Inici del tutorial

1. Creeu una carpeta anomenada *TextEditor* dins la carpeta *Projects* que hi ha al directori on hi ha instal·lat Delphi (normalment "C:\Archivos de programa\Borland\Delphi5").

2. Inicieu Delphi i creeu una nova aplicació.

A Delphi, cada aplicació està representada per un projecte. Quan iniciem Delphi, aquest crea un projecte nou per defecte. Aquest nou projecte, inicialment està format pels següents fitxers:

-*Project1.dpr*: fitxer de codi font Object Pascal associat al projecte.

-*Unit1.pas*: fitxer de codi font Object Pascal associat a la finestra principal del projecte.

-*Unit1.dfm*: fitxer que conté informació sobre els atributs de la finestra principal del projecte.

Cada finestra consta del seu fitxer de codi i d'atributs. Si creeu un nou formulari, es crearan automàticament els seus corresponents fitxers *Unit2.pas* i *Unit2.dfm*.

3. Escolliu *File* → *Save all* per guardar els fitxers a disc. Quan aparegui el corresponent quadre de diàleg, guardeu el codi de la unitat amb nom *Unit1.pas* i el codi del projecte amb nom *TextEditor.dpr*. El nom del fitxer executable de l'aplicació serà el mateix que el del fitxer del projecte, però amb extensió *.exe*.

Més endavant, podreu tornar a guardar els fitxers de l'aplicació amb *File* → *Save all*. Veureu que a mesura que avancem en el desenvolupament de l'aplicació, aniran apareixent nous fitxers amb extensions que no hem comentat: *TextEditor.dof* per guardar opcions de Delphi, *TextEditor.cfg* per guardar la configuració, *TextEditor.res* per guardar els recursos de l'aplicació, *\*.dcu* per guardar el codi compilat de les unitats, *\*.~\** per guardar còpies de seguretat de fitxers de codi que han estat modificats. Aquests fitxers no són importants. No cal que us amoïneu per ells.

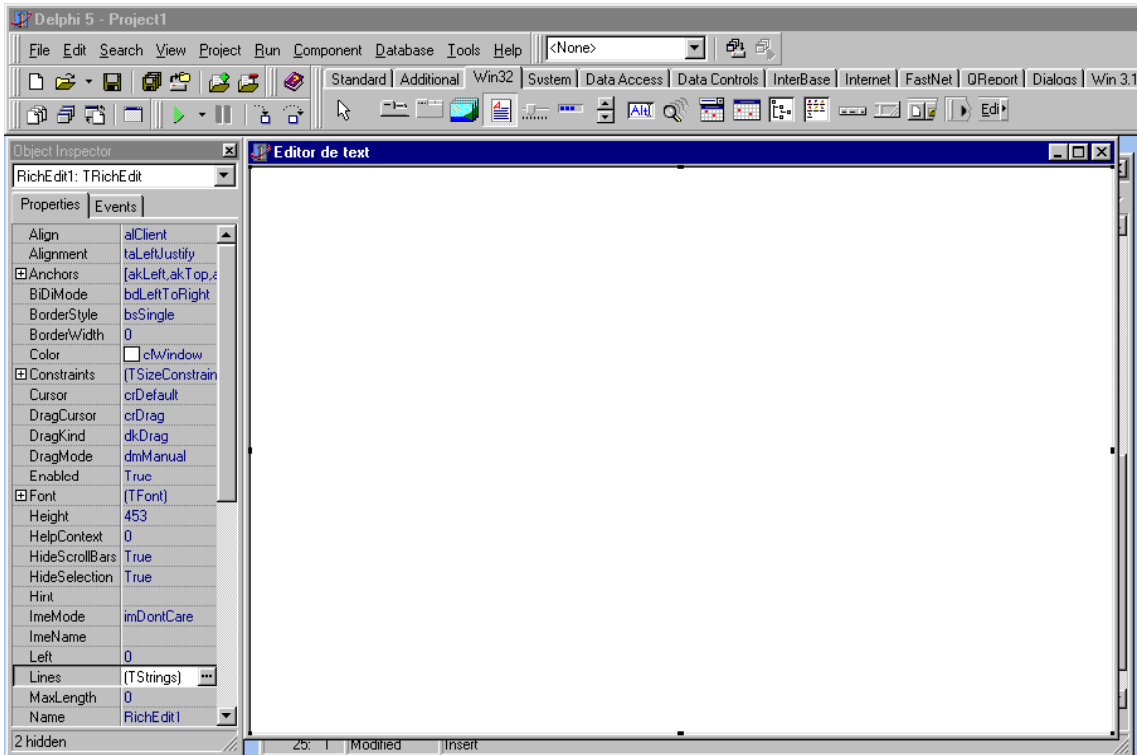
4. Modifiqueu la propietat *Caption* del formulari a l'inspector d'objectes i escriviu "Editor de text". Aquest serà el títol que apareixerà a la finestra.
5. Introduïu dins el formulari un component *RichEdit* (quadre d'edició de text amb format *rtf*) de la paleta de components. A l'inspector d'objectes, modifiqueu la propietat *Align* del quadre d'edició assignant-li el valor *alClient*. D'aquesta manera, el quadre d'edició omplirà l'espai lliure de la finestra, encara que aquesta canviï de mida. També ajusteu el valor de la seva propietat *ScrollBars* a *ssBoth*, per tal que quan el text no càpiga dins el quadre d'edició apareguin dins aquestes barres de desplaçament vertical i horitzontal. A continuació, modifiqueu la propietat *Lines* del quadre d'edició per esborrar el text que apareix per defecte. Per últim, poseu el valor *True* a la propietat *PlainText*, per indicar que llegirem de fitxer i guardarem a fitxer el text d'aquest component com a text pla i no com a text enriquit, és a dir, sense format.

De manera opcional, segons si voleu que les línies de text que siguin més amples que el quadre d'edició es trenquin i continuen a la següent línia o no, doneu un valor a la seva propietat *WordWrap* (per defecte es trenquen).

També de manera opcional, seleccioneu si voleu un tipus de lletra bonic pel text del quadre d'edició, amb la seva propietat *Font*. Una font equiespaiada (totes les lletres ocupen la mateixa amplada) com



*Courier New* pot quedar bé. Feu cas al vostre gust, però no escolliu tipus de lletra molt estranys que l'usuari de l'aplicació pugui no tenir instal·lats.



## Els components i

Anem a canviar una mica la manera de fer que em vist fins ara i a seguir una nova filosofia a l'hora de dissenyar la interfície gràfica d'una aplicació.

### Llistes d'imatges

El component `ImageList` ens permet, entre d'altres coses, gestionar de manera ràpida i eficient les icones de la nostra aplicació. Es tracta d'un vector d'imatges, totes de la mateixa dimensió, on podem emmagatzemar en temps de disseny les icones que emprarem a la nostra aplicació, i referenciar-les posteriorment per la seva posició dins el vector.

La manera de treballar amb aquest component és la següent. Imaginem els controls visuals amb icona (botons, opcions del menú, etc.) de la nostra aplicació. Enlloc de anar control per control associant la seva icona a una imatge que cerquem a un fitxer, el que farem serà inserir un component `ImageList` a la nostra aplicació. A continuació fem doble clic per editar aquest component. S'obrirà un quadre de diàleg que ens permetrà inserir totes les imatges que vulguem a la llista d'imatges. Un cop ja tenim editada la llista d'imatges, podem associar icones a un conjunt de components de la següent nova manera: a la propietat `Images` del conjunt de components seleccionem el nom de la llista d'imatges, i a la propietat `ImageIndex` dels components seleccionem l'índex de la imatge dins aquesta llista.

Què guanyem amb això? D'una banda, sembla més ràpid treballar en el desenvolupament d'una aplicació tenint totes les icones agrupades a un component. D'altra banda, no tindrem imatges repetides a memòria encara que dos controls tinguin associada una mateixa imatge.

### Llistes d'accions

Per entendre el significat del component `ActionList` observem com seria el disseny d'una aplicació estàndard tal com hem fet fins ara. A una aplicació podem realitzar una mateixa acció (per exemple, guardar un document) fent servir diferents controls (opció *guardar* al menú principal, botó *guardar* a la barra d'eines, opció *guardar* al menú contextual, ...). Segurament tots aquests controls que inicien

L'execució d'un cert codi tindran un mateix text, una mateixa icona i una mateixa ajuda emergent que caldrà associar a cadascun per separat quan dissenyem l'aplicació. Segurament quan no vulguem que el codi associat a aquests controls s'executi, haurem d'establir, un a un, la seva propietat `Enable` a `False`. Existeix una manera de fer això per a tots aquests controls alhora de manera automàtica? Sí, mitjançant llistes d'accions.

Una acció és una porció de codi que es pot executar, com si fos una funció o procediment, però que a més a més porta associades com a propietats, entre d'altres, un nom (`Caption`), una icona d'una llista d'imatges (`ImageIndex`), una línia d'ajuda emergent (`Hint`), una pàgina d'ajuda (`HelpContext`), una combinació de tecles per fer dreuera (`Shortcut`) i uns indicadors de si està seleccionada (`Checked`), habilitada (`Enable`) i visible (`Visible`). Quan associem una acció a un component, aquest adquirirà automàticament les propietats abans esmentades de l'acció, i el codi de l'acció s'executarà quan facin clic al component. Podem associar una mateixa acció a varis components.

Com a esdeveniments importants de l'acció tenim `onExecute` i `onUpdate`. L'esdeveniment `onExecute` conté el codi de l'acció a executar, mentre que l'esdeveniment `onUpdate` permet habilitar o inhabilitar els controls associats a l'acció, depenent de les condicions actuals. Per exemple, no té sentit tenir habilitats a una aplicació els botons de tallar i copiar al portafolis si no tenim res seleccionat. Tampoc no té gaire sentit tenir habilitat el botó de guardar un document a fitxer si no hem realitzat canvis sobre aquest document. Habilitar o inhabilitar controls associats a una acció segons els canvis que vagin succeint és tan senzill com col·locar els valors `True` o `False`, respectivament, a la propietat `Enabled` de l'acció, sempre dins el codi de l'esdeveniment `onUpdate` de l'acció o de la llista d'accions. Exemple:

```
procedure TForm1.GuardarActionExecute(Sender: TObject);
begin
  Memo1.Lines.SaveToFile(NomFitxer);
end;

procedure TForm1.GuardarActionUpdate(Sender: TObject);
begin
  GuardarAction.Enabled := Memo1.Modified and (Length(Memo1.Lines.Text) > 0);
end;
```

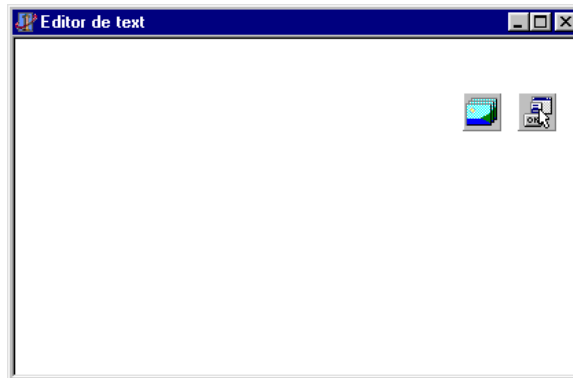
La manera de treballar amb llistes d'accions és la següent. Imaginem els controls visuals de la nostre aplicació que tenen associat, mitjançant un esdeveniment, un codi a executar. Enlloc de anar control per control ajustant les seves propietats principals, el que farem serà inserir un component `ActionList` a la nostre aplicació. A continuació fem doble clic per editar aquest component. S'obrirà un quadre de diàleg que ens permetrà inserir accions a la llista d'accions. Aquestes accions podran ser tant noves accions definides per nosaltres com accions ja predefinides de Delphi. De cadascuna d'aquestes accions ajustarem els valors de les propietats que ja hem esmentat, així com el seu codi a executar. Un cop ja tenim editada la llista d'accions, podem associar una acció a un component de la següent manera: a la propietat `Action` del component seleccionem el nom de l'acció. Com ja hem dit, aquest component adquirirà automàticament les propietats de l'acció, i al seu esdeveniment `onClick` s'associarà el codi de l'esdeveniment `onExecute` de l'acció.

### La nostre aplicació 'Editor de Text'

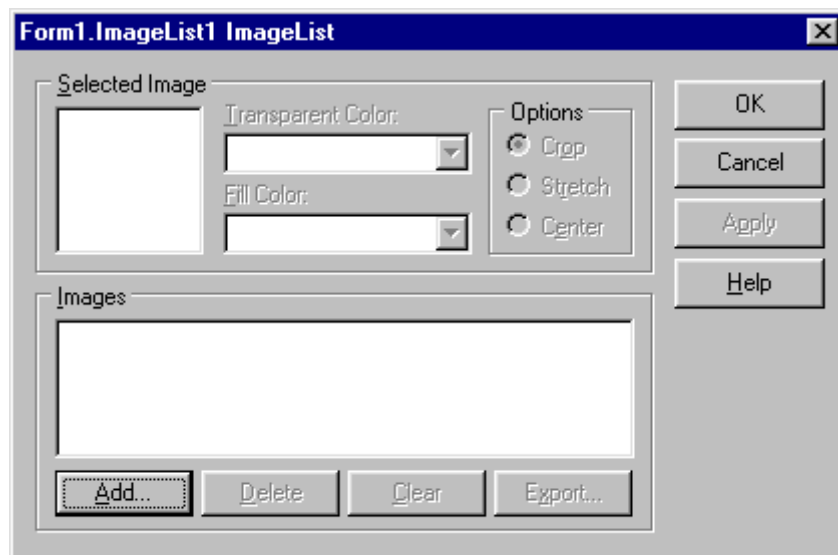
Provem tot això que acabem de veure amb un exemple. Continuem amb el nostre tutorial. El primer que hem de fer es pensar quines accions caldran al nostre editor de text:

| Acció       | Descripció                                      | Menú   | Icona          |
|-------------|---|--------|----------------|
| Crear       | Crear un nou document                           | Fitxer | 0 FileOpen.bmp |
| Obrir       | Obrir un document existent per editar           | Fitxer | 1 FileNew.bmp  |
| Guardar     | Guarda el document actual a disc                | Fitxer | 2 FileSave.bmp |
| Guardar com | Guarda el document actual a disc amb nom nou    | Fitxer | -              |
| Sortir      | Tanca l'aplicació                               | Fitxer | 3 DoorShut.bmp |
| Tallar      | Esborra text i el guarda al portaretalls        | Editar | 4 Cut.bmp      |
| Copiar      | Guarda text al portaretalls                     | Editar | 5 Copy.bmp     |
| Enganxar    | Insereix text del portaretalls                  | Editar | 6 Paste.bmp    |
| Continguts  | Mostra la taula de contingut de l'ajuda         | Ajuda  | 7 Help.bmp     |
| Índex       | Mostra l'índex de tòpics de l'ajuda             | Ajuda  | -              |
| A propòsit  | Mostra informació de l'aplicació a una finestra | Ajuda  | -              |

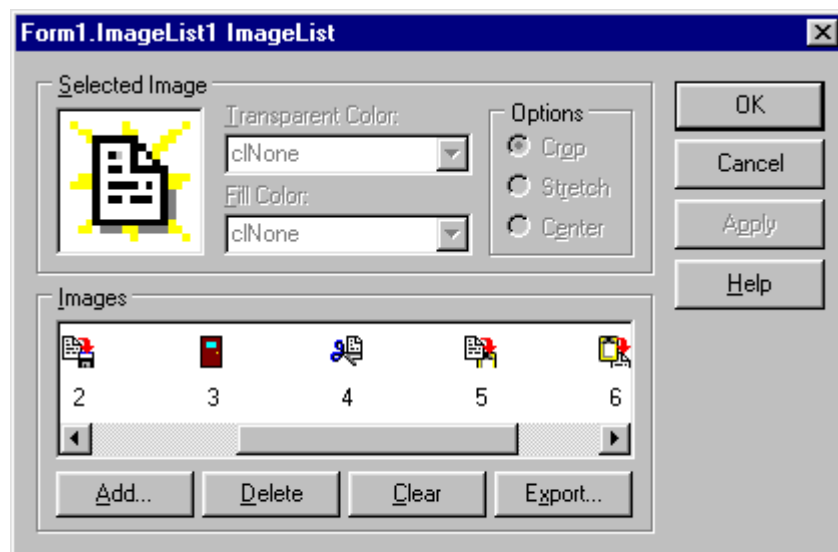
- Introduïu dins el formulari un component `ImageList` de la paleta de components *Win32*, i un component `ActionList` de la paleta de components *Standard*. Aquests components no són visuals, així que no importa on els col·loqueu sobre el formulari, ja que no apareixeran per pantalla en temps d'execució.



- Fem doble clic sobre el component `ImageList` per tal de carregar la llista d'imatges dels corresponents fitxers.

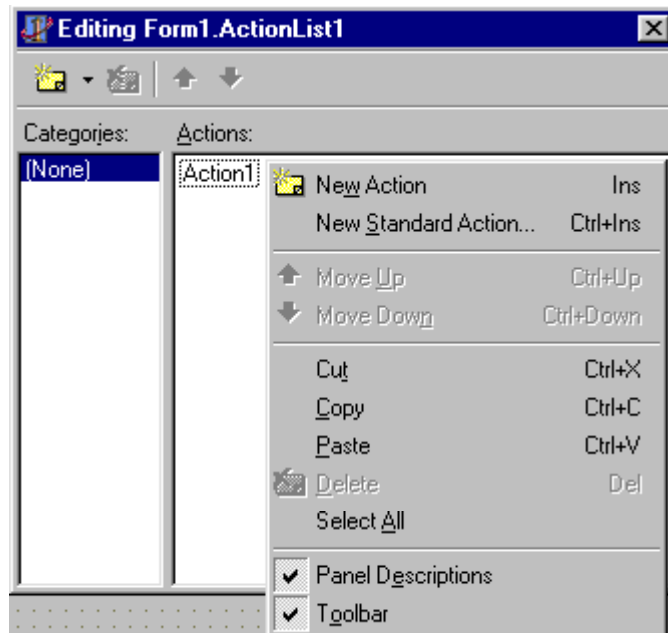


Aquests es troben a la carpeta "C:\Archivos de programa\Archivos comunes\Borland Shared\Images\Buttons\)", amb els noms FileOpen.bmp, FileNew.bmp, FileSave.bmp, DoorShut.bmp, Cut.bmp, Copy.bmp, Paste.bmp i Help.bmp.



Quan aparegui un missatge preguntant si voleu separar el mapa de bits en dos, responeu que sí. Cada icona contindrà una versió activa de la imatge i una versió inactiva (en gris). Elimineu aquesta última.

8. Fem doble clic sobre el component `ActionList` per tal de crear la nostre llista d'accions. Apareixerà el quadre de diàleg d'edició de llistes d'accions. Fem clic amb el botó dret del ratolí i seleccionem l'opció *New Action* al menú sensible al context que apareixerà.



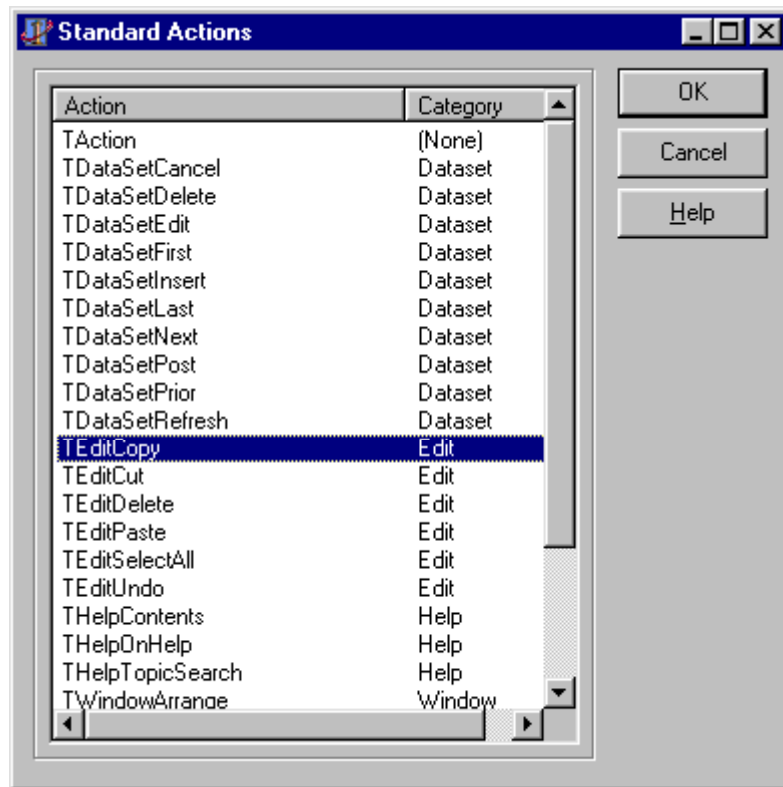
Fem clic sobre l'acció que hem creat per seleccionar-la i a l'inspector d'objectes modifiquem les seves següents propietats:

| Caption | Category | Hint                | ImageIndex | Name        |
|---------|----------|---------------------|------------|-------------|
| &Nou    | Fitxer   | Crear un nou fitxer | 0          | FitxerCrear |

Repetim el procés per a les següents noves accions:

| Caption      | Category | Hint                     | ImageIndex | Name             |
|--------------|----------|--------------------------|------------|------------------|
| &Obrir       | Fitxer   | Obrir un fitxer existent | 1          | FitxerObrir      |
| &Guardar     | Fitxer   | Guardar el fitxer actual | 2          | FitxerGuardar    |
| G&uardar com | Fitxer   | Guardar a un nou fitxer  | -1         | FitxerGuardarCom |
| &Sortir      | Fitxer   | Sortir del programa      | 3          | FitxerSortir     |
| &Contingut   | Ajuda    | Visualitzar l'ajuda      | 7          | AjudaContingut   |
| &Índex       | Ajuda    | Índex de l'ajuda         | -1         | AjudaIndex       |
| &A propòsit  | Ajuda    |                          | -1         | AjudaAProposit   |

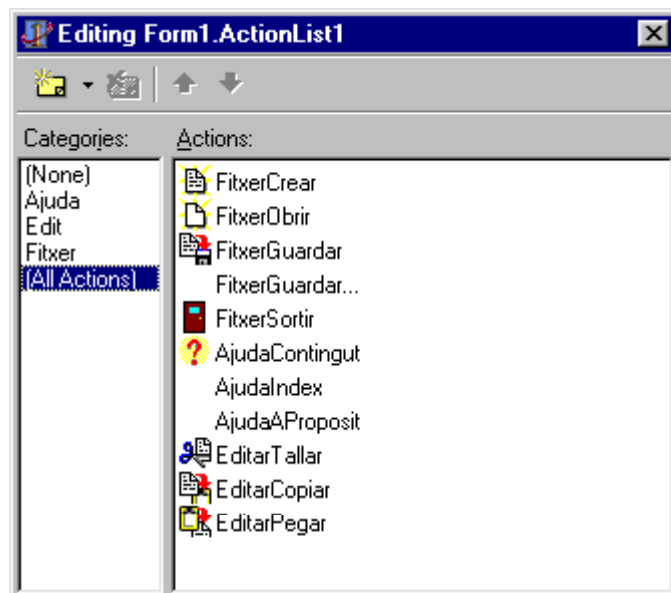
Ara afegirem accions ja predefinides de Delphi. Fem clic amb el botó dret del ratolí sobre l'editor de llistes d'accions i seleccionem l'opció *New Standard Action* al menú sensible al context que apareixerà.



Les noves accions estàndard a afegir són TEditCut, TEditCopy i TEditPaste. Modificarem les seves propietats:

| Caption | Category | Hint                    | ImageIndex | Name         |
|---------|----------|-------------------------|------------|--------------|
| &Tallar | Edit     | Tallar al portafolis    | 4          | EditarTallar |
| &Copiar | Edit     | Copiar al portafolis    | 5          | EditarCopiar |
| &Pegar  | Edit     | Enganxar del portafolis | 6          | EditarPegar  |

A aquestes alçades, a l'editor de llistes d'accions ens hauriem de trobar quelcom tal que així:



Quelcom falla? A la llista d'accions no apareixen les icones del gràfic anterior? Que ens hem deixat? Hem associat a les nostres accions l'index de les imatges, però no hem associat a la nostra llista d'accions la corresponent llista d'imatges. Fem clic sobre la nostra llista d'accions (per defecte ActionList1) per seleccionar-la, i a l'inspector d'objectes donem el nom de la nostre llista d'imatges (per defecte ImageList1) a la seva propietat Images. Ara sí, oi?

## Una aplicació estàndard

Gairebé tota aplicació estàndard de Windows està formada per una barra de menú i una barra d'eines a la part superior, una barra d'estat a la part inferior i un àrea de treball a la part central.

### El Component *MainMenu*

El component `MainMenu` permet afegir un menú principal a un formulari. Aquest menú principal consta de la barra de menú i les seves opcions de menú en cascada.

Per tal d'editar el menú principal, cal fer doble clic sobre el component `MainMenu` un cop s'ha afegit aquest al formulari. Ens apareixerà un editor de menús multinivell que ens permetrà canviar les propietats de cada opció del menú, així com inserir noves opcions (tecla `Insert`), nous submenús (combinació de tecles `Ctrl` + `→`) i esborrar opcions existents (tecla `Supr`). El text que apareixerà a cada opció del menú s'escriu a la corresponent propietat `Caption`. Si volem una línia de separació enlloc d'una opció de menú haurem d'escriure un guió '-' a l'esmentada propietat. Si volem associar a l'opció del menú una acció, escriurem el nom d'aquesta acció a la propietat `Action` de l'opció del menú.

Els menús s'organitzen en forma d'arbre. A la propietat `Items` de cada opció del menú, podem afegir noves opcions del menú (submenús). I així successivament, formant el que s'anomena menús en cascada.

### Component *ToolBar*

El component `ToolBar` ens permet agrupar conjunts de botons. Aquest component té varies propietats interessants però, com no tinc temps de comentar-les aquí, només les enumeraré per si voleu cercar vosaltres mateixos/es la informació: `ButtonHeight`, `ButtonWidth`, `Flat`, `Indent`, `List` i `ShowCaptions`. La única propietat seva que comentaré és `Images`, que conté el nom de la llista d'imatges que conté les icones a associar amb els botons de la barra.

Els botons de la barra d'eines són components del tipus `ToolButton`. Com a propietat nova i important d'aquests components trobem `Action`, que guarda el nom de l'acció associada al component, en el cas que haguem definit alguna llista d'accions. Si volem botons plans, enlloc de botons amb aspecte tridimensional, haurem d'establir el valor `True` a la propietat `Flat` de la barra d'eines.

La manera de treballar amb una barra d'eines és la següent: un cop inserida dins la finestra, fem clic sobre ella amb el botó dret del ratolí per desplegar el menú sensible al context. A aquest menú trobarem les opcions d'afegir un nou botó i d'afegir un espai separador de botons. Un cop afegit un nou botó, si fem clic a sobre d'ell el tindrem seleccionat i podrem editar les seves propietats mitjançant l'inspector d'objectes.

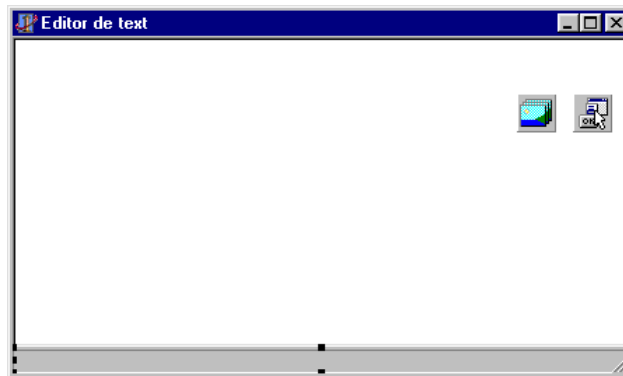
### Component *StatusBar*

L'ús de la barra d'estat és molt senzill. Un cop inserida dins la finestra, ens trobem amb dos modes de treball. Al mode més senzill la barra no està subdividida. En aquest cas la seva propietat `SimplePanel` conté el valor `True` i la seva propietat `SimpleText` conté la cadena de text a escriure. Per accedir en temps d'execució al text escrit dins aquesta barra d'estat haurem d'escriure quelcom semblant a `NomStatusBar.SimpleText := ...`. A l'altre mode de treball la barra està subdividida en dues o més seccions independents. En aquest cas, la seva propietat `SimplePanel` conté el valor `False` i si fem doble clic a la propietat `Panels` podem editar el nombre de subdivisions i la seva mida en píxels. Per accedir en temps d'execució al text escrit dins una subdivisió haurem d'escriure `NomStatusBar.Panels[index].Text := ...`, on l'índex 0 correspon a la primera subdivisió.

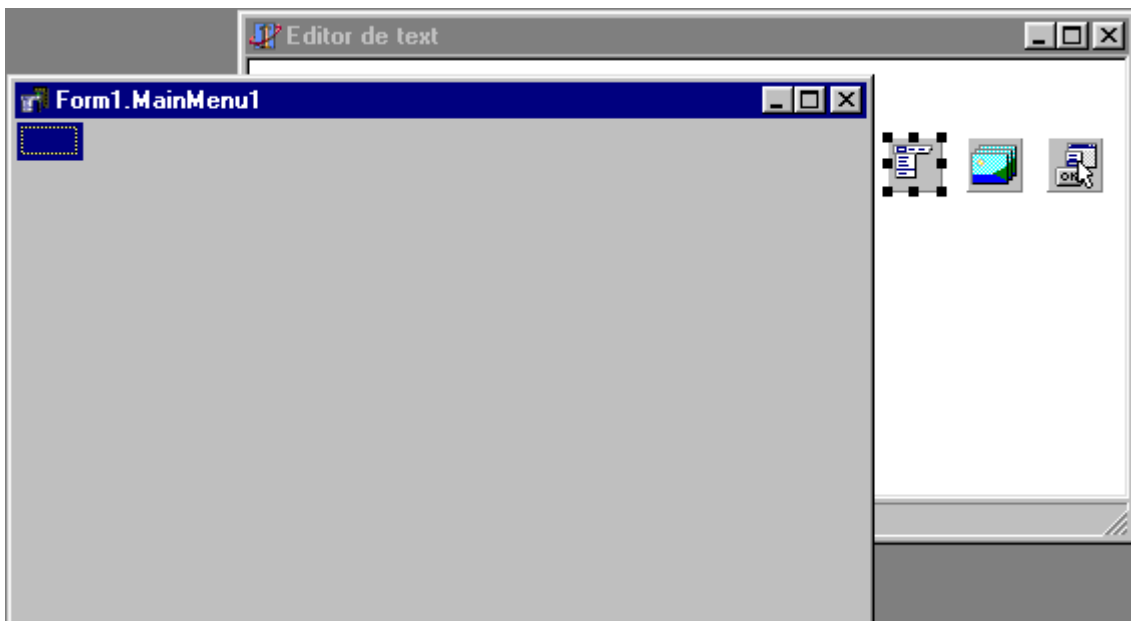
### La nostra aplicació 'Editor de Text'

9. Introduïu dins el formulari un component `StatusBar` de la paleta de components `Win32`. S'afegirà automàticament a baix de tot de la finestra. L'utilitzarem només per visualitzar el nom del fitxer que

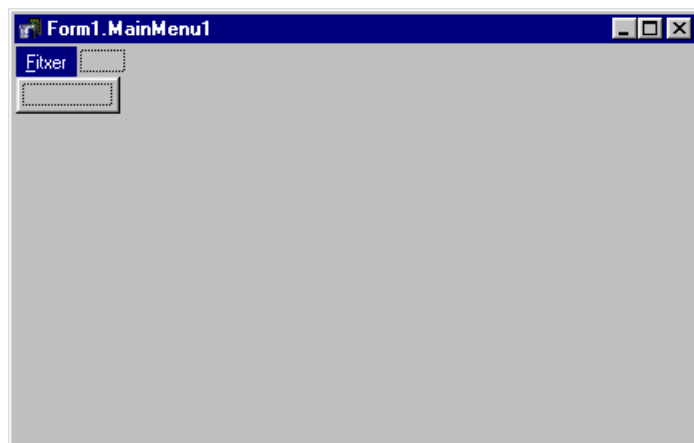
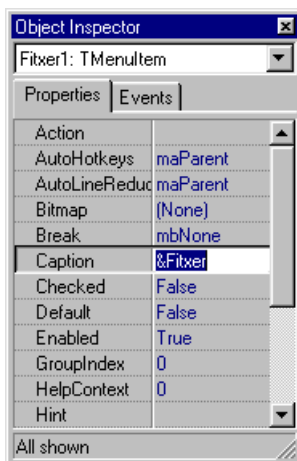
estem editant, així que amb ell treballarem en mode senzill. Posarem a True la seva propietat SimplePanel.



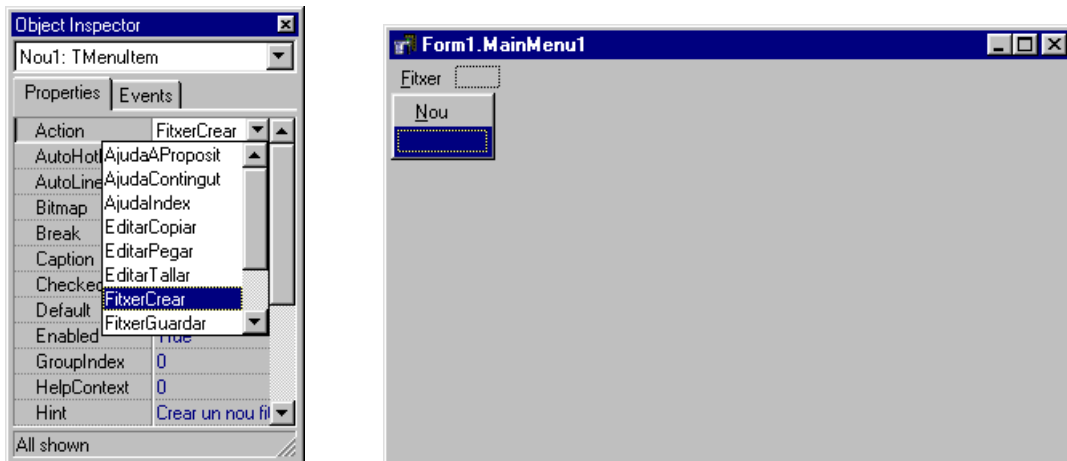
10. Introduïu dins el formulari un component MainMenu de la paleta de components *Standard*. Establirem el valor de la seva propietat Images a ImageList1, que és el nom de la nostra llista d'imatges. A continuació fem doble clic sobre el component per tal de obrir el dissenyador de menús.



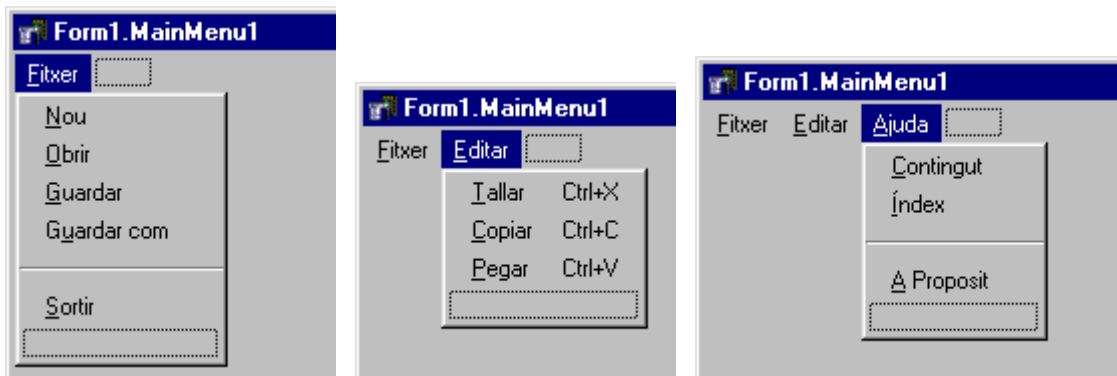
11. A l'inspector d'objectes, escriviu &Fitxer a la propietat Caption de la primera categoria del menú. Si a continuació seleccioneu la opció de menú que acabeu de crear, observareu que a sota apareix una opció del menú buida.



12. Seleccioneu aquesta opció del menú buida i a l'inspector d'objectes associeu l'acció `FitxerCrear` a la seva propietat `Action`.



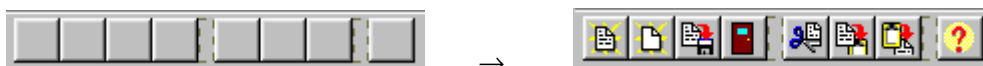
13. Repetim el procés per a les diferents accions de la nostra aplicació, inserint algun separador (caràcter '-' a la propietat `Caption`) de tant en tant.



14. Tanqueu el dissenyador de formularis, guardeu l'aplicació (`File → Save`) i premeu la tecla `[F9]` per compilar i executar el programa. Funciona? Encara no fa res perquè no em escrit el codi de les accions a executar. Tot arribarà, però abans tanqueu el programa en execució i anem a afegir una barra d'eines.
15. Introduïu dins el formulari un component `ToolBar` de la paleta de components `Win32`. Establirem el valor de la seva propietat `Images` a `ImageList1` (el nom de la nostra llista d'imatges), el valor de la seva propietat `Indent` a 4 (la separació en píxels del primer botó al marge esquerre) i el valor de la seva propietat `ShowHint` a `True` (opció de mostrar quadre d'ajut emergent).
16. A continuació afegirem els botons i separadors a la barra d'eines. Fem clic sobre la barra amb el botó dret del ratolí i seleccionem afegir nou botó quatre cops, afegir separador, afegir nou botó tres cops, afegir separador i afegir nou botó.



17. Consecutivament associeiem la propietat `Action` d'aquests botons a les accions `FitxerObrir`, `FitxerGuardar`, `FitxerSortir`, `EditarTallar`, `EditarCopiar`, `EditarPegar` i `AjudaContingut`.





18. Si ara premeu la tecla **F9** per compilar i executar el programa veureu que les accions estàndard de tallar, copiar i enganxar ja funcionen.

## Quadres de diàleg estàndard

Els components de la paleta *Dialogs* fan que els quadres de diàleg estàndard de Windows estiguin disponibles a les nostres aplicacions. D'una banda, això ens estalviarà molta feina, ja que no els haurem de dissenyar nosaltres. D'altra banda, aquests quadres de diàleg proporcionen una interfície consistent i ja coneguda per l'usuari per a tot un seguit d'operacions com: cercar i obrir fitxers, guardar fitxers, configurar les opcions d'impressió i la impressora, seleccionar fonts i colors, cercar i reemplaçar text, etc.



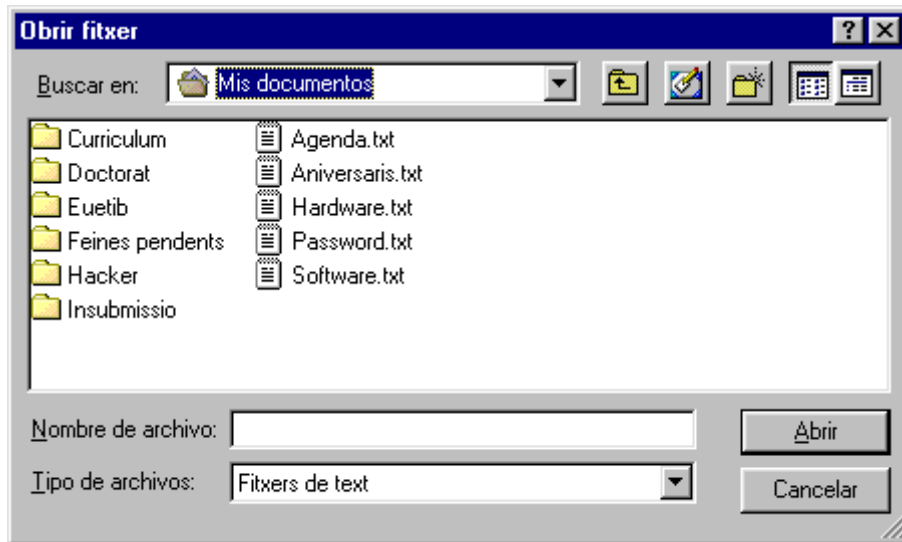
La manera de treballar amb aquests quadres de diàleg es la següent. De la paleta de components *Dialogs* inserim a la nostre finestra tots els quadres de diàleg que considerem necessaris. Encara que la seva icona aparegui a la nostre finestra, aquests quadres de diàleg no es visualitzaran fins que el seu mètode `Execute` sigui cridat. Aquest mètode `Execute` retornarà els valors booleans `True` o `False` segons s'hagi tancat el quadre de diàleg seleccionant un valor o cap, respectivament. El codi que normalment s'utilitza és:

```
// Seleccióem el conjunt de característiques del quadre (opcional)
NomQuadreDialog.Options := [...];

// Fem apareixer el quadre de diàleg
if NomQuadreDialog.Execute then
... // Codi a executar si s'ha seleccionat quelcom
else
... // Codi a executar si no s'ha seleccionat res (opcional)
```

Cada quadre de diàleg de la paleta *Dialogs* té propietats diferents. De moment aquí veurem els quadres de diàleg per obrir i tancar un fitxer: `OpenDialog` i `SaveDialog`. Els components `OpenPictureDialog` i `SavePictureDialog` són similars però especialitzats en fitxers d'imatges (disposen d'un quadre addicional per previsualitzar la imatge que conté el fitxer). Els components `PrintDialog` i `PrinterSetupDialog` els veurem a l'apartat dedicat a la impressió. La resta els haureu de veure pel vostre compte. Les propietats més importants de `OpenDialog` i `SaveDialog` són:

- `Title`: Especifica el text a la barra de títol del quadre de diàleg.
- `InitialDir`: Determina el directori inicial quan el quadre de diàleg s'obre.
- `Filter` i `FilterIndex`: La propietat `Filter` especifica les màscares per filtrar fitxers. S'edita fent doble clic sobre d'ella a l'inspector d'objectes. La propietat `FilterIndex` és un nombre enter que indica quina és la màscara per defecte quan s'obre el quadre de diàleg (1 per la primera, 2 per la segona, etc.).
- `DefaultExt`: Conté la extensió per defecte del fitxer. Si l'usuari no escriu cap extensió, o si la que escriu no està registrada, s'afegeix aquesta.
- `Options`: És un conjunt de moltes petites opcions booleanes (cert o fals) per controlar quines característiques del quadre de diàleg estan habilitades. És molt interessant mirar-se-les, però aquí no hi ha espai per comentar-les totes. Permeten, per exemple, seleccionar si volem avisar l'usuari de si el fitxer que ha seleccionat per guardar les dades ja existeix, si volem avisar l'usuari si intenta obrir un fitxer que ja està obert, etc.
- `FileName`: Conté el nom i camí del fitxer que ha seleccionat l'usuari.



### La nostre aplicació 'Editor de Text'

19. Anem a escriure el codi de l'aplicació. Primer de tot caldrà una variable per emmagatzemar el nom del fitxer de text que l'usuari editarà. Dintre de l'espai de declaracions públiques del nostre formulari declararem la variable *NomFitxer* de tipus *String* (text en cursiva).

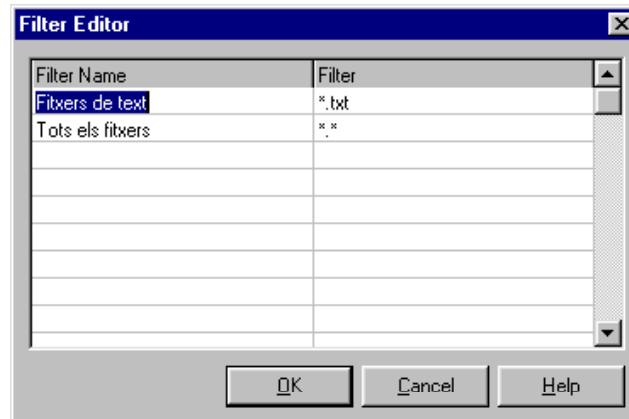
```
type
  TForm1 = class(TForm)
    ...
  private
    { Private declarations }
  public
    { Public declarations }
    NomFitxer: String;
  end;
```

A continuació fem doble clic sobre la llista d'accions per obrir l'editor de llistes d'accions.

20. Feu doble clic sobre l'acció *FitxerCrear*, de la categoria *Fitxer*. Automàticament apareixerà la capçalera del seu esdeveniment *onExecute*, on haureu d'afegir (text en cursiva):

```
procedure TForm1.FitxerCrearExecute(Sender: TObject);
begin
  RichEdit1.Clear;
  NomFitxer := 'Sensenom.txt';
  StatusBar1.SimpleText := NomFitxer;
end;
```

21. Introduïu dins el formulari un component *OpenDialog* de la paleta de components *Dialogs*. Seleccionem aquest component i a l'inspector d'objectes doneu el valor *txt* a la seva propietat *DefaultExt*, per indicar la extensió per defecte del fitxer quan l'usuari no especifica aquesta. Doneu també el valor *Obrir fitxer* a la seva propietat *title*, per indicar el títol que apareixerà al quadre de diàleg. A continuació feu doble clic sobre la propietat *Filter* per visualitzar l'editor de filtres, i escriviu:



Ara, a l'editor de llistes d'accions feu doble clic sobre l'acció *FitxerObrir*, de la categoria *Fitxer*. Automàticament apareixerà la capçalera del seu esdeveniment `onExecute`, on haureu d'afegir (text en cursiva):

```

procedure TForm1.FitxerObrirExecute(Sender: TObject);
begin
    if OpenFileDialog1.Execute then
        begin
            RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
            RichEdit1.Modified := False;
            NomFitxer := OpenFileDialog1.FileName;
            StatusBar1.SimpleText := NomFitxer;
        end;
end;

```

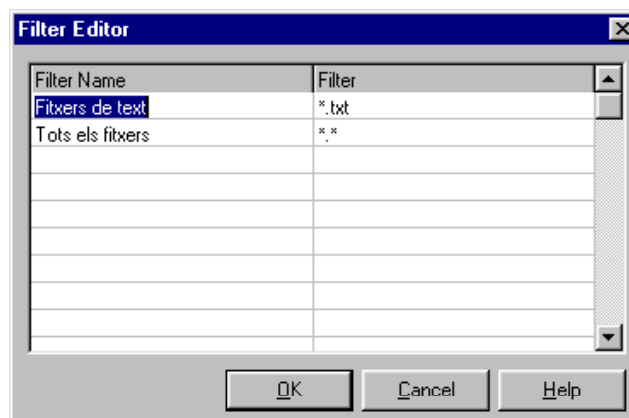
22. Feu doble clic sobre l'acció *FitxerGuardar*, de la categoria *Fitxer*. Automàticament apareixerà la capçalera del seu esdeveniment `onExecute`, on haureu d'afegir (text en cursiva):

```

procedure TForm1.FitxerGuardarExecute(Sender: TObject);
begin
    if NomFitxer = 'Sensenom.txt' then
        FitxerGuardarComExecute(nil)
    else
        begin
            RichEdit1.Lines.SaveToFile(NomFitxer);
            RichEdit1.Modified := False;
        end;
end;

```

23. Introduïu dins el formulari un component `SaveDialog` de la paleta de components *Dialogs*. Seleccionem aquest component i a l'inspector d'objectes doneu el valor `txt` a la seva propietat `DefaultExt`, per indicar la extensió per defecte del fitxer quan l'usuari no especifica aquesta. Doneu també el valor `Guardar` com a la seva propietat `title`, per indicar el títol que apareixerà al quadre de diàleg. A continuació feu doble clic sobre la propietat `Filter` per visualitzar l'editor de filtres, i escriviu:



Ara, a l'editor de llistes d'accions feu doble clic sobre l'acció *FitxerGuardarCom*, de la categoria *Fitxer*. Automàticament apareixerà la capçalera del seu esdeveniment *onExecute*, on haureu d'afegir (text en cursiva):

```
procedure TForm1.FitxerGuardarComExecute(Sender: TObject);  
begin  
  SaveDialog1.FileName := NomFitxer;  
  SaveDialog1.InitialDir := ExtractFilePath(NomFitxer);  
  if SaveDialog1.Execute then  
    begin  
      RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);  
      RichEdit1.Modified := False;  
      NomFitxer := SaveDialog1.FileName;  
      StatusBar1.SimpleText := NomFitxer;  
    end;  
end;
```

24. Feu doble clic sobre l'acció *FitxerSortir*, de la categoria *Fitxer*. Automàticament apareixerà la capçalera del seu esdeveniment *onExecute*, on haureu d'afegir (text en cursiva):

```
procedure TForm1.FitxerSortirExecute(Sender: TObject);  
begin  
  Close;  
end;
```

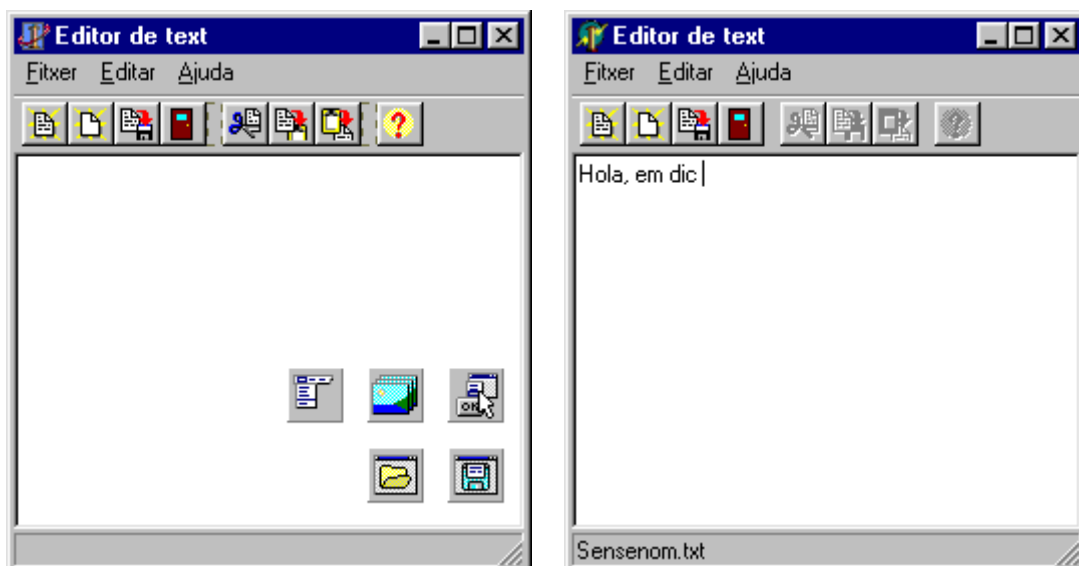
25. Seleccioneu la llista d'accions. A la paleta d'esdeveniments de l'inspector d'objectes, cerqueu l'esdeveniment *onUpdate* de la llista d'accions i feu doble clic sobre ell. Aquest esdeveniment s'executa quan la nostra aplicació entra en inactivitat i no està processant cap altre esdeveniment. Haureu d'escriure el següent codi (text en cursiva):

```
procedure TForm1.ActionList1Update(Action: TBasicAction;  
                                   var Handled: Boolean);  
begin  
  FitxerGuardar.Enabled := RichEdit1.Modified and  
                           (Length(RichEdit1.Lines.Text) > 0);  
  FitxerGuardarCom.Enabled := FitxerGuardar.Enabled;  
end;
```

26. Seleccioneu el formulari. A la paleta d'esdeveniments de l'inspector d'objectes, cerqueu l'esdeveniment *onCreate* del formulari i feu doble clic sobre ell. Aquest esdeveniment s'executa quan es crea la finestra, en iniciar l'aplicació. Haureu d'escriure el següent codi (text en cursiva):

```
procedure TForm1.FormCreate(Sender: TObject);  
begin  
  FitxerCrearExecute(nil);  
end;
```

27. Premeu la tecla **F9** per compilar i executar el programa. Molt millor, oi?



## Ús del portafolis

El portafolis és un dels mecanismes més habituals que utilitza l'usuari per intercanviar informació entre aplicacions. Es tracta d'un espai on l'usuari pot enviar informació seleccionada a la seva aplicació ('tallar' i 'copiar'), per després inserir-la dins la mateixa aplicació o una altra.

A la col·lecció de components VCL, el portafolis ve representat per la classe `TClipboard` i, per defecte, ja existeix un objecte d'aquesta classe anomenat `Clipboard`. Per tal que la nostra aplicació pugui treballar amb el portafolis no cal inserir cap component dins la nostra finestra. Basta afegir `Clipbrd` a la clàusula `uses` de la unitat: `uses ..., Clipbrd;`

El portafolis no només conté informació, sinó també un identificador del format de la informació continguda, per tal que les aplicacions sàpiguen si poden enganxar aquesta informació i decideixin com han de fer-ho. Cal tenir en compte, però, que el portafolis de Windows pot contenir informació de diferents tipus alhora, com per exemple text i una imatge. Cal tenir en compte, també, que una mateixa informació pot trobar-se al portafolis de Windows codificada en diferents formats alhora, com per exemple una imatge de línies que es trobi al portafolis en format *Metafile* i també en format *Bitmap*. Els mètodes i propietats de la classe `TClipboard` que ens permeten obtenir informació sobre els formats acceptats i continguts al portafolis són:

- `HasFormat`: Aquest mètode pregunta si el portafolis conté informació en un format especificat. Per exemple:

```
if Clipboard.HasFormat(CF_TEXT) then
    Edit1.Text := Clipboard.AsText
else
    MessageDlg('No hi ha text al portafolis', mtInformation, [mbOK], 0);
```

- `Formats` i `FormatCount`: Aquestes propietats només són accessibles en temps d'execució. La propietat `FormatCount` indica el nombre de formats diferents de la informació actualment continguda al portafolis, mentre que la propietat `Formats` és un vector que permet, mitjançant un índex numèric entre 0 i `FormatCount-1`, saber quins són aquests formats. Per exemple:

```
for i := 0 to Clipboard.FormatCount-1 do
    ListBox1.Items.Add(IntToStr(Clipboard.Formats[i]));
```

Quan enganxem informació del portafolis, podem emprar aquestes propietats per conèixer el format més adient d'una informació que pot estar codificada de diferents maneres.

- `GetClipboardFormatName` (*IdentificadorFormat*, *BufferNom*, *LongitudBuffer*): És una funció de Windows que ens permet obtenir el nom d'un format registrat, donat el seu nombre identificador de format.

Per poder treballar al portafolis amb un format nou creat per la nostra aplicació, cal registrar prèviament aquest nou format amb la funció `RegisterClipboardFormat` (*NomNouFormat*) de Windows, que retorna el nombre identificador de format assignat.

### Treballar amb el portafolis

Sempre que copiem informació al portafolis, haurem de bloquejar aquest per tal que cap altra aplicació pugui també copiar informació al mateix temps que la nostra. Per això emprem el mètode `Clipboard.Open` abans de començar les operacions de copiat i el mètode `Clipboard.Close` en finalitzar-les.

El primer que haurem de fer en enganxar informació a la nostra aplicació des del portafolis serà comprovar que aquesta informació està en el format que l'aplicació reconeix, mitjançant el mètode `Clipboard.HasFormat()`, i a continuació obrar en conseqüència.

Si en algun moment desitgem esborrar la informació continguda dins el portafolis, farem servir el mètode `Clipboard.Clear`.

### Copiar i enganxar text

Una manera de copiar i enganxar text al portafolis, és emprar els mètodes `Clipboard.GetTextBuf` i `Clipboard.SetTextBuf` per copiar a un buffer intermedi la cadena de text, i després operar amb aquest buffer. Per exemple:

```
var Text: array [0..16383] of Char; // Fins 16 Kbytes de text
...
Clipboard.Open; // Copiar
Memo1.GetTextBuf(text, 16384);
Clipboard.SetTextBuf(text);
Clipboard.Close;
...
if Clipboard.HasFormat(CF_TEXT) then // Enganxar
begin
Clipboard.GetTextBuf(text, 16384);
Memo1.SetTextBuf(text);
end
```

Per enganxar text, també tenim el mètode `Clipboard.AsText`, que ens permet representar el contingut del portafolis com una cadena de caràcters. Com hem vist a un exemple anterior:

```
if Clipboard.HasFormat(CF_TEXT) then
Edit1.Text := Clipboard.AsText
else
MessageDlg('No hi ha text al portafolis', mtInformation, [mbOK], 0);
```

Tanmateix, alguns controls ja disposen de mètodes especials per copiar al portafolis el text seleccionat dins ells i per enganxar el text contingut al portafolis a la posició del cursor dins ells, com és el cas dels mètodes `CutToClipboard`, `CopyToClipboard` i `PasteFromClipboard` dels components `Edit`, `Memo` i `RichEdit`.

### Copiar i enganxar imatges

Emprem el mètode `Clipboard.Assign(NomImatge)` per copiar imatges al portafolis, i el mètode `NomImatge.Assign(Clipboard)` per enganxar imatges des del portafolis. Per exemple:

```
Clipboard.Assign(SpeedButton1.Glyph); // Copia la icona d'un botó
```

Per treballar amb imatges i el portafolis, també existeixen els mètodes `LoadFromClipboardFormat` i `SaveToClipboardFormat`, però aquests són més complicats d'emprar. Per exemple:

```
var Bitmap: TBitmap;
...
Bitmap := TBitmap.create;
try
Bitmap.LoadFromClipboardFormat(cf_BitMap, Clipboard.GetAsHandle(cf_BitMap), 0);
Canvas.draw(0,0,Bitmap);
finally
Bitmap.free;
end;
```

### Copiar i enganxar components

No és gaire freqüent, però si volguéssiu copiar i enganxar controls visuals al portafolis, hauríeu d'emprar els mètodes `Clipboard.GetComponent` i `Clipboard.SetComponent`. Per copiar el component només cal passar com a argument el nom del component a copiar. Per enganxar el component, cal passar com a arguments el propietari i el contenidor. Abans, però, cal haver registrat el component amb la funció `RegisterClasses`. Un petit exemple:

```
RegisterClasses([TButton]); // Registra el botó
Clipboard.SetComponent(Button1); // Copia el botó al portafolis
Button1.Name := 'OriginalButton'; // Canvia el nom del botó
Clipboard.GetComponent(Self, GroupBox1); // Enganxa el botó a una groupbox
```

### Exemple

Aquí tenim una petita aplicació que permet copiar al portafolis el text i la imatge que conté, i també enganxar dins seu el text i la imatge que contingui el portafolis.



El codi corresponent als botons de copiar i enganxar seria:

```

procedure TForm1.ButtonCopiarClick(Sender: TObject);
var
    Text: array [0..16383] of Char;      // Fins 16 Kbytes de text
begin
    Clipboard.Open;
    Memo1.GetTextBuf(Text, 16384);
    Clipboard.SetTextBuf(Text);
    Clipboard.Assign(Image1.Picture);
    Clipboard.Close;
end;

procedure TForm1.ButtonPegarClick(Sender: TObject);
var
    Text: array [0..16383] of Char;      // Fins 16 Kbytes de text
begin
    // Comprovar si hi ha un text al portapapers
    if Clipboard.HasFormat(CF_TEXT) then
        begin
            Clipboard.GetTextBuf(Text, 16384);
            Memo1.SetTextBuf(Text);
        end
    else
        Memo1.Clear;

    // Comprovar si hi ha un gràfic al portapapers
    if Clipboard.HasFormat(CF_BITMAP) or Clipboard.HasFormat(CF_PICTURE) then
        Image1.Picture.Assign(Clipboard)
    else
        begin
            Image1.Canvas.Brush.Color := Color;
            Image1.Canvas.FillRect(Image1.Canvas.ClipRect);
        end;
end;

```

### La nostra aplicació 'Editor de Text'

28. A la nostra aplicació no cal incloure codi per les accions de tallar, copiar i enganxar, ja que hem utilitzat accions estàndards. Si no haguéssim fet servir accions estàndards el codi a incloure hagués estat:

```

// Acció de tallar cap al portapapers
procedure TForm1.EditarTallarExecute(Sender: TObject);

```

```
begin
  RichEdit1.CutToClipboard;
end;

// Acció de copiar cap al portapapers
procedure TForm1.EditarCopiarExecute(Sender: TObject);
begin
  RichEdit1.CopyToClipboard;
end;

// Acció de pegar des d'el portapapers
procedure TForm1.EditarPegarExecute(Sender: TObject);
begin
  RichEdit1.PasteFromClipboard;
end;
```

## Impressió de dades

### Ús de la impressora

Tenim quatre maneres de gestionar la impressió a la nostra aplicació:

- Si estem fent servir informació procedent de bases de dades, la manera més elegant i còmode de treballar és generar un informe amb els components de la paleta *QuickReport*. Aquest mode de treball el veureu al tema dedicat a les bases de dades.
- Una manera poc corrent de treballar, però de vegades ràpida i efectiva, és imprimir una imatge del contingut de la finestra amb el mètode `Print` de la classe `TForm`. Aquest mètode només imprimeix l'àrea de treball de la finestra, és a dir, la finestra sense la barra de títol ni els marges. La propietat `PrintScale` de la classe `TForm` ens permetrà controlar l'escala en que la finestra s'imprimirà: amb el valor `poNone` dibuixarà un punt d'impressora per cada píxel de pantalla, així que segurament quedarà imprès més petit; amb el valor `poProportional` intentarà que quedi a la mateixa mida que a pantalla; i amb el valor `poPrintToFit` escalarà proporcionalment el dibuix per tal que s'ajusti a les dimensions del paper.

La propietat `PaintTo` de tot control visual també permet imprimir la seva àrea de treball, escrivint la instrucció `NomControl.PaintTo(Printer.Handle, CoordX, CoordY)`.

- Si només hem d'imprimir text, podem emprar la impressora en mode text. Aquest mètode és molt ràpid i no consumeix gaire memòria. Per contra, l'aspecte de la impressió de vegades no és gaire polit. El programa *Bloc de Notes* que s'instal·la amb Windows és un exemple d'aplicació que imprimeix en mode text. Veurem aquest mètode més en profunditat a continuació.
- La manera normal de treballar és emprar la impressora en mode gràfic. Veurem aquest mètode més en profunditat a continuació.

### El component Printer

Totes les opcions d'impressió que veurem deriven, d'una manera o d'una altra, del component `TPrinter`. Per fer servir aquesta funcionalitat a la nostra aplicació no caldrà crear un objecte de la classe `TPrinter`, ja que per defecte ja existeix un objecte anomenat `Printer` que podem emprar. El primer pas, però, que hauré de donar abans de poder emprar cap funció d'impressió, serà afegir el mòdul `Printers` a la clàusula `uses` del nostre programa: `uses ..., Printers;`


A continuació veurem algunes de les propietats més importants de la classe `TPrinter`. Els mètodes més importants els veurem a l'apartat d'impressió en mode gràfic.

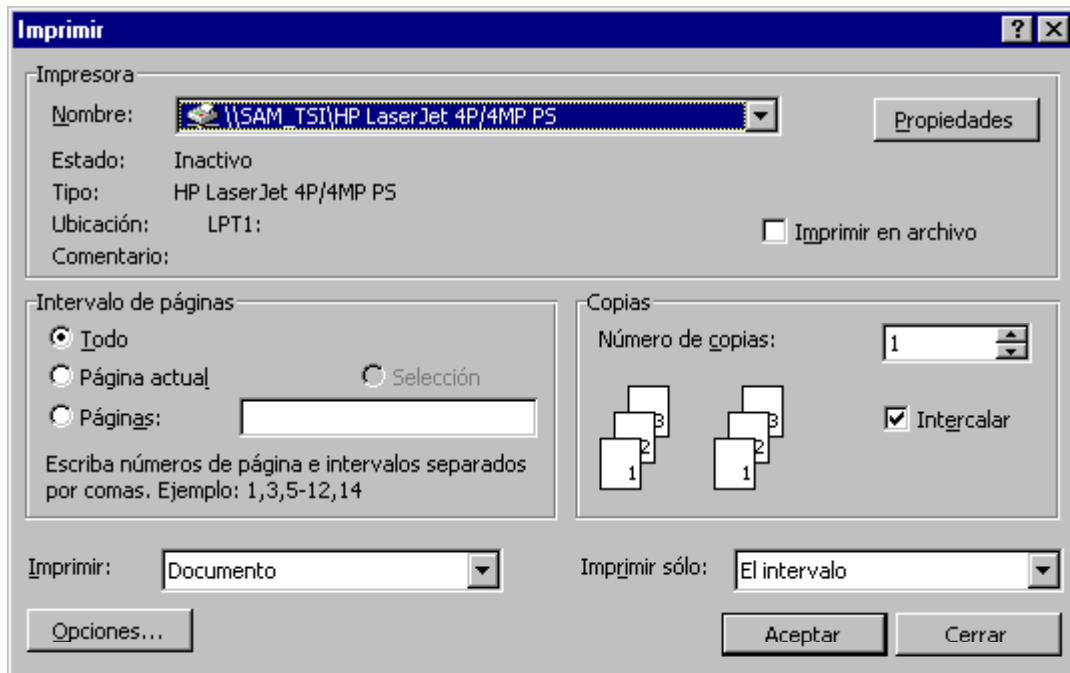
- `Printers` i `PrinterIndex`: La primera és la llista d'impressores del sistema disponibles, mentre que la segona és un nombre enter que denota quina impressora d'aquesta llista tenim seleccionada (-1 per la impressora per defecte, 0 per la primera de la llista, 1 per la segona de la llista, ...).
- `Fonts`: És la llista de fonts tipogràfiques de que disposa la impressora seleccionada.
- `Title`: És el nom que identifica l'actual treball d'impressió dins la cua d'impressió.



- Canvas: Representa la superfície de dibuix de la pàgina a imprimir.
- PageHeight i PageWidth: L'alçada i l'amplada, respectivament, mesurada en punts d'impressora de la pàgina a imprimir. Varien segons la mida i la orientació del paper, i la resolució de la impressora.
- PageNumber: El número de la pàgina que s'està editant actualment per enviar a la cua d'impressió.

### El quadre de diàleg per configurar la impressió


El Component `TPrintDialog`  de la paleta *Dialogs*, en executar-se abans d'iniciar la impressió d'un document, mostra un quadre de diàleg per configurar les opcions de la impressió: seleccionar impressora, nombre de còpies, etc.

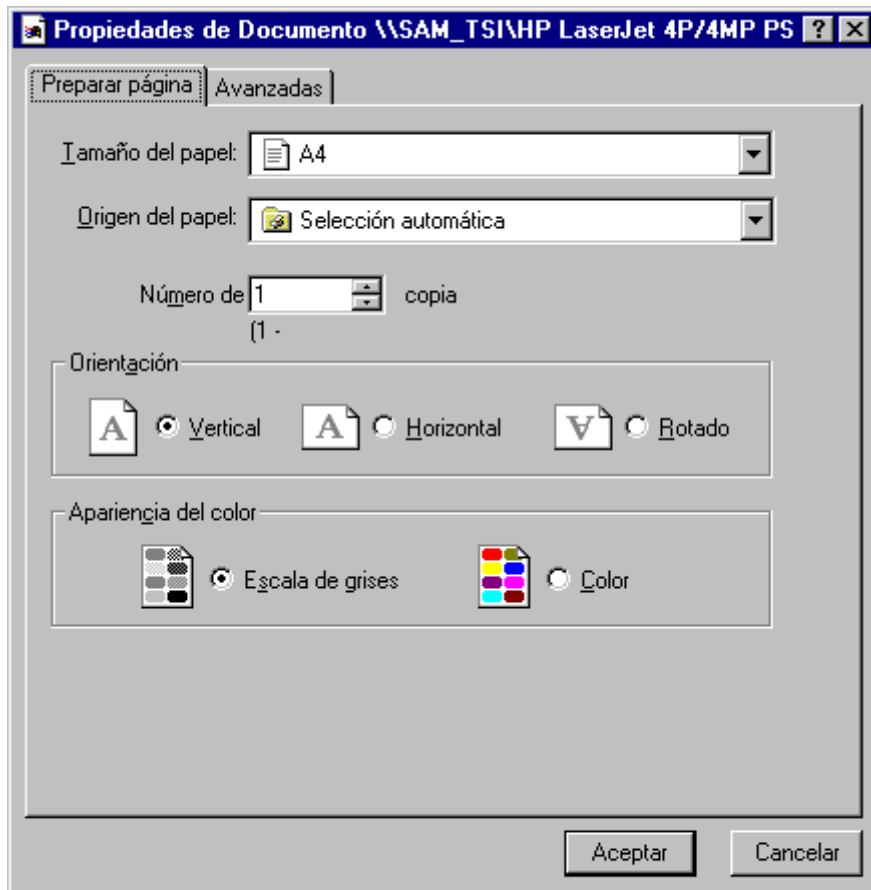


Les propietats més importants de `PrintDialog` són:

- Copies: Especifica el nombre de còpies a imprimir.
- Collate: Especifica si les còpies s'imprimiran intercalades.
- PrintRange: Informa de quina opció tenim seleccionada a l'interval de pàgines a imprimir.
- FromPage i ToPage: Especificquen, respectivament, la pàgina d'inici i final a imprimir, en cas que l'usuari hagi seleccionat que vol imprimir un rang de pàgines.
- Options: És un conjunt de moltes petites opcions booleanes (cert o fals) per controlar quines característiques del quadre de diàleg estan habilitades. És molt interessant mirar-se-les, però aquí no hi ha espai per comentar-les totes. Permeten, per exemple, seleccionar si volem que hi hagi un botó d'ajuda, per visualitzar la opció d'imprimir a un fitxer, etc.

### El quadre de diàleg per configurar la impressora

El Component `TPrinterSetupDialog`  de la paleta *Dialogs* mostra, en executar-se, un quadre de diàleg per configurar les opcions de la impressora seleccionada: dimensions del paper, orientació del paper, qualitat de la impressió, etc. El quadre de diàleg mostrat depèn dels controladors de la impressora i, per tant, variarà d'impressora a impressora.



Com que aquest component no té propietats, l'únic que podem fer amb ell és executar-lo amb la instrucció `PrinterSetupDialog1.Execute`;

### Impressió en mode text

La impressió de text consisteix en treballar amb la impressora com si es tractés d'un fitxer de text obert en mode d'escriptura. Recordeu com funcionaven els fitxers a Pascal?

- Primer haurem de declarar una variable del tipus `TextFile`. Per exemple: `var Impresora: TextFile;`
- Quan vulguem imprimir, haurem d'associar a aquesta variable la impressora seleccionada, mitjançant la instrucció `AssignPrn`. Per exemple: `AssignPrn(Impresora);`
- A continuació toca obrir la impressora amb la instrucció `Rewrite`. Per exemple: `Rewrite(Impresora);`
- Anirem imprimint les diferents cadenes de text amb la instrucció `WriteLn`. Per exemple: `WriteLn(Impresora, Text);`
- Per últim, tancarem la impressora amb la instrucció `CloseFile`. Per exemple: `CloseFile(Impresora);`

Veiem aquí un petit exemple d'impressió en mode text de la informació continguda a un quadre d'edició. Abans d'iniciar la impressió, apareixerà un quadre de diàleg per configurar la impressió, on l'usuari escollirà el nombre de còpies i si vol imprimir tot el text o només el text seleccionat.

```
procedure TForm1.FitxerImprimirExecute(Sender: TObject);  
  
var  
    Impresora: TextFile;  
    i, NumCopies: Integer;  
  
begin  
    // Establim les opcions del PrintDialog
```

```

PrintDialog1.Options := [poSelection];

if PrintDialog1.Execute then
begin

    // Associar la impressora amb la variable
    AssignPrn(Impresora);

    // Establir el tipus de lletra
    Printer.Canvas.Font := Memo1.Font;

    // El títol a la cua d'impressió serà el de la finestra
    Printer.Title := Caption + ' - ' + NomFitxer;

    // Imprimirem tantes còpies com s'hagin indicat
    for NumCopies := 1 to PrintDialog1.Copies do
    begin

        // Iniciar la impressió
        Rewrite(Impresora);

        if PrintDialog1.PrintRange = prSelection then
            // S'ha escollit imprimir el text seleccionat
            WriteLn(Impresora, Memo1.SelText)
        else
            // S'ha escollit imprimir totes les línies
            for i := 0 to Memo1.Lines.Count do
                WriteLn(Impresora, Memo1.Lines[i]);

        // Finalitzar la impressió
        CloseFile(Impresora);
    end;
end;
end;

```

Tanmateix, existeixen dos petits truc per facilitar la impressió en mode text:

- El primer és que el component RichEdit (quadre d'edició per text amb format) disposa d'un mètode Print que li permet imprimir el seu contingut. Per exemple: RichEdit1.Print('títol: llegeix.txt');
- El segon és que podem emprar l'API de Windows per enviar un fitxer de text a la impressora per defecte: ShellExecute(Handle, 'print', NomFitxer, nil, nil, SW\_HIDE);

### Impressió en mode gràfic

La impressió en mode gràfic consisteix en treballar amb el canvas de la impressora, que representa l'àrea imprimible del paper, i anar dibuixant sobre aquest canvas tota la informació que volem imprimir a la pàgina.

- Quan vulguem imprimir, haurem d'indicar que comencem a enviar un document a cua d'impressió, mitjançant el mètode BeginDoc de l'objecte Printer. Per exemple: Printer.BeginDoc;
- A partir d'aquest moment dibuixem sobre la propietat Canvas de l'objecte Printer el que volem imprimir a la pàgina actual. Per exemple: Printer.Canvas.Rectangle(10,10,80,95);
- Per enviar la pàgina, un cop dibuixada, a la cua d'impressió i començar a treballar a una nova pàgina, emparem el mètode NewPage de l'objecte Printer. Per exemple: Printer.NewPage;
- Per últim, indicarem a la cua d'impressió que hem acabat d'enviar el document amb el mètode EndDoc de l'objecte Printer. Per exemple: Printer.EndDoc;
- Si en un moment donat hem de cancel·lar la impressió que havíem iniciat, emparem el mètode Abort de l'objecte Printer, enlloc del mètode EndDoc. Per exemple: Printer.Abort;

Veiem aquí un petit exemple d'impressió en mode gràfic, on s'envia cinc còpies a la cua d'impressió de la impressora seleccionada un document de dues pàgines, que conté una imatge a cada pàgina. Si mentre s'imprimeix l'usuari prem la tecla **ESC**, la impressió s'aturarà.

```
procedure TForm1.FitxerImprimirExecute(Sender: TObject);
var
  i: Integer;
begin
  // Per no haver d'escriure Printer tota l'estona fem servir un with
  with Printer do
    begin
      // El títol que apareixerà a la cua d'impressió
      Title := 'Dues imatges';

      // Imprimirem 5 cops
      for i := 1 to 5 do
        begin
          // Iniciar la impressió
          BeginDoc;

          // Imprimim un mapa de bits (tipus TBitmap)
          Canvas.Draw( (PageWidth - Bitmap1.Width) div 2,
            (PageHeight - Bitmap1.Height) div 2, Bitmap);

          // Passem a una nova pàgina
          NewPage;

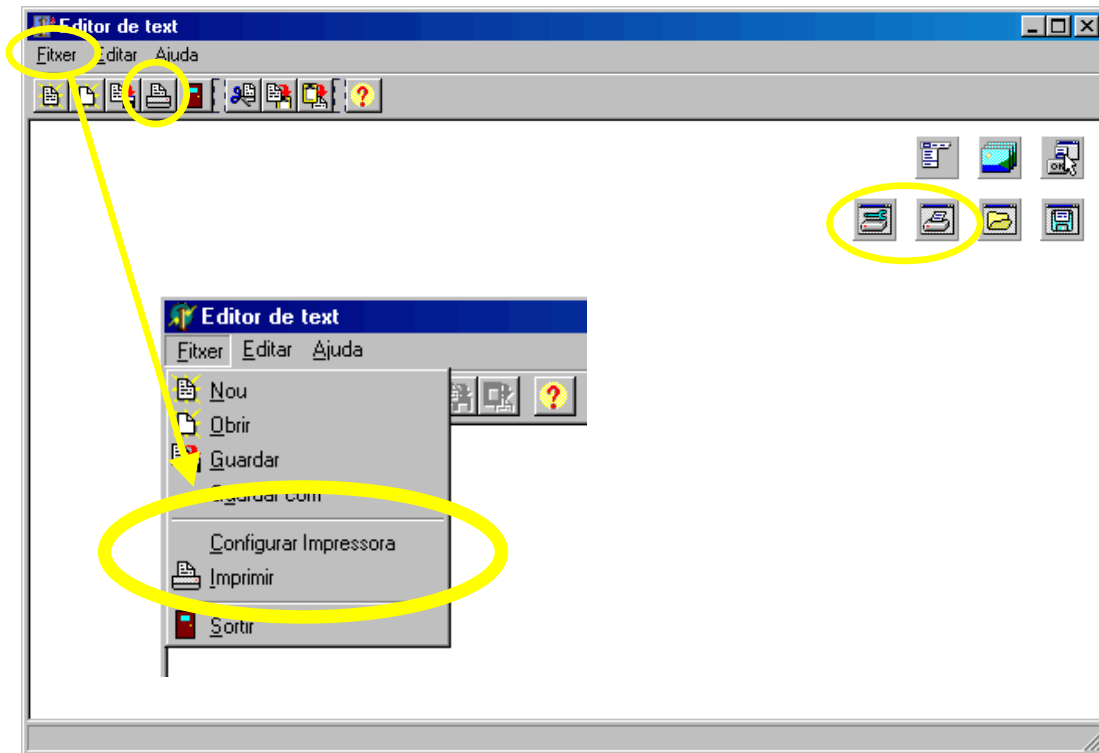
          // Imprimim una imatge (tipus TImage)
          Canvas.CopyRect(Canvas.ClipRect, Image1.Canvas,
            Image1.Canvas.ClipRect);

          // Finalitzar la impressió
          EndDoc;

        end;
      end;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
  if (Key=VK_ESCAPE) and Printer.Printing then
    begin
      Printer.Abort;
      MessageDlg('Impressió avortada', mtInformation, [mbOK], 0);
    end;
end;
```

### ***La nostre aplicació 'Editor de Text'***



Inseriu a la finestra de la nostra aplicació els components `TPrinterSetupDialog` i `TPrintDialog` de la paleta de components *Dialog*.

29. Afegiu la icona d'una impressora al component `TImageList`. Trobareu aquesta icona a la carpeta "C:\Archivos de programa\Archivos comunes\Borland Shared\Images\Buttons\", amb el nom `Print.bmp`.
30. Afegiu les accions `FitxerConfigurarImpressora` i `FitxerImprimir` al component `TActionList`. Ajusteu les propietats de les noves accions.

| Caption                | Category | Hint                     | ImageIndex | Name                       |
|------------------------|----------|--------------------------|------------|----------------------------|
| &Configurar Impressora | Fitxer   | Configurar la impressora | -1         | FitxerConfigurarImpressora |
| &Imprimir              | Fitxer   | Imprimir el fitxer       | 8          | FitxerImprimir             |

Feu doble clic sobre aquestes accions per tal d'escriure el codi a executar:

```

procedure TForm1.FitxerConfigurarImpressoraExecute(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;

procedure TForm1.FitxerImprimirExecute(Sender: TObject);
begin
    if PrintDialog1.Execute then
        RichEdit1.Print('Editor de Text - ' + NomFitxer);
end;

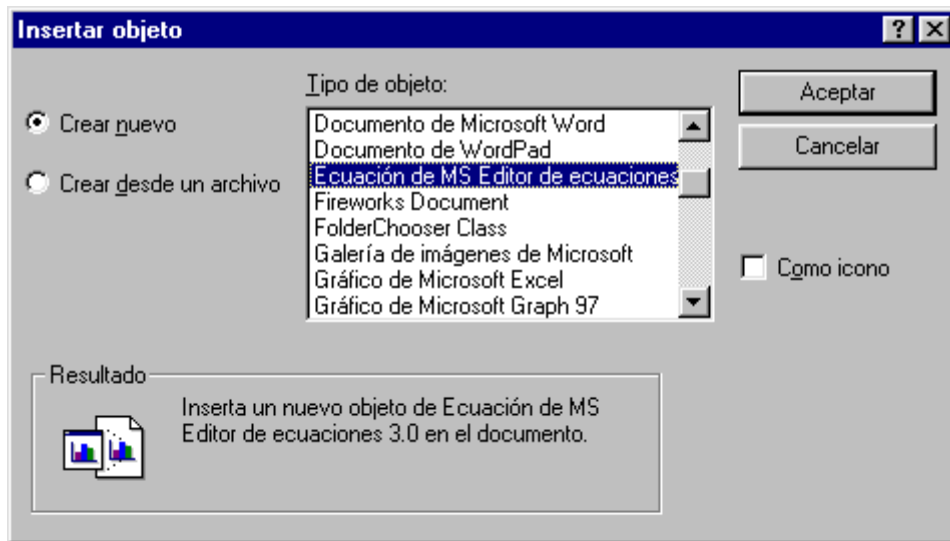
```

31. Afegiu un botó per imprimir al component `TToolBar`, entre els botons de guardar i de sortir. Associeu aquest nou botó a l'acció `FitxerImprimir`.
32. Afegiu una opció per configurar impressora, una opció per imprimir i un separador (caràcter '-' a la propietat `Caption`), al component `TMainMenu`, entre les opcions de guardar i de sortir. Associeu aquestes noves opcions a les accions `FitxerConfigurarImpressora` i `FitxerImprimir`, respectivament.

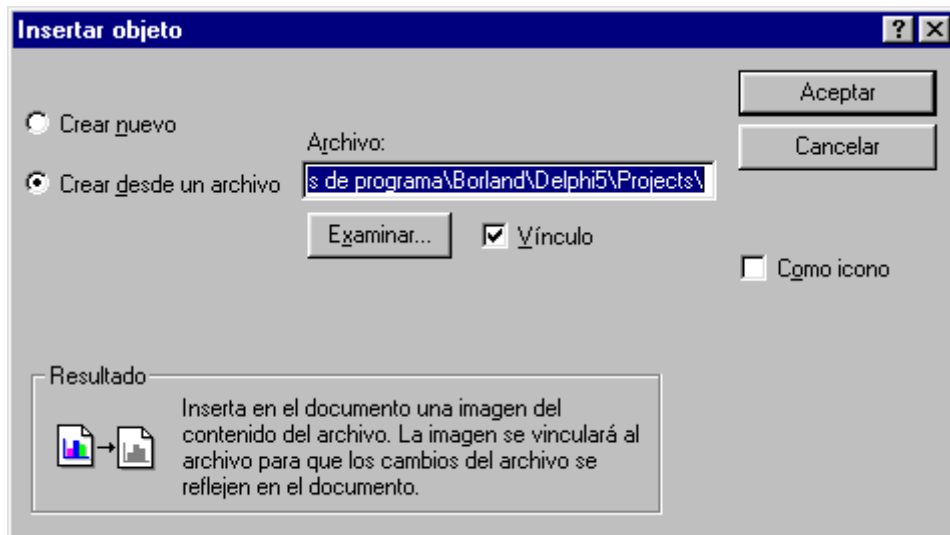
## Enllaç i inserció d'objectes OLE

**OLE** El Component `TOleContainer` permet treballar de la manera més senzilla possible amb la tecnologia OLE 2.0 de Microsoft. Aquesta tecnologia permet inserir dins un document o aplicació objectes que després seran processats per l'aplicació que els va crear. Podeu veure els objectes registrats al vostre sistema si obriu el processador de text *Microsoft Word* i seleccioneu l'opció del menú *Inserir* → *Objecte...* Per exemple: podem inserir un arxiu de so a un document de Word i sentir-lo fent doble clic sobre la seva icona, o inserir una equació a un document de Word i modificar-la fent doble clic sobre la seva imatge.

Un cop inserit un component `TOleContainer` dins el nostre formulari, cal fer doble clic sobre ell per seleccionar a quina aplicació pertany (si creem un objecte OLE nou) o a quin arxiu es troba (si ja existia l'objecte OLE). Si hem creat un objecte OLE nou, caldrà tornar a fer doble clic sobre aquest per obrir la seva aplicació associada i editar-lo.



Si en canvi ja existia un arxiu amb l'objecte, haurem de especificar el camí fins aquest arxiu i decidir si el volem vincular o no. En cas que escollim vincular l'arxiu, l'objecte OLE quedarà enllaçat, i quan modifiquem l'objecte quedarà modificat l'arxiu, i a l'inrevés. En cas que escollim no vincular l'arxiu, l'objecte OLE quedarà encastat, i quan modifiquem l'objecte no es guardaran els canvis a l'arxiu original.

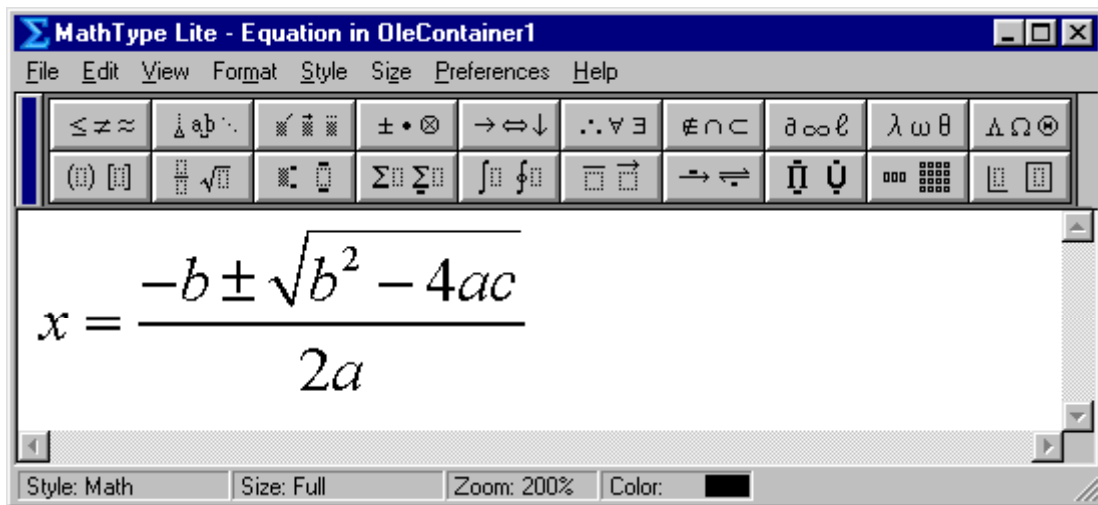
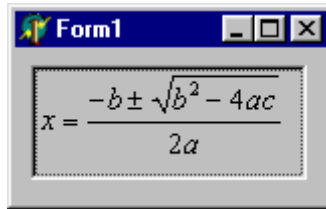


Les propietats més importants d'un objecte de la classe `TOleContainer` són:

- `Linked`: Indica si el fitxer de l'objecte està enllaçat o encastat.
- `Modified`: Indica si l'objecte ha estat modificat.
- `SourceDoc`: Indica el nom i camí del fitxer d'un objecte OLE enllaçat.

- **State:** Descriu l'estat del contenidor OLE (sense objecte OLE, amb objecte OLE i aplicació associada tancada, amb objecte OLE i aplicació associada en funcionament, etc.).

Per exemple, podem inserir dins un formulari un objecte OLE creat per l'editor d'equacions que acompanya a l'editor de text *Microsoft Word*. Quan executem l'aplicació, si fem doble clic sobre aquest objecte OLE a la finestra, s'obrirà l'editor d'equacions, que és l'aplicació que té associada per editar-lo.



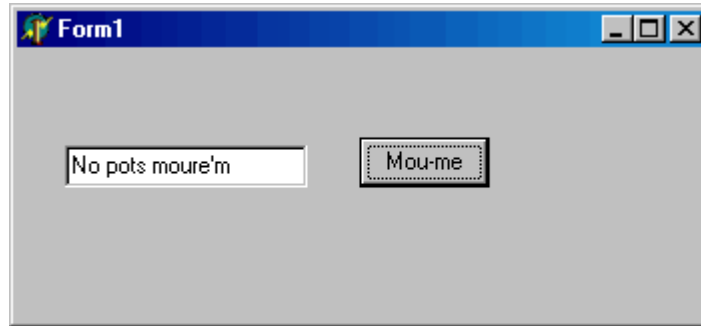
## Ús de arrossegar i deixar anar

### Arrossegar i deixar anar components

No és gaire corrent a una aplicació, però potser un dia ens trobem en el cas que haguem de permetre a l'usuari canviar de posició els controls que resideixen dins la finestra. Per aconseguir aquest efecte haurem de seguir els següents passos:

- Haurem de seleccionar els controls de la finestra que volem que l'usuari pugui moure, i a l'inspector d'objectes canviar el valor de la seva propietat `DragKind` de `dkDock` a `dkDrag` (ja està així per defecte), i també canviar el valor de la seva propietat `DragMode` de `dmManual` a `dmAutomatic`. Opcionalment, podem deixar a la seva propietat `DragCursor` el cursor que volem que aparegui quan el control sigui arrossegat.
- Haurem de seleccionar els contenidors que hauran de rebre els controls quan l'usuari els canviï de lloc (normalment serà el mateix formulari) i escriure el codi al seu esdeveniment `OnDragOver` per indicar si accepta l'objecte arrossegat o no, i al seu esdeveniment `OnDragDrop` per indicar que fer amb l'objecte arrossegat.

### Exemple



Tenim un formulari amb un botó que volem que es pugui arrossegar i deixar anar, i una caixa d'edició que no. Per aconseguir-ho, a l'inspector d'objectes posem la propietat `DragMode` del botó i la caixa d'edició al valor `dmAutomatic`. A continuació, el codi a escriure serà:

```
// Indiquem al formulari que pot acceptar el botó
procedure TForm1.FormDragOver(Sender, Source: TObject; X, Y: Integer;
                               State: TDragState; var Accept: Boolean);
begin
    Accept := Source is TButton;
end;

// Indiquem al formulari que ha de fer quan deixin anar el botó
procedure TForm1.FormDragDrop(Sender, Source: TObject; X, Y: Integer);
begin
    (Source as TButton).Left := X;
    (Source as TButton).Top := Y;
end;
```

Quan s'executi el programa, l'usuari podrà intentar arrossegar tots dos controls, però només el botó serà acceptat pel formulari i canviarà de lloc.

### Arrossegar i deixar anar des de l'explorador de Windows

Molt més freqüent és el cas en que l'usuari arrossegui un fitxer des de l'escriptori o l'explorador de Windows fins a la finestra de la nostra aplicació, per tal que la nostra aplicació obri aquest fitxer i processi la informació que conté.

No existeix a Delphi cap esdeveniment per tractar aquesta acció de l'usuari, així que l'haurèm de crear nosaltres seguint els següents passos:

- A l'esdeveniment `onCreate` del formulari, cal escriure la línia de codi `DragAcceptFiles(Handle, True)` per indicar-li que accepta que l'usuari arrossegui fitxers des de l'explorador fins a la finestra. Per emprar aquesta instrucció de l'API de Windows, cal afegir a la clàusula `uses` la unitat `ShellAPI`: `uses ..., ShellAPI;`
- Quan l'usuari arrossegui un fitxer de l'explorador a la nostra finestra, Windows enviarà un missatge `WM_DROPFILES` a la nostra finestra, amb informació sobre el fitxer arrossegat. Si a la nostra classe finestra definim un mètode que gestioni aquest missatge, ja haurèm creat l'esdeveniment necessari. Això es fa de la següent manera: a la part de declaracions privades de la nostra classe finestra escrivim la capçalera del mètode

```
| Procedure NomMetode(var Missatge: TMessage); message WM_DROPFILES;
```

i més a baix, a la part d'implementació, escrivim el codi a executar per aquest mètode. Dins aquest codi podem emprar la funció `DragQueryFile` per obtenir el nombre d'arxius arrossegats i el seu nom, i la funció `DragFinish` per alliberar la informació passada pel missatge. Per exemple:

```
// En rebre el missatge WM_DROPFILES
procedure TForm1.NomMetode;
var
    NomArxiu: array [0..255] of char;
    i: integer;
begin
    // Obtenim el nombre d'arxius
```



```

for i := 0 to DragQueryFile(Missatge.WParam, $FFFFFFFF, nil, 0) - 1 do
begin
  // Obtenim el nom i camí de l'arxiu número 'i'
  DragQueryFile(Missatge.WParam, i, NomArxiu, 256);

  // Comprovem si l'extensió de l'arxiu és l'esperada
  if LowerCase(ExtractFileExt(StrPas(NomArxiu))) = '.xxx' then
    ...
  else
    ...
end;

// Alliberem el bloc de dades
DragFinish(Missatge.WParam);
end;

```

### La nostra aplicació 'Editor de Text'

33. Per tal que l'editor de text obri automàticament fitxers amb extensió *.txt* en arrossegar-los sobre la seva finestra principal, el codi a incloure (text en cursiva) fora el següent:

```

uses ... , ShellAPI;

type
  TForm1 = class(TForm)
    ...
  private
    // Mètode per gestionar el missatge generat en deixar anar els arxius
    procedure WMDropFiles(var Missatge: TMessage); message WM_DROPFILES;
    ...
  end;

// En crear-se el formulari
procedure TForm1.FormCreate(Sender: TObject);
begin
  DragAcceptFiles(Handle, True);
  ...
end;

// En rebre el missatge WM_DROPFILES
procedure TForm1.WMDropFiles;
var NomArxiu: array [0..255] of char;
begin
  // Obtenim el nom del primer arxiu
  DragQueryFile(Missatge.WParam, 0, NomArxiu, 256);

  // Alliberem el bloc de dades
  DragFinish(Missatge.WParam);

  // Si es tracta d'un arxiu de text, carreguem-lo
  if LowerCase(ExtractFileExt(StrPas(NomArxiu))) = '.txt' then
    begin
      RichEdit1.Lines.LoadFromFile(NomArxiu);
      NomFitxer := StrPas(NomArxiu);
      StatusBar1.SimpleText := NomFitxer;
    end
  else
    ShowMessage('Tipus d''arxiu desconegut');
end;

```

## Ús del registre

### Introducció al registre de Windows

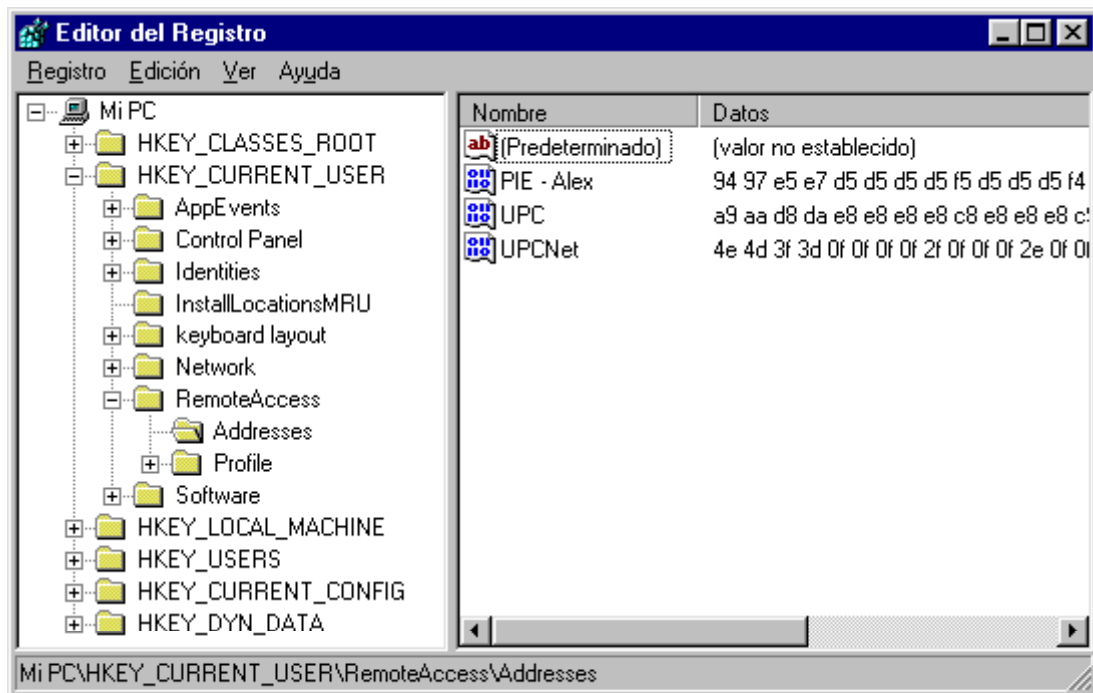
A les primeres versions de Windows (3.x i anteriors) la configuració del hardware, del sistema operatiu i dels diferents programes es guardava a arxius individuals amb extensió *.ini*. A les següents versions de

Windows (9x i NT) s'utilitza una base de dades unificada i centralitzada, anomenada el registre de configuració, per guardar de forma jeràrquica tota aquesta informació. Com a avantatges més bàsics d'aquest nou sistema tenim:

- La informació de configuració està centralitzada, enlloc d'en múltiples fitxers localitzats per tot el disc.
- Permet accedir local o remotament a aquesta informació, tot aplicant unes normes de seguretat d'accés.
- En sistemes multiusuari (Windows NT) permet registrar informació dels usuaris individuals.

Encara i així, els fitxers *.ini* continuen existint per mantenir la compatibilitat amb les aplicacions de 16 bits.

Una imatge val més que mil paraules, així que fem una ullada al registre mitjançant una eina que proporciona el mateix sistema: l'editor del registre. Aquesta eina apareixerà quan escrivim *regedit* o *regedit32* a la línia de comandes de Windows (*Menú Inici → Executar → Regedit*).



Com veiem, la informació està estructurada en forma d'arbre, com si de l'explorador de Windows es tractés. Dins de cada carpeta (claus) ens podem trobar noves carpetes (subclaus) o bé valors amb les seves dades corresponents. Aquestes dades poden ser de tipus binari, enters de quatre bytes, variables i text. Amb l'editor del registre podem crear i esborrar claus, crear i esborrar valors, modificar dades associades a valors i realitzar cerques dins el registre. Ha de quedar clar, però, que sempre que realitzem canvis al registre hem de 'refrescar-lo' prement la tecla **F5** per tal que aquests canvis tinguin lloc. Si això no funciona, haurem de reiniciar el sistema.

Una altra manera que tenim de modificar el registre es afegir la informació a un fitxer de text amb extensió *.reg*, que s'executarà en fer doble clic sobre ell o mitjançant la instrucció 'regedit /s fitxer.reg'. Aquesta última manera es especialment interessant de cara a afegir la informació automàticament cada cop que es reiniciï la màquina.

El fitxer *.reg* ha de tenir la següent estructura:

```
REGEDIT4

; comentari          +-
[Aquí el nom d'una clau] |
"valor"="dada"        +-> Aquestes línies es van repetint
"valor"="dada"        |
...                  +-
```

```
; comentari          -+
[-Nom de clau a esborrar]  +-> Aquestes línies es van repetint
...                    -+
```

El registre a Windows NT està format per cinc arbres principals.

- HKEY\_LOCAL\_MACHINE. Conté la informació del sistema de l'equip local, tant a nivell software com a nivell hardware: controladors de dispositius instal·lats i serveis del sistema (subarbre SYSTEM), hardware instal·lat (subarbre HARDWARE), software instal·lat (subarbre SOFTWARE), base de dades del sistema de seguretat (subarbre SECURITY) i comptes de grups i usuaris locals (subarbre SAM).
- HKEY\_CLASSES\_ROOT. Conté l'associació entre fitxers i objectes OLE. En realitat es un apuntador a l'arbre HKEY\_LOCAL\_MACHINE\Software\Classes.
- HKEY\_USERS. Conté dades de l'entorn de l'usuari actualment identificat a la màquina i de l'usuari per defecte, però no dels usuaris que han accedit remotament a la màquina. Hi han dades per defecte (subarbre DEFAULT) i dades associades a identificadors d'usuaris (subarbres SID).
- HKEY\_CURRENT\_USER. Conté dades de l'entorn de l'usuari actualment en sessió a la màquina. En realitat és un apuntador a l'arbre HKEY\_USERS\SID\_usuari.
- HKEY\_CURRENT\_CONFIG. Conté la informació de la configuració actual del hardware. En realitat és un apuntador o drecera a l'arbre HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Hardware\Profiles\Current.

La informació d'aquests arbres es guarda als fitxers dels directoris "c:\winnt\system32\config" i "c:\winnt\profiles\nom\_usuari", o allà on s'hagi instal·lat Windows NT (ho indica la variable d'entorn %systemroot%).

El registre a Windows 9x està format per sis arbres principals:

- HKEY\_LOCAL\_MACHINE. Conté la informació del sistema de l'equip local, tant a nivell software com a nivell hardware: dades de diferents configuracions de hardware (subarbre CONFIG), dades dels dispositius instal·lats (subarbre ENUM), controladors de dispositius instal·lats i serveis del sistema (subarbre SYSTEM), dades del microprocessador i ports sèrie (subarbre HARDWARE), software instal·lat (subarbre SOFTWARE), dades de xarxa (subarbre NETWORK) i dades de seguretat de xarxa (subarbre SECURITY).
- HKEY\_CLASSES\_ROOT. Conté l'associació entre fitxers i objectes OLE. En realitat es un apuntador a l'arbre HKEY\_LOCAL\_MACHINE\Software\Classes.
- HKEY\_USERS. Conté dades de l'entorn dels usuaris de la màquina. Hi han dades de l'usuari per defecte (subarbre .DEFAULT) i preferències associades a cada usuari (un subarbre per a cada usuari). Dins aquests subarbres d'usuari trobem els fitxers de so associats a esdeveniments (subarbre APPEVENT), dades del pannel de control (subarbre CONTROL PANEL), la disposició del teclat (subarbre KEYBOARD LAYOUT), accessos a xarxa (subarbre NETWORK), informació d'accés remot (subarbre REMOTE ACCESS) i les preferències de cada programa (subarbre SOFTWARE).
- HKEY\_CURRENT\_USER. Conté dades de l'entorn de l'usuari actualment en sessió a la màquina. En realitat es un apuntador a l'arbre HKEY\_USERS\Default o HKEY\_USERS\Clau\_usuari.
- HKEY\_CURRENT\_CONFIG. Conté la informació de la configuració actual del hardware. En realitat es un apuntador a l'arbre HKEY\_LOCAL\_MACHINE\Config.
- HKEY\_DYN\_DATA. Conté part d'informació del registre que, degut a que aquesta canvia freqüentment, es manté a memòria en lloc del disc dur (caché). Per exemple: els codis de problemes de hardware (subarbre CONFIG MANAGER) i dades de rendiment del sistema i la xarxa (subarbre PERFSTATS).

La informació d'aquests arbres es guarda als fitxers "c:\windows\user.dat" i "c:\windows\system.dat", o allà on s'hagi instal·lat Windows (ho indica la variable d'entorn %windir%).

### Els components *TRegistry* i *TRegistryIniFile*

L'API de Windows proporciona algunes funcions per a la manipulació del registre: *RegCreateKey*, *RegOpenKey*, *RegQueryValue*, *RegSetValue*, *RegDeleteKey*, etc. Treballar amb el registre a nivell d'API és bastant tediós. La biblioteca de components VCL proporciona dos components que encapsulen les operacions amb el registre: *TRegistry* i *TRegistryIniFile*. Per poder emprar-los, cal afegir *Registry* a la clàusula *uses* de la unitat: **uses ... , Registry;**

El component *TRegistry* encapsula l'accés al registre de Windows a la nostra aplicació. La seva propietat *RootKey* conté l'arbre principal, és a dir, la jerarquia de subarbres als que l'aplicació podrà accedir. Quan es crea un objecte de la classe *TRegistry*, aquesta propietat val per defecte *HKEY\_CURRENT\_USER*, que en un principi és on una aplicació hauria de guardar les seves dades. Un objecte de la classe *TRegistry* només pot accedir a una clau alhora, i aquesta clau ve indicada per la seva propietat *CurrentKey*. Els mètodes d'aquest component permeten a l'aplicació obrir, tancar, guardar, moure, copiar i esborra claus, preguntar si una clau conté dades, llegir les dades d'una clau i escriure dades a una clau. Com a exemple, veiem com utilitzar aquest component per associar extensions de fitxer a una aplicació.

```
uses ... , Registry, ShlObj;

procedure RegisterFileType(Ext, Key, Descr, Icon, App: string);
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;
  try
    Reg.RootKey := HKEY_CLASSES_ROOT;

    Reg.OpenKey(Ext, True);
    if Key = '' then
      Key := Copy(Ext, 2, Length(Ext)-1) + '_auto_file';
    Reg.WriteString('', Key);
    Reg.CloseKey;

    Reg.OpenKey(Key, True);
    Reg.WriteString('', Descr);
    Reg.CloseKey;

    if Icon <> '' then
      begin
        Reg.OpenKey(Key + '\DefaultIcon', True);
        Reg.WriteString('', Icon);
        Reg.CloseKey;
      end;

    Reg.OpenKey(Key + '\shell\open\command', True);
    Reg.WriteString('', App+' "%1"');
    Reg.CloseKey;
  finally
    Reg.Free;
  end;
  SHChangeNotify(SHCNE_ASSOCCHANGED, SHCNF_IDLIST, nil, nil);
end;

begin
  RegisterFileType('.txt', 'txt_auto_file', 'Fitxer de text',
    Application.ExeName+',0', Application.ExeName);
end.
```

El component *TRegistryIniFile* presenta una manera més senzilla d'accedir al registre del sistema i amaga la necessitat de conèixer la estructura interna del registre. Aquest component permet manipular el registre com si es tractés d'un fitxer INI de Windows 3.x. Això fa que podem adaptar amb un mínim de canvis aplicacions que treballaven amb fitxers INI per tal que treballin amb el registre. Com a exemple, veiem com utilitzar aquest component per associar extensions de fitxer a una aplicació.

```
uses ... , Registry, ShlObj;
```

```

procedure RegisterFileType(Ext, Key, Descr, Icon, App: string);
var
    Reg: TRegistryIniFile;
begin
    try
        Reg := TRegistryIniFile.Create('');
        Reg.RegIniFile.RootKey := HKEY_CLASSES_ROOT;
        if Key = '' then
            Key := Copy(Ext, 2, Length(Ext)-1) + '_auto_file';
        Reg.WriteString(Ext, '', Key);
        Reg.WriteString(Key, '', Descr);
        if Icon <> '' then
            Reg.WriteString(Key + '\DefaultIcon', '', Icon);
            Reg.WriteString(Key + '\shell\open\command', '', App+' "%1"');
        finally
            Reg.Free;
        end;
        SHChangeNotify(SHCNE_ASSOCCHANGED, SHCNF_IDLIST, nil, nil);
    end;

begin
    RegisterFileType('.txt', 'txt_auto_file', 'Fitxer de text',
        Application.ExeName+'.0', Application.ExeName);
end.

```

## Ús de varies finestres

Pot ser que la interfície gràfica de la nostra aplicació estigui formada per vàries finestres. En aquest cas cal tenir les idees ben clares i parar sempre a pensar quin serà el codi més adient, per tal de no trobar-nos amb problemes més endavant.

El codi principal de l'aplicació és al fitxer de projecte (extensió *.dpr*). Un exemple típic de fitxer de projecte amb quatre finestres seria:

```

program Project1;

uses Forms, Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2},
    Unit3 in 'Unit3.pas' {Form3}, Unit4 in 'Unit4.pas' {Form4};

{$R *.RES}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.CreateForm(TForm2, Form2);
    Application.CreateForm(TForm3, Form3);
    Application.CreateForm(TForm4, Form4);
    Application.Run;
end.

```

L'aplicació primer executarà el codi de la secció d'inicialització de les diferents unitats de la clàusula *uses*. A continuació executarà el mètode *Initialize*, que normalment no fa res.

Cada mètode *CreateForm* crea una finestra, és a dir, crea a memòria l'estructura per un objecte d'una classe derivada de *TForm*, i executa el codi del seu esdeveniment *onCreate*, però no mostra per pantalla aquesta finestra. La primera finestra creada amb el mètode *CreateForm* l'anomenem finestra principal de l'aplicació.

Per últim, el mètode *Run* mostra per pantalla la finestra principal de l'aplicació. Només quan aquesta finestra es tanqui finalitzarà l'aplicació.

### Mostrar i amagar finestres

A qualsevol part del codi, podem mostrar les finestres mitjançant els mètodes *Show* i *ShowModal* de que aquestes disposen. La diferència entre aquests dos mètodes rau en que *Show* mostra una finestra permetent accedir a la resta de finestres de l'aplicació que continuen obertes, mentre que amb

ShowModal s'ha de tancar la finestra que s'ha mostrat per poder tornar a accedir a la resta de finestres de l'aplicació. ShowModal s'acostuma a utilitzar per quadres de diàleg, mentre que Show s'acostuma a utilitzar per finestres de treball.

Tota finestra disposa també d'un mètode Hide per amagar-se. És millor mostrar i amagar una finestra cridant a aquests mètodes que jugant amb la seva propietat Visible.

També cal tenir en compte que quan una finestra que no es la finestra principal de l'aplicació es tanca, realment no es destrueix sinó que només s'amaga, i no s'allibera la memòria que aquesta finestra ocupa fins que acaba l'aplicació. Si voleu canviar aquest comportament haureu de donar el valor caFree al paràmetre de sortida Action de l'esdeveniment onClose del formulari. Per exemple:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  ...
  Action := caFree
end;
```

De totes maneres, a l'hora de dissenyar la interfície gràfica de la vostra aplicació, recordeu que les aplicacions més elegants només tenen una finestra amb tota l'àrea de treball, i que l'intercanvi de dades addicional amb l'usuari es produeix a través de quadres de diàleg. Tenir varies finestres obertes alhora és molest i dificulta el treball de l'usuari.

### SplashScreen

S'anomenen així les pantalles de presentació de les aplicacions. Sí, el dibuix al mig de la pantalla que apareix per distreure'ns l'estona que tarda una aplicació en carregar-se, és a dir, en crear les estructures de les seves finestres, reservar memòria dinàmica i inicialitzar les variables.



Exemple: pantalla de presentació que apareix quan es carrega Delphi 5.

Quin codi hem d'afegir a la nostra aplicació per tal que aparegui una finestra a l'inici? Observem el codi de l'anterior fitxer de projecte i pensem. Volem una finestra que es creï la primera, però que no sigui la finestra principal de l'aplicació. Només hi ha una solució: no crear-la amb el mètode CreateForm, sinó amb el mètode Create que té tota classe. Per exemple, el següent codi mostra una pantalla de presentació amb una barra de progrés mentre l'aplicació crea quatre formularis:

```
program Project1;
uses Forms, Unit5 in 'Unit5.pas' {FormSplash},
    Unit1 in 'Unit1.pas' {Form1}, Unit2 in 'Unit2.pas' {Form2},
```

```
Unit3 in 'Unit3.pas' {Form3}, Unit4 in 'Unit4.pas' {Form4};  
{ $R *.RES }  
begin  
Application.Initialize;  
FormSplash := TFormSplash.Create(nil);  
try  
FormSplash.ProgressBar1.Max := 100;  
FormSplash.Show;  
  
FormSplash.Label1.Caption := 'Creant la primera finestra';  
FormSplash.Update;  
Application.CreateForm(TForm1, Form1);  
FormSplash.ProgressBar1.StepBy(25);  
  
FormSplash.Label1.Caption := 'Creant la segona finestra';  
FormSplash.Update;  
Application.CreateForm(TForm2, Form2);  
FormSplash.ProgressBar1.StepBy(25);  
  
FormSplash.Label1.Caption := 'Creant la tercera finestra';  
FormSplash.Update;  
Application.CreateForm(TForm3, Form3);  
FormSplash.ProgressBar1.StepBy(25);  
  
FormSplash.Label1.Caption := 'Creant la quarta finestra';  
FormSplash.Update;  
Application.CreateForm(TForm4, Form4);  
FormSplash.ProgressBar1.StepBy(25);  
finally  
FormSplash.Free;  
end;  
Application.Run;  
end.
```



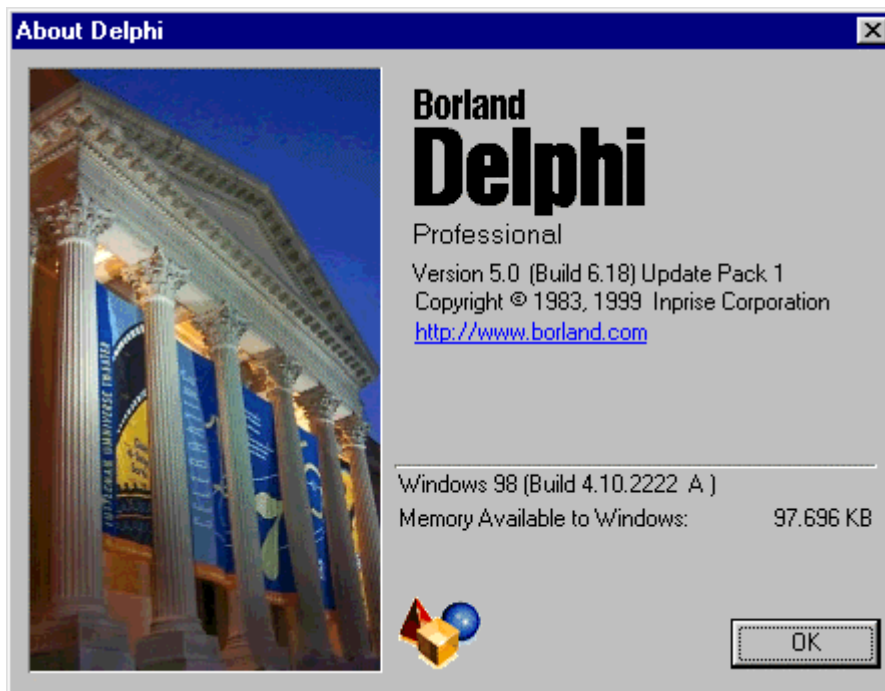
Penseu que una pantalla de presentació potser no té gaire sentit si l'aplicació no tarda en carregar-se. Si encara i així volem una pantalla de presentació que es vegi durant uns instants en iniciar l'aplicació, haurem d'afegir un component `Timer` (cronòmetre) per comptar aquest temps.

Un últim detall: per tal que la finestra aparegui centrada a pantalla haurem d'assignar el valor `poScreenCenter` a la seva propietat `Position`, i per tal que la finestra aparegui sense marges ni barra de títol haurem d'assignar el valor `bsNone` a la seva propietat `BorderStyle`.

### AboutBox

S'anomenen així les finestres que contenen breu informació sobre la gent que va elaborar l'aplicació: logotip de l'empresa, pàgina web, telèfon, versió de l'aplicació, etc.

Normalment es criden des d'una opció del menú de la finestra principal (*Ajuda* → *A propòsit de ...*).



Exemple: finestra d'informació de Delphi 5.

Si volem que la nostra aplicació disposi d'una *AboutBox* haurem d'inserir un nou formulari amb l'opció del menú *File* → *New Form*, o inserir un formulari *AboutBox* ja mig construït amb l'opció del menú *File* → *New...* → pestanya *Forms* → *About box*. Modifiquem llavors aquesta finestra a voluntat, fins donar-li l'aspecte desitjat.

A continuació, a la clàusula *uses* de la finestra principal afegim el nom de la unitat de l'*AboutBox*. Per mostrar aquesta *AboutBox* per pantalla, haurem d'escriure a alguna part del codi de la finestra principal *NomAboutBox.ShowModal*.

### **Vàries finestres del mateix tipus**

Quan elaborem una aplicació amb Delphi, per a cada tipus de finestra de l'aplicació creem una nova classe derivada de *TForm*, i que té com a atributs tots els components que anem arrossegant sobre la finestra. Generalment, després només treballarem amb una finestra d'aquell tipus, però si volguéssim podríem treballar amb vàries finestres del mateix tipus, de la mateixa manera que podem tenir varis objectes d'una mateixa classe i vàries variables d'un mateix tipus de dades.

Aconseguir-ho no es gens difícil. Podem tenir codi com aquest:

```
// Crea tres instàncies de finestres del mateix tipus
var FormTauler1, FormTauler2, FormTauler3: TFormTauler;
begin
  Application.Initialize;
  Application.CreateForm(TFormTauler, FormTauler1);
  Application.CreateForm(TFormTauler, FormTauler2);
  Application.CreateForm(TFormTauler, FormTauler3);
  Application.Run;
end.
```

o com aquest:

```
// Crea una instància d'una finestra cada cop que es prem un botó
procedure TFormPrincipal.Button1Click(Sender: TObject);
begin
  with TFormTauler.Create(Self) do Show;
end;
```

Però per tal que funcioni bé cal tenir en compte dues coses molt importants:



- Quan declarem una classe, mai hem d'escriure al codi de la classe referències a objectes d'aquesta classe. Ho veurem millor amb un exemple:

```

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin
  Form1.Caption := 'hola';      // MAI!!!!   Form1
end;

```

Si volguéssim crear un segon objecte de la classe TForm1 anomenat Form2, funcionaria aquest correctament? Què faria el codi de l'anterior esdeveniment a l'objecte Form2? Molt millor sense la referència a Form1:

```

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var Form1, Form2: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin
  Caption := 'hola';      // Ara funciona per qualsevol finestra.
end;

```

I encara millor si els objectes d'una mateixa classe no es declaren a la unitat que defineix la classe, sinó al fitxer que crida aquesta unitat. En el nostre exemple, la declaració `var Form1, Form2: TForm1;` és preferible que es trobi al fitxer de projecte que no pas al de la unitat de la finestra, encara que Delphi per defecte crea una declaració a la unitat.

- Si tenim informació addicional necessària pel funcionament de la finestra, penseu si la voleu com a variable global (en aquest cas hi haurà una variable per a la unitat i totes les finestres d'aquella classe compartiran la mateixa informació):

```

type
  TFormDocument = class(TForm)
    ...
  private
    ...
  public
    ...
  end;

var
  Form1, Form2: TFormDocument;
  NomFitxer: String; // Un únic NomFitxer que empraran totes les finestres

```

O com atribut de la finestra (cada finestra d'aquella classe tindrà el seu propi atribut amb la informació):

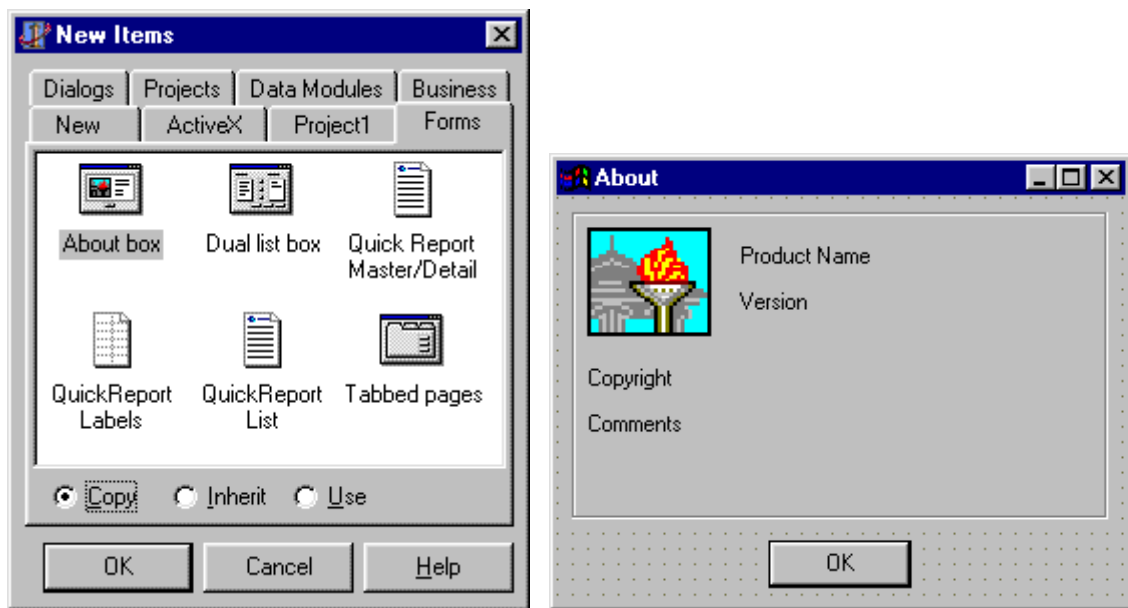
```
type
  TFormDocument = class(TForm)
    NomFitxer: String; // Un NomFitxer diferent per a cada finestra
    ...
  private
    ...
  public
    ...
  end;

var
  Form1, Form2: TFormDocument;
```

### La nostre aplicació 'Editor de Text'

Anem a incloure al nostre editor de text una caixa per visualitzar la informació del producte: nom, versió, logotip, el nostre nom, la nostre pàgina web, i d'altre informació general.

34. Al menú de Delphi, escolliu inserir un formulari *AboutBox* mig construït amb l'opció *File* → *New...* → pestanya *Forms* → *About box*.



35. Modifiqueu la propietat *Caption* de les seves etiquetes, la propietat *Caption* del formulari i la propietat *Picture* de la imatge fins a donar-li l'aspecte desitjat.



36. Si voleu que alguna de les etiquetes sigui un enllaç a una pàgina web, haureu de seleccionar la etiqueta i fer el següent: modifiqueu la seva propietat *Font* per tal que sembli un enllaç a Internet

(normalment blau i subratllat), modifiqueu la seva propietat `Cursor` per tal que aparegui un dit (valor `crHandPoint`) enlloc d'una fletxa (valor `crDefault`), i al seu esdeveniment `onClic` escriviu:

```
procedure TAboutBox.LabelWebClick(Sender: TObject);
begin
    ShellExecute(Handle, 'open', 'http://www.lsi.upc.es/~acastan/',
        nil, nil, SW_SHOWNORMAL);
end;
```

Caldrà, però, que canvieu el tercer paràmetre de la instrucció `ShellExecute` per la vostra pàgina web desitjada. Per poder emprar la instrucció `ShellExecute`, haureu de afegir la paraula `ShellAPI` a la clàusula `uses` de l'*AboutBox*:

```
uses
    Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
    Buttons, ExtCtrls, ShellAPI;
```

37. Escolliu *File* → *Save as...* per guardar la unitat de l'*AboutBox* a disc amb el nom *About.pas*. Ara, a l'editor de codi de Delphi hauríeu de tenir dues pestanyes, una per la unitat *Unit1* i una altra per la unitat *About*. Seleccioneu la pestanya *Unit1*. A la clàusula `uses` d'aquesta unitat afegim l'*AboutBox*. Hauria de quedar:

```
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    ActnList, ImgList, ComCtrls, StdCtrls, StdActns, Menus, ToolWin, About;
```

38. Afegiu l'acció *AjudaAProposit* al component *TActionList*. Ajusteu les propietats de la nova acció.

| Caption     | Category | Hint | ImageIndex | Name           |
|-------------|----------|------|------------|----------------|
| &A Proposit | Ajuda    |      | -1         | AjudaAProposit |

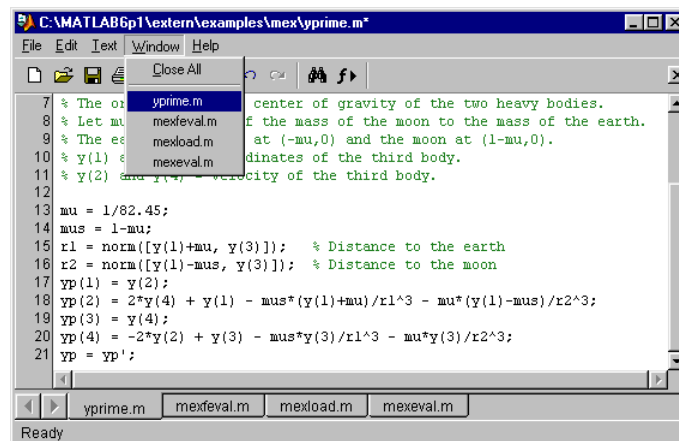
Feu doble clic sobre aquesta acció per tal d'escriure el codi a executar:

```
procedure TForm1.AjudaAPropositExecute(Sender: TObject);
begin
    AboutBox.ShowModal;
end;
```

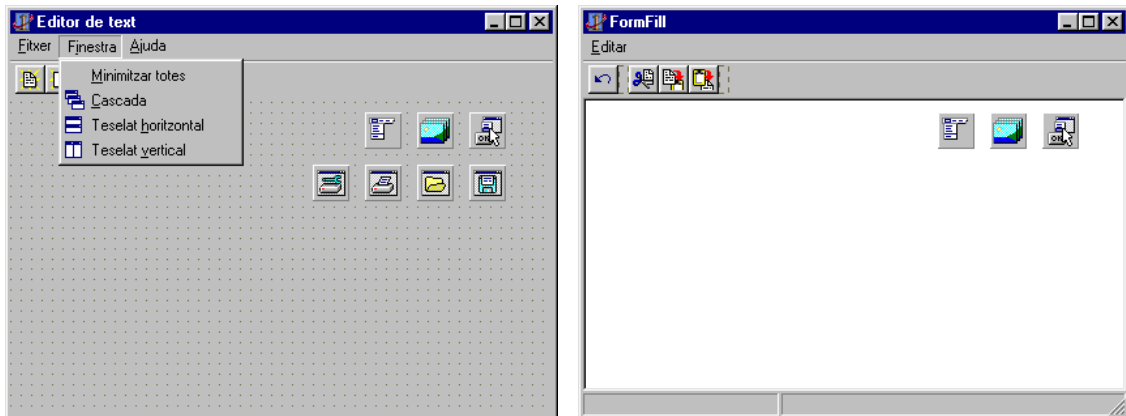
39. Al component *TMainMenu*, afegiu un separador (caràcter '-' a la propietat `Caption`) i una opció per visualitzar l'*AboutBox*, després de les opcions de contingut i índex del menú d'ajuda. Associeu aquesta nova opció a l'acció *AjudaAProposit* que acabeu de crear.

## Treballar amb varis documents alhora – Aplicacions MDI

Tots i totes hem treballat algun cop amb aplicacions que poden tenir oberts dintre seu varis documents alhora. Normalment, aquestes aplicacions permeten a l'usuari escollir amb quin document dels oberts treballarà mitjançant una opció del menú principal, encara que podem trobar altres aplicacions que permetin accedir mitjançant pestanyes i d'altres elements gràfics.



Per crear una aplicació multidocument (MDI a partir d'ara) necessitarem crear dos tipus de finestres: un tipus de finestra que contindrà dintre seu els documents, i que a partir d'ara anomenarem finestra pare; i un altre tipus de finestra que serà la plantilla per crear els documents, i que a partir d'ara anomenarem finestra fill.



Per indicar que la finestra que dissenyem és una finestra pare, a la seva propietat `FormStyle` escollirem el valor `fsMDIForm` (formulari principal - contenidor), mentre que si és una finestra filla escollirem el valor `fsMDIChild` (formulari document - plantilla).

Amb la finestra pare treballarem com normalment hem treballat amb tota finestra: es crearà com sempre al fitxer de projecte amb la instrucció `CreateForm`, etc. En canvi, les finestres filles es crearan i destruiran en temps d'execució, ja que no sabem a priori quants documents voldrà crear o obrir l'usuari.

Per crear un nou formulari fill dins del formulari pare, al codi del formulari pare haurem d'escriure quelcom semblant a això:

```
with TFormFill.Create(Self) do  
begin  
...  
end;
```

Per eliminar el formulari fill de dins del formulari pare, al codi del formulari fill haurem de cridar al seu mètode `Free` enlloc del seu mètode `Close`, o bé, al seu esdeveniment `onClose` canviar el valor del paràmetre de sortida `Action` a `caFree`. Si no fem això, per defecte quan tanquem una finestra fill aquesta només es minimitzarà i no desapareixerà ni de memòria ni de pantalla:

```
procedure TFormFill.FormClose(Sender: TObject; var Action: TCloseAction);  
begin  
...  
Action := caFree;  
end;
```

Per treballar amb els formularis fills dins del formulari pare, haurem de considerar les propietats `MDIChildCount`, `MDIChildren` i `ActiveMDIChild` que el formulari pare té. `MDIChildCount` indica quantes finestres filla hi han obertes dintre seu, facilitant l'accés individual a cadascuna d'elles mitjançant un índex numèric. D'aquesta manera es possible realitzar en temps d'execució operacions que afecten a múltiples finestres MDI filles sense necessitat de fer referència directa a cadascuna d'elles. `MDIChildren` és el vector que ens permet accedir a les finestres MDI filles mitjançant l'índex. La seva base és zero. Si volem saber quina és la finestra MDI filla activa en aquell moment, aquest ve indicat per la propietat `ActiveMDIChild`.

Per exemple, per realitzar una operació sobre tots els documents:

```
for i := 0 to MDIChildCount do  
with (MDIChildren[i] as TFormFill) do  
begin  
...  
end;
```

Per exemple, per realitzar una operació sobre el document actiu:

```
with (ActiveMDIChild as TFormFill) do  
begin
```

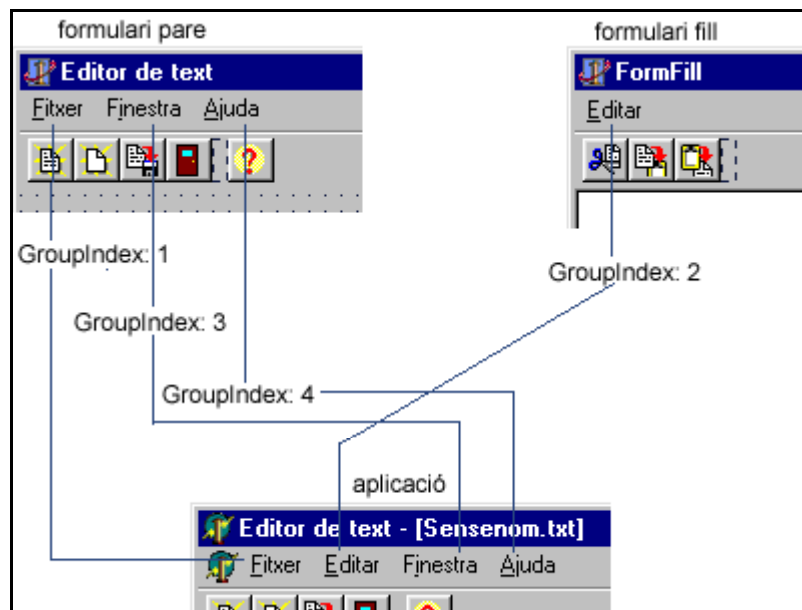
```
...  
end;
```

El formulari pare té, a més a més d'aquestes tres propietats, cinc mètodes per treballar amb formularis fill: Cascade, Tile, ArrangeIcons, Previous i Next.

- Cascade: Endreça els documents sobreposant un a sobre de l'altre.
- Tile: Endreça els documents formant un mosaic sobre el formulari pare. Segons el valor de la propietat TileMode del formulari pare, el tessel·lat serà horitzontal o vertical.
- ArrangeIcons: Endreça els documents minimitzant-los tots.
- Previous i Next: Passen, respectivament, al document anterior o posterior al document actiu.

Per tal que l'usuari pugui seleccionar amb quin document dels que té oberts vol treballar, crearem una opció de menú on apareguin com a subopcions els noms de les finestres a escollir. Aconseguir això a Delphi és tan senzill com crear una nova opció al menú principal del formulari pare i a la propietat WindowMenu del formulari pare posar el nom d'aquesta nova opció del menú que hem creat.

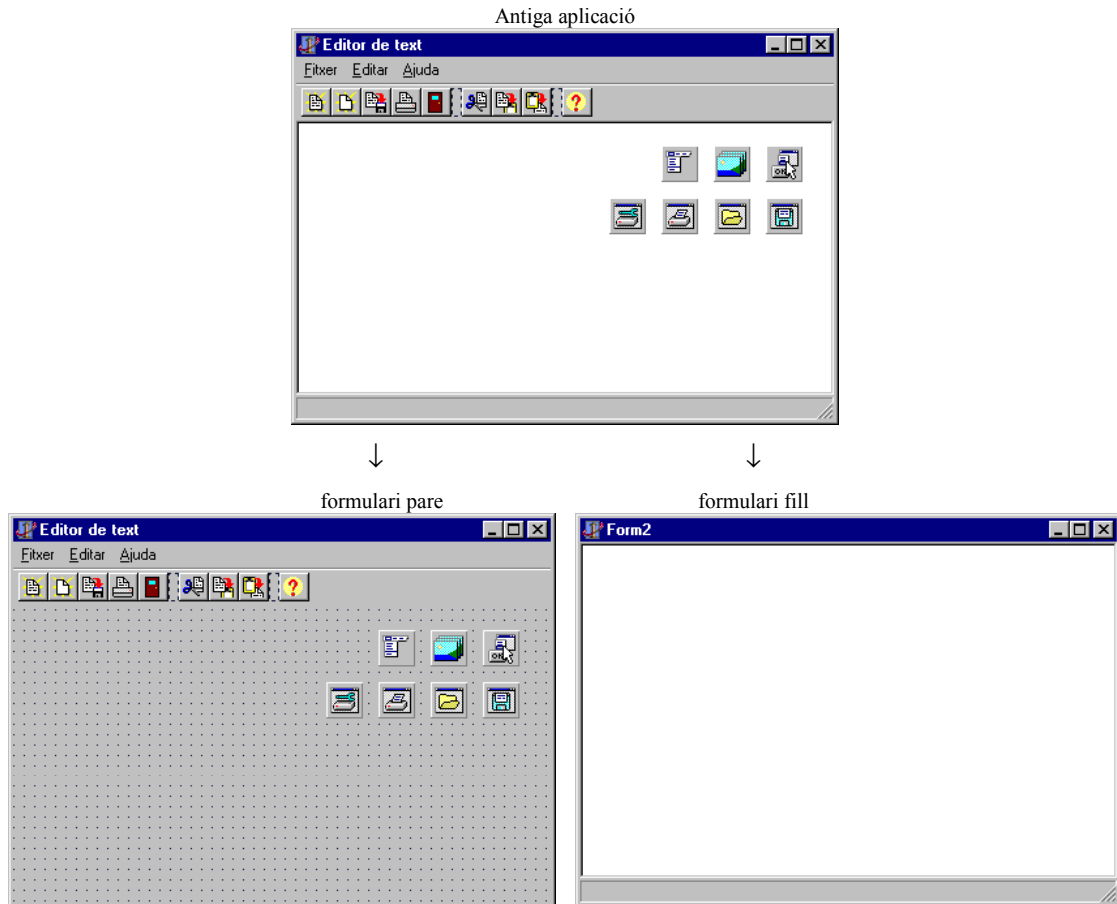
Cal dir, per últim, que ens podem trobar amb el cas hi hagin diferents tipus de documents fills dins una aplicació MDI. Imaginem, per exemple, un sistema d'informació geogràfica multidocument on puguem tenir oberts al mateix temps múltiples bases de dades, varies imatges rasteritzades, varies imatges vectorials, etc. Això no és cap problema. Però si volem que les opcions principals del menú de l'aplicació variïn segons el document obert haurem de fer el que s'anomena fusió de menús. Consisteix en crear un menú principal al formulari pare i un altre als diferents tipus de formularis fills. El menú del formulari pare contindrà opcions comunes a tota l'aplicació, mentre que el menú dels formularis fills tindrà opcions específiques del tipus de document. A la propietat GroupIndex de les opcions principals dels menús ficarem un valor numèric que indiqui l'ordre en que aquestes opcions apareixeran.



### La nostra aplicació 'Editor de Text'

Ara dividireu el formulari de l'aplicació en dos formularis: el formulari 'pare', que contindrà el menú principal, les barres d'eines i un espai per allotjar documents; i el formulari 'fill', que serà la plantilla del document, i contindrà el quadre d'edició i la barra d'estat

40. Creeu un nou formulari a l'aplicació, amb l'opció del menú de Delphi *File* → *New Form*. Seleccioneu el components barra d'estat i quadre d'edició del formulari de l'aplicació i porteu-los al nou formulari. La manera més ràpida de fer això conservant les seves propietats és amb tallar (tecles **Ctrl** + **x**) i enganxar (tecles **Ctrl** + **v**).



41. Al fitxer de projecte esborreu la instrucció `Application.CreateForm(TForm2, Form2)` que s'haurà creat per defecte.

Al fitxer de codi del nou formulari esborreu la declaració de variable `var Form2: TForm2` que s'haurà creat per defecte.

Com que des del codi de la classe formulari pare haurem de fer referència a la classe formulari fill, i des del codi de la classe formulari fill haurem de fer referència a la classe formulari pare, heu d'incloure el nom de cada unitat a la clàusula `uses` de l'altre. A la clàusula `uses` de la unitat del formulari pare afegireu el nom de la unitat del formulari fill:

```
uses  
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
Dialogs, ActnList, ImgList, ComCtrls, StdCtrls, StdActns, Menus,  
ToolWin, Printers, ShellAPI, About, Unit2;
```

A la clàusula `uses` de la unitat del formulari fill afegireu el nom de la unitat del formulari pare. No ho podeu fer a la clàusula `uses` que ja existeix o crearíeu un error de referència circular (per compilar la primera unitat caldria incloure i compilar la segona, però per compilar la segona unitat caldria incloure i compilar la primera). Caldria incloure una nova clàusula `uses` a la part d'implementació:

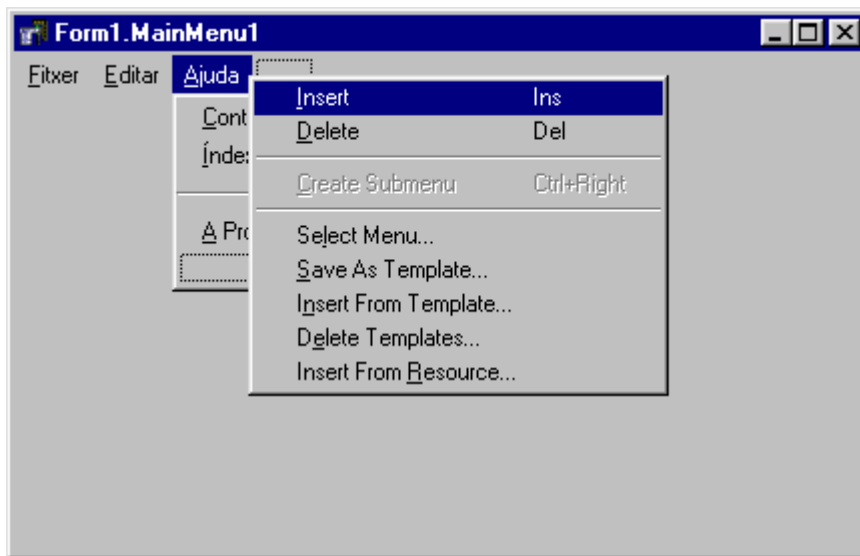
```
implementation  
uses Unit1;
```

42. Seleccioneu el formulari pare. A la seva propietat `FormStyle` seleccioneu el valor `fsMDIForm` per indicar que és un contenidor de documents MDI, i a la seva propietat `Name` escriviu `FormPare`, que és un nom més adient que el que tenia.
43. Seleccioneu el formulari fill. A la seva propietat `FormStyle` seleccioneu el valor `fsMDIChild` per indicar que és una plantilla de documents MDI, i a la seva propietat `Name` escriviu `FormFill`, que és un nom més adient que el que tenia. Per últim, si voleu que quan es creï un document aquest ocupi tota l'àrea de treball del formulari pare, escolliu el valor `wsMaximized` a la propietat `WindowState`.

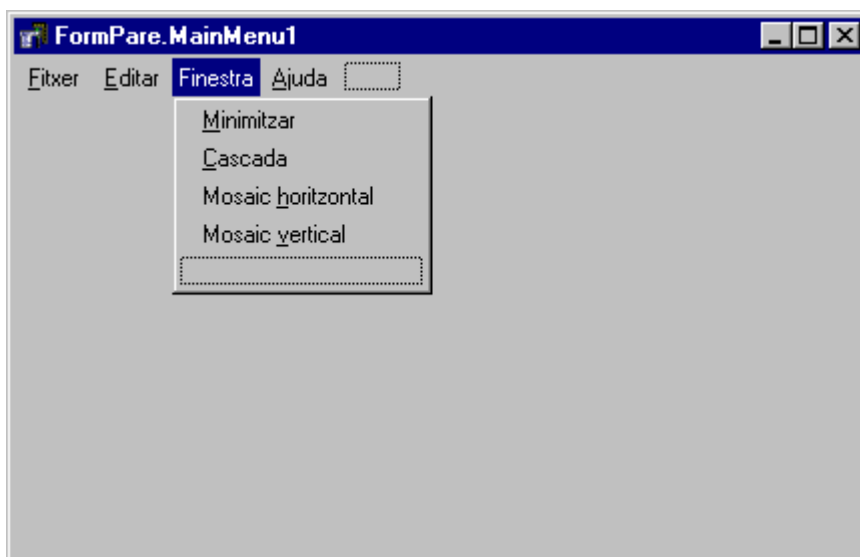
44. Afegiu a la llista d'accions del formulari pare les accions estàndard WindowMinimizeAll, WindowCascade, WindowTileHorizontal i WindowTileVertical. Observeu que en ser accions estàndard, no només no haureu d'escriure el seu codi, sinó que a la llista d'imatges associada a la llista d'accions per defecte s'incorporen icones estàndard per aquestes noves accions. Ajusteu les propietats d'aquestes noves accions:

| Caption             | Category | Hint                     | ImageIndex | Name                |
|---------------------|----------|--------------------------|------------|---------------------|
| &Minimitzar         | Window   | Minimitzar les finestres | -1         | FinestraMinimitzar  |
| &Cascada            | Window   | Finestres en cascada     | 8          | FinestraCascada     |
| Mosaic &horitzontal | Window   | Tessel·lat horitzontal   | 9          | FinestraMosaicHoriz |
| Mosaic &vertical    | Window   | Tessel·lat vertical      | 10         | FinestraMosaicVert  |

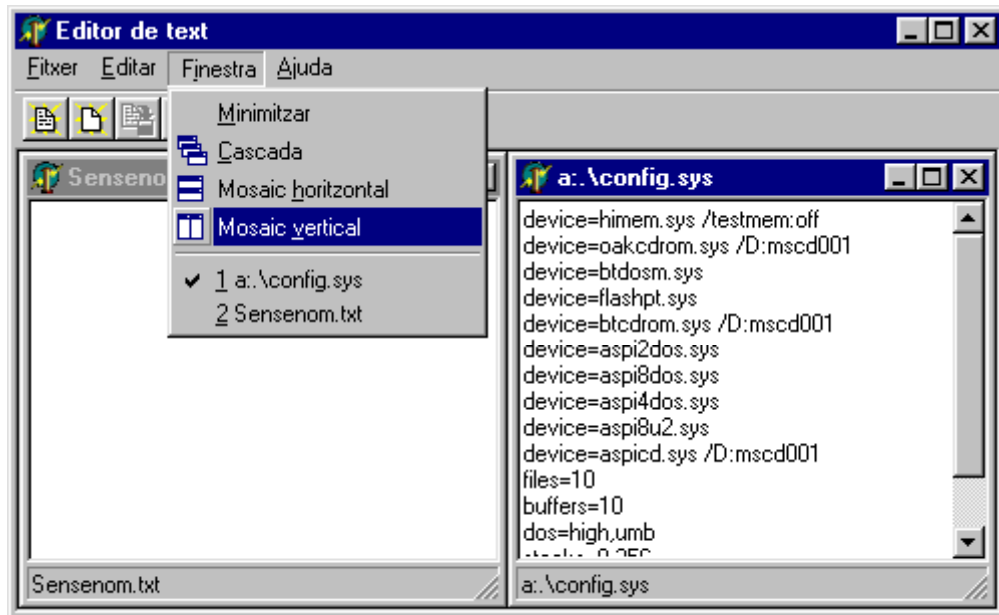
45. Afegireu una nova opció al menú per tal de treballar amb finestres. Haureu de fer doble clic sobre el petit quadre que representa el component menú principal, per tal d'obrir l'editor de menús. Seleccioneu l'opció *Ajuda* del menú principal, feu clic amb el botó dret del ratolí sobre ella i escolliu *Inserir* al menú contextual.



Obtindreu espai per una nova opció del menú, entre les opcions *Editar* i *Ajuda*. Escriviu *&Finestra* a la propietat *Caption* d'aquesta nova opció. Associeu les noves accions creades al punt anterior a la propietat *Action* de cadascuna de les subopcions que apareixen sota aquesta opció del menú.



Per últim, a la propietat *WindowMenu* del formulari principal associeu aquesta nova opció del menú, per tal de poder visualitzar una llista amb els documents oberts quan s'executi l'aplicació.



46. Els canvis al codi del programa seran els següents (en color més fosc):

#### Unitat de formulari pare

```
unit Unit1;  
  
interface  
  
uses  
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,  
  Dialogs, ActnList, ImgList, ComCtrls, StdCtrls, StdActns, Menus,  
  ToolWin, Printers, ShellAPI, Unit2, About;  
  
type  
  TFormPare = class(TForm)  
    ...  
  private  
    procedure WMDropFiles(var Message: TMessage); message WM_DROPFILES;  
  public  
    // NomFitxer ara serà un atribut del formulari fill  
    end;  
  
var  
  FormPare: TFormPare;  
  
implementation  
  
{ $R *.DFM }  
  
procedure TFormPare.FitxerCrearExecute(Sender: TObject);  
begin  
  with TFormFill.Create(Self) do  
    begin  
      RichEdit1.Clear;  
      NomFitxer := 'Sensenom.txt';  
      StatusBar1.SimpleText := NomFitxer;  
      Caption := NomFitxer;  
    end;  
end;  
  
procedure TFormPare.FitxerObrirExecute(Sender: TObject);  
begin  
  if OpenFileDialog1.Execute then  
    with TFormFill.Create(Self) do
```



```

        begin
            RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
            RichEdit1.Modified := False;
            NomFitxer := OpenDialog1.FileName;
            StatusBar1.SimpleText := NomFitxer;
            Caption := NomFitxer;
        end;
    end;

procedure TFormPare.FitxerGuardarExecute(Sender: TObject);
begin
    with (ActiveMDIChild as TFormFill) do
        if NomFitxer = 'Sensenom.txt' then
            FitxerGuardarComExecute(nil)
        else
            begin
                RichEdit1.Lines.SaveToFile(NomFitxer);
                RichEdit1.Modified := False;
            end;
    end;
end;

procedure TFormPare.FitxerGuardarComExecute(Sender: TObject);
begin
    with (ActiveMDIChild as TFormFill) do
        begin
            SaveDialog1.FileName := NomFitxer;
            SaveDialog1.InitialDir := ExtractFilePath(NomFitxer);
            if SaveDialog1.Execute then
                begin
                    RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
                    RichEdit1.Modified := False;
                    NomFitxer := SaveDialog1.FileName;
                    StatusBar1.SimpleText := NomFitxer;
                    Caption := NomFitxer;
                end;
        end;
    end;
end;

procedure TFormPare.FitxerConfigurarImpressoraExecute(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;

procedure TFormPare.FitxerImprimirExecute(Sender: TObject);
begin
    if PrintDialog1.Execute then
        (ActiveMDIChild as TFormFill).RichEdit1.Print(Caption);
end;

procedure TFormPare.FitxerSortirExecute(Sender: TObject);
begin
    Close;
end;

procedure TFormPare.AjudaAPropositExecute(Sender: TObject);
begin
    AboutBox.ShowModal;
end;

procedure TFormPare.ActionList1Update(Action: TBasicAction;
                                       var Handled: Boolean);
begin
    FitxerGuardar.Enabled := (MDIChildCount > 0) and
        (ActiveMDIChild as TFormFill).RichEdit1.Modified and
        (Length((ActiveMDIChild as TFormFill).RichEdit1.Lines.Text) > 0);
    FitxerGuardarCom.Enabled := FitxerGuardar.Enabled
    FitxerImprimir.Enabled := (MDIChildCount > 0);
end;

```

```
procedure TFormPare.FormCreate(Sender: TObject);
begin
  DragAcceptFiles(Handle, True);
end;

// En rebre el missatge WM_DROPFILES
procedure TFormPare.WMDropFiles;
var
  NomArxiu: array [0..255] of char;
  i: integer;
begin
  // Obtenim el nombre d'arxius
  for i := 0 to DragQueryFile(Missatge.WParam, $FFFFFFFF, nil, 0) - 1 do
  begin
    // Obtenim el nom i camí de l'arxiu número 'i'
    DragQueryFile(Missatge.WParam, i, NomArxiu, 256);

    // Si es tracta d'un arxiu de text, carreguem l'arxiu
    if LowerCase(ExtractFileExt(StrPas(NomArxiu))) = '.txt' then
      with TFormFill.Create(Self) do
      begin
        RichEdit1.Lines.LoadFromFile(NomArxiu);
        RichEdit1.Modified := False;
        NomFitxer := StrPas(NomArxiu);
        StatusBar1.SimpleText := NomFitxer;
        Caption := NomFitxer;
      end
    else // Si no es ninguna de las extensiones anteriores
      ShowMessage(NomArxiu + ': tipus d''arxiu desconegut');

  end;

  // Alliberem el bloc de dades
  DragFinish(Missatge.WParam);
end;
```

### Unitat de formulari fill

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, StdCtrls;

type
  TFormFill = class(TForm)
    ...
  private
  public
    NomFitxer: String;
  end;

implementation

uses Unit1;

{$R *.DFM}

// Comprovem abans de tancar si hi han canvis per guardar
procedure TFormFill.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if RichEdit1.Modified then
    case MessageDlg('Guardar canvis a '+NomFitxer, mtConfirmation,
      [mbYes, mbNo, mbCancel], 0) of
      mrYes: (Owner as TFormPare).FitxerGuardarExecute(Self);
    end;
end;
```

```
        mrCancel: CanClose := False;
        end
    end;

// Tanquem el document destruint-lo i alliberant la memòria
procedure TFormFill.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Action := caFree;
end;
```

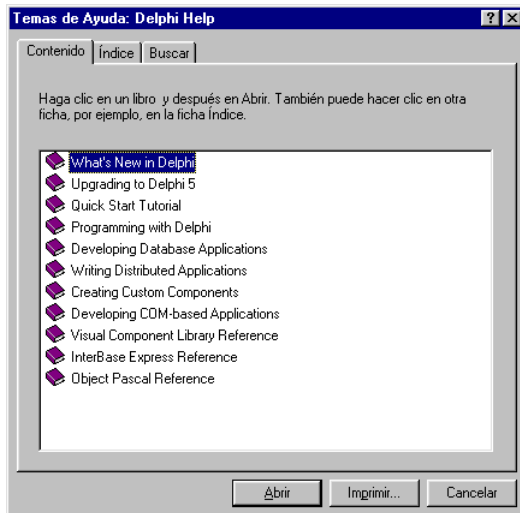
## Creació del fitxer d'ajuda

### **Abans de tot ... deu consells per escriure millor la documentació i els fitxers d'ajuda**

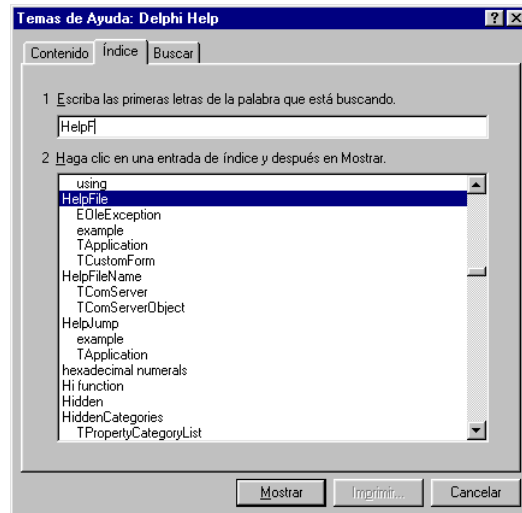
10. Tracteu la documentació com una part fonamental de l'aplicació. És l'únic que es troba entre l'usuari i la interfície.
11. Planegeu la documentació d'ençà el principi. Escriure documentació porta el seu temps i pot afectar la línia de temps de desenvolupament de l'aplicació.
12. Conegueu a l'audiència: Qui seran els usuaris? Quines seran les seves necessitats quan accedeixin a la documentació?
13. Les tres Ces: escriviu documentació concisa, completa i correcta.
14. A la documentació, expliqueu a l'usuari que s'ha d'esperar quan segueix un procediment. No escriviu a l'usuari que faci clic sobre el botó OK i ja està. Comenteu que és el que passarà després.
15. A la documentació, cal avisar a l'usuari quan hi hagi risc de causar pèrdues de dades o dany al sistema.
16. Estil consistent. Per exemple, no empreu a un capítol lletra en negreta per donar èmfasi, i lletra en cursiva pel mateix a un altre capítol.
17. En moltes ocasions una imatge val més que mil paraules, però de vegades no. Si una imatge proporciona informació addicional que no apareix al text o redueix substancialment la quantitat de text, empreu-la. Si la imatge senzillament està per fer bonic, no la empreu, especialment si ocupa 500 Kb. o més!
18. Llegiu sempre el que heu escrit, i feu que ho llegeixi també algú altre.
19. Proveu totes les explicacions com a mínim un cop, per estar segurs que no heu oblidat cap pas crucial o afegit passos innecessaris.

### **Els fitxers d'ajuda**

Existeixen dos tipus de fitxer d'ajuda "estàndard" de Microsoft. El primer tipus, anomenat *WinHelp*, es va emprar a Windows 3.1, Windows 95 i Windows NT 4.0, però encara avui en dia és el més conegut i emprat. Una ajuda d'aquest tipus consisteix en un arxiu amb extensió *.hlp* que conté les pàgines d'ajuda, i un arxiu amb el mateix nom i amb extensió *.cnt* que conté la taula de contingut. El format d'aquests dos arxius és "secret" (Microsoft no l'ha fet mai públic) i la manera de crear-los és editar un fitxer en format *rtf* amb qualsevol editor de text i escriure les diferents pàgines d'ajuda. Dins aquestes pàgines d'ajuda caldrà inserir caràcters especials per indicar quin element és un enllaç a una altra pàgina, quin element és el nombre de pàgina, etc. A continuació cal compilar aquest fitxer amb un programa proporcionat per Microsoft que crearà els dos fitxers d'ajuda que hem comentat.



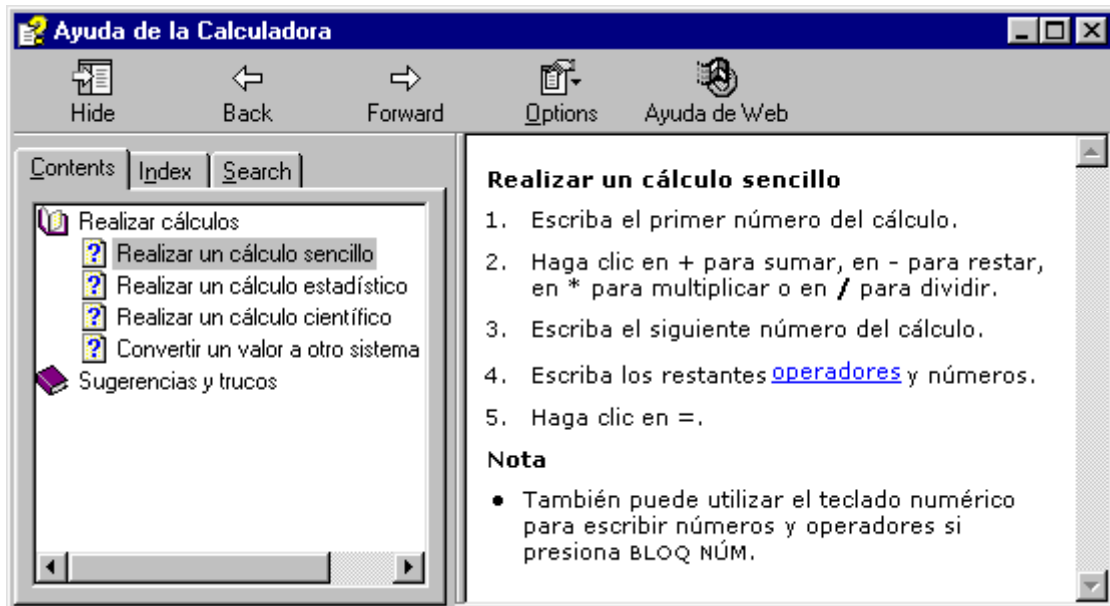
Ajuda WinHelp - Taula de continguts (fitxer .cnt)



Ajuda WinHelp - Índice (fitxer .hlp)

El segon tipus de fitxer d'ajuda, anomenat *HtmlHelp*, es va començar a emprar a Windows 98 i havia de ser el successor de *WinHelp*, però mai es va imposar. El fet que gairebé no aportés res de nou sobre l'antic tipus d'ajuda ha contribuït a que es continués emprant aquesta última, a la que els usuaris ja estan acostumats. El fet que el seu format continués essent secret i les companyies de software haguessin de dependre d'un compilador d'ajuda proporcionat per Microsoft i no poguessin afegir característiques noves a l'ajuda ha contribuït a que les grans companyies de software, com per exemple Adobe o Macromedia, creïn els seus propis sistemes d'ajuda.

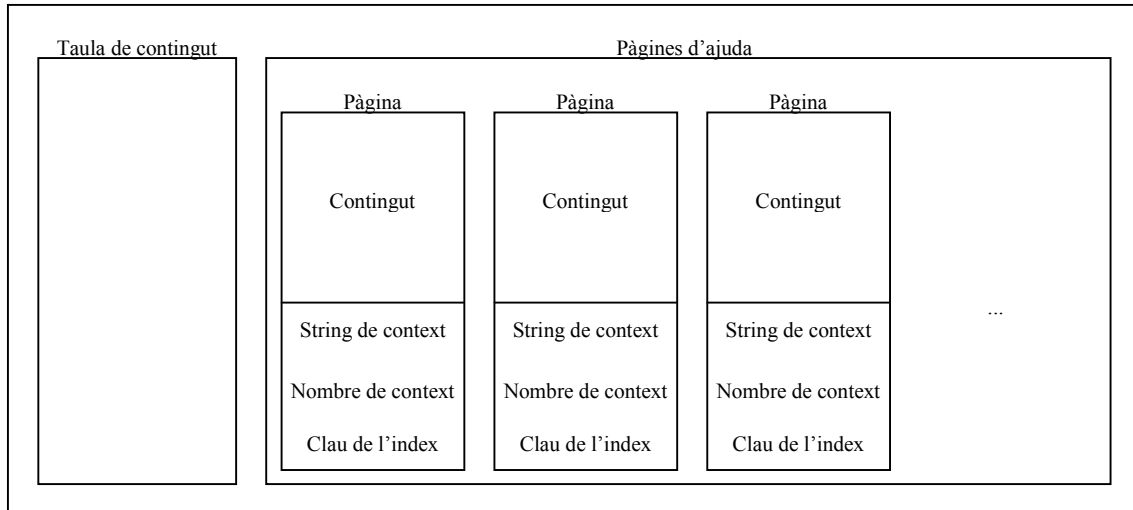
Una ajuda del tipus *HtmlHelp* està formada per un arxiu amb extensió *chm* que conté les pàgines d'ajuda i la taula de continguts. El format d'aquest arxiu és secret, com ja hem dit, i la manera de crear-lo és similar a la manera en que creem una ajuda *WinHelp*, amb la diferència que enlloc de treballar inicialment amb un document en format *rtf*, es treballa amb un document en format *html*.



Ajuda HtmlHelp (fitxer .chm)

Per últim, anem a entrar més en detall amb el format d'una pàgina d'ajuda, ja sigui *WinHelp* o *HtmlHelp*. Cada pàgina d'ajuda consta d'un contingut, que és el que veu l'usuari, i de tres camps d'informació interna, que resten amagats a l'usuari. Un d'aquests camps és la cadena de text que apareixerà a l'índex. Els altres dos camps són un nombre enter i una cadena de text que identifiquen la pàgina, de cara a elaborar ajuda sensible al context.

Ajuda



### Procés de creació

Per temps i per espai, d'aquí en endavant ens centrarem únicament en el tipus d'ajuda *WinHelp* i deixarem de banda el tipus d'ajuda *HtmlHelp*. Tanmateix, gairebé tot el que es dirà per un serveix per l'altre.

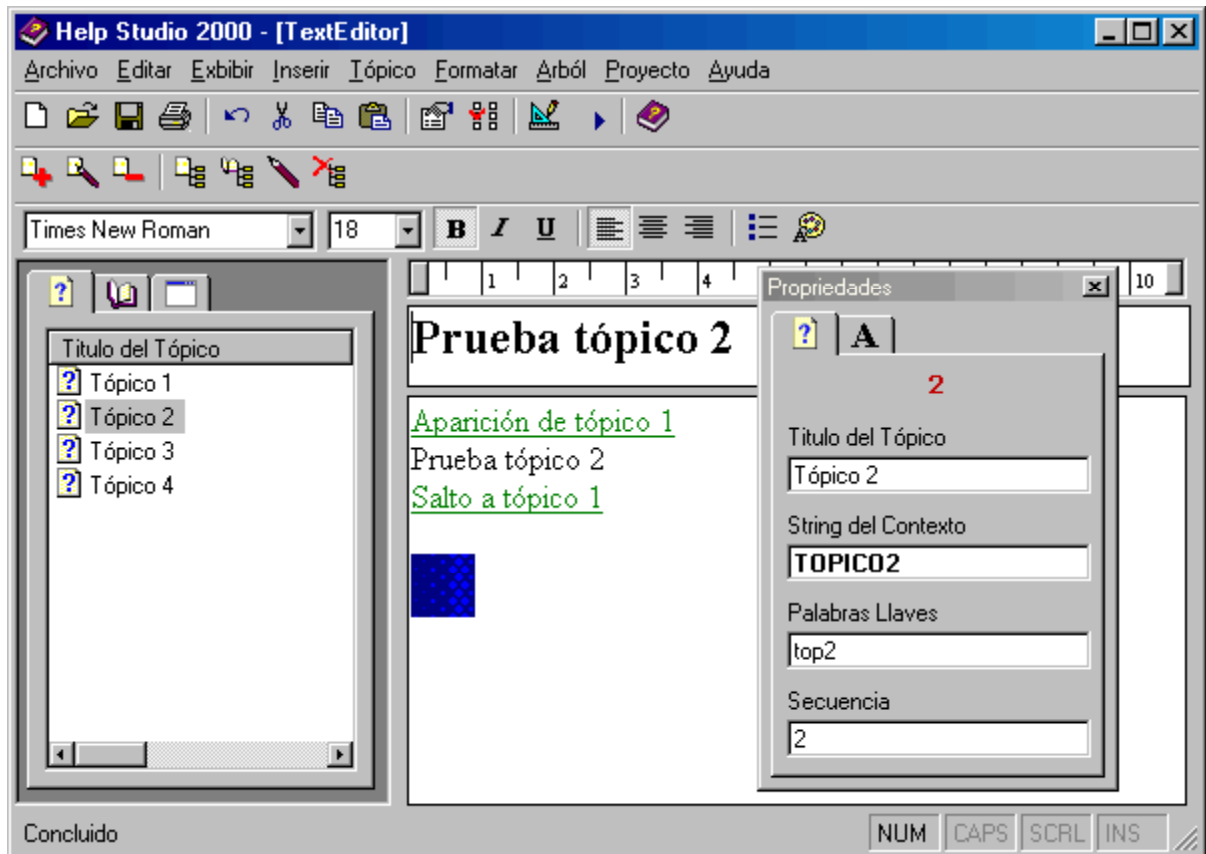
El mètode descrit anteriorment per crear un fitxer d'ajuda és molt pesat. Si voleu seguir aquest mètode, Delphi proporciona les eines i la documentació adequada al directori "C:\Archivos de programa\Borland\Delphi5\Help\Tools". El més còmode, però, és emprar un programa editor de fitxers d'ajuda, que és una mena d'editor de text que ens permet crear el contingut de les pàgines d'ajuda, però que també permet d'una manera clara i ràpida inserir la informació addicional per crear l'índex, la taula de continguts i l'ajut sensible al context. De programes d'aquest tipus s'hi poden trobar molts a Internet i, si fa no fa, tots amb les mateixes capacitats. Per la senzillesa i claredat de la seva interfície d'usuari jo he emprat Help Studio 2000 v2.6. A continuació teniu una llista, ni completa ni actualitzada, d'editors de fitxers d'ajuda que podeu trobar a Internet:

| Nom                   | Cost   | Requeriments              | Pàgina web  | Sortida                                       |
|-----------------------|--|---------------------------|---|---|
| AnetHelp              | Shareware (\$149.95 to \$199.95)                             | Win 95                    | <a href="http://www.anetsoft.com/engl/helptool/prhlptl.htm">http://www.anetsoft.com/engl/helptool/prhlptl.htm</a>   | Windows 3.x, Windows 95, HTML Help, Java Help |
| Astrohelp             | Freeware   | Win 3.x/95, Word 6/95/97  | <a href="http://www10.pair.com/vsap/AstroHelp.html">http://www10.pair.com/vsap/AstroHelp.html</a>                   | Windows 3.x, Windows 95                       |
| AuthorIT              | Commercial demo. Limited to a small number of topics         | Win 95/NT4                | <a href="http://www.oscl.com/">http://www.oscl.com/</a>   | Windows 95, HTML Help                         |
| Doc-To-Help           | 30-day commercial demo                                       | Win 95/NT, Word 95/97     | <a href="http://www.wextech.com/doc2help.htm">http://www.wextech.com/doc2help.htm</a>                               | Windows 3.x, Windows 95, HTML Help, JavaHelp  |
| EasyHelp              | 30-day commercial demo                                       | Win 95/NT, Word 6/95/97   | <a href="http://www.eon-solutions.com/easyhelp/easyhelp.htm">http://www.eon-solutions.com/easyhelp/easyhelp.htm</a> | Windows 95, HTML Help                         |
| eAuthor Help          | 30-day commercial demo                                       | Win 95/NT 4               | <a href="http://www.hyperact.com/eAuthorHelp.html">http://www.hyperact.com/eAuthorHelp.html</a>                     | HTML Help                                     |
| FAR                   | \$38 (shareware, download is fully functional)               | Win 95                    | <a href="http://helpware.net/FAR/index.html">http://helpware.net/FAR/index.html</a>                                 | HTML Help compiled from Web sites             |
| ForeHelp              | Crippled demos (20 topics max) (~\$100 to \$700)             | Win 95                    | <a href="http://www.forehelp.com/">http://www.forehelp.com/</a>   | HTML Help, WinHelp, and many others           |
| HELLLP!               | Shareware (\$30 to \$100)                                    | Win 95/NT 4, Word 97/2000 | <a href="http://mindlink.net/Ed_Guy/helllp.html">http://mindlink.net/Ed_Guy/helllp.html</a>                         | Windows 95                                    |
| Help Authoring Expert | Shareware (\$65). The unregistered version has a topic limit | Win 95/NT 4, Word 95/97   | <a href="http://www.stevesamuelson.com">http://www.stevesamuelson.com</a>   | Windows 95                                    |
| Help Express          | Shareware (\$149). Unregistered version has an advertisement | Win 95                    | <a href="http://www.chainware.com/docs/he3.html">http://www.chainware.com/docs/he3.html</a>                         | Windows 95                                    |

**Programació Visual amb Delphi**  
Creació d'una aplicació

|                       |   |                                    |   |                                    |
|-----------------------|---|------------------------------------|---|------------------------------------|
|                       | embedded in every help file   |                                    |   |                                    |
| Help Maker Plus       | Shareware (\$45). Unregistered version is limited to a small number of topics | Win 3.x/95/2000, Word 6/95/97/2000 | <a href="http://www.exhedra.com/exhedra/helpmakerplus/features.asp">http://www.exhedra.com/exhedra/helpmakerplus/features.asp</a> | Windows 3.x, Windows 95            |
| Help Pad              | Shareware (\$80)  | Win 95                             | <a href="http://www.genet.com/bw/hpad/">http://www.genet.com/bw/hpad/</a>   | Windows 95, HTML Help              |
| HelpBreeze            | Commercial demo. Limited to a small number of topics                          | Win 95/NT, Word 6/95               | <a href="http://www.solutionsoft.com/">http://www.solutionsoft.com/</a>   | Windows 95, HTML Help              |
| Helpburger            | Shareware (\$40). The unregistered version has a topic limit.                 | Win 95                             | <a href="http://www.langdaledesigns.co.uk/hb/helpburger.htm">http://www.langdaledesigns.co.uk/hb/helpburger.htm</a>               | Windows 3.x, Windows 95            |
| HelpGen               | Shareware (\$30)  | Win 3.x/95/NT                      | <a href="http://www.rimrocksoftware.com/helpgen.html">http://www.rimrocksoftware.com/helpgen.html</a>                             | Windows 95                         |
| HelpHikes Pro         | Shareware (\$35)  | Win 3.x/95                         | <a href="http://web.superb.net/helphikes/">http://web.superb.net/helphikes/</a>   | Windows 3.x, Windows 95            |
| HelpMe                | Freeware  | Win 95                             | <a href="http://home.wxs.nl/~verho037/jfpprogramming.htm">http://home.wxs.nl/~verho037/jfpprogramming.htm</a>                     | Windows 3.x, Windows 95            |
| HTML Help Workshop    | Free  | Win 95/NT 4                        | <a href="http://www.microsoft.com/workshop/author/htmlhelp">http://www.microsoft.com/workshop/author/htmlhelp</a>                 | HTML Help                          |
| HyperText Studio      | Commercial demo. Limited to a small number of topics.                         | Win 95                             | <a href="http://webnz.com/olson/ProductsFrameset.htm">http://webnz.com/olson/ProductsFrameset.htm</a>                             | Windows 3.x, Windows 95, HTML Help |
| JavaHelp              | Free  | Win 95/NT, Solaris, MacOS 8.1      | <a href="http://www.javasoft.com/products/javahelp/index.html">http://www.javasoft.com/products/javahelp/index.html</a>           | JavaHelp                           |
| NetHelp2              | Free  | Win 95                             | <a href="http://home.netscape.com/eng/help/home/home.htm">http://home.netscape.com/eng/help/home/home.htm</a>                     | NetHelp2                           |
| RoboHELP              | 15-day commercial demo (~\$2,000)   | Win 95/98                          | <a href="http://www.ehelp.com">http://www.ehelp.com</a>   | Most help formats                  |
| SOS Help! Info-Author | 30-day commercial demo  | Win 95                             | <a href="http://www.lamaura.com/">http://www.lamaura.com/</a>   | Windows 3.x, Windows 95            |
| Visual Help Pro       | 30-day commercial demo  | Win 95                             | <a href="http://www.winwareinc.com/">http://www.winwareinc.com/</a>   | Windows 3.1, Windows 95, HTML Help |
| Windows Help Designer | Shareware (\$49 to \$79)  | Win 95/NT                          | <a href="http://www.visagesoft.com/">http://www.visagesoft.com/</a>   | Windows 95, HTML Help              |

Anem a veure com crear un fitxer d'ajuda amb Help Studio 2000 v2.6.



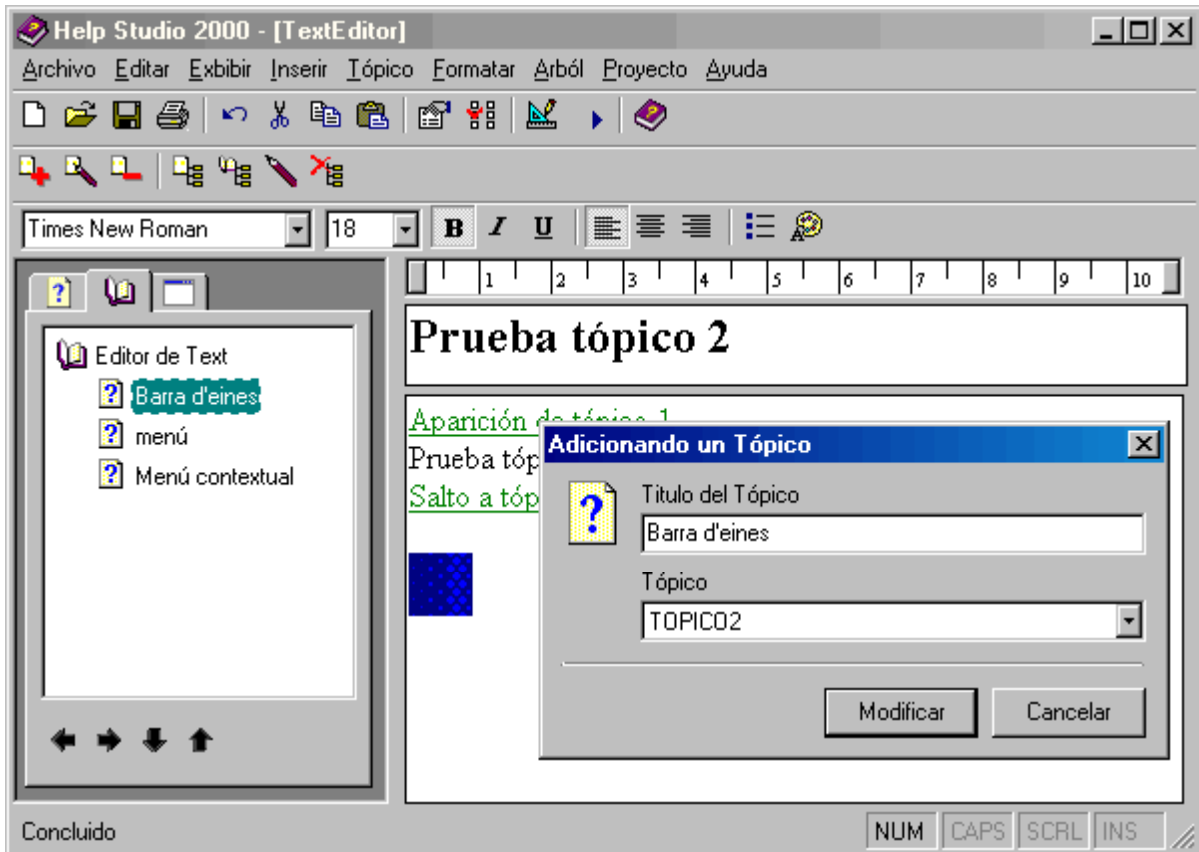
L'espai de treball de Help Studio 2000, editant una pàgina d'ajuda

Al quadre de l'esquerra podem veure totes les pàgines que tenim, seleccionar una pàgina, afegir una de nova o esborrar-ne.

A l'espai central editem el contingut de la pàgina. Una pàgina d'ajuda pot contenir text amb format, imatges i enllaços a altres pàgines o a Internet.

Al quadre de diàleg que flota a la dreta introduïrem el nom de la pàgina d'ajuda que apareixerà a l'índex ("Palabras Llaves") i el nom ("String del Contexto") i el nombre ("Secuencia") per l'ajut sensible al context. Aquest quadre té una segona pestanya per crear enllaços a altres pàgines d'ajuda o a Internet.



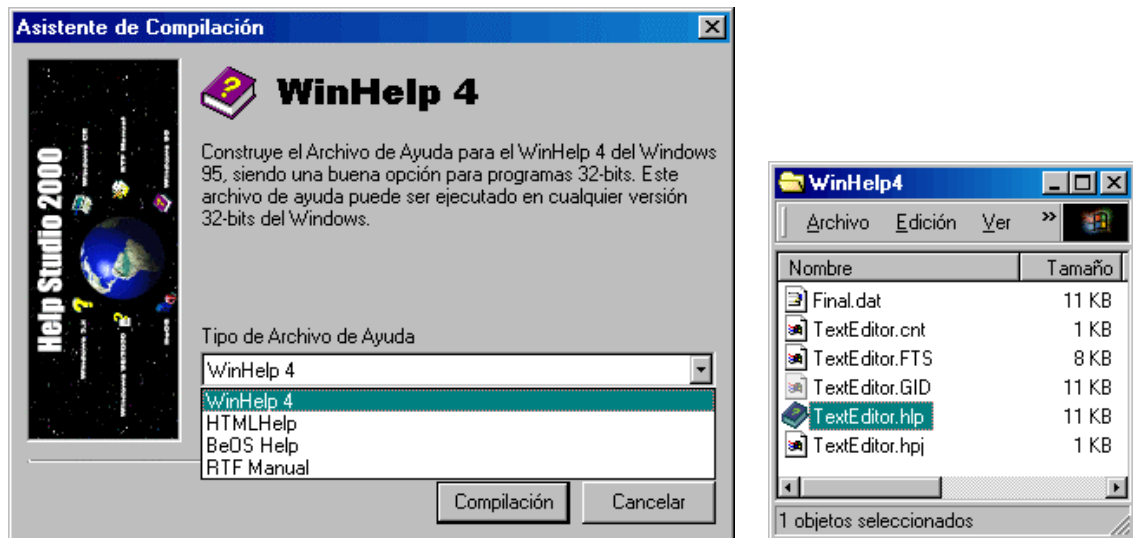


L'espai de treball de Help Studio 2000, creant la taula de continguts

La taula de continguts està formada per llibres, on cada llibre pot contenir altres llibres i pàgines d'ajuda.

Al quadre de l'esquerra podem veure com està organitzada la taula de continguts, afegir i esborrar llibres de la taula, i afegir i esborrar de la taula pàgines d'ajuda prèviament creades.

Al quadre de diàleg que flota a la dreta escriurem el text que voldrem que aparegui a la taula de contingut pel llibre o pàgina d'ajuda seleccionada.

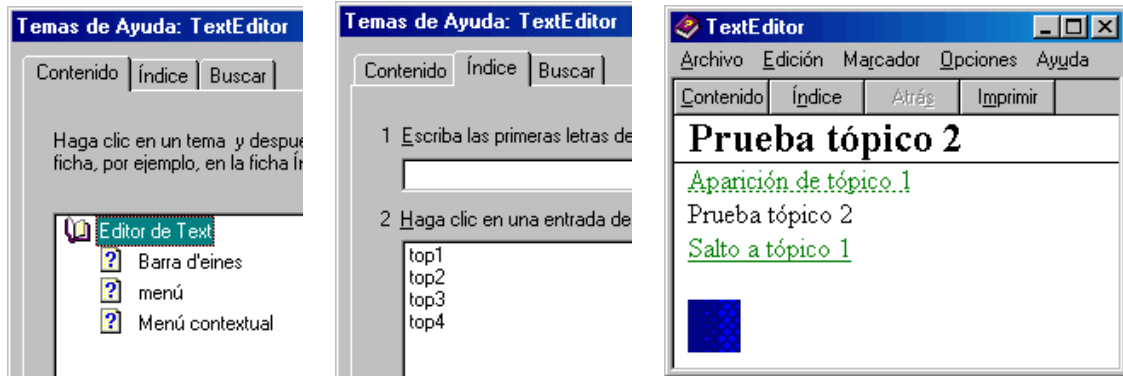


Compilador d'ajuda de Help Studio 2000, creant el tipus d'ajuda desitjada

Un cop hem escrit les pàgines d'ajuda cal compilar-les per crear el fitxer d'ajuda utilitzable per l'aplicació. Help Studio ens permet seleccionar quin tipus de fitxer d'ajuda volem. Si seleccionem l'ajuda



WinHelp ens crearà els fitxers *.hlp* i *.cnt* que haurem de moure després al lloc on es trobi la nostra aplicació. Fem doble clic sobre el fitxer *.hlp* per veure si funciona.



### Associar el fitxer d'ajuda a una aplicació

Podem especificar el fitxer d'ajuda d'una aplicació en temps de disseny, mitjançant l'opció del menú de Delphi Project → Options... → Application → Help file.

També podem especificar el fitxer d'ajuda d'una aplicació en temps d'execució, assignant el seu nom a la propietat HelpFile de l'objecte Application. Per exemple:

```
| Application.HelpFile := ExtractFilePath(Application.ExeName) + 'NomFitxerAjuda.hlp';
```

Tanmateix, tinguem en compte que una aplicació pot tenir associat més d'un fitxer d'ajuda. L'aplicació en sí en tindrà un, tal i com hem explicat al paràgraf anterior, però cadascuna de les seves finestres també en pot tenir un de propi diferent associat a la seva propietat HelpFile.

Associat un fitxer d'ajuda a una finestra o a l'aplicació sencera, quan executem aquesta aplicació l'ajuda apareixerà en prémer la tecla **F1**. Queda, però, associar les pàgines del fitxer d'ajuda als controls i accions de la finestra, per tal que l'ajuda sigui sensible al context, és a dir, que aparegui una pàgina d'ajuda diferent segons s'hagi sol·licitat ajuda tenint el focus un control de la interfície gràfica o un altre. Per aconseguir això treballarem amb la propietat HelpContext dels controls i accions, que contindrà l'índex de la pàgina d'ajuda associada (cada pàgina d'ajuda té un nombre i un nom que la identifica).

Si volem que l'usuari pugui accedir a l'ajuda d'altres maneres, com per exemple prement un botó o escollint una opció del menú, haurem d'emprar alguna d'aquestes instruccions:

- Inserir a la nostra llista d'accions les accions estàndard THelpContents i THelpTopicSearch que mostren, respectivament, la taula de continguts i l'índex del fitxer d'ajuda.
- El mètode HelpCommand de l'objecte Application, que permet accedir directament a l'API WinHelp. Aquest mètode rep dos paràmetres que són la comanda a executar i les dades.

| Comanda           | Acció  | Dada                 |
|-------------------|--|----------------------|
| HELP_INDEX        | Visualitza la taula de continguts                    |                      |
| HELP_FINDER       | Visualitza l'índex                                   |                      |
| HELP_HELPONHELP   | Visualitza ajuda sobre com emprar l'ajuda de Windows |                      |
| HELP_KEY          | Visualitza ajuda sobre un títol de l'índex           | Adreça de la paraula |
| HELP_CONTEXT      | Visualitza una pàgina d'ajuda                        | Nombre de la pàgina  |
| HELP_CONTEXTPOPUP | Visualitza a una finestra emergent un títol d'ajuda  | Nombre de la pàgina  |
| HELP_SETWINPOS    | Estableix la posició i dimensions de l'ajuda         | Adreça HELPWININFO   |
| HELP_QUIT         | Tanca l'ajuda  |                      |

Per exemple:

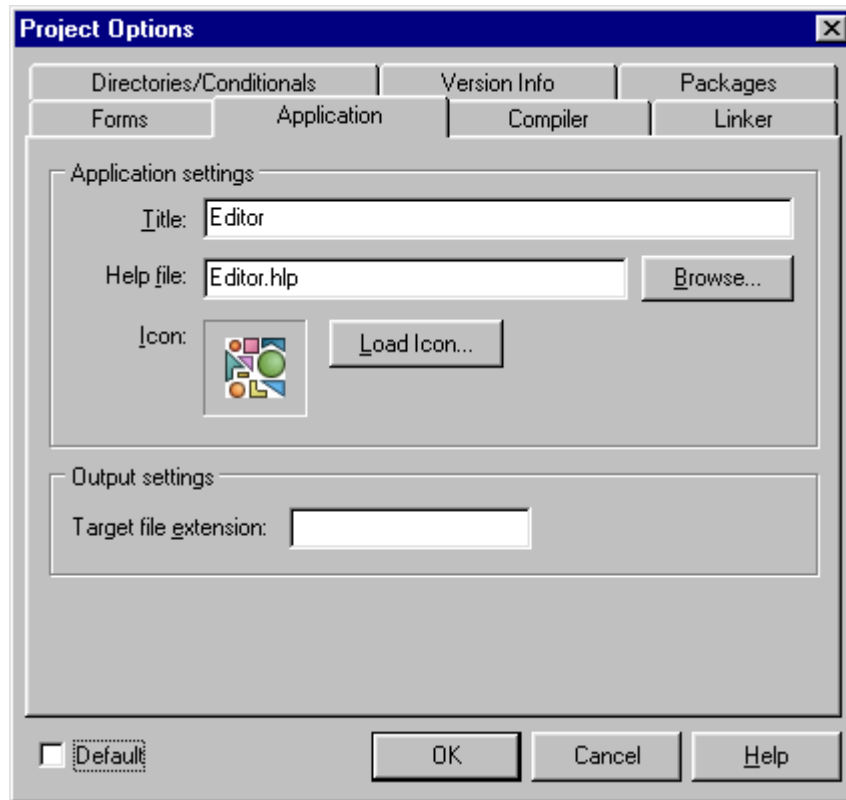
```
| Application.HelpCommand(HELP_FINDER, 0); //Continguts
| Application.HelpCommand(HELP_INDEX, 0); //Index
```

- Els mètodes HelpContext i HelpJump que permeten accedir a una pàgina del fitxer d'ajuda, segons el nombre identificador de la pàgina i el nom identificador de la pàgina, respectivament. Per exemple:

```
| Application.HelpContext(2);
| Application.HelpJump(TOPICO2);
```

### La nostre aplicació 'Editor de Text'

47. Creeu un petit fitxer d'ajuda en format *WinHelp* (amb tres pàgines n'hi ha prou per fer una prova) amb el vostre programa preferit de creació d'ajuda. Copieu els fitxers *.hlp* i *.cnt* generats a la carpeta on guardeu l'aplicació de l'editor de text i doneu-los un nom adient, com per exemple *Editor.hlp* i *Editor.cnt*.
48. Associeu el fitxer d'ajuda a l'aplicació, bé sigui mitjançant l'opció del menú de Delphi *Project* → *Options...* → *Application* → *Help file*:



o bé sigui mitjançant codi:

```
program Exemple6_1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {FormPare},
  Unit2 in 'Unit2.pas' {FormFill},
  About in 'About.pas' {AboutBox};

{$R *.RES}

begin
  Application.Initialize;
  Application.Title := 'Editor';
  Application.HelpFile := 'Editor.hlp';
  Application.CreateForm(TFormPare, FormPare);
  Application.CreateForm(TAboutBox, AboutBox);
  Application.Run;
end.
```

49. Anteriorment heu creat a l'aplicació dues accions per treballar amb el fitxer d'ajuda, AjudaContingut i AjudaIndex, i les heu associat a dues opcions del menú i un botó de la barra d'eines. Com que no les hem escollit com accions estàndard, ara ens tocarà escriure el seu codi:

```
procedure TFormPare.AjudaContingutExecute(Sender: TObject);
begin
  Application.HelpCommand(HELP_INDEX, 0);
end;
```

```
procedure TFormPare.AjudaIndexExecute(Sender: TObject);  
begin  
    Application.HelpCommand(HELP_FINDER, 0);  
end;
```

50. Si voleu ajut sensible al context, associeu a la propietat `HelpContext` de l'acció o control seleccionat el número de pàgina d'ajuda corresponent.

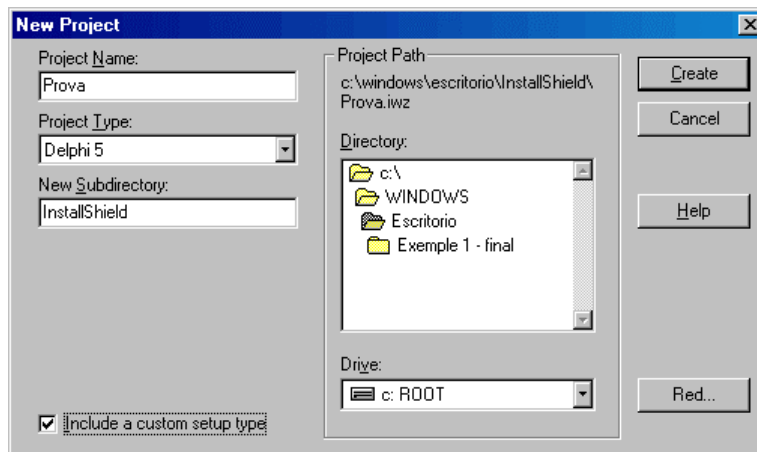
## Creació de l'assistent d'instal·lació

Gairebé tota aplicació d'avui en dia es distribueix comprimida dins un assistent d'instal·lació. Aquest assistent, en executar-se, dona l'opció a l'usuari d'escollir a quina carpeta voldrà instal·lar l'aplicació i quines parts de l'aplicació s'instal·laran, associa l'aplicació i una icona a alguna extensió de fitxer, inclou al sistema una opció per desinstal·lar de manera automàtica l'aplicació, etc.

D'entre les moltes i bones utilitats que existeixen al mercat per crear assistents d'instal·lació, una de les més populars per la seva senzillesa és InstallShield Express. Delphi ve acompanyat d'una versió d'InstallShield Express especialment preparada per treballar amb aplicacions creades amb Delphi. Delphi 5 ve acompanyat d'InstallShield Express 2.12, Delphi 6 ve acompanyat d'InstallShield Express 3, i al moment d'escriure aquestes línies ha sortit al mercat la versió 4 d'InstallShield Express. Nosaltres treballarem amb la versió 2.12 d'InstallShield Express que acompanya Delphi 5. Cal aclarir, però, que aquesta eina no s'instal·la per defecte amb Delphi, sinó de manera separada.



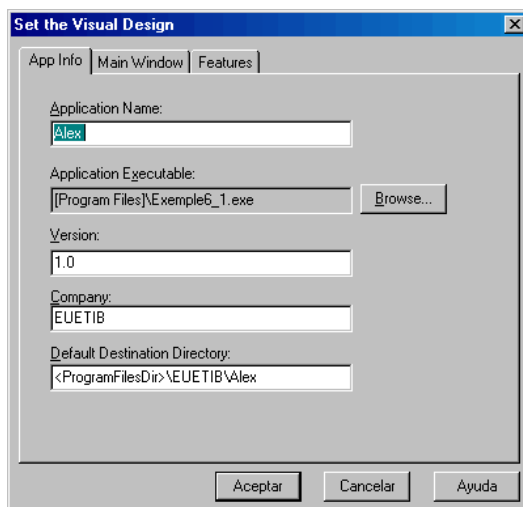
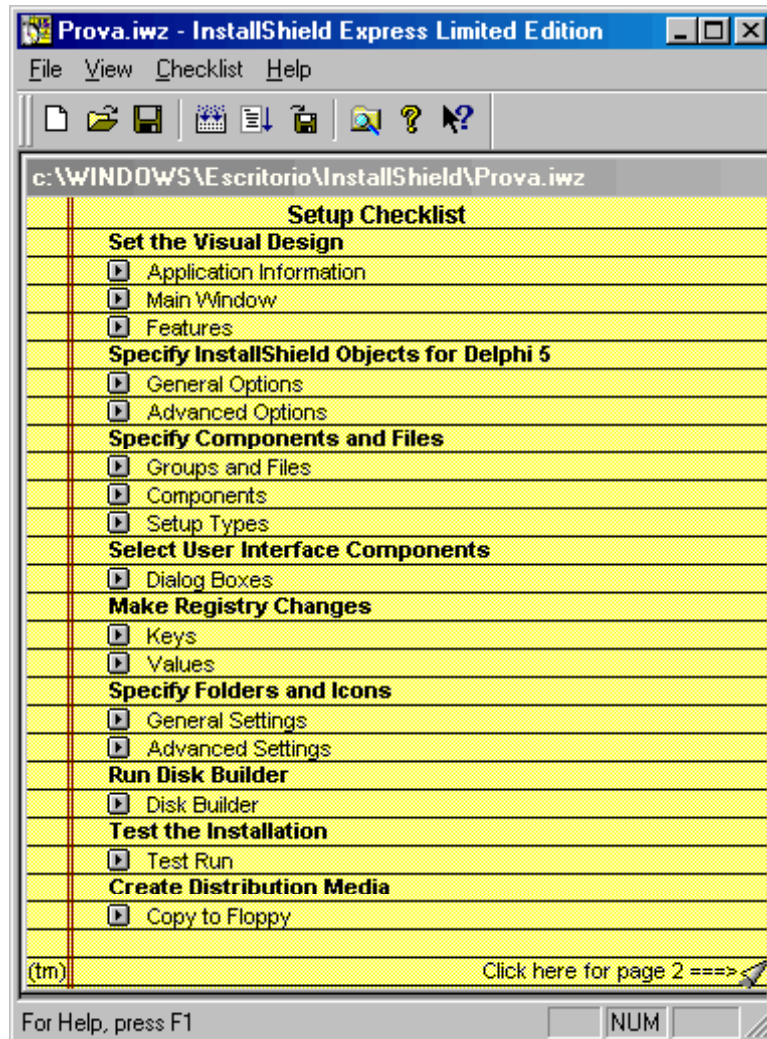
En començar a emprar InstallShield per crear un nou assistent d'instal·lació cal especificar el nom del projecte, la carpeta on es guardarà tant el projecte com els arxius generats per InstallShield i si es voldrà una instal·lació personalitzada o no.



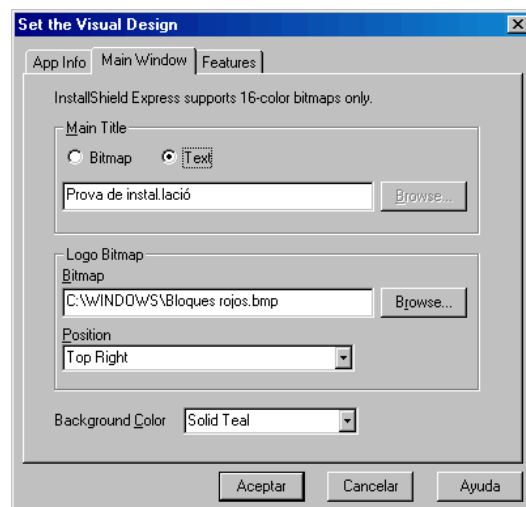
## Programació Visual amb Delphi

### Creació d'una aplicació

A continuació apareix una llista de tasques a omplir per especificar com serà la instal·lació. Haurem d'anar completant una a una les tasques de la llista, encara que no cal que sigui en l'ordre que apareixen. El millor és aprendre un mateix com funciona la creació de l'assistent d'instal·lació jugant amb la llista i explorant les seves opcions. Els quadres de diàleg que aniran apareixent són prou intuïtius.

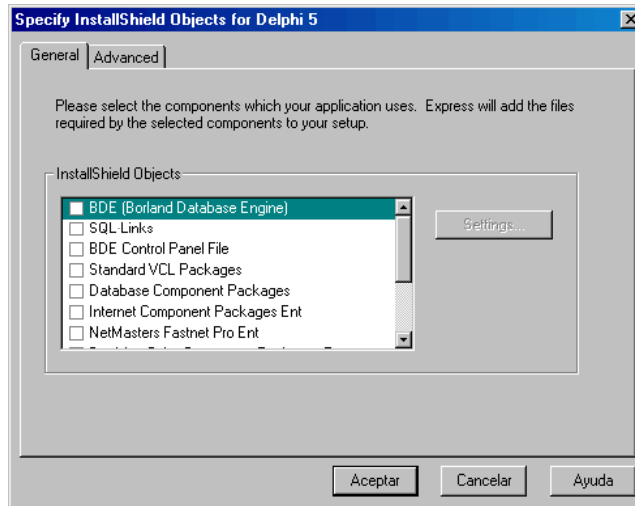


Especificar dades de l'aplicació: fitxer executable, directori on s'instal·larà per defecte, etc.

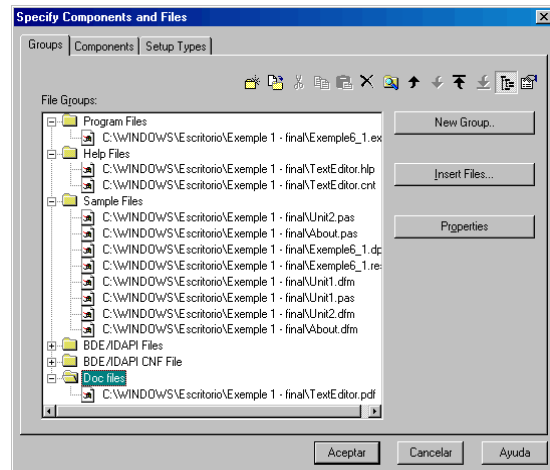
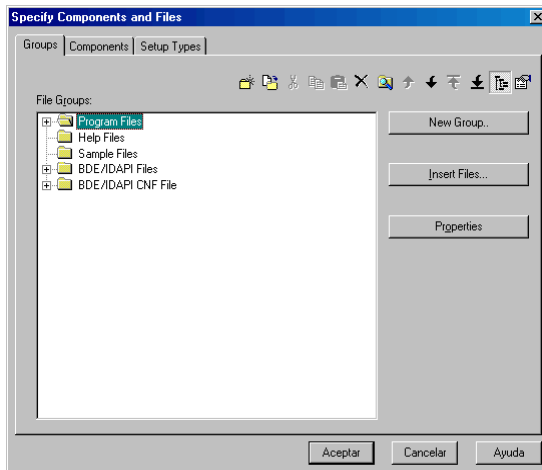


Especificar com serà la finestra del programa d'instal·lació: Color de fons, imatge de fons, text de fons.

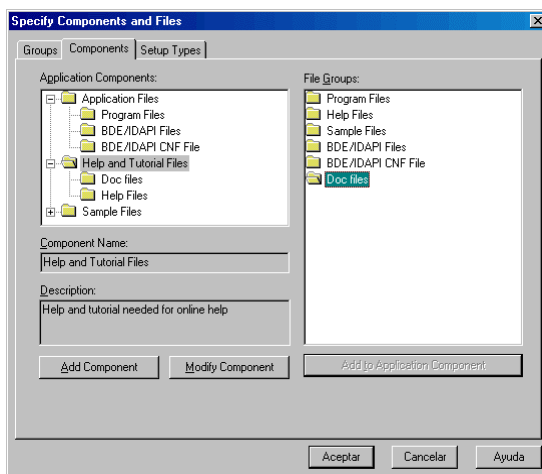




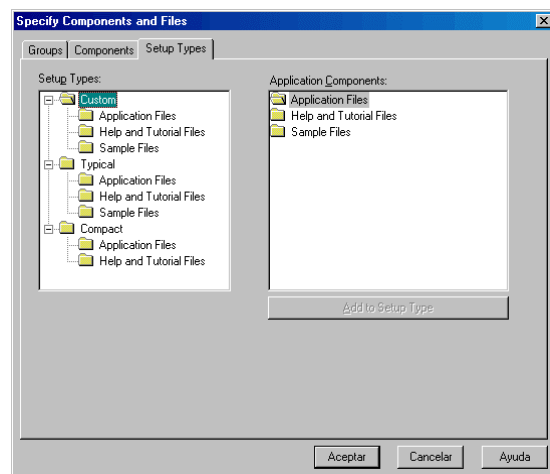
Especificar elements opcionals de Delphi que cal instal·lar per tal que l'aplicació funcioni. Per exemple, si l'aplicació treballa amb bases de dades, caldrà instal·lar també el motor de bases de dades de Borland (BDE).



Especificar els grups de fitxers, que és el conjunt d'arxius que el programa instal·larà a un directori específic. Per exemple, els arxius d'ajuda els podem ficar a un directori separat dels fitxers executables



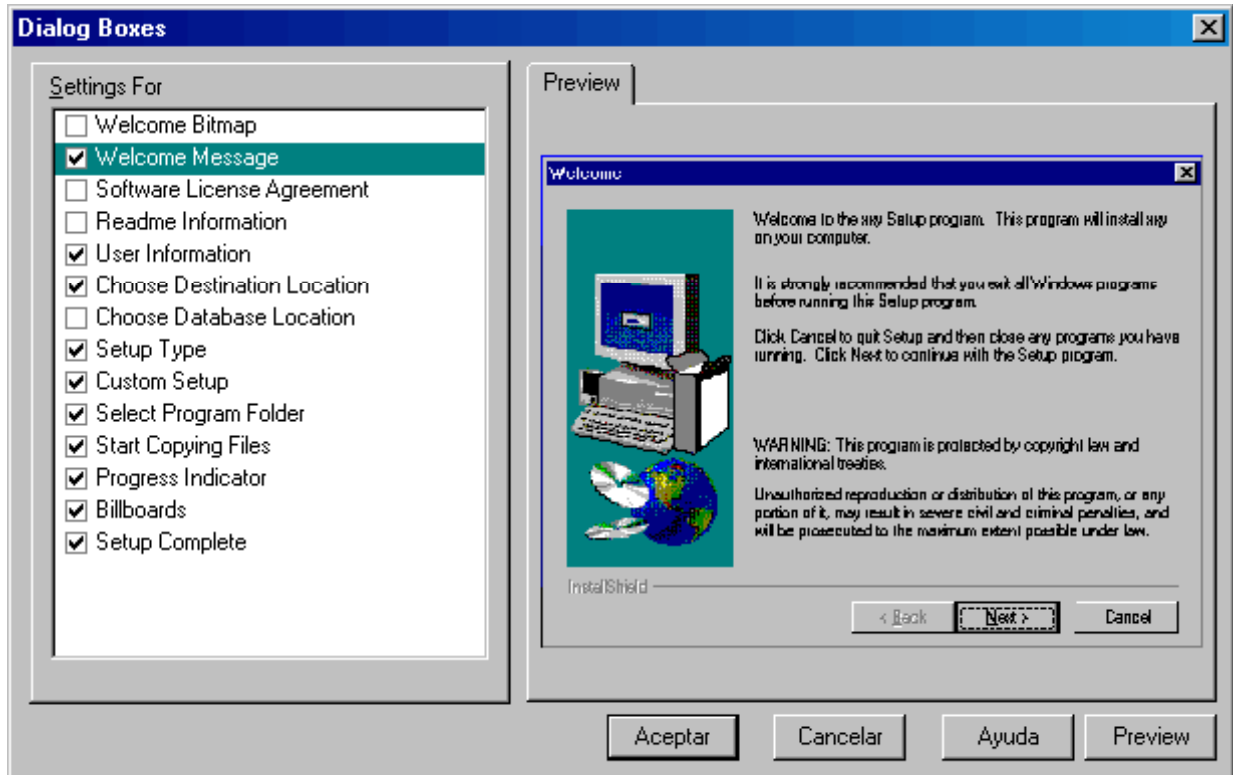
Especificar els components, que són conjunts de grups d'arxius agrupats en base a les seves funcions lògiques. Per exemple, podem ficar dins un mateix component anomenat *documentació* el grup d'arxius *fitxers d'ajuda* i el grup d'arxius *tutorial*.



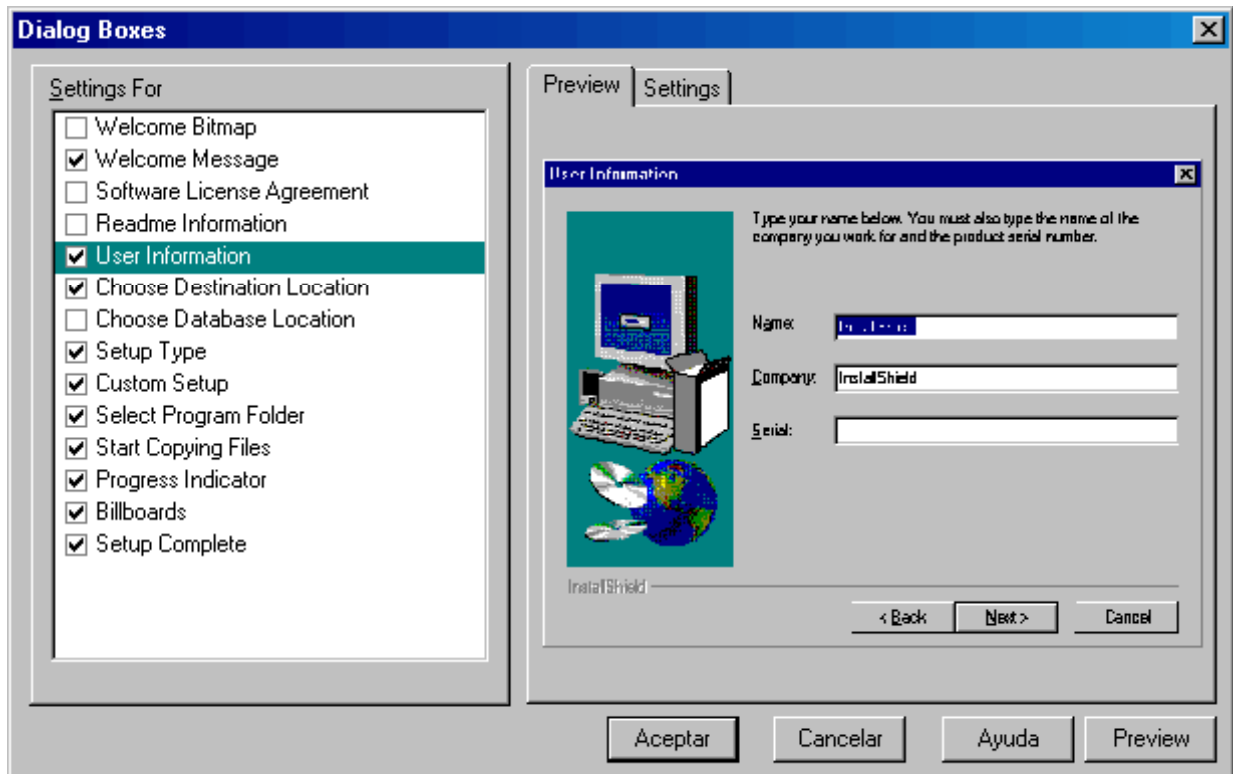
Especificar quins components tindrà cada tipus d'instal·lació.



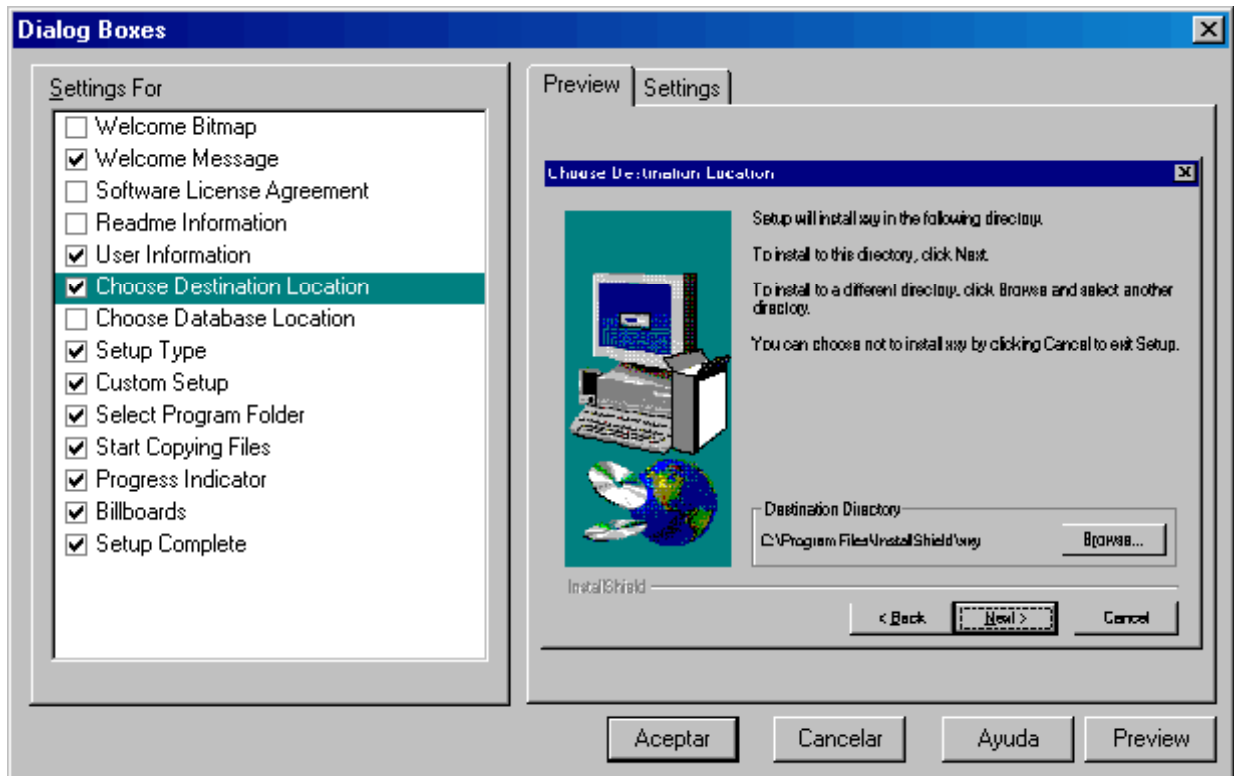
Especificar els quadres de diàleg estàndard del fitxer d'instal·lació que apareixeran.  
Exploreu-los tots. Alguns tenen opcions que cal especificar.



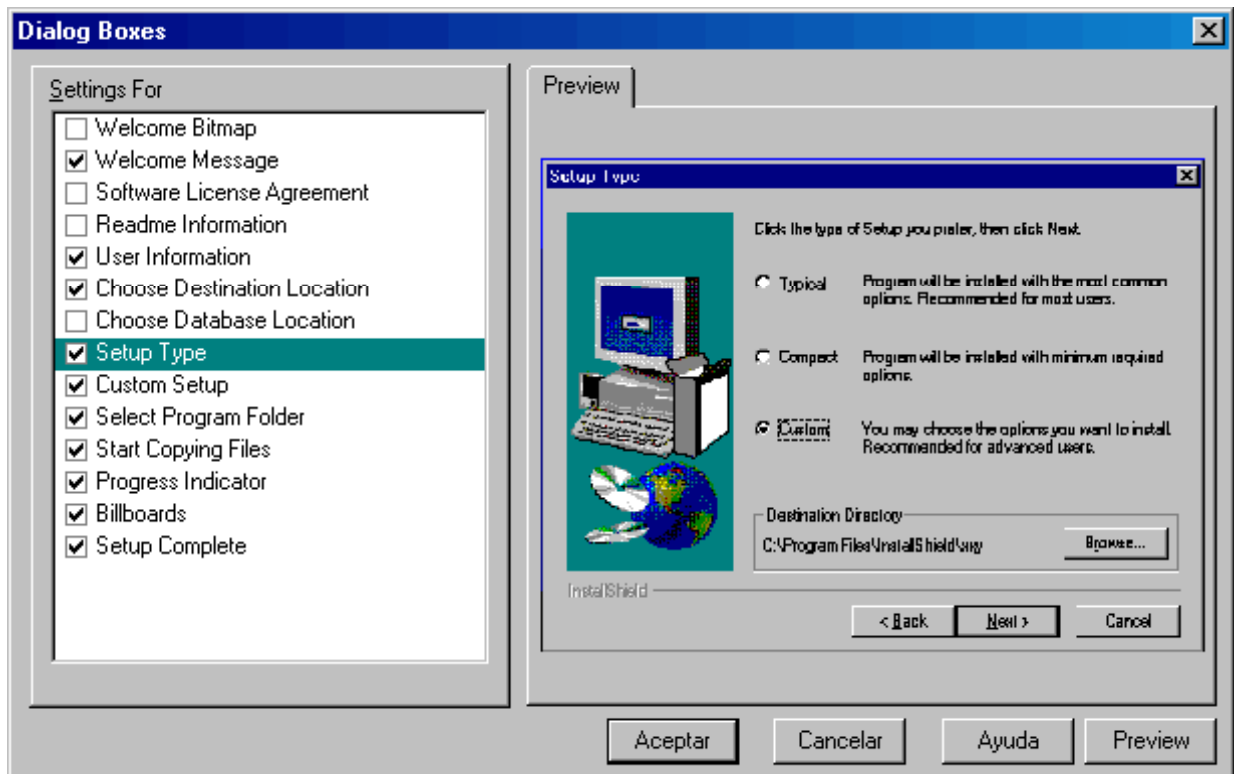
Missatge de benvinguda per l'usuari



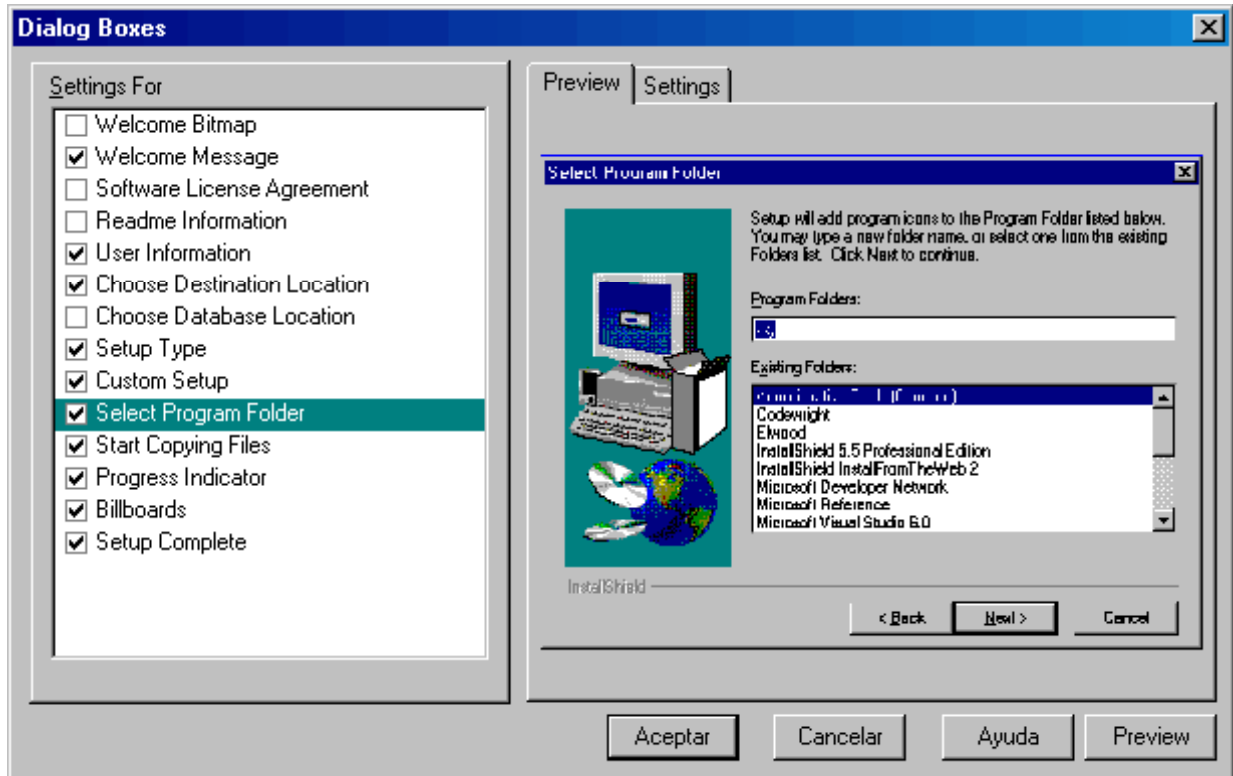
L'usuari pot especificar el seu nom i el número de sèrie



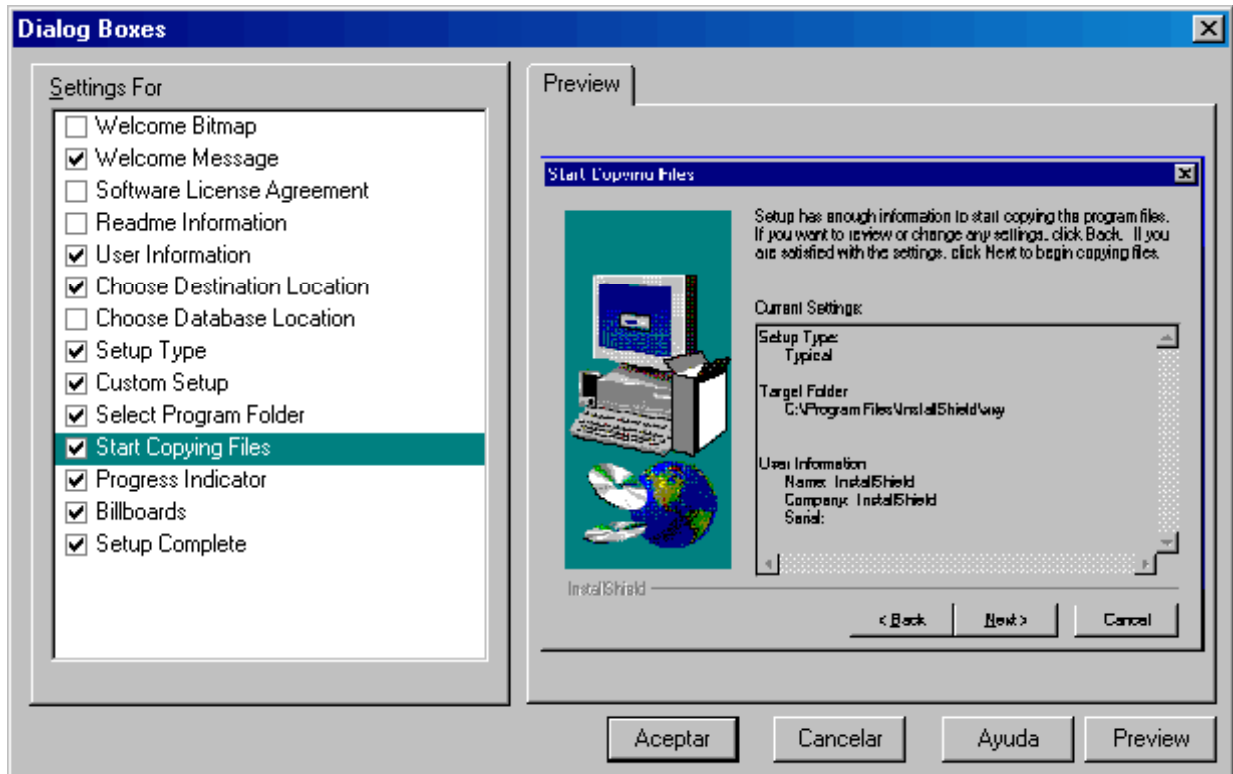
L'usuari pot escollir a quina carpeta s'instal·larà l'aplicació



L'usuari pot escollir el tipus d'instal·lació

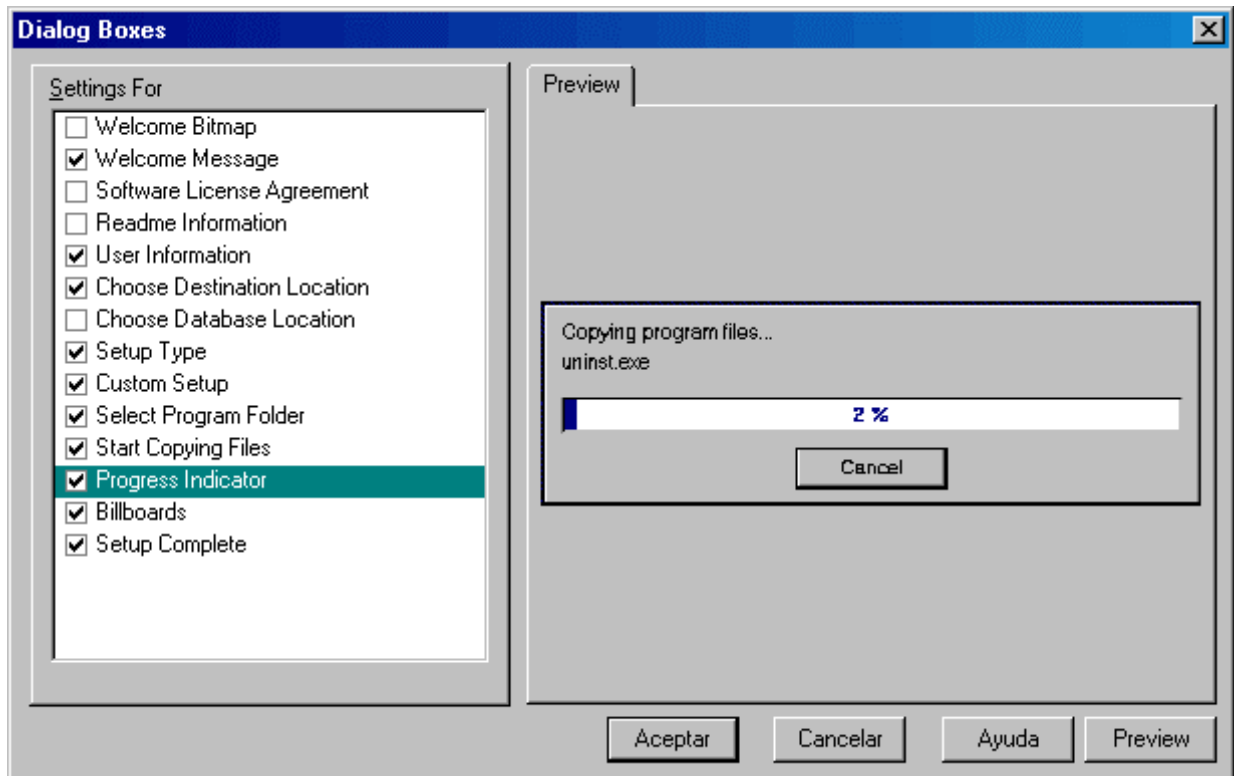


L'usuari pot escollir la carpeta del menú d'inici on apareixeran les icones de l'aplicació

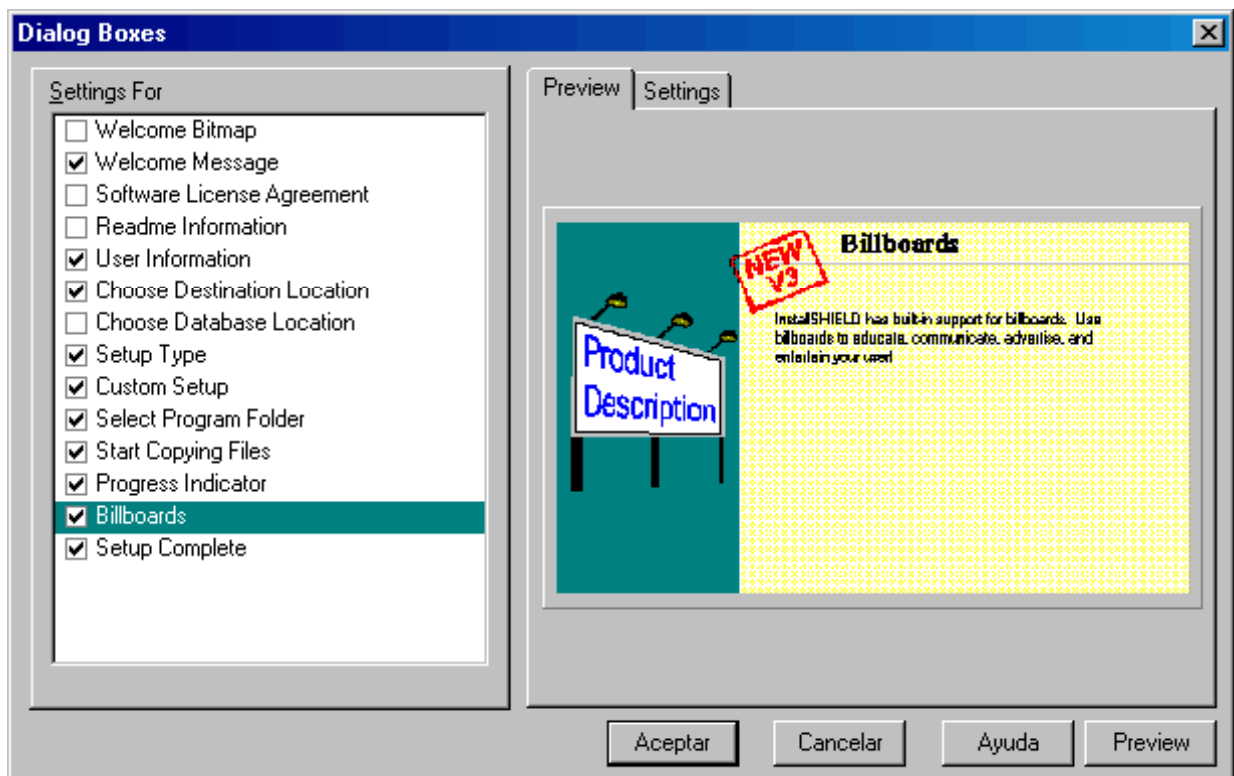


L'usuari veurà un resum de les opcions que a escollit per a la instal·lació





L'usuari veurà una barra de progrés de la instal·lació

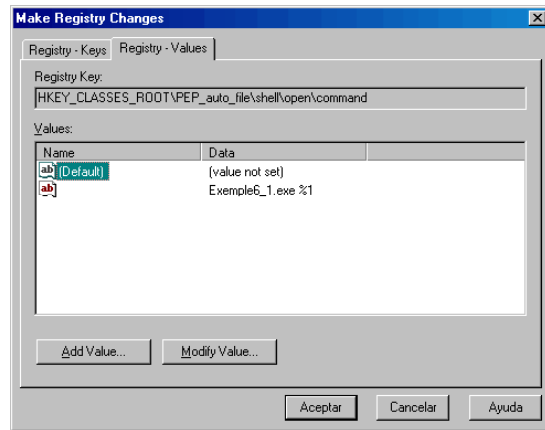
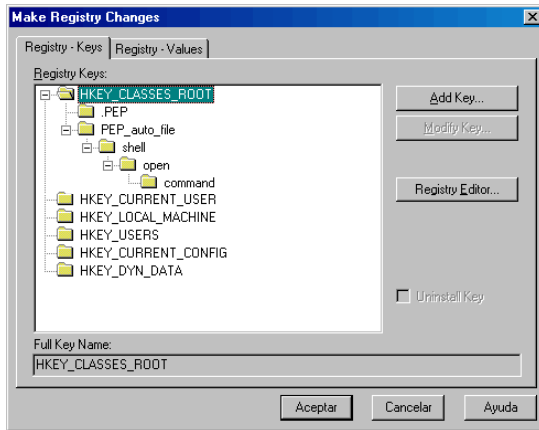


Pantalles per entretenir l'usuari mentre dura la instal·lació

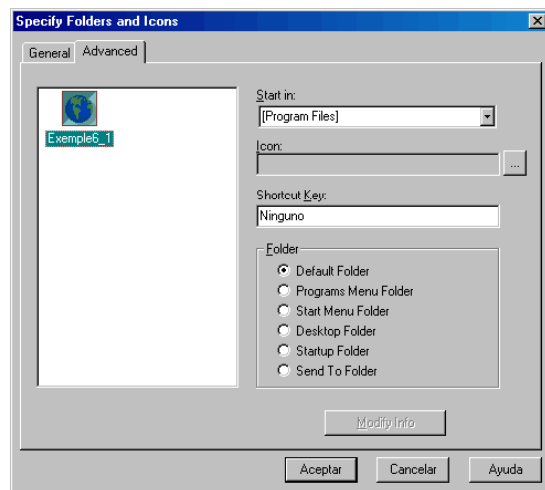
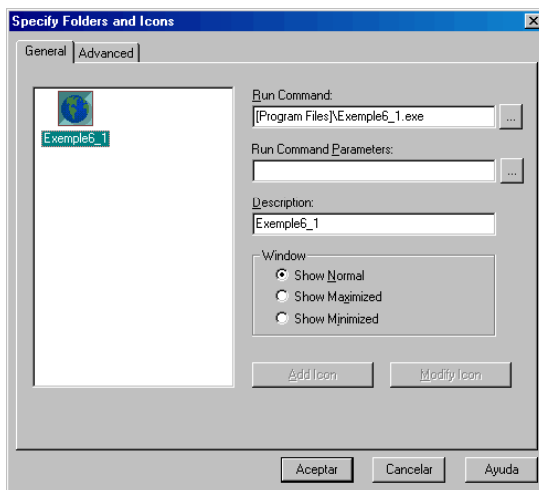


## Programació Visual amb Delphi

### Creació d'una aplicació

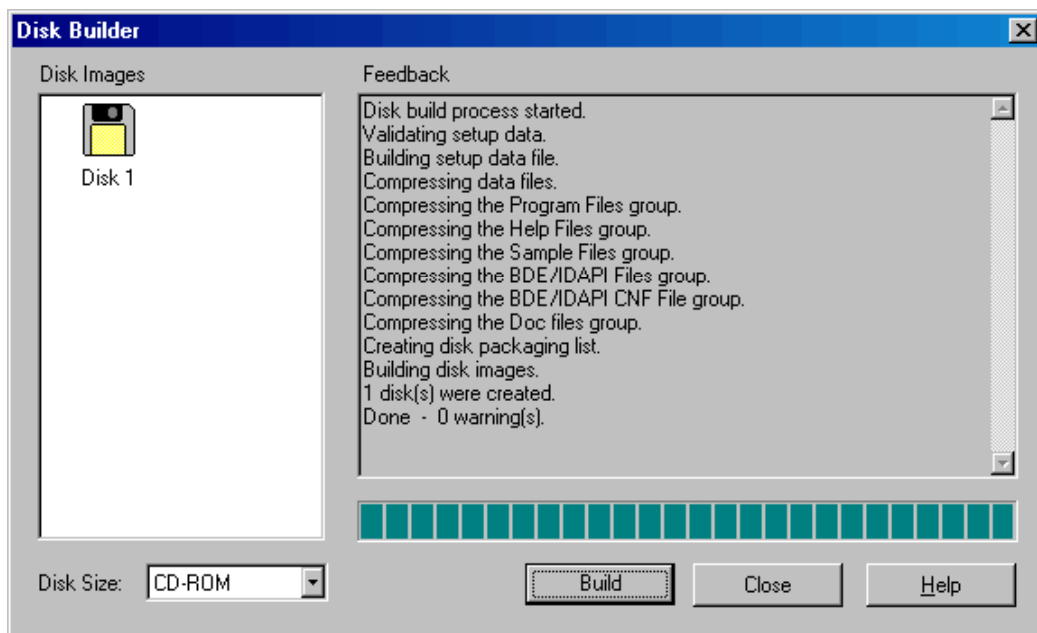


Realitzar canvis al registre. Es pot emprar, per exemple, per associar una extensió de fitxer a una icona i a la nostra aplicació, o per especificar la configuració inicial de la nostra aplicació.



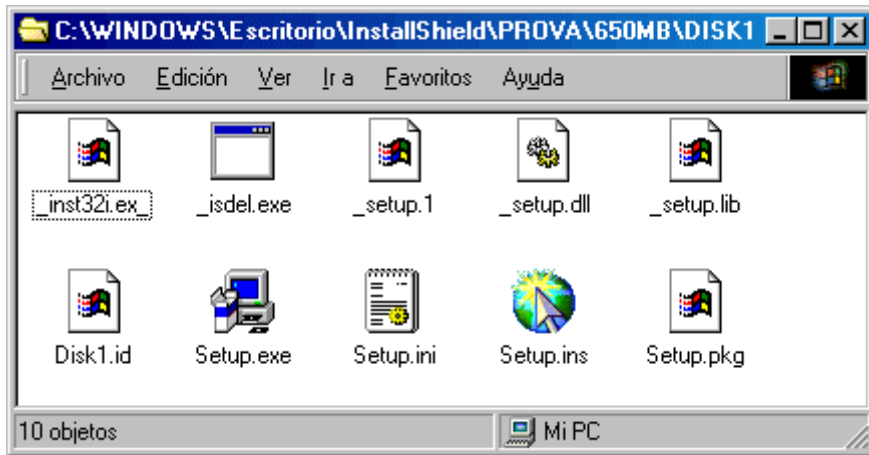
Especificar paràmetres de la línia de comandes de l'aplicació.

Especificar on apareixerà una icona addicional de l'aplicació



Genera les imatges i fitxers del programa d'instal·lació, a partir de les opcions especificades als quadres de diàleg anteriors.

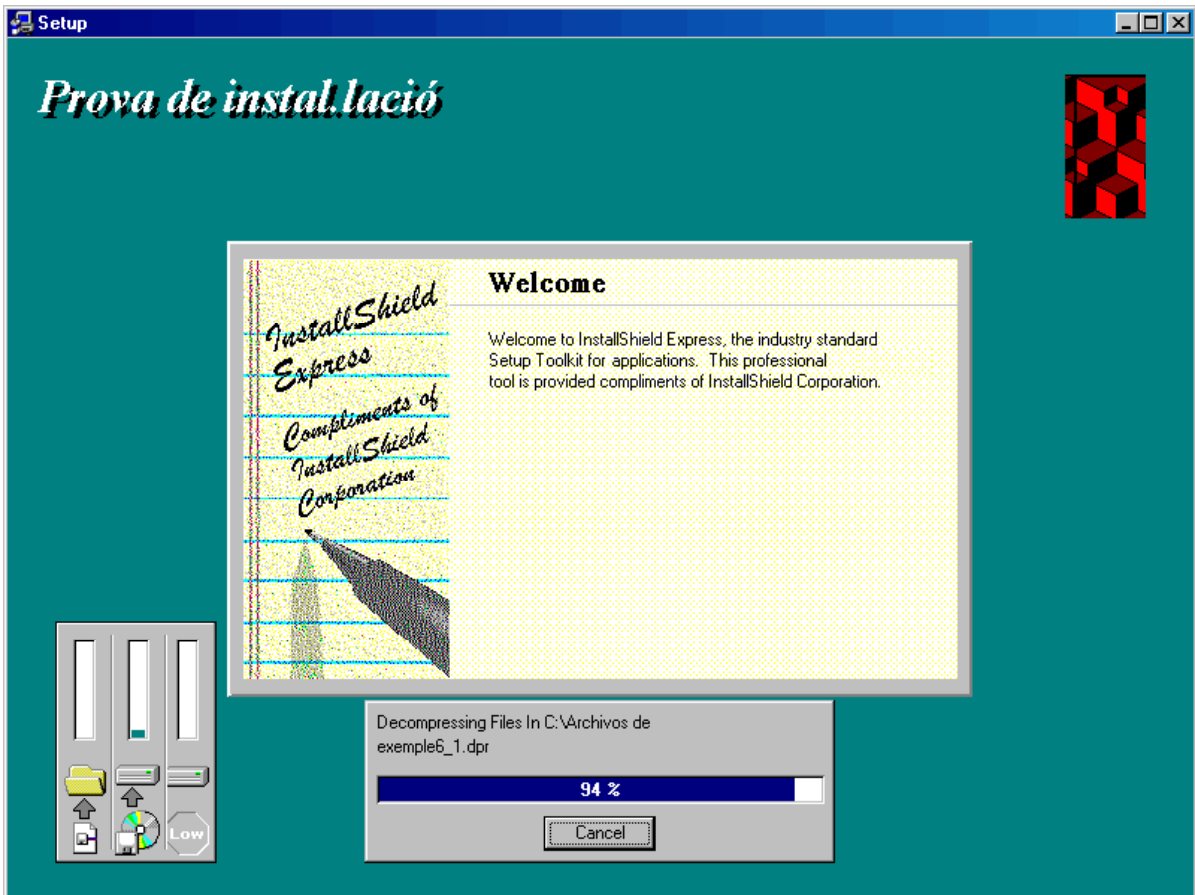




Imatges generades



Prova



### La nostre aplicació 'Editor de Text'

51. Repetint el procés que acabeu de veure, creeu un assistent d'instal·lació per l'aplicació que, seguint els passos del tutorial, heu creat. Els fitxers a instal·lar poden ser l'executable, els fitxers d'ajuda, els fitxers de codi font, etc.

## **Exercici 6-1: creació d'una aplicació estàndard**

Repetiu per a l'exercici 5-2 (adquisició de dades pel port sèrie) els passos del tutorial que heu seguit a aquest capítol. L'aplicació que obtindreu haurà de complir els següents requisits:

- Treballar internament amb llistes d'imatges i llistes d'accions.
- Comptar amb un menú principal i una barra d'eines.
- Comptar amb una barra d'estat, que s'emprarà per visualitzar el temps que dura l'experiment, i els resultats dels càlculs de la mitjana i la desviació.
- Possibilitat de intercanviar dades amb altres aplicacions mitjançant el portafolis. Les dades a intercanviar poden ser les dades de l'experiment en mode text i la gràfica.
- Possibilitat d'obrir fitxers de l'aplicació arrossegant-los des de l'escriptori.
- Possibilitat de llençar l'aplicació fent doble clic a qualsevol fitxer associat.
- Possibilitat d'impressió.
- Possibilitat de treballar amb més d'un document alhora. Cada document MDI utilitzarà un port sèrie diferent. D'aquesta manera l'aplicació pot rebre dades de varis experiments alhora.
- Comptar amb un menú d'ajuda.
- Instal·lar-se en un nou ordinador mitjançant un assistent d'instal·lació.

# Llibreries d'enllaç dinàmic i l'API de Windows

## El sistema operatiu Windows i les finestres

El sistema operatiu Windows, en les seves diverses variants (Windows 95, 98, Me, NT i XP), utilitza els conceptes de finestres i pas de missatges. Parlant planerament, una finestra és una taula de dades a memòria que defineix les característiques visibles i invisibles del que veiem com a finestra a la pantalla. Quan creem un formulari amb algun llenguatge de programació visual, estem creant una finestra. Les propietats d'aquesta finestra, com l'amplada i l'alçada, es guarden a una taula interna. És feina del sistema operatiu tractar amb els assumptes de la interfície d'usuari dels que el programador no hauria de preocupar-se, com per exemple conèixer d'una determinada finestra quan té el focus d'atenció, quan està a primer pla, quan s'ha de redibuixar, quan es tancada, quan es minimitzada o maximitzada, etc.

Quan escrivim un programa no ens hem de preocupar en determinar quan succeeixen aquests esdeveniments. Quan passen, el sistema operatiu ja li notifica a la finestra mitjançant missatges. Un missatge és una petita porció de dades (com un tipus de variable definit per l'usuari) i el seu significat ve definit pel seu valor numèric. Els missatges no són només els mitjans de comunicació entre el sistema operatiu i les finestres, també són el mitjà de comunicacions entre diferents finestres i entre qualsevol aplicació i qualsevol finestra.

Totes les finestres són identificades per un manipulador o *handle*, que és un nombre sencer de 32 bits que el sistema operatiu assigna a una finestra quan aquesta es crea. Quan volem enviar un missatge a una finestra, hem d'especificar aquest manipulador a la rutina que envia el missatge. Exemples d'altres entitats de Windows identificades per un manipulador són mòduls, processos, fils d'execució, marcs, menús, imatges de mapa de bits, icones, cursors i l'espai de colors.

Un formulari i qualsevol component que es registra com a finestra, tenen un manipulador i esdeveniments associats com *MouseMove*, *MouseDown*, *MouseUp*, *Paint*, *Refresh*, etc. Aquests esdeveniments són rutines que es criden automàticament quan la finestra (formulari, component visual, etc.) rep un missatge. Per exemple, quan un usuari fa clic sobre una finestra amb el botó esquerre del ratolí, el sistema operatiu ho detecta i genera un missatge *WM\_LBUTTONDOWN* indicant a la corresponent aplicació quina acció a succeït i especificant sobre quina finestra mitjançant el corresponent manipulador. Un cop l'aplicació lliura el missatge a la finestra, es dispara l'esdeveniment *MouseDown* d'aquesta. Els missatges de Windows proporcionen el mecanisme d'interacció emprat per lliurar entrada a varis objectes representats pel seu corresponent manipulador, bé sigui aquesta entrada accions de l'usuari, canvis del sistema o informació directa enviada d'una aplicació a una altre. La clau per que el pas de missatges funcioni és conèixer el manipulador de l'entitat a la que s'ha de lliurar el missatge.

El sistema de missatges d'una aplicació del sistema operatiu Windows consta de tres components principals:

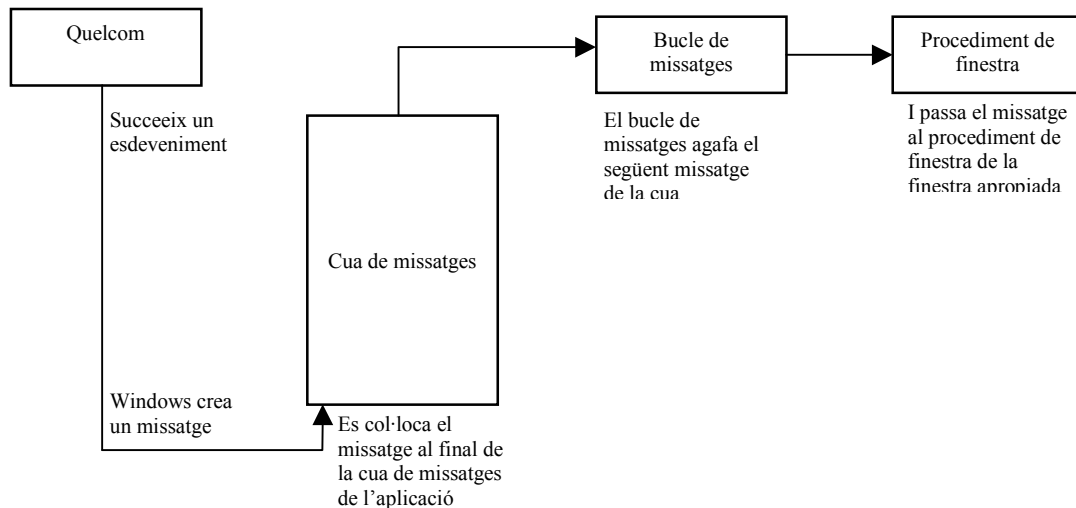
- Cua de missatges. Windows manté una cua de missatges per a cada aplicació. Una aplicació de Windows ha d'agafar missatges d'aquesta cua i despatxar-los cap a la finestra apropiada.
- Bucle de missatges. És el mecanisme iteratiu d'un programa de Windows que va a buscar un missatge de la cua de l'aplicació i ho despatxa cap a la finestra apropiada, va a buscar el següent missatge i ho despatxa cap a la finestra apropiada, i així successivament.
- Procediment de finestra. Cada finestra de l'aplicació té un procediment de finestra que rep cada missatge passat a través del bucle de missatges. La feina del procediment de finestra és agafar cada missatge per la finestra i respondre conseqüentment. Després de processar el missatge, usualment retorna un valor al sistema operatiu Windows.

Des del moment que succeeix algun esdeveniment i es crea un missatge fins que una finestra de la nostre aplicació respon al missatge, es completa el següent procés de cinc passos (veure figura 7.1):

1. Succeeix algun esdeveniment en el sistema.

2. El sistema operatiu Windows transforma aquest esdeveniment en un missatge i el col·loca en la cua de missatges de la nostre aplicació.
3. La nostre aplicació recupera el missatge de la cua i el col·loca en un registre del tipus TMSG.
4. La nostre aplicació traspasa el missatge al procediment de finestra de la finestra apropiada de la nostre aplicació.
5. El procediment de finestra realitza alguna acció en resposta al missatge.

Els passos 3 i 4 estructuraven el bucle de missatges de l'aplicació. Si no hi ha missatges a la cua de la nostre aplicació, Windows permet a les altres aplicacions processar els seus missatges.



**Figura 7.1** El sistema de missatges de Windows.

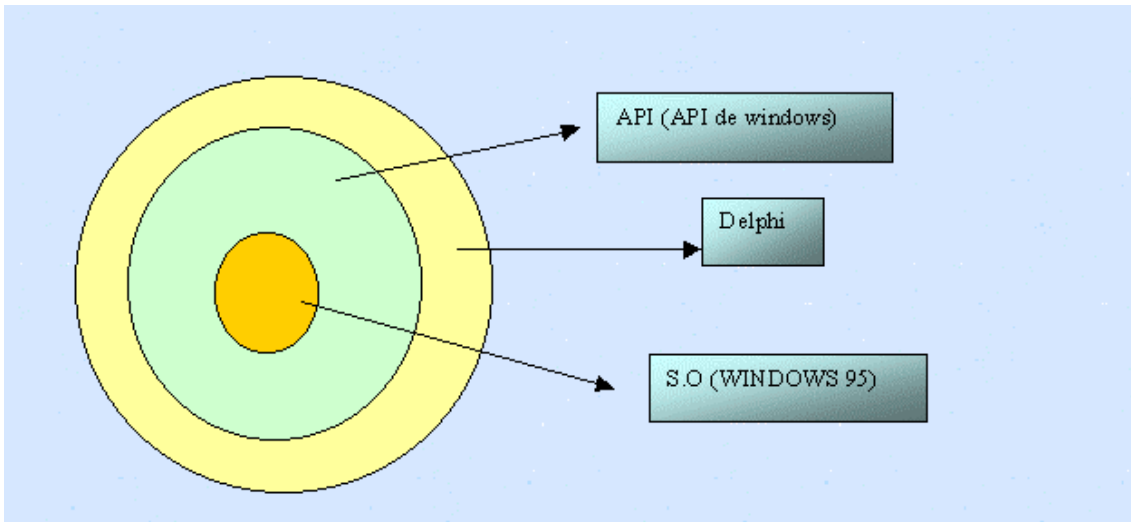
## Les DLL de Windows

Una DLL o biblioteca de vincles dinàmics (*Dinamic Link Library*) és un conjunt de procediments, externs a una aplicació, als que aquesta aplicació pot cridar. Les DLL no estan enllaçades a l'arxiu executable de l'aplicació i, així doncs, poden vincular-se en temps d'execució en lloc de carregar-se en temps de compilació. Això vol dir que les biblioteques poden actualitzar-se independentment de l'aplicació i que múltiples aplicacions poden compartir una mateixa DLL.

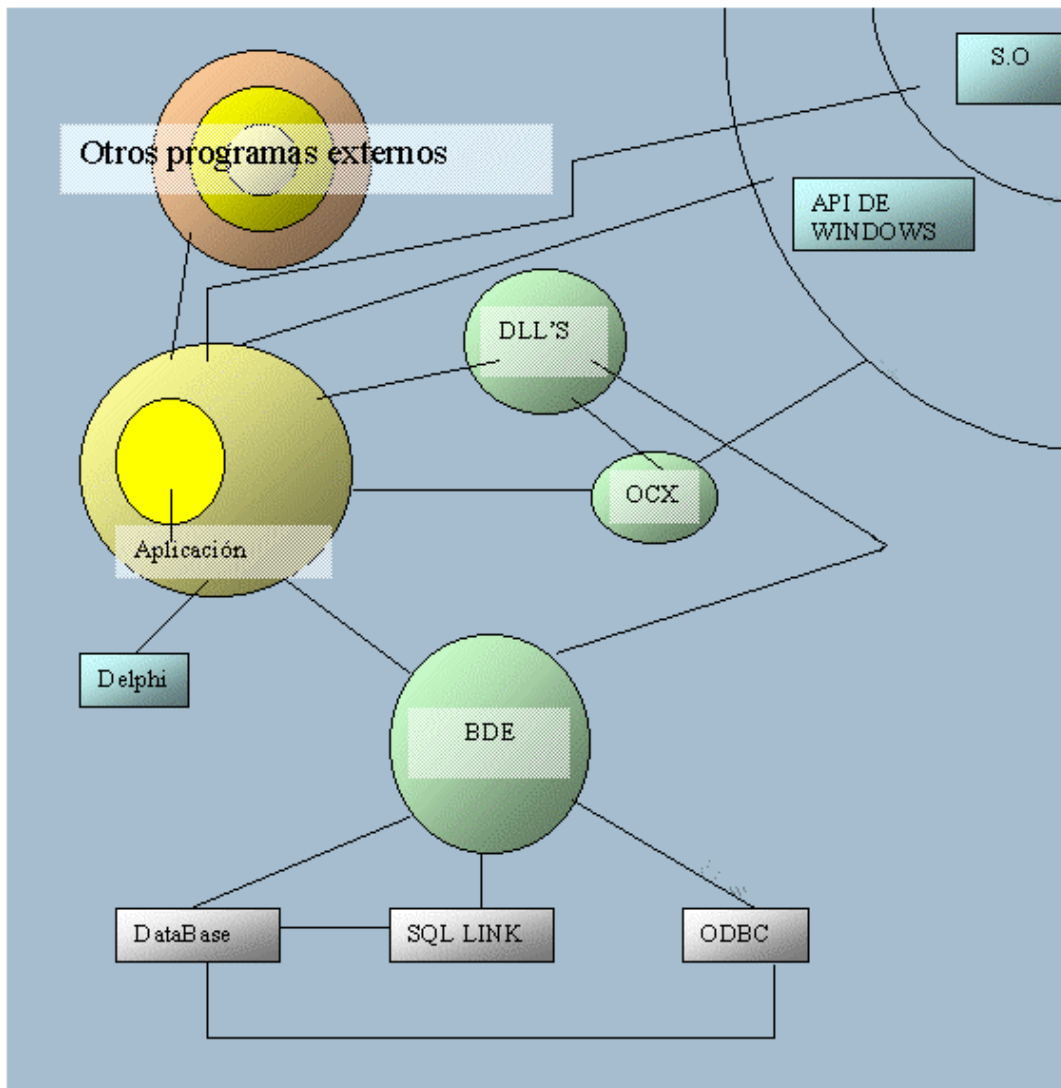
Les versions de 32 bits del sistema operatiu Windows (Windows 95, 98, Me, NT i XP) consten de tres DLL principals que proporcionen la major part de la funcionalitat de l'API de Windows. Aquestes són User32.dll, GDI32.dll i Kernel32.dll, però hi han moltes altres DLL que amplien la funcionalitat dels llenguatges de programació.

## L'API de Windows

API són les sigles d'*Application Programming Interface*. L'API de Windows és un enorme conjunt de rutines en les que es recolzen els desenvolupadors de software per accedir als diferents serveis oferts pel sistema operatiu a l'hora de crear una aplicació. Això inclou serveis com les finestres de diàleg més comunes, impressió i gràfics. Les API són els blocs amb que es construeix una aplicació de Windows, són llibreries (una mena de caixa d'eines) que formen una capa d'ajuda al programador (veure figures 7.2 i 7.3).



**Figura 7.2** La nostra aplicació escrita en Delphi, un cop compilada, està formada per instruccions en ensamblador i crides a l'API de Windows. L'API de Windows, a la seva vegada, està formada per instruccions en ensamblador i crides al nucli del sistema operatiu Windows.



**Figura 7.3** Delphi, no es basa únicament en l'API de Windows, sinó que també utilitza el BDE (*Borland Database Engine*) i DLL particulars.

L'API de Windows es pot agrupar en vuit àrees funcionals:

- *Maneig de finestres* (també anomenada *User*), que tracta les tasques relacionades amb l'administració de l'aplicació, finestres, menús, controls i quadres de diàleg. Les seves funcions les proporciona la llibreria User32.dll. Com a exemple, es pot cridar al procediment *FlashWindow* d'aquesta API per fer que una finestra parpellegi.
- *Serveis del sistema* (també anomenada *Kernel*), que s'encarrega de les tasques del sistema operatiu i té el màxim control del hardware. Les seves funcions permeten a les aplicacions manegar memòria i d'altres recursos, accedir a fitxers, carpetes i dispositius d'entrada/sortida, suportar execució multitasca i multifil, registrar esdeveniments, i manipular errors i excepcions. La majoria de les seves funcions les proporciona la llibreria Kernel32.dll. Com a exemple, es pot cridar al procediment *GetWindowsDirectory* d'aquesta API per obtenir la ruta d'accés a la carpeta de Windows.
- *Interfície de dispositius gràfics* (també anomenada *GDI*), que s'encarrega de les tasques de presentació gràfica, permetent a les finestres dibuixar imatges, línies, text, manegar el color i les fonts tipogràfiques. Les seves funcions les proporciona la llibreria Gdi32.dll.
- *Serveis multimèdia*, que permet a les aplicacions incorporar elements multimèdia, com so i vídeo. Les seves funcions les proporcionen la llibreria del sistema multimèdia Mmsystem.dll, la llibreria de vídeo per Windows Msvfw32.dll, la llibreria de compressió d'àudio Msacm32.dll, i d'altres.
- *Controls i diàlegs comuns*, que proporciona una col·lecció de controls predefinitos a través de la llibreria Comctl32.dll i una col·lecció de quadres de diàleg estàndard a través de la llibreria Comdlg32.dll.
- *Característiques de l'entorn*, que dona suport a l'associació de fitxers a l'extensió de fitxers, a arrossegat fitxers dins una aplicació, a associar icones a fitxers, etc. Les seves funcions les proporciona la llibreria Shell32.dll.
- *Característiques d'internacionalització*, que dona suport al conjunt de caràcters Unicode de 16 bits per treballar amb símbols tècnics i els símbols alfabètics de diferents idiomes. Les seves funcions les proporciona la llibreria Imm32.dll.
- *Serveis de xarxa*, que permet la comunicació entre aplicacions a diferents ordinadors d'una xarxa. Les seves funcions les proporcionen la llibreria d'interfície de xarxa Mpr.dll (treball en xarxa de Windows), la llibreria del sistema bàsic d'entrada/sortida en xarxa Netapi32.dll (NetBIOS), la llibreria d'intercanvi dinàmic de dades a través de xarxa Nddeapi.dll (DDE), la llibreria del servei d'accés remot Rasapi32.dll (RAS), la llibreria de sockets Wsock32.dll i Ws2\_32.dll (Winsock), i d'altres.

Les tres primeres estructures, que com hem comentat es troben als arxius del sistema operatiu User32.dll, GDI32.dll i Kernel32.dll, són les més importants i formen part del nucli de Windows.

| Procediment de l'API de Windows          | Finalitat  |
|--|--|
| <i>BringWindowToTop, SetActiveWindow</i> | Establir el focus d'atenció en una determinada finestra.                                 |
| <i>DragAcceptFiles, DragFinish</i>       | Proporcionar les característiques d'arrossegar i col·locar arxius.                       |
| <i>ExtractIcon, DrawIcon i LoadIcon</i>  | Manipular icones.  |
| <i>FindExecutable</i>                    | Cercar i recuperar el nom de l'arxiu executable associat a un determinat arxiu.          |
| <i>FindWindow, ShowWindow</i>            | Comprovar si una aplicació determinada està o no en execució actualment.                 |
| <i>GetActiveWindow, IsWindow</i>         | Determinar quan una funció del nucli ha terminat de carregar un programa.                |
| <i>GetSystemDirectory</i>                | Obtenir la ruta de accés del directori del sistema de Windows.                           |
| <i>GetSystemMetrics</i>                  | Obtenir l'alt i l'amplada dels elements de pantalla de Microsoft Windows.                |
| <i>GetTempFileName</i>                   | Obtenir un nom d'arxiu temporal i una ruta d'accés mitjançant la variable d'entorn TEMP. |



|   |  |
|---|--|
| <i>GetWindowPlacement,</i><br><i>SetWindowPlacement</i> | Obtenir o establir l'estat de presentació i les posicions normal (restaurada), minimitzada i maximitzada d'una finestra. |
| <i>GetWindowText</i>                                    | Obtenir el títol d'una finestra o el text d'un control, donat un manipulador de finestra.                                |
| <i>SendMessage, PostMessage</i>                         | Enviar missatges de Windows per controlar aplicacions. Hi ha centenars de missatges diferents.                           |

**Taula 7.1** Alguns procediments de l'API de Windows emprats sovint. Per obtenir informació addicional sobre els procediments de l'API de Windows, pot resultar útil la següent publicació: *Microsoft Windows Programmer's Reference*, publicat per Microsoft Press.

## Exemples basats en preguntes freqüents d'alumnes

### Com podem crear finestres transparents?

Per aconseguir finestres totalment transparents no cal emprar crides a l'API de Windows. Quan creem la finestra hem d'afegir les següents dues línies de codi: `Brush.Style := bsClear` i `BorderStyle := bsNone`; i redefinir el seu mètode `CreateParams`. El codi quedaria semblant a:

```
type
  TForm1 = class(TForm)
    ...
    procedure CreateParams(var Params: TCreateParams); override;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.CreateParams(var Params: TCreateParams);
begin
  inherited CreateParams(Params);
  Params.ExStyle := WS_EX_TRANSPARENT;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Brush.Style := bsClear;      // fons transparent
  BorderStyle := bsNone;     // sense barra de finestra
end;
```

### Com podem crear finestres amb formes irregulars?

Emprant la funció `SetWindowRgn` de l'API de Windows, que estableix l'àrea de la finestra on el sistema operatiu permet dibuixar. Qualsevol part de la finestra fora d'aquesta àrea no serà dibuixada. Si cerqueu ajuda sobre aquesta funció, trobareu que la seva sintaxi és:

```
int SetWindowRgn(
    HWND  hWnd,      // handle a la finestra de la que establirem la regió
    HRGN  hRgn,     // handle de la regió
    BOOL  bRedraw   // indica si a continuació es redibuixa la finestra
);
```

Per exemple, per establir una finestra amb forma d'el·lipse podem emprar la instrucció `SetWindowRgn(Handle, CreateEllipticRgn(0,0,100,200), True)` a l'esdeveniment `onCreate` del formulari.

Un exemple amb una regió més complexa seria el següent:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Reg1, Reg2, Reg: HRGN;
  Apoint: array [0..2] of TPoint;
begin
  // Crea una regió triangular
  APoint[0] := Point(40, ClientHeight - 30);
  APoint[1] := Point(30, ClientHeight);
  APoint[2] := Point(20, ClientHeight - 20);
  Reg1 := CreatePolygonRgn(Apoint, 3, WINDING);

  // Crea una regió rectangular amb cantonades rodones
  Reg2 := CreateRoundRectRgn(Width - ClientWidth, Height - ClientHeight,
    ClientWidth, ClientHeight - 10, 55, 55);

  // Uneix les dues regions creades en una de sola
  Reg := CreateRectRgn(150, 50, 200, 100);
  CombineRgn(Reg, Reg1, Reg2, RGN_OR);

  // Estableix la nova regió de la finestra
  SetWindowRgn(handle, Reg, true);

  // Esborra les regions creades, excepte l'emprada per establir la regió
  // de la finestra, perquè ara aquesta regió pertany al sistema operatiu
  DeleteObject(Reg1);
  DeleteObject(Reg2);
end;

// Si fem clic sobre la finestra la podrem arrossegar,
// encara que no es vegi la barra de títol d'aquesta.
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  begin
    ReleaseCapture;
    Perform(WM_syscommand, $F012, 0);
  end;
end;
```

### Com podem canviar l'aparença dels controls de la finestra?

Aquest punt està pendent d'elaboració. Part falta i part és incorrecte (només serveix per alguns controls).

Quan veiem aplicacions com *WinAmp*, de seguida ens preguntem com podem canviar l'aparença dels diferents controls visuals de l'aplicació per tal d'aconseguir interfícies com aquesta.



El sistema operatiu Windows és l'encarregat de dibuixar a pantalla tots aquests controls de la nostra aplicació que anomenem controls estàndard: opcions de menú, caixes d'edició, botons, etc.. Si canviem la propietat `OwnerDraw` d'un control a `True`, llavors Windows ja no dibuixarà més el control, sinó que generarà esdeveniments per a cada element visible del control, i serà l'aplicació la responsable de gestionar aquests esdeveniments per dibuixar els elements. És aquesta característica la que ens permetrà

alterar l'aparença del control. La finestra pare d'un control d'aquest tipus rebrà missatges WM\_DRAWITEM quan una porció del control hagi de ser dibuixada. Llavors emprarem un gestor d'esdeveniment per canviar la mida dels elements visibles, si fos necessari, i dibuixar-los.

Un exemple per dibuixar una opció d'un menú contextual seria el següent:

```

procedure TForm1.mnuBoldMeasureItem(Sender: TObject; ACanvas: TCanvas;
                                     var Width, Height: Integer);
begin
  Width := 140;
end;

procedure TForm1.mnuBoldDrawItem(Sender: TObject; ACanvas: TCanvas;
                                  ARect: TRect; Selected: Boolean);
var ImgID: integer;
begin
  if Selected then
    ACanvas.Brush.Color := clHighlight
  else
    ACanvas.Brush.Color := clMenu;

  // move the rect to make place for the side bar
  ARect.Left := 20;
  ACanvas.FillRect(ARect);

  if mnuBold.Checked then
    ImgID := 1 // thumb up in the ImageList
  else
    ImgID := 0; // thumb down in the ImageList

  ImageList1.Draw(ACanvas, 22, ARect.Top + 2, ImgID);
  ACanvas.Font.Style := [fsBold];

  // user defined text drawing function:
  DrawItemText(45, ACanvas, ARect, 'Bold');
end;

```

In the OnMeasureItem event we set the size of owner-draw menu items. The OnDrawItem event contains parameters indicating the index of the item to draw, the rectangle in which to draw, and usually some information about the state of the item (such as whether the item has focus - is selected). This event is fired when Windows first displays the items and each time the status changes (mouse moves over an item). Take a look at the following OnDrawItem procedure for the "Bold" menu item (the other five handlers for menu items use similar code): The ACanvas parameter provides a drawing surface on which to draw the menu item. The Selected parameter indicates whether the menu item is selected and therefore the menu item's background is filled with the appropriate color (clSelected if selected, clMenu otherwise). If the Checked property of menu item is True (indicating the Bold font for the Memo component) the "Thumb-Up" picture is displayed by painting it with the Draw method of the ImageList component. Finally, the word "Bold" is drawn onto the Canvas with the DrawItemText procedure.

### **Com podem comprovar si ja s'està executant una còpia d'una aplicació**

Una manera és emprar la funció FindWindow de l'API de Windows, que donat el nom d'una classe finestra i el seu Caption comprova si existeix alguna finestra visible amb aquestes característiques. Retorna zero si no la troba o el *handle* de la finestra si la troba. A continuació, la funció SetForegroundWindow de l'API de Windows, activa la finestra donat el seu *handle*, si aquesta no es troba minimitzada. Per exemple, al fitxer de projecte podem escriure:

```

var
  Hwnd: THandle;
begin
  Hwnd := FindWindow('TForm1', nil);
  if Hwnd = 0 then
    begin
      Application.Initialize;
    end;

```

```
Application.CreateForm(TForm1, Form1);
Application.Run;
end
else
SetForegroundWindow(Hwnd)
end.
```

Una altre manera és emprar un objecte d'exclusió mútua, o *mutex*. En executar una aplicació aquesta pot crear un *mutex* amb un nom donat, i després comprovar si aquest ja pertanyia a alguna aplicació mitjançant la funció `WaitForSingleObjectWindows` de l'API de Windows. Si el *mutex* no tenia propietari, l'aplicació es converteix en el seu propietari. En cas contrari, l'aplicació espera que transcorri un cert temps i després retorna un codi d'error. Per exemple, al fitxer de projecte podem escriure:

```
var
hMutex: THandle;
begin
HMutex := CreateMutex(nil, False, 'ElMeuMutex');
if WaitForSingleObject(hMutex, 0) <> WAIT_TIMEOUT then
begin
Application.Initialize;
Application.CreateForm(TForm1, Form1);
Application.Run;
end;
end.
```

### **Com podem obrir un document amb l'aplicació associada per defecte?**

Emprant la funció `ShellExecute` de l'API de Windows, que permet obrir, imprimir i executar fitxers. La seva sintaxi és:

```
HINSTANCE ShellExecute(
    HWND hwnd,           // handle a la finestra pare
    LPCTSTR lpOperation, // string que especifica l'operació a realitzar
    LPCTSTR lpFile,      // string amb el nom del fitxer o carpeta
    LPCTSTR lpParameters, // string amb els paràmetres del fitxer executable
    LPCTSTR lpDirectory, // string que especifica el directori per defecte
    INT nShowCmd         // whether file is shown when opened
);
```

El paràmetre `lpOperation` pot tenir els valors 'open', 'print' o 'explore'. Per exemple, si volem obrir el navegador d'Internet amb una determinada pàgina web, podem emprar la instrucció `ShellExecute(Handle, 'open', 'http://www.upc.es', nil, nil, SW_SHOWNORMAL)`.

### **Com podem tancar el sistema operatiu?**

Emprant la funció `ExitWindowsEx` de l'API de Windows:

|   |                                     |
|---|-------------------------------------|
| <code>ExitWindowsEx(EWX_LOGOFF, 0)</code>   | Tanca la sessió                     |
| <code>ExitWindowsEx(EWX_SHUTDOWN, 0)</code> | Apaga el sistema.                   |
| <code>ExitWindowsEx(EWX_POWEROFF, 0)</code> | Apaga el sistema i tanca la corrent |
| <code>ExitWindowsEx(EWX_REBOOT, 0)</code>   | Reinicia el sistema.                |

### **Com podem obtenir informació de la memòria disponible a Windows?**

Emprant el procediment `GlobalMemoryStatus` de l'API de Windows, retorna informació tant de la memòria física com de la virtual. Un exemple del seu ús seria:

```
var MS: TMemoryStatus;
begin
GlobalMemoryStatus(MS);
LabelPhysMem.Caption := 'Memòria física total: ' +
    FormatFloat('#,###" KB"', MS.dwTotalPhys/1024);
LabelAvailMem.Caption := 'Memòria física disponible: ' +
```

```
FormatFloat('#,###" KB"', MS.dwAvailPhys/1024);
LabelFreeRes.Caption := 'Percentatge de memòria en ús:' +
Format('%d %%', [MS.dwMemoryLoad]);
end;
```

## Varis

Més exemples de crides a l'API de Windows:

```
// Tanquem una aplicació de Windows (la calculadora)
if IsWindow(FindWindow(nil, 'Calculadora')) = True then
  PostMessage(FindWindow(nil, 'Calculadora'), WM_QUIT, 0, 0);

// Canviem la configuració de la data de Windows
SetLocaleInfo(LOCALE_USER_DEFAULT, LOCALE_SSHORTDATE, 'dd/mm/aa');

// Realitzem un desplaçament en un control TRichEdit
SendMessage(RichEdit1.Handle, EM_LINESCROLL, 0, RichEdit1.Lines.Count-1);

// Obrim un fitxer .bmp amb la seva aplicació associada
ShellExecute(Self.Handle, 'open', 'ondas.bmp', nil, nil, sw_showdefault);

// Desactivem el botó de tancat d'un formulari
DeleteMenu(GetSystemMenu(Handle, false), SC_CLOSE, MF_BYCOMMAND)

// Ejecutar qualsevol comanda del command.com
// En aquest cas copiem el fitxer autoexec.bat a prova.txt
WinExec('c:\command.com /c copy c:\autoexec.bat c:\prova.txt',
  SW_SHOWNORMAL);

// Despleguem un control TComboBox
SendMessage(ComboBox1.Handle, CB_SHOWDROPDOWN, 1, 1);

// Comprovar si estan actives la tecla de inserció, ...
if (GetKeyState(VK_INSERT) and 1) > 0 then ...
if (GetKeyState(VK_CAPITAL) and 1) > 0 then ...
if (GetKeyState(VK_NUMLOCK) and 1) > 0 then ...

// Reiniciem Windows
if not ExitWindow(EW_RestartWindows, 0) then
  ShowMessage('Una aplicació no va poder ser tancada');

// Apaguem el sistema
if not ExitWindow(EW_RebootSystem, 0) then
  ShowMessage('Una aplicació no va poder ser tancada');
```

## Tècniques de gràfics i so

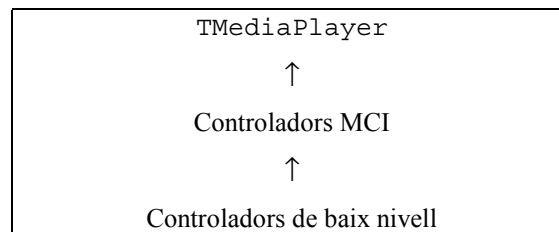
### El component MediaPlayer



#### Definició

El component `TMediaPlayer` o “reproductor multimèdia” permet a l’aplicació reproduir i controlar dispositius multimèdia (un dispositiu multimèdia pot ser hardware o software), com per exemple: lector de CD-ROM, escàner, seqüenciador MIDI, enregistrador i reproductor de vídeo, etc.

El reproductor multimèdia inclou un conjunt de botons (Back, Eject, Next, Pause, Play, Prev, Record, Step i Stop) per controlar el dispositiu multimèdia.



#### Propietats

- **DeviceType:** Especifica el tipus de dispositiu multimèdia (`dtAutoSelect`, `dtAVIVideo`, `dtCDAudio`, `dtDAT`, `dtDigitalVideo`, `dtMMMovie`, `dtOther`, `dtOverlay`, `dtScanner`, `dtSequencer`, `dtVCR`, `dtVideodisc`, `dtWaveAudio`).
- **Capabilities:** Especifica quines operacions pot fer i quines no el reproductor multimèdia (obrir, reproduir, enregistrar, tenir finestra, pas endavant, i pas enrera).
- **EnabledButtons** i **VisibleButtons:** Especifiquen, respectivament, quins botons del reproductor es poden utilitzar i quins són visibles.
- **AutoEnable:** Indica si el reproductor activa i desactiva automàticament els botons.
- **Display** i **DisplayRect:** Especifiquen, respectivament, quin control s’emprarà per visualitzar la sortida i quina àrea rectangular d’aquest control.
- **Error** i **ErrorMessage:** Especifiquen, respectivament, el codi i el text de l’error produït durant l’execució d’un mètode de control del dispositiu multimèdia.
- **FileName:** Conté el nom del fitxer a ser obert amb el mètode `Open` i guardat amb el mètode `Save`.
- **Frames:** Especifica el nombre de quadres a avançar amb el mètode `Step` i a retrocedir amb el mètode `Back`. Per defecte és el 10% de la longitud total del medi.
- **Length:** Conté la longitud del medi a un dispositiu multimèdia obert.
- **Position:** Especifica la posició actual dins el medi carregat. Si el medi consta de diverses pistes, llavors indica la posició dins la primera pista.
- **StartPos** i **EndPos:** Especifiquen, respectivament, les posicions a les que començarà i parerà de reproduir o enregistrar el reproductor.

- **Tracks, TrackLength i TrackPosition:** La primera propietat indica el nombre de pistes reproduïbles al dispositiu multimèdia obert. La segona i la tercera indiquen, respectivament, la longitud i la posició d'inici de la pista especificada per un índex.
- **Mode:** Indica l'estat d'un dispositiu multimèdia obert (`mpNotReady`, `mpStopped`, `mpPlaying`, `mpRecording`, `mpSeeking`, `mpPaused`, `mpOpen`).
- **Notify i NotifyValue:** La primera propietat indica si es genera l'esdeveniment `OnNotify` després de l'execució d'un mètode de control del dispositiu multimèdia. La segona propietat informa de quin ha estat el resultat de la crida a l'esmentat mètode: comanda executada satisfactòriament, comanda cancel·lada per l'usuari, comanda fallida i comanda sobreixida per una altre comanda.
- **Wait:** Indica si després de llençar l'execució d'un mètode de control del dispositiu multimèdia es retorna el control a l'aplicació o s'espera a que aquest mètode acabi la seva execució.
- **Shareable:** Indica si vàries aplicacions poden compartir o no el dispositiu multimèdia.
- **TimeFormat:** Determina el format emprat per especificar informació sobre la posició (propietats `StartPos`, `Length`, `Position`, `Start`, `EndPos`, etc.). Els formats disponibles són: `tfHMS`, `tfMSF`, `tfFrames`, `tfSMPTTE24`, `tfSMPTTE25`, `tfSMPTTE30`, `tfSMPTTE30Drop`, `tfBytes`, `tfMilliseconds` i `tfTMSF`.

### Esdeveniments

- **OnClick:** Es produeix quan l'usuari selecciona un botó del reproductor. Per defecte es crida al mètode corresponent.
- **OnPostClick:** Es produeix després de seleccionar un botó del reproductor o després de realitzar l'acció associada per defecte, segons sigui el valor de la propietat `Wait`.
- **OnNotify:** Es produeix després d'executar-se un mètode (`Back`, `Close`, `Eject`, `Next`, `Open`, `Pause`, `PauseOnly`, `Play`, `Previous`, `Resume`, `Rewind`, `StartRecording`, `Step` o `Stop`) quan la propietat `Notify` conté el valor `True`.

### Mètodes

- **Play:** Reprodueix el contingut del dispositiu multimèdia.
- **StartRecording:** Inicia l'enregistrament des de la posició actual o des de la posició especificada per la propietat `StartPos`.
- **Pause i Resume:** Pausa i reprèn, respectivament, la reproducció o enregistrament al dispositiu multimèdia.
- **Stop:** Atura la reproducció o enregistrament al dispositiu multimèdia.
- **Back i Step:** Avancen cap enrera o cap endavant, respectivament, el nombre de quadres especificat per la propietat `Frames`.
- **Next i Previous:** Desplacen el reproductor, respectivament, cap a l'inici de la pista següent o la pista anterior del medi.
- **Rewind:** Posiciona el reproductor a l'inici del medi, la posició del qual ve especificat per la propietat `Start`.
- **Eject:** Expulsa el medi del dispositiu multimèdia obert.
- **Open i Close:** Obre i tanca, respectivament, el dispositiu multimèdia.
- **Save:** Guarda el medi carregat dins un fitxer especificat per la propietat `FileName`.

### Exemple

```
OpenDialog1.DefaultExt := 'AVI';  
OpenDialog1.FileName := '*.avi';  
if OpenDialog1.Execute then
```

```
with MediaPlayer1 do
begin
  MediaPlayer1.FileName := OpenFileDialog1.FileName;
  DeviceType := dtAVIVideo;
  Display := Panell1;
  VisibleButtons := [btPlay, btStop];
  try
    Open;
    EndPos := TrackLength[1] div 2;
    Play;
  except
    MessageDlg('Error Code: ' + IntToStr(Error) + #13#10 + ErrorMessage,
              mtError, [mbOk], 0);
  end;
end;
```

## Transparències

- Propietat DeviceType: dtAutoSelect, dtDAT, dtAVIVideo, dtCDAudio, dtDigitalVideo, dtMMMovie, dtScanner, dtSequencer, dtVCR, dtVideodisc, dtWaveAudio, dtOther, ...
- Propietat DeviceId: nombre enter que identifica el dispositiu.
- Propietat FileName: fitxer que conté la informació a reproduir o a gravar. No tot dispositiu multimèdia treballa amb fitxers.
- Nou botons: cada botó realitza una acció per defecte.

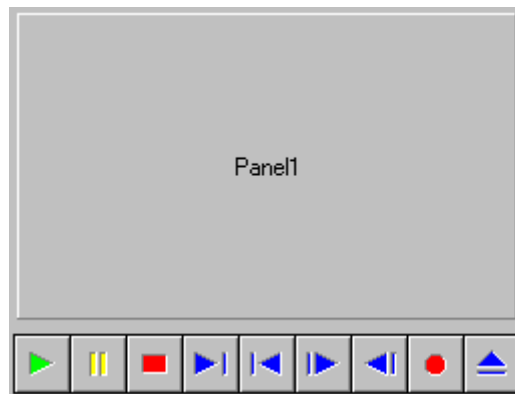


Propietats `VisibleButtons`, `EnabledButtons` i `ColoredButtons`.

- Obrir dispositiu → Treballar → Tancar
  - Manual: mètodes `Open()` i `Close()`.
  - Automàtic: propietat `AutoOpen := True`.
- Mètodes DeviceType: `Play`, `Pause`, `Resume`, `Stop`, `Next`, `Previous`, `Step`, `Back`, `StartRecording`, `Eject`, `Rewind`, `Save`, ...
- Compartir: propietat `Shareable := True`.
- Errors: excepció `EMCIDeviceError` i propietats `Error` i `ErrorMessage`.
- Possibilitats del dispositiu multimèdia: propietat `Capabilities`: `[mpCanStep, mpCanEject, mpCanPlay, mpCanRecord, mpUsesWindow]`.
- Estat del dispositiu multimèdia: propietat `Mode`: `(mpNotReady, mpStopped, mpPlaying, mpRecording, mpSeeking, mpPaused, mpOpen)`.



- Dispositius amb finestra: propietat Display.



- Dispositius amb pistes: propietats Tracks, TrackPosition i TrackLenght.
- Formats de temps: propietat TimeFormat: (tfMilliseconds, tfHMS, tfMSF, tfTMSF, tfFrames, tfSMPTE24, tfBytes, tfSamples, ...).
- Propietats StartPos, EndPos, Length i Position.

```
MediaPlayer1.StartPos := MCI_MAKE_TMSF(2,1,30,0);
```

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MediaPlayer1.FileName := 'c:\Windows\Media\Ding.wav';
  try
    MediaPlayer1.Open;
    MediaPlayer1.Play;
    MediaPlayer1.Close;
  except
    ShowMessage('Error');
  end;
end;
```



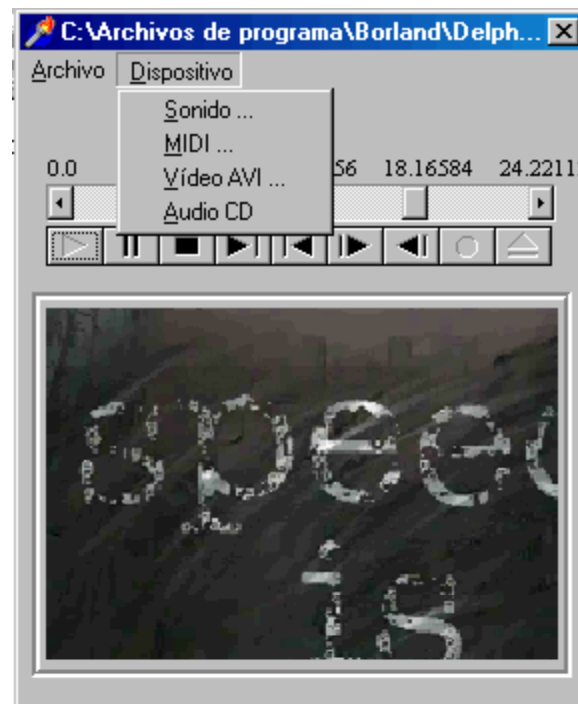
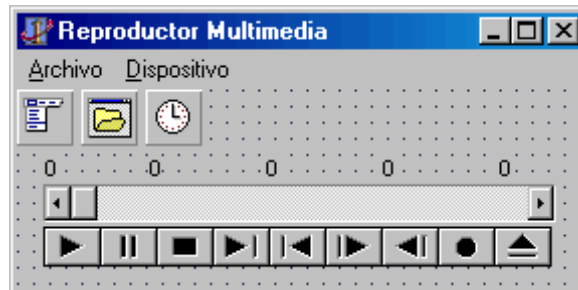
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MediaPlayer1.FileName := 'c:\Windows\Media\Ding.wav';
  try
    MediaPlayer1.Open;
    MediaPlayer1.Wait := True;
    MediaPlayer1.Play;
    MediaPlayer1.Close;
  except
    ShowMessage('Error');
  end;
end;
```



```
procedure TForm1.Button1Click(Sender: TObject);
begin
  MediaPlayer1.FileName := 'c:\Windows\Media\Ding.wav';
  try
    MediaPlayer1.Open;
    MediaPlayer1.Notification := True;
    MediaPlayer1.Play;
  except
    ShowMessage('Error');
  end;
end;
```

```
procedure TForm1.MediaPlayer1Notify(Sender: TObject);  
begin  
    MediaPlayer1.Close;  
end;
```

## Exemple 7-1: explorador multimèdia



```
// Al crear la ficha comprobaremos qué dispositivos son los que están  
// disponibles, eliminando del menú las opciones que correspondan.  
procedure TForm1.FormCreate(Sender: TObject);  
begin  
    with Control do  
        begin  
            // Intentamos abrir el dispositivo de audio  
            DeviceType := dtWaveAudio;  
            try  
                Open;  
                Close;  
            except  
                OpcionSonido.Visible := False;  
            end;  
  
            // El mateix amb els altres dispositius  
            ...  
        end  
    end;
```

```

    end;
end;

// Presenta el cuadro de diàlego para la apertura del archivo
procedure TForm1.AbreArchivo(Patron: String);
begin
    AbrirArchivo.Filter := Patron;
    if AbrirArchivo.Execute then
        begin
            Control.FileName := AbrirArchivo.FileName;
            Control.DeviceType := dtAutoSelect;
            AbrirDispositivo;
        end;
end;

// Este procedimiento se encargará de abrir los dispositivos.
procedure TForm1.AbrirDispositivo;
begin
    // Si la propiedad Tag tiene algún valor es porque tenemos la ficha abierta,
    // mostrando la pantalla de visualización de secuencias de vídeo
    if Tag <> 0 then
        begin
            Height := Tag;    // restablecemos el alto de la ficha
            Tag := 0;
        end;

    // Desactivamos la opción Salvar del menú Archivo
    OpcionSalvar.Enabled := False;

    with Control do
        try
            Open;
            TimeFormat := tfMilliseconds;
            if FileName = '' then
                Caption := Titulo
            else
                begin
                    Caption := FileName;
                    if LowerCase(ExtractFileExt(FileName)) = '.avi' then
                        begin
                            Display := Pantalla;
                            DisplayRect := Rect(0, 0, 319, 219);
                            // Guardamos el alto actual de la ficha en la propiedad Tag, y
                            // ampliamos la altura para permitir ver la pantalla de vídeo.
                            Form1.Tag := Form1.Height;
                            Form1.Height := Form1.Height+Pantalla.Height+32;
                        end
                    end;
                end;
            EstableceMedidas;
        except
            ShowMessage(ErrorMessage);
            Caption := Titulo;
        end;
    end;

// Función que ajustar los límites de la barra de desplazamiento
procedure TForm1.EstableceMedidas;
var N, Segundos, Decimas: Integer;
begin
    // La barra de desplazamiento se basará en segundos
    Posicion.Min := 0;
    Posicion.Max := Control.Length div 1000;
    Posicion.Position := 0;

    if Control.DeviceType <> dtCDAudio then
        // Si el dispositivo no es un CD calculamos
        // cinco puntos de tiempo y los mostramos

```

```
    for N := 1 to 5 do
    begin
        Segundos := Control.Length div 5 * (N-1) div 1000;
        Decimas := Control.Length div 5 * (N-1) - Segundos*100;
        (FindComponent('Punto'+IntToStr(N)) as TLabel).Caption :=
            IntToStr(Segundos) + '.' + IntToStr(Decimas);
        end
    else
    // En caso de que el dispositivo sea un CD
    begin
        // Mostramos el número de temas y la longitud del disco
        Punto1.Caption := '0s';
        Punto2.Caption := IntToStr(Control.Tracks)+ ' temas';
        if Control.Length < 3600000 then
            Punto4.Caption := IntToStr(Control.Length div 60000) + 'min.'
        else
            Punto4.Caption := '1h' + IntToStr((Control.Length-3600000) Div 60000) +
                'min.';
            Punto3.Caption := '';
            Punto5.Caption := '';
        end;
    end;
end;

// Cada vez que el Timer produzca un evento, indicamos que la modificación ha
// sido del programa y actualizamos la posición de la barra de desplazamiento.
procedure TForm1.EventoPosicionTimer(Sender: TObject);
begin
    Modificando := True;
    Posicion.Position := Control.Position div 1000;
end;

// Al pulsar sobre los botones del TMediaPlayer
procedure TForm1.ControlClick(Sender: TObject;
    Button: TMPBtnType; var DoDefault: Boolean);
begin
    // Activamos el Timer si se ha iniciado la reproducción
    EventoPosicion.Enabled := (Button = btPlay);

    // Si se ha pulsado el botón Record, activamos la opción
    // Salvar, para poder conservar las modificaciones
    if Button = btRecord then
        OpcionSalvar.Enabled := True;
end;

// Si recibimos una notificación del control multimedia
procedure TForm1.ControlNotify(Sender: TObject);
begin
    // Desactivamos el temporizador
    EventoPosicion.Enabled := False;
end;

// Si se ha cambiado la posición en la barra
procedure TForm1.PosicionChange(Sender: TObject);
begin
    if not Modificando then
        // Si el cambio lo ha producido el usuario
        Control.Position := Posicion.Position * 1000
    else
        // en caso contrario desactivamos el indicador
        // Modificado, puesto que ya se ha controlado el evento
        Modificando := False;
end;

// Al seleccionar la opción Salir
procedure TForm1.OpcionSalirClick(Sender: TObject);
begin
    Close;
end;
```

```
// Al seleccionar la opció Salvar
procedure TForm1.OpcionSalvarClick(Sender: TObject);
begin
    Control.Save; // Usamos el método Save
end;

// Si se selecciona sonido en formato WAVE
procedure TForm1.OpcionSonidoClick(Sender: TObject);
begin
    AbreArchivo('Archivos de audio (*.wav)|*.wav');
end;

// Si se selecciona música en formato MIDI
procedure TForm1.OpcionMIDIClick(Sender: TObject);
begin
    AbreArchivo('Archivos MIDI (*.mid)|*.mid');
end;

// Si se selecciona vídeo en formato AVI
procedure TForm1.OpcionVideoAVIClick(Sender: TObject);
begin
    AbreArchivo('Archivos de vídeo (*.avi)|*.avi');
end;

// Si se selecciona el CD
procedure TForm1.OpcionAudioCDClick(Sender: TObject);
begin
    Control.DeviceType := dtCDAudio;
    Control.FileName := '';
    AbrirDispositivo;
end;
```

## Accés i maneig de bases de dades

# Programació per Internet

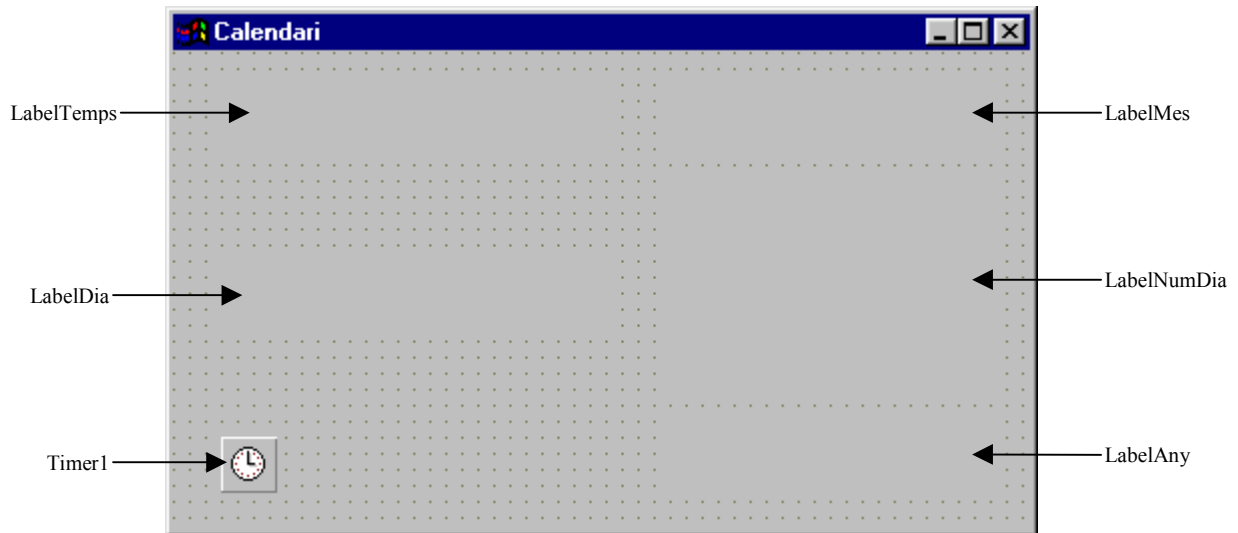
## Programació multifil



## Solucions als exercicis

### Exercici 3-1: calendari

Una possible solució seria la següent:



#### Propietats:

##### TForm **FormCalendari:**

|              |             |
|--------------|-------------|
| BorderStyle  | bsDialog    |
| Caption      | 'Calendari' |
| ClientHeight | 238         |
| ClientWidth  | 428         |

##### TLabel **LabelTemps:**

|             |                   |
|-------------|-------------------|
| Caption     |                   |
| Left        | 24                |
| Top         | 16                |
| Width       | 194               |
| Height      | 36                |
| Alignment   | taCenter          |
| AutoSize    | False             |
| Font.Color  | clWindowText      |
| Font.Height | 32                |
| Font.Name   | 'Times New Roman' |
| Font.Style  | fsBold            |

##### TLabel **LabelDia:**

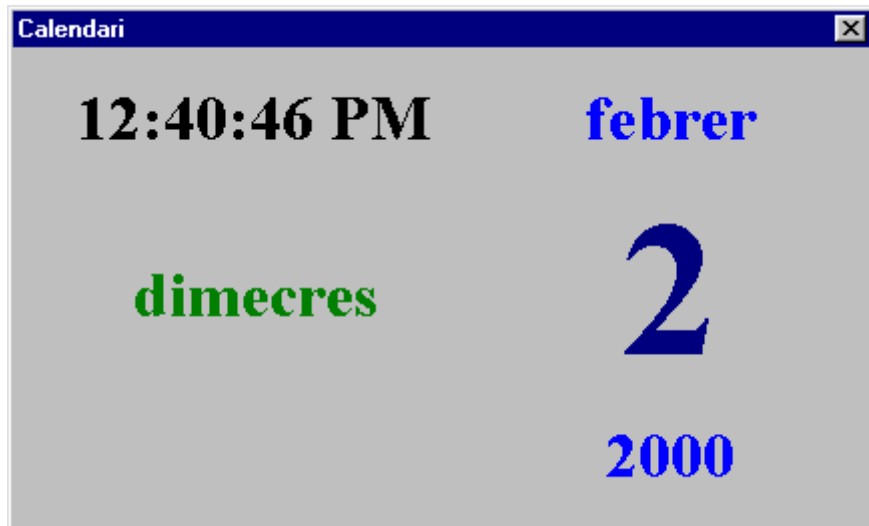
|             |          |
|-------------|----------|
| Caption     |          |
| Left        | 24       |
| Top         | 104      |
| Width       | 194      |
| Height      | 36       |
| Alignment   | taCenter |
| AutoSize    | False    |
| Font.Color  | clGreen  |
| Font.Height | 32       |

|                            |                   |
|----------------------------|-------------------|
| Font.Name                  | 'Times New Roman' |
| Font.Style                 | fsBold            |
| <b>TLabel LabelMes:</b>    |                   |
| Caption                    |                   |
| Left                       | 248               |
| Top                        | 16                |
| Width                      | 161               |
| Height                     | 36                |
| Alignment                  | taCenter          |
| AutoSize                   | False             |
| Font.Color                 | clBlue            |
| Font.Height                | 32                |
| Font.Name                  | 'Times New Roman' |
| Font.Style                 | fsBold            |
| <b>TLabel LabelNumDia:</b> |                   |
| Caption                    |                   |
| Left                       | 248               |
| Top                        | 64                |
| Width                      | 161               |
| Height                     | 109               |
| Alignment                  | taCenter          |
| AutoSize                   | False             |
| Font.Color                 | clNavy            |
| Font.Height                | 96                |
| Font.Name                  | 'Times New Roman' |
| Font.Style                 | fsBold            |
| <b>TLabel LabelAny:</b>    |                   |
| Caption                    |                   |
| Left                       | 248               |
| Top                        | 184               |
| Width                      | 161               |
| Height                     | 36                |
| Alignment                  | taCenter          |
| AutoSize                   | False             |
| Font.Color                 | clBlue            |
| Font.Height                | 32                |
| Font.Name                  | 'Times New Roman' |
| Font.Style                 | fsBold            |
| <b>TTimer Timer1:</b>      |                   |
| Interval                   | 1000              |
| OnTimer                    | Timer1Timer       |

Codi:

```
procedure TFormCalendari.Timer1Timer(Sender: TObject);  
var  
    Temps : TDateTime;  
begin  
    Temps := Now();  
    LabelNumDia.Caption := FormatDateTime('d', Temps);  
    LabelDia.Caption := FormatDateTime('dddd', Temps);  
    LabelMes.Caption := FormatDateTime('mmmm', Temps);  
    LabelAny.Caption := FormatDateTime('yyyy', Temps);  
    LabelTemps.Caption := FormatDateTime('h:nn:ss AM/PM', Temps);  
end;
```

Resultat:



Aquest resultat ha emprat els noms dels dies i dels mesos proporcionats pel sistema operatiu i, per tant, apareixen en el mateix idioma. Si volguéssim que els noms dels dies i dels mesos apareguessin en un idioma diferent, hauríem de canviar el codi de l'esdeveniment `TFormCalendari.Timer1Timer`.

Una altre interfície gràfica molt elegant per aquest exercici la podem aconseguir amb el component *Calendar*:



## Exercici 4-1: POO – creació d'una classe

Una possible solució seria la següent:

```
unit Experiment;  
  
interface  
  
uses SysUtils, Cua;  
  
type TExperiment = class
```

```
private
  Dades : TCua;
public
  constructor Create(Capacitat : integer);
  procedure Obtenir();
  function Mitjana():Real;
  function Desviacio():Real;
  destructor Destroy; override;
end;

implementation

constructor TExperiment.Create(Capacitat : integer);
begin
  inherited Create;
  Dades := TCua.Create(Capacitat);
end;

procedure TExperiment.Obtenir();
var
  x:TElement;
begin
  if Dades.Plena() then
    Dades.Treure(x);

    // de moment genero les dades aleatòriament
    x := random(100);

    Dades.Ficar(x);
end;

function TExperiment.Mitjana():Real;
var
  i, N : Integer;
  Suma : Real;
begin
  N := Dades.NumElements();
  if N > 0 then
    begin
      Suma := 0;
      for i := 1 to N do
        Suma := Suma + Dades.Element(i);
      Result := Suma/N
    end
  else
    Result := 0;
end;

function TExperiment.Desviacio():Real;
var
  i, N : Integer;
  Suma : Real;
begin
  N := Dades.NumElements();
  if N > 1 then
    begin
      Suma := 0;
      for i := 1 to N do
        Suma := Suma + Sqr(Dades.Element(i));
      Result := (Suma - N*Sqr(Mitjana())) / (N-1);
    end
  else
    Result := 0;
end;

destructor TExperiment.Destroy;
begin
  Dades.Destroy;
```

```

    inherited Destroy;
end;

end.

```

## Exercici 4-2: POO – herència

Una possible solució seria la següent:

```

program Exercici4_2;

{$APPTYPE CONSOLE}

uses SysUtils;

type TCercle = class
    private
        Radi : real;
    public
        constructor Create();
        procedure LlegirRadi();
        procedure VeureArea();
        procedure VeurePerimetre();
    end;

type TEsfera = class(TCercle)
    public
        constructor Create();
        procedure VeureArea();
        procedure VeureVolum();
    end;

type TCilindre = class(TCercle)
    private
        Alcada : real;
    public
        constructor Create();
        procedure LlegirAlcada();
        procedure VeureArea();
        procedure VeureVolum();
    end;

type TCilindreBuit = class(TCilindre)
    private
        RadiIntern : real;
    public
        constructor Create();
        procedure LlegirRadiIntern();
        procedure VeureArea();
        procedure VeureVolum();
    end;

constructor TCercle.Create();
begin
    inherited Create;
    LlegirRadi();
end;

procedure TCercle.LlegirRadi();
begin
    Write('Radi ? ');
    Readln(Radi);
end;

```

```
procedure TCercle.VeureArea();
begin
  Writeln('Area : ', PI*Radi*Radi :4:2);
end;

procedure TCercle.VeurePerimetre();
begin
  Writeln('Perimetre : ', 2*PI*Radi :4:2);
end;

constructor TEsfere.Create();
begin
  inherited Create;
end;

procedure TEsfere.VeureArea();
begin
  Writeln('Area : ', 4*PI*Radi*Radi :4:2);
end;

procedure TEsfere.VeureVolum();
begin
  Writeln('Volum : ', 4*PI*Radi*Radi*Radi/3 :4:2);
end;

constructor TCilindre.Create();
begin
  inherited Create;
  LlegirAlcada();
end;

procedure TCilindre.LlegirAlcada();
begin
  Write('Alcada ? ');
  Readln(Alcada);
end;

procedure TCilindre.VeureArea();
begin
  Writeln('Area : ', 2*PI*Radi*Alcada + 2*PI*Radi*Radi :4:2);
end;

procedure TCilindre.VeureVolum();
begin
  Writeln('Volum : ', PI*Radi*Radi*Alcada :4:2);
end;

constructor TCilindreBuit.Create();
begin
  inherited Create;
  LlegirRadiIntern();
end;

procedure TCilindreBuit.LlegirRadiIntern();
begin
  Write('Radi intern ? ');
  Readln(RadiIntern);
end;

procedure TCilindreBuit.VeureArea();
begin
  Writeln('Area : ', 2*PI*(Radi + RadiIntern)*Alcada +
    2*PI*(Radi*Radi - RadiIntern*RadiIntern) :4:2);
end;

procedure TCilindreBuit.VeureVolum();
begin
```

```

Writeln('Volum : ', PI*(Radi*Radi - RadiIntern*RadiIntern)*Alçada :4:2);
end;

var
  Ce : TCercle;
  Es : TEsfera;
  Ci : TCilindre;
  CiB : TCilindreBuit;

// Programa principal
begin
  Writeln('Creant un cercle ...');
  Ce := TCercle.Create();
  Ce.VeureArea();
  Ce.VeurePerimetre();
  Ce.Free;
  Writeln;

  Writeln('Creant una esfera ...');
  Es := TEsfera.Create();
  Es.VeureArea();
  Es.VeureVolum();
  Es.Free;
  Writeln;

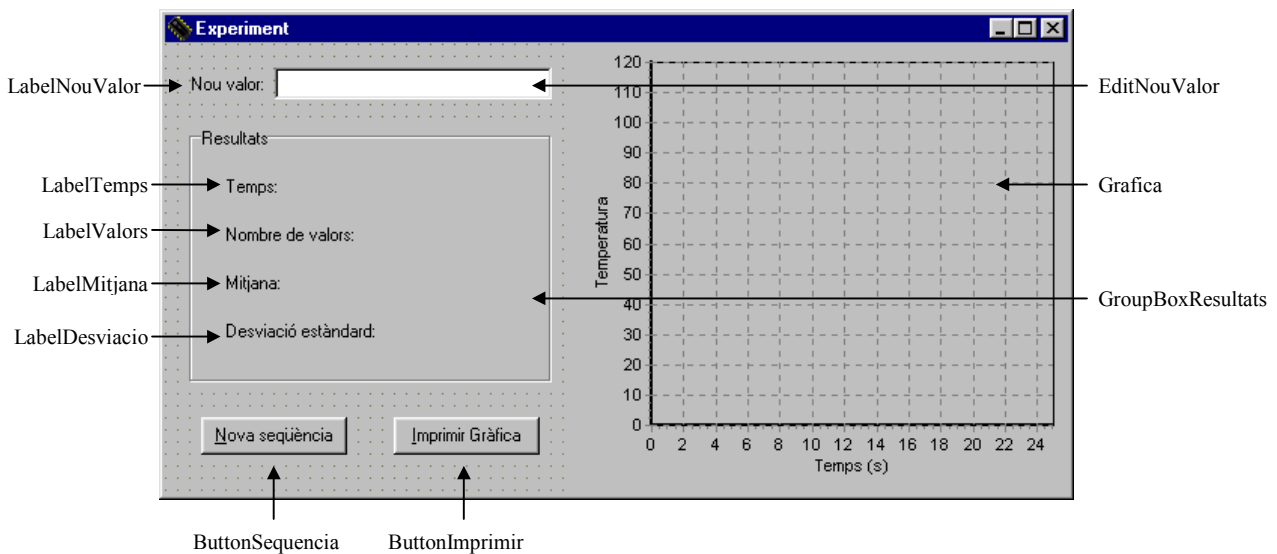
  Writeln('Creant un cilindre ...');
  Ci := TCilindre.Create();
  Ci.VeureArea();
  Ci.VeureVolum();
  Ci.Free;
  Writeln;

  Writeln('Creant un cilindre buit ...');
  CiB := TCilindreBuit.Create();
  CiB.VeureArea();
  CiB.VeureVolum();
  CiB.Free;
  Readln;
end.

```

## Exercici 5-1: càlcul de la mitjana i la desviació estàndard

Una possible solució seria la següent:



Propietats finestra principal:

**TForm FormExperiment:**

|                       |              |
|-----------------------|--------------|
| Left                  | 204          |
| Top                   | 262          |
| Caption               | 'Experiment' |
| Height                | 326          |
| Width                 | 608          |
| Constraints.MinHeight | 326          |
| Constraints.MinWidth  | 608          |

**TLabel LabelNouValor:**

|         |              |
|---------|--------------|
| Left    | 17           |
| Top     | 20           |
| Width   | 49           |
| Height  | 13           |
| Caption | 'Nou valor:' |

**TEdit EditNouValor:**

|            |                      |
|------------|----------------------|
| Left       | 73                   |
| Top        | 16                   |
| Width      | 184                  |
| Height     | 21                   |
| OnKeyPress | EditNouValorKeyPress |

**TGroupBox GroupBoxResultats:**

|         |                           |
|---------|---------------------------|
| Left    | 16                        |
| Top     | 56                        |
| Width   | 241                       |
| Height  | 169                       |
| Anchors | [akLeft, akTop, akBottom] |
| Caption | 'Resultats'               |

**TLabel LabelTemps:**

|         |          |
|---------|----------|
| Left    | 24       |
| Top     | 32       |
| Width   | 35       |
| Height  | 13       |
| Caption | 'Temps:' |

**TLabel LabelValors:**

|         |                     |
|---------|---------------------|
| Left    | 24                  |
| Top     | 64                  |
| Width   | 86                  |
| Height  | 13                  |
| Caption | 'Nombre de valors:' |

**TLabel LabelMitjana:**

|         |            |
|---------|------------|
| Left    | 24         |
| Top     | 96         |
| Width   | 37         |
| Height  | 13         |
| Caption | 'Mitjana:' |

**TLabel LabelDesviacio:**

|         |                        |
|---------|------------------------|
| Left    | 24                     |
| Top     | 128                    |
| Width   | 100                    |
| Height  | 13                     |
| Caption | 'Desviació estàndard:' |

**TButton ButtonSequencia:**

|      |    |
|------|----|
| Left | 24 |
|------|----|



|                                |                                    |
|--------------------------------|------------------------------------|
| Top                            | 248                                |
| Width                          | 97                                 |
| Height                         | 25                                 |
| Anchors                        | [akLeft, akBottom]                 |
| Caption                        | '&Nova seqüència'                  |
| OnClick                        | ButtonSequenciaClick               |
| <b>TButton ButtonImprimir:</b> |                                    |
| Left                           | 152                                |
| Top                            | 248                                |
| Width                          | 97                                 |
| Height                         | 25                                 |
| Anchors                        | [akLeft, akBottom]                 |
| Caption                        | '&Imprimir Gràfica'                |
| OnClick                        | ButtonImprimirClick                |
| <b>TChart Grafica:</b>         |                                    |
| Left                           | 272                                |
| Top                            | 0                                  |
| Width                          | 328                                |
| Height                         | 299                                |
| Anchors                        | [akLeft, akTop, akRight, akBottom] |
| AllowZoom                      | False                              |
| Title.Text.Strings             | ('')                               |
| Title.Visible                  | False                              |
| BottomAxis.Title.Caption       | 'Temps (s)'                        |
| LeftAxis.Automatic             | False                              |
| LeftAxis.Maximum               | 120                                |
| LeftAxis.Title.Caption         | 'Temperatura'                      |
| Legend.Visible                 | False                              |
| View3D                         | False                              |
| <b>TLineSeries Serie:</b>      |                                    |
| Marks.Visible                  | True                               |
| SeriesColor                    | clRed                              |

Codi finestra principal:

```

const MaxDades = 5;

var
  e: TExperiment;
  t: Double;

// Procediment que inicialitza les variables i el text del formulari
procedure TFormExperiment.Inicialitzar();
begin
  e := TExperiment.Create(MaxDades);
  t := 0;
  Serie.Clear();
  LabelTemps.caption := 'Temps: 0 s.';
  LabelValors.caption := 'Nombre de valors: 0';
  LabelMitjana.caption := 'Mitjana: ';
  LabelDesviacio.caption := 'Desviació estàndard: ';
  EditNouValor.text := '';
end;

// Procediment que calcula la mitjana i la desviació estàndard
procedure TFormExperiment.Actualitzar();
var N: integer;
begin
  // Actualitzem el temps
  LabelTemps.caption := 'Temps: ' + FloatToStr(t) + ' s.';

  // Actualitzem el nombre de valors

```

```
N := e.NumDades();
LabelValors.caption := 'Nombre de valors: ' + FloatToStr(N);

// Actualitzem la mitjana
if N > 0 then
  LabelMitjana.caption := 'Mitjana: ' + FloatToStr(e.Mitjana());

// Actualitzem la desviació estàndard
if N > 1 then
  LabelDesviacio.caption := 'Desviació estàndard: ' +
    FloatToStr(e.Desviacio);

if serie.count = MaxDades then
  Serie.Delete();
  Serie.AddXY(t, StrToFloat(EditNouValor.text), '', clRed);
end;

// Quan es crea el formulari, fem una crida a la rutina d'inicialització.
procedure TFormExperiment.FormCreate(Sender: TObject);
begin
  Inicialitzar();
end;

// Quan seleccionem aquest botó, fem una crida a la rutina d'inicialització.
procedure TFormExperiment.ButtonSequenciaClick(Sender: TObject);
begin
  e.Free;
  Inicialitzar();
  EditNouValor.SetFocus();
end;

// Quan seleccionem aquest botó, imprimim la gràfica.
procedure TFormExperiment.ButtonImprimirClick(Sender: TObject);
begin
  Grafica.Print();
end;

// Quan l'usuari prem ENTER, acceptem el valor i calculem els resultats.
procedure TFormExperiment.EditNouValorKeyPress(Sender: TObject; var Key: Char);
begin
  if key = #13 then
    try
      t := t + 1;
      e.Obtenir( StrToInt(EditNouValor.text) );
      Actualitzar();
      EditNouValor.text := '';
    except
      on EConvertError do ShowMessage('Valor incorrecte!');
    end;
  end;
end;
```



Propietats about box:

|                              |                |
|------------------------------|----------------|
| <b>TAboutBox AboutBox:</b>   |                |
| ActiveControl                | EditNouValor   |
| BorderStyle                  | bsDialog       |
| Caption                      | 'Experiment'   |
| ClientHeight                 | 213            |
| ClientWidth                  | 298            |
| Position                     | poScreenCenter |
| <b>TLabel LabelNouValor:</b> |                |
| Left                         | 17             |
| Top                          | 20             |
| Width                        | 49             |
| Height                       | 13             |
| Caption                      | 'Nou valor:'   |
| <b>TEdit EditNouValor:</b>   |                |
| Left                         | 120            |
| Top                          | 120            |
| Width                        | 121            |
| Height                       | 21             |
| PasswordChar                 | '*'            |
| <b>TButton OKButton:</b>     |                |
| Left                         | 111            |
| Top                          | 180            |
| Width                        | 75             |
| Height                       | 25             |
| Caption                      | 'OK'           |
| Default                      | True           |
| OnClick                      | OKButtonClick  |

...

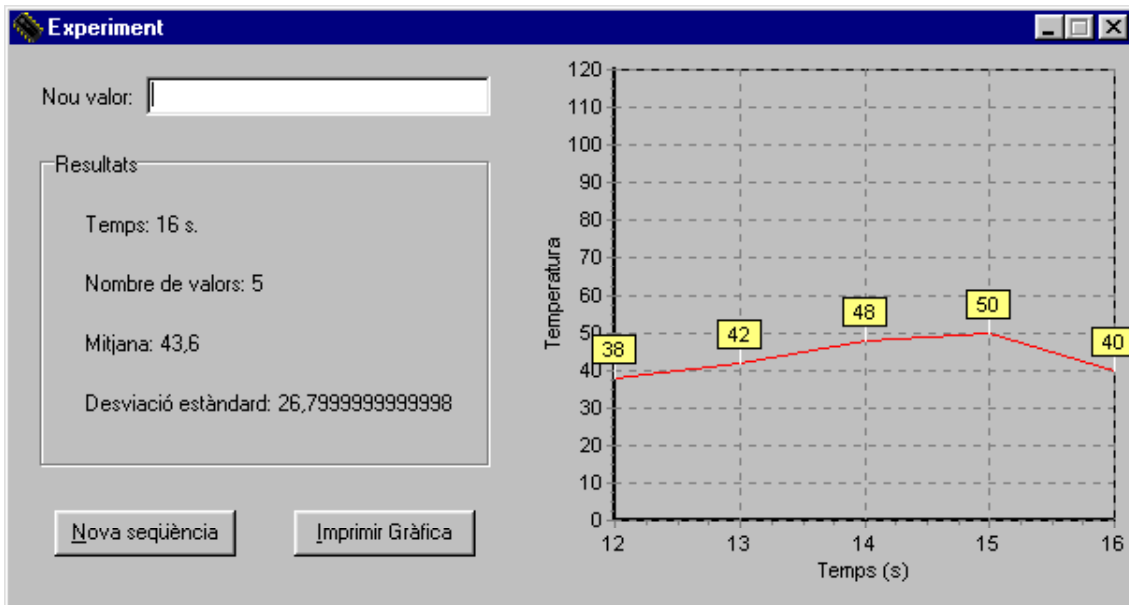
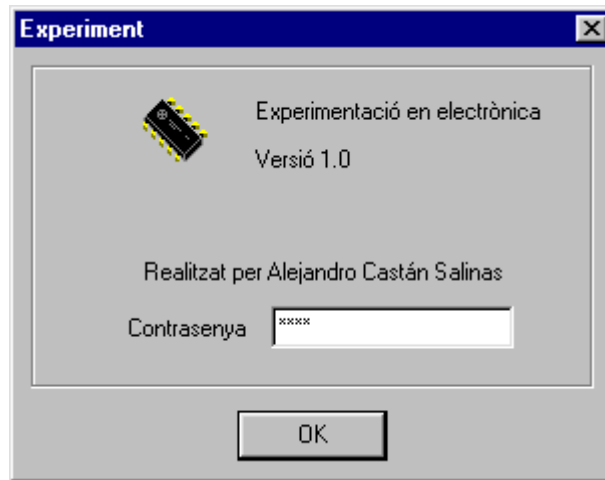
Codi about box:

```
// Comprova si la contrasenya és correcta.
procedure TAboutBox.OKButtonClick(Sender: TObject);
const
  CONTRASENYA = 'Alex';
  NUM_INTENTS = 3;
begin
  if Edit1.Text <> CONTRASENYA then
    begin
      Tag := Tag + 1;
      if Tag = NUM_INTENTS then
        begin
          MessageDlg('Accés denegat'+#13+'Fi de programa', mtError, [mbOk], 0);
          ModalResult := mrCancel;
        end
      else
        begin
          MessageDlg('Intent ' + IntToStr(Tag) + #13 +
            'Contrasenya incorrecta', mtWarning, [mbOk], 0);
          Edit1.Text := '';
          Edit1.SetFocus;
        end
      end
    else
      ModalResult := mrOK;
  end;
```

Codi programa principal:

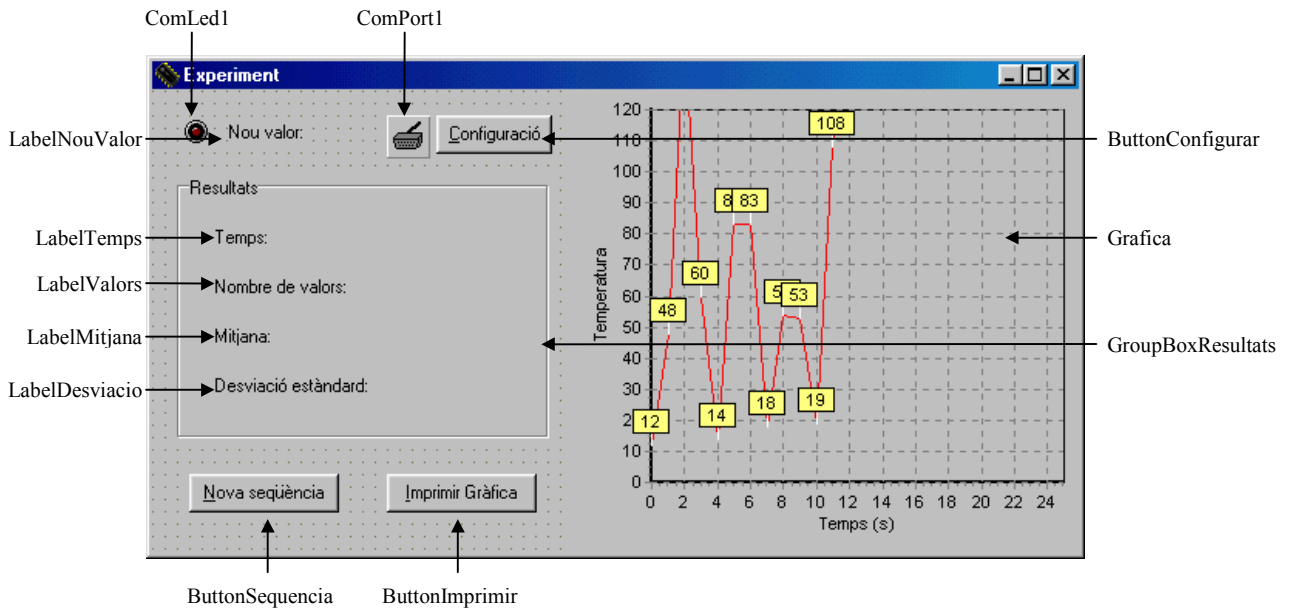
```
Application.Initialize;  
AboutBox := TAboutBox.Create(nil);  
if AboutBox.ShowModal = mrOK then  
begin  
Application.CreateForm(TFormExperiment, FormExperiment);  
Application.Run;  
end;  
AboutBox.Free;
```

Resultat:



## Exercici 5-2: instal·lació i ús d'un nou component

Trobareu components per treballar amb el port sèrie a <http://www2.arnes.si/~sopecrni> (ComPort) i a <http://www.tmssoftware.com> (Async32), entre d'altres. Una possible solució, escollint el component TComPort, seria la següent:



Propietats finestra principal:

**TForm FormExperiment:**

|                       |              |
|-----------------------|--------------|
| Left                  | 204          |
| Top                   | 262          |
| Caption               | 'Experiment' |
| Height                | 326          |
| Width                 | 608          |
| Constraints.MinHeight | 326          |
| Constraints.MinWidth  | 608          |

**TLabel LabelNouValor:**

|         |              |
|---------|--------------|
| Left    | 17           |
| Top     | 20           |
| Width   | 49           |
| Height  | 13           |
| Caption | 'Nou valor:' |

**TComLed ComLed1:**

|           |            |
|-----------|------------|
| Left      | 16         |
| Top       | 14         |
| Width     | 25         |
| Height    | 25         |
| ComPort   | ComPort1   |
| LedSignal | lsRx       |
| Kind      | lkRedLight |

**TGroupBox GroupBoxResultats:**

|         |                           |
|---------|---------------------------|
| Left    | 16                        |
| Top     | 56                        |
| Width   | 241                       |
| Height  | 169                       |
| Anchors | [akLeft, akTop, akBottom] |
| Caption | 'Resultats'               |

**TLabel LabelTemps:**

|         |          |
|---------|----------|
| Left    | 24       |
| Top     | 32       |
| Width   | 35       |
| Height  | 13       |
| Caption | 'Temps:' |

|                                 |                                    |
|---------------------------------|------------------------------------|
| <b>TLabel LabelValors:</b>      |                                    |
| Left                            | 24                                 |
| Top                             | 64                                 |
| Width                           | 86                                 |
| Height                          | 13                                 |
| Caption                         | 'Nombre de valors:'                |
| <b>TLabel LabelMitjana:</b>     |                                    |
| Left                            | 24                                 |
| Top                             | 96                                 |
| Width                           | 37                                 |
| Height                          | 13                                 |
| Caption                         | 'Mitjana:'                         |
| <b>TLabel LabelDesviacio:</b>   |                                    |
| Left                            | 24                                 |
| Top                             | 128                                |
| Width                           | 100                                |
| Height                          | 13                                 |
| Caption                         | 'Desviació estàndard:'             |
| <b>TButton ButtonSequencia:</b> |                                    |
| Left                            | 24                                 |
| Top                             | 248                                |
| Width                           | 97                                 |
| Height                          | 25                                 |
| Anchors                         | [akLeft, akBottom]                 |
| Caption                         | '&Nova seqüència'                  |
| OnClick                         | ButtonSequenciaClick               |
| <b>TButton ButtonImprimir:</b>  |                                    |
| Left                            | 152                                |
| Top                             | 248                                |
| Width                           | 97                                 |
| Height                          | 25                                 |
| Anchors                         | [akLeft, akBottom]                 |
| Caption                         | '&Imprimir Gràfica'                |
| OnClick                         | ButtonImprimirClick                |
| <b>TButton ButtonSequencia:</b> |                                    |
| Left                            | 184                                |
| Top                             | 16                                 |
| Width                           | 75                                 |
| Height                          | 25                                 |
| Anchors                         | [akLeft, akBottom]                 |
| Caption                         | '& Configuració'                   |
| OnClick                         | ButtonConfigurarClick              |
| <b>TComPort ComPort1:</b>       |                                    |
| Port                            | 'COM1'                             |
| BaudRate                        | br9600                             |
| DataBits                        | dbEight                            |
| Parity.Bits                     | prNone                             |
| StopBits                        | sbOneStopBit                       |
| OnRxChar                        | ComPort1RxChar                     |
| <b>TChart Grafica:</b>          |                                    |
| Left                            | 272                                |
| Top                             | 0                                  |
| Width                           | 328                                |
| Height                          | 299                                |
| Anchors                         | [akLeft, akTop, akRight, akBottom] |
| AllowZoom                       | False                              |
| Title.Text.Strings              | (')                                |

```

Title.Visible           False
BottomAxis.Title.Caption 'Temps (s)'
LeftAxis.Automatic      False
LeftAxis.Maximum        120
LeftAxis.Title.Caption  'Temperatura'
Legend.Visible          False
View3D                  False

TLineSeries Serie:
Marks.Visible           True
SeriesColor              clRed

```

Codi finestra principal:

```

const MaxDades = 5;

var
  e: TExperiment;
  v: integer;
  t: TDateTime;

// Procediment que inicialitza les variables i el text del formulari
procedure TFormExperiment.Inicialitzar();
begin
  e := TExperiment.Create(MaxDades);
  t := Time();
  Serie.Clear();
  LabelNouValor.caption := 'Nou valor: ';
  LabelTemps.caption := 'Temps: ';
  LabelValors.caption := 'Nombre de valors: 0';
  LabelMitjana.caption := 'Mitjana: ';
  LabelDesviacio.caption := 'Desviació estàndard: ';
end;

// Procediment que calcula la mitjana i la desviació estàndard
procedure TFormExperiment.Actualitzar();
var N: integer;
begin
  // Actualitzem el temps
  LabelTemps.caption := 'Temps: ' + TimeToStr(Time()-t);

  // Actualitzem el nombre de valors
  N := e.NumDades();
  LabelValors.caption := 'Nombre de valors: ' + FloatToStr(N);

  // Actualitzem la mitjana
  if N > 0 then
    LabelMitjana.caption := 'Mitjana: ' + FloatToStr(e.Mitjana());

  // Actualitzem la desviació estàndard
  if N > 1 then
    LabelDesviacio.caption := 'Desviació estàndard: '+FloatToStr(e.Desviacio);

  if serie.count = MaxDades then
    Serie.Delete(0);
    Serie.AddXY(86400*(Time()-t), v, '', clRed);
end;

// Quan es crea el formulari, fem una crida a la rutina d'inicialització.
procedure TFormExperiment.FormCreate(Sender: TObject);
begin
  Inicialitzar();
  ComPort1.Open();
end;

// Quan seleccionem aquest botó, fem una crida a la rutina d'inicialització.
procedure TFormExperiment.ButtonSequenciaClick(Sender: TObject);

```

```
begin
  e.Free();
  Inicialitzar();
end;

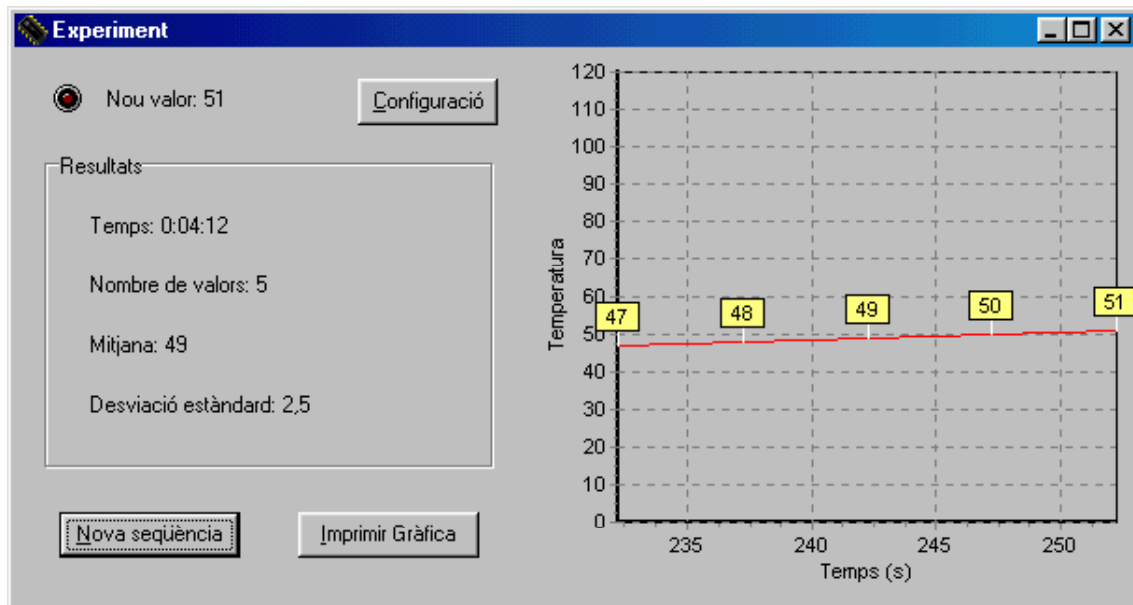
// Quan seleccionem aquest botó, imprimim la gràfica.
procedure TFormExperiment.ButtonImprimirClick(Sender: TObject);
begin
  Grafica.Print();
end;

// Quan seleccionem aquest botó, configurem les comunicacions.
procedure TFormExperiment.ButtonConfigurarClick(Sender: TObject);
begin
  ComPort1.ShowSetupDialog();
end;

// Esdeveniment que s'activa cada cop que rebem informació pel port
procedure TFormExperiment.ComPort1RxChar(Sender: TObject; Count: Integer);
var Str: String;
begin
  try
    ComPort1.ReadStr(Str, Count);
    LabelNouValor.caption := 'Nou valor: ' + Str;
    v := StrToInt(Str);
    e.Obtenir(v);
    Actualitzar();
  except
    LabelNouValor.caption := 'Nou valor: ERROR!';
  end;
end;

// Quan tanquem el formulari tanquem també les comunicacions
procedure TFormExperiment.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  ComPort1.Close();
end;
```

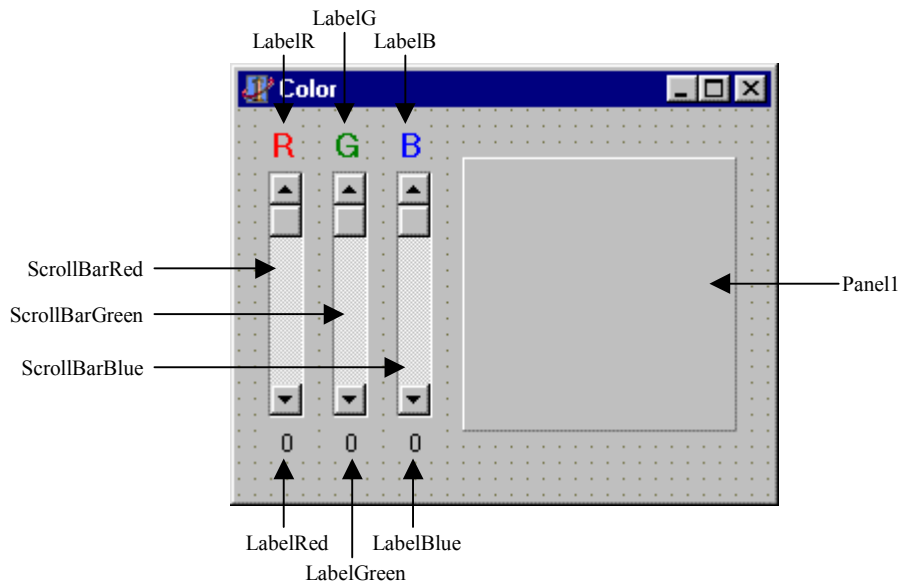
Resultat:





## Exercici 5-3: creació de colors

Una possible solució seria la següent:



### Propietats:

TForm **Form1**:  
 BorderStyle bsSingle  
 BorderIcons [biSystemMenu, biMinimize]  
 Caption 'Color'  
 Height 221  
 Width 274  
 Left 192  
 Top 107  
 OnCreate ScrollBarChange

TLabel **LabelR**:  
 Left 16  
 Top 8  
 Width 14  
 Height 20  
 Caption 'R'  
 Font.Color clRed  
 Font.Style [fsBold]

TLabel **LabelG**:  
 Left 48  
 Top 8  
 Width 15  
 Height 20  
 Caption 'G'  
 Font.Color clGreen  
 Font.Style [fsBold]

TLabel **LabelB**:  
 Left 80  
 Top 8  
 Width 13  
 Height 20  
 Caption 'B'

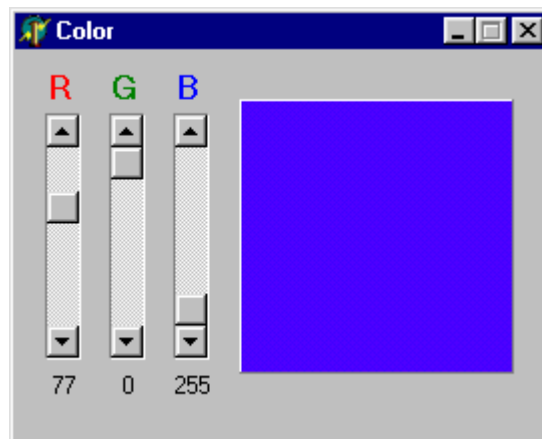
|                                   |                 |
|-----------------------------------|-----------------|
| Font.Color                        | clBlue          |
| Font.Style                        | [fsBold]        |
| <b>TLabel LabelRed:</b>           |                 |
| Left                              | 15              |
| Top                               | 160             |
| Width                             | 18              |
| Height                            | 13              |
| Alignment                         | taCenter        |
| AutoSize                          | False           |
| Caption                           | '0'             |
| <b>TLabel LabelGreen:</b>         |                 |
| Left                              | 47              |
| Top                               | 160             |
| Width                             | 18              |
| Height                            | 13              |
| Alignment                         | taCenter        |
| AutoSize                          | False           |
| Caption                           | '0'             |
| <b>TLabel LabelBlue:</b>          |                 |
| Left                              | 79              |
| Top                               | 160             |
| Width                             | 18              |
| Height                            | 13              |
| Alignment                         | taCenter        |
| AutoSize                          | False           |
| Caption                           | '0'             |
| <b>TLabel LabelRed:</b>           |                 |
| Left                              | 15              |
| Top                               | 160             |
| Width                             | 18              |
| Height                            | 13              |
| Alignment                         | taCenter        |
| AutoSize                          | False           |
| Caption                           | '0'             |
| <b>TScrollBar ScrollBarRed:</b>   |                 |
| Left                              | 16              |
| Top                               | 32              |
| Width                             | 16              |
| Height                            | 121             |
| Kind                              | sbVertical      |
| LargeChange                       | 15              |
| Max                               | 255             |
| PageSize                          | 0               |
| OnChange                          | ScrollBarChange |
| <b>TScrollBar ScrollBarGreen:</b> |                 |
| Left                              | 48              |
| Top                               | 32              |
| Width                             | 16              |
| Height                            | 121             |
| Kind                              | sbVertical      |
| LargeChange                       | 15              |
| Max                               | 255             |
| PageSize                          | 0               |
| OnChange                          | ScrollBarChange |
| <b>TScrollBar ScrollBarBlue:</b>  |                 |
| Left                              | 80              |
| Top                               | 32              |

|                        |                 |
|------------------------|-----------------|
| Width                  | 16              |
| Height                 | 121             |
| Kind                   | sbVertical      |
| LargeChange            | 15              |
| Max                    | 255             |
| PageSize               | 0               |
| OnChange               | ScrollBarChange |
| TPanel <b>Panel1</b> : |                 |
| Left                   | 112             |
| Top                    | 24              |
| Width                  | 137             |
| Height                 | 137             |

Codi:

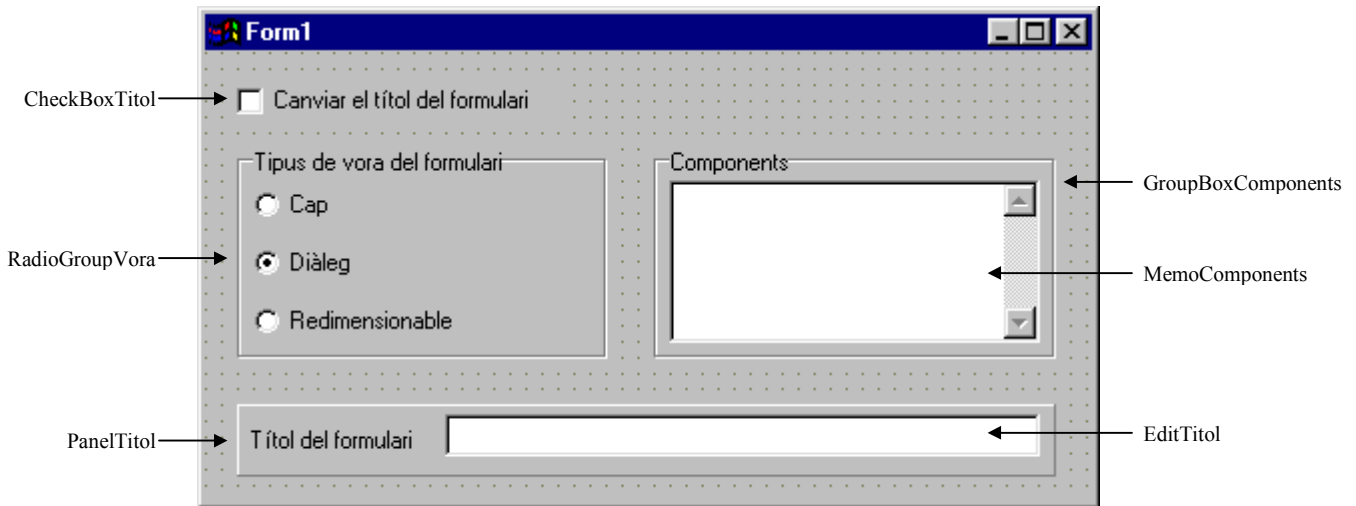
```
procedure TForm1.ScrollBarChange(Sender: TObject);  
begin  
  LabelRed.Caption := IntToStr(ScrollBarRed.Position);  
  LabelGreen.Caption := IntToStr(ScrollBarGreen.Position);  
  LabelBlue.Caption := IntToStr(ScrollBarBlue.Position);  
  Panel1.Color := RGB(ScrollBarRed.Position, ScrollBarGreen.Position,  
    ScrollBarBlue.Position);  
end;
```

Resultat:



## Exercici 5-4: dissenyador de formularis

Una possible solució seria la següent:



Propietats:

TForm **Form1:**

|                       |            |
|-----------------------|------------|
| BorderStyle           | bsDialog   |
| Caption               | 'Form1'    |
| Left                  | 200        |
| Top                   | 108        |
| Height                | 251        |
| Width                 | 451        |
| Constraints.MinHeight | 251        |
| Constraints.MinWidth  | 451        |
| OnCreate              | FormCreate |

TCheckBox **CheckBoxTitol:**

|          |                                  |
|----------|----------------------------------|
| Left     | 16                               |
| Top      | 16                               |
| Width    | 161                              |
| Height   | 17                               |
| Caption  | 'Canviar el títol del formulari' |
| TabOrder | 0                                |
| OnClick  | CheckBoxTitolClick               |

TRadioGroup **RadioGroupVora:**

|               |                                      |
|---------------|--------------------------------------|
| Left          | 16                                   |
| Top           | 48                                   |
| Width         | 185                                  |
| Height        | 105                                  |
| Anchors       | [akLeft, akTop, akBottom]            |
| Caption       | 'Tipus de vora del formulari'        |
| Items.Strings | ('Cap', 'Diàleg', 'Redimensionable') |
| ItemIndex     | 1                                    |
| TabOrder      | 1                                    |
| OnClick       | RadioGroupVoraClick                  |

TGroupBox **GroupBoxComponents:**

|          |                                    |
|----------|------------------------------------|
| Left     | 224                                |
| Top      | 48                                 |
| Width    | 201                                |
| Height   | 105                                |
| Anchors  | [akLeft, akTop, akRight, akBottom] |
| Caption  | 'Components'                       |
| TabOrder | 2                                  |

TMemo **MemoComponents:**

|      |   |
|------|---|
| Left | 8 |
|------|---|

|            |                                    |
|------------|------------------------------------|
| Top        | 16                                 |
| Width      | 185                                |
| Height     | 81                                 |
| Anchors    | [akLeft, akTop, akRight, akBottom] |
| ScrollBars | ssVertical                         |

**TPanel PanelTitol:**

|             |                             |
|-------------|-----------------------------|
| Left        | 16                          |
| Top         | 176                         |
| Width       | 409                         |
| Height      | 36                          |
| BorderWidth | 5                           |
| Alignment   | taLeftJustify               |
| Anchors     | [akLeft, akRight, akBottom] |
| Caption     | 'Títol del formulari'       |
| Visible     | False                       |
| TabOrder    | 3                           |

**TEdit EditTitol:**

|            |                   |
|------------|-------------------|
| Left       | 104               |
| Top        | 5                 |
| Width      | 297               |
| Height     | 21                |
| Anchors    | [akLeft, akRight] |
| OnChange   | EditTitolChange   |
| OnKeyPress | EditTitolKeyPress |

Codi:

```
// En crear el formulari, escriu el nom dels components a MemoComponents
procedure TForm1.FormCreate(Sender: TObject);
var i: integer;
begin
    GroupBoxComponents.Caption := IntToStr(ComponentCount) + ' components: ';
    for i := 0 to ComponentCount - 1 do
        MemoComponents.Lines.Add(Components[i].Name);
end;

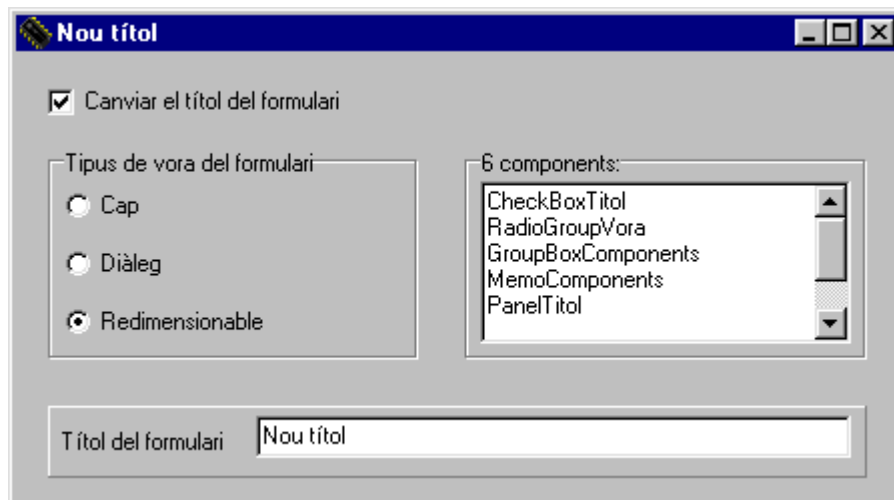
// En prèmer sobre el CheckBox, mostra i oculta el panel inferior
procedure TForm1.CheckBoxTitolClick(Sender: TObject);
begin
    PanelTitol.Visible := CheckBoxTitol.Checked;
    if CheckBoxTitol.Checked then EditTitol.SetFocus;
end;

// En prèmer sobre el RadioGroup, canvia la vora del formulari
procedure TForm1.RadioGroupVoraClick(Sender: TObject);
begin
    CheckBoxTitol.Enabled := RadioGroupVora.ItemIndex <> 0;
    EditTitol.Enabled := RadioGroupVora.ItemIndex <> 0;
    case RadioGroupVora.ItemIndex of
        0: // En prèmer sobre el primer RadioButton, treu les vores del formulari
            BorderStyle := bsNone;
        1: // En prèmer sobre el segon RadioButton, posa vores no redimensionables
            BorderStyle := bsDialog;
        2: // En prèmer sobre el tercer RadioButton, posa vores redimensionables
            BorderStyle := bsSizeable;
    end;
end;

// Controla cada pulsació de tecla en EditTitol, filtrant els caràcters
// numèrics i canviant les a's per e's i a l'inrevés.
procedure TForm1.EditTitolKeyPress(Sender: TObject; var Key: Char);
begin
    case Key of
```

```
'A': Key := 'E';  
'a': Key := 'e';  
'E': Key := 'A';  
'e': Key := 'a';  
'0'..'9': Key := #0;  
end;  
end;  
  
// En canviar el text d'EditTitol, l'assigna al títol del formulari  
procedure TForm1.EditTitolChange(Sender: TObject);  
begin  
    Caption := EditTitol.Text;  
end;
```

Resultat:



## Apèndix I. Excepcions

**Exception**—This is the ancestor of all exception classes. There is nothing wrong with using this class to raise exceptions in quick-and-dirty code, but in production code you'll want to be able to distinguish between the multitude of families of errors that your application can encounter. The best way to distinguish a family of related error conditions from the rest of the pack is to use a custom exception class to report those related errors.

**EAbort**—Referred to as Delphi's "silent" exception, this exception is trapped by the VCL default exception handler, but VCL does not inform the user that the exception occurred. Use **EAbort** when you want to take advantage of the exception's capability to abort and unwind out of a complicated process but you don't want the user to see an error message. Remember, the terms *exception* and *error* are *not* equivalent: Exceptions are a means of changing program flow to facilitate error handling...among other things.

**EAccessViolation**—An Access Violation has occurred in the operating system. Usually caused by a Nil or "wild" pointer.

**EAssertionFailed**—The statement passed to the **Assert()** procedure evaluated to **False**.

**EBitsError**—Raised when the **Bits** or **Size** properties of a **TBits** object is out of bounds.

**EComponentError**—This exception is raised in two situations:

- When you use **RegisterClasses()** to attempt to register a component outside the **Register()** procedure.
- When the name of your component is invalid or not unique.

**EControlC**—The user has interrupted with the Ctrl+C key combination. This exception only occurs within console mode applications.

**EDbEditError**—The user entered text into a **TMaskEdit** or **TDbEdit** component that is incompatible with the current edit mask.

**EDdeError**—An error occurred during a DDE operation with any of the **TDdeClientConv**, **TDdeClientItem**, **TDdeServerConv**, or **TDdeServerItem** components.

**EExternalException**—This exception occurs when an unrecognized exception is raised by the operating system.

**EInOutError**—This exception is raised when any I/O error occurs in your program. This exception will only occur when I/O checking is enabled using **{SI+}** in code or **Options | Project | Compiler | I/O Checking** in IDE.

**EIntError**—This is the ancestor of all integer math exceptions. The descendents of this class are:

**EDivByZero**—This exception is raised when you divide an integral number by zero. This exception is raised as a result of runtime error 200. This code sample will cause an **EDivByZero** exception:

```
var
  I: integer;
begin
  I := 0;
  I := 10 div I; { exception raised here }
end;
```

**EIntOverflow**—This exception is raised when you attempt to perform an operation that overflows an integral variable beyond that variable type's capacity. This exception is raised as a result of runtime error 215. This exception will only be raised if overflow checking is enabled using **{SQ+}** in code or **Options | Project | Compiler | Overflow Checking** in the IDE. The following code will cause this exception to be raised

```
var
  l: longint;
begin
  l := MaxLongint;
  l := l * l; { exception raised here }
end;
```

**ERangeError**—This exception is raised when you attempt to index an array beyond its declared bounds or when you attempt to store a value that's too large in an integral type variable. This exception is raised as a result of runtime error 201. Range checking must be enabled with {\$R+} in code or Options | Project | Compiler | Range Checking in the IDE for this error to occur. The following sample will cause Delphi to raise this exception:

```
var
  a: array[1..16] of integer;
  i: integer;
begin
  i := 17;
  a[i] := 1; { exception raised here }
end;
```

**EIntfCastError**—An attempt was made to cast an object or interface to an unsupported interface.

**EInvalidCast**—This exception is raised when you attempt to use the as operator to typecast a class as an incompatible class. This exception is raised as a result of runtime error 219. The following code will cause this exception to be raised:

```
var
  B: TObject;
begin
  B := TButton.Create(nil);
  with B as TMemo do { exception raised here -
                    TMemo is not an ancestor of TButton }
  ...
end;
```

**EInvalidGraphic**—This exception is raised when you attempt to use LoadFromFile() on a file that is not a compatible graphics format into a class expecting a graphics file.

**EInvalidGraphicOperation**—This exception is raised when you attempt to perform an illegal operation on a graphic object. For example, resizing a TIcon is illegal.

**EInvalidOperation**—This exception occurs when you try to display or perform any other operation that requires a window handle on a control without a Parent. For example:

```
var
  b: TBitBtn;
begin
  b := TBitBtn.Create(Self);
  b.SetFocus; { exception raised here }
end;
```

**EInvalidPointer**—This exception is raised usually when you attempt to free an invalid or already-freed portion of memory in a call to Dispose(), FreeMem(), or a class destructor. This example causes an EInvalidPointer exception to be raised:

```
var
  p: pointer;
begin
  GetMem(p, 8);
  FreeMem(p, 8);
  FreeMem(p, 8); { exception raised here }
end;
```

**EListError**—This exception will be raised if you try to index past the end of a TList descendant. For example:

```
var
  S: TStringList;
  Strng: String;
```



```
begin
  S := TStringList.Create;
  S.Add('One String');
  Strng := S.Strings[2]; { exception raised here }
end;
```

**EMathError**—This is the ancestor object from which the floating-point exceptions are derived.

**EInvalidOp**—This exception is raised when an invalid instruction is sent to the numeric coprocessor. This exception is uncommon unless you control the coprocessor directly with BASM code.

**EOverflow**—This exception is raised as a result of floating-point overflow, when a value becomes too large to hold in a floating point variable. This exception corresponds to runtime error 205.

**EUnderflow**—Raised as a result of floating-point underflow, when a value becomes too small to hold in a floating point variable. This exception corresponds to runtime error 206.

**EZeroDivide**—Raised when a floating point number is divided by zero.

**EMCIDeviceError**—The exception indicates that an error occurred in the `TMediaPlayer` component. Most commonly, this exception is raised when the user attempts to play some media whose type is unsupported by the hardware.

**EMenuError**—This is a generic exception that occurs in almost any error condition involving a `TMenu`, `TMenuItem`, or `TPopupMenu` component.

**EOleCtrlError**—This exception is reserved for ActiveX control wrapper errors, but it is currently not being used in VCL.

**EOleError**—This exception is raised when an OLE automation error occurs.

**EOleSysError**—This exception is raised by the `OleCheck()` and `OleError()` routines when an error occurs calling an OLE API function.

**EOleException**—Raised when an error occurs inside a `safecall` function or procedure.

**EOutlineError**—This is a generic exception that is raised when an error occurs while working with a `TOutline` component.

**EOutOfMemory**—This exception is raised when you call `New()`, `GetMem()`, or a class constructor and not enough memory is available on the heap for the allocation. This exception corresponds to runtime error 203.

**EOutOfResources**—This exception occurs when Windows cannot fill an allocation request for a Windows resource, such as a Window handle. This exception often reflects bugs in your video driver, especially if you're running in a high-color (32K or 64K colors) mode. If this error goes away when you switch to using the standard Windows VGA driver or to a lesser mode of your normal video driver, it's very likely that you've found a bug in your video driver. Contact your video card manufacturer for a driver update.

**EPackageError**—Raised when an error occurs loading, initializing, or finalizing a package.

**EParserError**—Raised when Delphi is unable to parse your text form file back to the binary `.dfm` format. Generally, this is the result of a syntax error while editing the form in the IDE.

**EPrinter**—This is a generic exception that will be raised when an error occurs while you are trying to use the `TPrinter` object.

**EPrivilege**—This exception indicates that an attempt was made to execute a privileged instruction.

**EPropertyError**—This exception is raised when an error occurs inside a component property editor.

**ERegistryException**—The `TRegistry` and `TRegIniFile` objects raise this exception when an error occurs while reading from or writing to the System Registry.

**EStackOverflow**—This exception represents a serious operating environment-level error in management of the stack. Compiling your program with stack checking enabled (`{ $S+ }` in code or Options | Project | Compiler | Stack Checking in the IDE) will help spot low stack conditions before a

processor stack fault occurs. Even with stack checking enabled, a processor stack fault still can occur if you call a stack-intensive Windows API function (such as `DrawText` or `FloodFill`) when your available stack space is very low.

**EReportError**—This is a generic exception for an error that occurs while working with a Report component.

**EResNotFound**—This exception is raised when there are problems loading a form from a `.dfm` file. This exception usually indicates that you have edited the `.dfm` file to make it invalid, the DFM or EXE file has become corrupted, or the DFM file was not linked into the EXE. Make sure you haven't deleted or altered the `{ $R * .DFM }` directive in your form unit.

**EStreamError**—This exception is the base class of all stream exceptions. This exception usually indicates a problem loading a `TStrings` from a stream or setting the capacity of a memory stream. The following descendent exception classes signal other specific error conditions:

**ECreateError**—Raised when an error occurs while creating a stream file. This exception often indicates that a file can't be created because the filename is invalid or in use by another process.

**EFileError**—This exception is raised when you attempted to register the same class twice using the `RegisterClasses()` procedure. This class also serves as the base for other filer-related exceptions:

**EClassNotFound**—This exception is raised when Delphi reads a component class name from a stream but cannot find a declaration for the component in its corresponding unit. Remember that code and declarations that are not used by a program will not be copied into the EXE file by Delphi's smart linker.

**EInvalidImage**—This exception is raised when you attempt to read components from an invalid resource file.

**EMethodNotFound**—This exception is raised when a method specified in the `.dfm` file or resource does not exist in the corresponding unit. This can happen if you have deleted code from the unit, recompiled the EXE, ignored the many warnings about the DFM file containing references to deleted code, and run the EXE anyway.

**EReadError**—This exception occurs when your application doesn't read the number of bytes from a stream that it is supposed to (for example, unexpected end-of-file) or when Delphi cannot read a property.

**EFOpenError**—This exception is raised when the specified stream file cannot be opened, and usually occurs when the file does not exist.

**EStringListError**—This is a generic exception that is raised when an error condition results while working with a `TStringList` object.

**EThread**—This is a `TThread`-related exception. Currently, this exception is only raised when a user attempts to call `Synchronize()` on a waiting thread.

**ETreeViewError**—This exception is raised when you pass an invalid item index to a `TTreeView` method or property.

**EWin32Error**—This exception is raised when an error occurs calling a Win32 API function. The message associated with this exception has error code and error string information.

## Apèndix II. Rutines predefinides d'Object Pascal

### Rutines estàndard

La següent taula llista els procediments i funcions més freqüentment emprats de les llibreries de Delphi. No és pas un inventari exhaustiu de rutines estàndard.

| Procediment o funció   | Descripció   |
|------------------------|--|
| <i>Abort</i>           | Ends the process without reporting an error.                                   |
| <i>Addr</i>            | Returns a pointer to a specified object.                                       |
| <i>AllocMem</i>        | Allocates a memory block and initializes each byte to zero.                    |
| <i>ArcTan</i>          | Calculates the arctangent of the given number.                                 |
| <i>Assert</i>          | Tests whether a boolean expression is True.                                    |
| <i>Assigned</i>        | Tests for a nil (unassigned) pointer or procedural variable.                   |
| <i>Beep</i>            | Generates a standard beep using the computer speaker.                          |
| <i>Break</i>           | Causes control to exit a for, while, or repeat statement.                      |
| <i>ByteToCharIndex</i> | Returns the position of the character containing a specified byte in a string. |
| <i>Chr</i>             | Returns the character for a specified ASCII value.                             |
| <i>Close</i>           | Terminates the association between a file variable and an external file.       |
| <i>CompareMem</i>      | Performs a binary comparison of two memory images.                             |
| <i>CompareStr</i>      | Compares strings case sensitively.   |
| <i>CompareText</i>     | Compares strings by ordinal value and is not case sensitive.                   |
| <i>Continue</i>        | Returns control to the next iteration of for, while, or repeat statements.     |
| <i>Copy</i>            | Returns a substring of a string or a segment of a dynamic array.               |
| <i>Cos</i>             | Calculates the cosine of an angle.   |
| <i>CurrToStr</i>       | Converts a currency variable to a string.                                      |
| <i>Date</i>            | Returns the current date.  |
| <i>DateTimeToStr</i>   | Converts a variable of type TDateTime to a string.                             |
| <i>DateToStr</i>       | Converts a variable of type TDateTime to a string.                             |
| <i>Dec</i>             | Decrements an ordinal variable.  |
| <i>Dispose</i>         | Releases memory allocated for a dynamic variable.                              |
| <i>ExceptAddr</i>      | Returns the address at which the current exception was raised.                 |
| <i>Exit</i>            | Exits from the current procedure.  |
| <i>Exp</i>             | Calculates the exponential of X.   |
| <i>FillChar</i>        | Fills contiguous bytes with a specified value.                                 |
| <i>Finalize</i>        | Uninitializes a dynamically allocated variable.                                |
| <i>FloatToStr</i>      | Converts a floating point value to a string.                                   |
| <i>FloatToStrF</i>     | Converts a floating point value to a string, using specified format.           |
| <i>FmtLoadStr</i>      | Returns formatted output using a resourced format string.                      |
| <i>FmtStr</i>          | Assembles a formatted string from a series of arrays.                          |
| <i>Format</i>          | Assembles a string from a format string and a series of arrays.                |
| <i>FormatDateTime</i>  | Formats a date-and-time value.   |
| <i>FormatFloat</i>     | Formats a floating point value.  |
| <i>FreeMem</i>         | Disposes of a dynamic variable.  |
| <i>GetMem</i>          | Creates a dynamic variable and a pointer to the address of the block.          |
| <i>GetParentForm</i>   | Returns the form or property page that contains a specified control.           |
| <i>Halt</i>            | Initiates abnormal termination of a program.                                   |
| <i>Hi</i>              | Returns the high-order byte of an expression as an unsigned value.             |
| <i>High</i>            | Returns the highest value in the range of a type, array, or string.            |
| <i>Inc</i>             | Increments an ordinal variable.  |
| <i>Initialize</i>      | Initializes a dynamically allocated variable.                                  |
| <i>Insert</i>          | Inserts a substring at a specified point in a string.                          |
| <i>Int</i>             | Returns the integer part of a real number.                                     |

|                          |  |
|--------------------------|--|
| <i>IntToStr</i>          | Converts an integer to a string.   |
| <i>Length</i>            | Returns the length of a string or array.                                       |
| <i>Lo</i>                | Returns the low-order byte of an expression as an unsigned value.              |
| <i>Low</i>               | Returns the lowest value in the range of a type, array, or string.             |
| <i>LowerCase</i>         | Converts an ASCII string to lowercase.   |
| <i>MaxIntValue</i>       | Returns the largest signed value in an integer array.                          |
| <i>MaxValue</i>          | Returns the largest signed value in an array.                                  |
| <i>MinIntValue</i>       | Returns the smallest signed value in an integer array.                         |
| <i>MinValue</i>          | Returns smallest signed value in an array.                                     |
| <i>New</i>               | Creates a new dynamic variable and references it with a specified pointer.     |
| <i>Now</i>               | Returns the current date and time.   |
| <i>Ord</i>               | Returns the ordinal value of an ordinal-type expression.                       |
| <i>Pos</i>               | Returns the index of the first character of a specified substring in a string. |
| <i>Pred</i>              | Returns the predecessor of an ordinal value.                                   |
| <i>Ptr</i>               | Converts a specified address to a pointer.                                     |
| <i>Random</i>            | Generates random numbers within a specified range.                             |
| <i>ReallocMem</i>        | Reallocates a dynamic variable.  |
| <i>Round</i>             | Returns the value of a real rounded to the nearest whole number.               |
| <i>SetLength</i>         | Sets the dynamic length of a string variable or array.                         |
| <i>SetString</i>         | Sets the contents and length of the given string.                              |
| <i>ShowException</i>     | Displays an exception message with its address.                                |
| <i>ShowMessage</i>       | Displays a message box with an unformatted string and an OK button.            |
| <i>ShowMessageFmt</i>    | Displays a message box with a formatted string and an OK button.               |
| <i>Sin</i>               | Returns the sine of an angle in radians.                                       |
| <i>SizeOd</i>            | Returns the number of bytes occupied by a variable or type.                    |
| <i>Sqr</i>               | Returns the square of a number.  |
| <i>Sqrt</i>              | Returns the square root of a number.   |
| <i>Str</i>               | Formats a string and returns it to a variable.                                 |
| <i>StrToCurr</i>         | Converts a string to a currency value.   |
| <i>StrToDate</i>         | Converts a string to a date format (TDateTime).                                |
| <i>StrToDateTime</i>     | Converts a string to a TDateTime.  |
| <i>StrToFloat</i>        | Converts a string to a floating-point value.                                   |
| <i>StrToInt</i>          | Converts a string to an integer.   |
| <i>StrToTime</i>         | Converts a string to a time format (TDateTime).                                |
| <i>StrUpper</i>          | Returns a string in upper case.  |
| <i>Succ</i>              | Returns the successor of an ordinal value.                                     |
| <i>Sum</i>               | Returns the sum of the elements from an array.                                 |
| <i>Time</i>              | Returns the current time.  |
| <i>TimeToStr</i>         | Converts a variable of type TDateTime to a string.                             |
| <i>Trunc</i>             | Truncates a real number to an integer.   |
| <i>UniqueString</i>      | Makes a string unique.   |
| <i>UpCase</i>            | Converts a character to uppercase.   |
| <i>UpperCase</i>         | Returns a string in uppercase.   |
| <i>VarArrayCreate</i>    | Creates a variant array.   |
| <i>VarArrayCreate</i>    | Returns number of dimensions of a variant array.                               |
| <i>VarArrayDimCount</i>  | Returns number of dimensions of a variant array.                               |
| <i>VarARrayHighBound</i> | Returns high bound for a dimension in a variant array.                         |
| <i>VarArrayLock</i>      | Locks a variant array and returns a pointer to the data.                       |
| <i>VarArrayLowBound</i>  | Returns the low bound of a dimension in a variant array.                       |
| <i>VarArrayOf</i>        | Creates and fills a one-dimensional variant array.                             |
| <i>VarArrayRedim</i>     | Resizes a variant array.   |
| <i>VarArrayRef</i>       | Returns a reference to the passed variant array.                               |
| <i>VarArrayUnlock</i>    | Unlocks a variant array.   |
| <i>VarAsType</i>         | Converts a variant to specified type.  |
| <i>VarCast</i>           | Converts a variant to a specified type, storing the result in a variable.      |
| <i>VarClear</i>          | Clears a variant.  |
| <i>VarCopy</i>           | Copies a variant.  |
| <i>VarToStr</i>          | Converts variant to string.  |
| <i>VarType</i>           | Returns type code of specified variant.  |

## Rutines de tractament de fitxers

| Procediment o funció | Descripció  |
|----------------------|---|
| <i>Append</i>        | Opens an existing text file for appending.  |
| <i>AssignFile</i>    | Assigns the name of an external file to a file variable.  |
| <i>BlockRead</i>     | Reads one or more records from an untyped file.   |
| <i>BlockWrite</i>    | Writes one or more records into an untyped file.  |
| <i>ChDir</i>         | Changes the current directory.  |
| <i>CloseFile</i>     | Closes an open file.  |
| <i>Eof</i>           | Returns the end-of-file status of a file.   |
| <i>Eoln</i>          | Returns the end-of-line status of a text file.  |
| <i>Erase</i>         | Erases an external file.  |
| <i>FilePos</i>       | Returns the current file position of a typed or untyped file.   |
| <i>FileSize</i>      | Returns the current size of a file; not used for text files.  |
| <i>Flush</i>         | Flushes the buffer of an output text file.  |
| <i>GetDir</i>        | Returns the current directory of a specified drive.   |
| <i>IOResult</i>      | Returns an integer value that is the status of the last I/O function performed.                           |
| <i>MkDir</i>         | Creates a subdirectory.   |
| <i>Read</i>          | Reads one or more values from a file into one or more variables.  |
| <i>Readln</i>        | Does what Read does and then skips to beginning of next line in the text file.                            |
| <i>Rename</i>        | Renames an external file.   |
| <i>Reset</i>         | Opens an existing file.   |
| <i>Rewrite</i>       | Creates and opens a new file.   |
| <i>Rmdir</i>         | Removes an empty subdirectory.  |
| <i>Seek</i>          | Moves the current position of a typed or untyped file to a specified component. Not used with text files. |
| <i>SeekEof</i>       | Returns the end-of-file status of a text file.  |
| <i>SeekEoln</i>      | Returns the end-of-line status of a text file.  |
| <i>SetTextBuf</i>    | Assigns an I/O buffer to a text file.   |
| <i>Truncate</i>      | Truncates a typed or untyped file at the current file position.   |
| <i>Write</i>         | Writes one or more values to a file.  |
| <i>Writeln</i>       | Does the same as Write, and then writes an end-of-line marker to the text file.                           |

## Rutines de tractament de cadenes acabades en nul

| Procediment o funció | Descripció  |
|----------------------|---|
| <i>StrAlloc</i>      | Allocates a character buffer of a given size on the heap.                     |
| <i>StrBufSize</i>    | Returns the size of a character buffer allocated using StrAlloc or StrNew.    |
| <i>StrCat</i>        | Concatenates two strings.   |
| <i>StrComp</i>       | Compares two strings.   |
| <i>StrCopy</i>       | Copies a string.  |
| <i>StrDispose</i>    | Disposes a character buffer allocated using StrAlloc or StrNew.               |
| <i>StrECopy</i>      | Copies a string and returns a pointer to the end of the string.               |
| <i>StrEnd</i>        | Returns a pointer to the end of a string.                                     |
| <i>StrFmt</i>        | Formats one or more values into a string.                                     |
| <i>StrIComp</i>      | Compares two strings without case sensitivity.                                |
| <i>StrLCat</i>       | Concatenates two strings with a given maximum length of the resulting string. |
| <i>StrLComp</i>      | Compares two strings for a given maximum length.                              |
| <i>StrLCopy</i>      | Copies a string up to a given maximum length.                                 |
| <i>StrLen</i>        | Returns the length of a string.   |
| <i>StrLFmt</i>       | Formats one or more values into a string with a given maximum length.         |
| <i>StrLIComp</i>     | Compares two strings for a given maximum length without case sensitivity.     |
| <i>StrLower</i>      | Converts a string to lowercase.   |
| <i>StrMove</i>       | Moves a block of characters from one string to another.                       |
| <i>StrNew</i>        | Allocates a string on the heap.   |
| <i>StrPCopy</i>      | Copies a Pascal string to a null-terminated string.                           |

|                  |   |
|------------------|---|
| <i>StrPLCopy</i> | Copies a Pascal string to a null-terminated string with a given maximum length. |
| <i>StrPos</i>    | Returns a pointer to the first occurrence of a given substring within a string. |
| <i>StrRScan</i>  | Returns a pointer to the last occurrence of a given character within a string.  |
| <i>StrScan</i>   | Returns a pointer to the first occurrence of a given character within a string. |
| <i>StrUpper</i>  | Converts a string to uppercase.   |

## Apèndix III. Gramàtica del llenguatge Object Pascal

```
Goal -> (Program | Package | Library | Unit)

Program -> [PROGRAM Ident ['(' IdentList ')'] ';' ]
          ProgramBlock '.'

Unit -> UNIT Ident ';'
       InterfaceSection
       ImplementationSection
       InitSection '.'

Package -> PACKAGE Ident ';'
          [RequiresClause]
          [ContainsClause]
          END '.'

Library -> LIBRARY Ident ';'
          ProgramBlock '.'

ProgramBlock -> [UsesClause]
               Block

UsesClause -> USES IdentList ';'

InterfaceSection -> INTERFACE
                  [UsesClause]
                  [InterfaceDecl]...

InterfaceDecl -> ConstSection
                -> TypeSection
                -> VarSection
                -> ExportedHeading

ExportedHeading -> ProcedureHeading ';' [Directive]
                 -> FunctionHeading ';' [Directive]

ImplementationSection -> IMPLEMENTATION
                       [UsesClause]
                       [DeclSection]...

Block -> [DeclSection]
        CompoundStmt

DeclSection -> LabelDeclSection
              -> ConstSection
              -> TypeSection
              -> VarSection
              -> ProcedureDeclSection

LabelDeclSection -> LABEL LabelId

ConstSection -> CONST (ConstantDecl ';' )...

ConstantDecl -> Ident '=' ConstExpr
              -> Ident ':' typeId '=' TypedConstant

TypeSection -> TYPE (TypeDecl ';' )...

TypeDecl -> Ident '=' Type
```

```

-> Ident '=' RestrictedType

TypedConstant -> (ConstExpr | ArrayConstant | RecordConstant)

ArrayConstant -> '(' TypedConstant/', '... ')

RecordConstant -> '(' RecordFieldConstant/';'... ')

RecordFieldConstant -> Ident ':' TypedConstant

Type -> TypeId
-> SimpleType
-> StructType
-> PointerType
-> StringType
-> ProcedureType
-> VariantType
-> ClassRefType

RestrictedType -> ObjectType
-> ClassType
-> InterfaceType

ClassRefType -> CLASS OF TypeId

SimpleType -> (OrdinalType | RealType)

RealType -> REAL48
-> REAL
-> SINGLE
-> DOUBLE
-> EXTENDED
-> CURRENCY
-> COMP

OrdinalType -> (SubrangeType | EnumeratedType | OrdIdent)

OrdIdent -> SHORTINT
-> SMALLINT
-> INTEGER
-> BYTE
-> LONGINT
-> INT64
-> WORD
-> BOOLEAN
-> CHAR
-> WIDECHAR
-> LONGWORD
-> PCHAR

VariantType -> VARIANT
-> OLEVARIANT

SubrangeType -> ConstExpr '..' ConstExpr

EnumeratedType -> '(' IdentList ')

StringType -> STRING
-> ANSISTRING
-> WIDESTRING
-> STRING '[' ConstExpr ']'

StructType -> [PACKED] (ArrayType | SetType | FileType | RecType)

ArrayType -> ARRAY ['[' OrdinalType/', '... ']] OF Type

RecType -> RECORD [FieldList] END

```



```

FieldList -> FieldDecl/';'... [VariantSection] [';']

FieldDecl -> IdentList ':' Type

VariantSection -> CASE [Ident ':'] TypeId OF RecVariant/';'...

RecVariant -> ConstExpr/','... ':' '(' [FieldList] ')'

SetType -> SET OF OrdinalType

FileType -> FILE OF TypeId

PointerType -> '^' TypeId

ProcedureType -> (ProcedureHeading | FunctionHeading) [OF OBJECT]

VarSection -> VAR (VarDecl ';')...

VarDecl -> IdentList ':' Type [(ABSOLUTE (Ident | ConstExpr)) | '=' ConstExpr]

Expression -> SimpleExpression [RelOp SimpleExpression]...

SimpleExpression -> ['+' | '-'] Term [AddOp Term]...

Term -> Factor [MulOp Factor]...

Factor -> Designator ['(' ExprList ')']
      -> '' Designator
      -> Number
      -> String
      -> NIL
      -> '(' Expression ')'
      -> NOT Factor
      -> SetConstructor
      -> TypeId '(' Expression ')'

RelOp -> '>'
      -> '<'
      -> '<='
      -> '>='
      -> '<>'
      -> IN
      -> IS
      -> AS

AddOp -> '+'
      -> '-'
      -> OR
      -> XOR

MulOp -> '*'
      -> '/'
      -> DIV
      -> MOD
      -> AND
      -> SHL
      -> SHR

Designator -> QualId ['.' Ident | '[' ExprList ']' | '^']...

SetConstructor -> '[' [SetElement/','...] ']'

SetElement -> Expression ['..' Expression]

ExprList -> Expression/','...

Statement -> [LabelId ':'] [SimpleStatement | StructStmt]
  
```

```

StmtList -> Statement/';'...

SimpleStatement -> Designator ['(' ExprList ')']
                  -> Designator ':=' Expression
                  -> INHERITED
                  -> GOTO LabelId

StructStmt -> CompoundStmt
              -> ConditionalStmt
              -> LoopStmt
              -> WithStmt

CompoundStmt -> BEGIN StmtList END

ConditionalStmt -> IfStmt
                  -> CaseStmt

IfStmt -> IF Expression THEN Statement [ELSE Statement]

CaseStmt -> CASE Expression OF CaseSelector/';'... [ELSE Statement] [';'] END

CaseSelector -> CaseLabel/','... ':' Statement

CaseLabel -> ConstExpr ['..' ConstExpr]

LoopStmt -> RepeatStmt
            -> WhileStmt
            -> ForStmt

RepeatStmt -> REPEAT Statement UNTIL Expression

WhileStmt -> WHILE Expression DO Statement

ForStmt -> FOR QualId ':=' Expression (TO | DOWNTO) Expression DO Statement

WithStmt -> WITH IdentList DO Statement

ProcedureDeclSection -> ProcedureDecl
                       -> FunctionDecl

ProcedureDecl -> ProcedureHeading ';' [Directive]
                 Block ';'

FunctionDecl -> FunctionHeading ';' [Directive]
                 Block ';'

FunctionHeading -> FUNCTION Ident [FormalParameters] ':' (SimpleType | STRING)

ProcedureHeading -> PROCEDURE Ident [FormalParameters]

FormalParameters -> '(' FormalParm/';'... ')'

FormalParm -> [VAR | CONST | OUT] Parameter

Parameter -> IdentList [':' ([ARRAY OF] SimpleType | STRING | FILE)]
              -> Ident ':' SimpleType '=' ConstExpr

Directive -> CDECL
              -> REGISTER
              -> DYNAMIC
              -> VIRTUAL
              -> EXPORT
              -> EXTERNAL
              -> FAR
              -> FORWARD
              -> MESSAGE
              -> OVERRIDE

```

```

-> OVERLOAD
-> PASCAL
-> REINTRODUCE
-> SAFECALL
-> STDCALL

ObjectType -> OBJECT [ObjHeritage] [ObjFieldList] [MethodList] END

ObjHeritage -> '(' QualId ')'

MethodList -> (MethodHeading [';' VIRTUAL])/';'...

MethodHeading -> ProcedureHeading
-> FunctionHeading
-> ConstructorHeading
-> DestructorHeading

ConstructorHeading -> CONSTRUCTOR Ident [FormalParameters]

DestructorHeading -> DESTRUCTOR Ident [FormalParameters]

ObjFieldList -> (IdentList ':' Type)/';'...

InitSection -> INITIALIZATION StmtList [FINALIZATION StmtList] END
-> BEGIN StmtList END
-> END

ClassType -> CLASS [ClassHeritage]
[ClassFieldList]
[ClassMethodList]
[ClassPropertyList]
END

ClassHeritage -> '(' IdentList ')'

ClassVisibility -> [PUBLIC | PROTECTED | PRIVATE | PUBLISHED]

ClassFieldList -> (ClassVisibility ObjFieldList)/';'...

ClassMethodList -> (ClassVisibility MethodList)/';'...

ClassPropertyList -> (ClassVisibility PropertyList ';'')...

PropertyList -> PROPERTY Ident [PropertyInterface] PropertySpecifiers

PropertyInterface -> [PropertyParameterList] ':' Ident

PropertyParameterList -> '[' (IdentList ':' TypeId)/';'... ']'

PropertySpecifiers -> [INDEX ConstExpr]
[READ Ident]
[WRITE Ident]
[STORED (Ident | Constant)]
[(DEFAULT ConstExpr) | NODEFAULT]
[IMPLEMENTS TypeId]

InterfaceType -> INTERFACE [InterfaceHeritage]
[ClassMethodList]
[ClassPropertyList]
END

InterfaceHeritage -> '(' IdentList ')'

RequiresClause -> REQUIRES IdentList... ';'

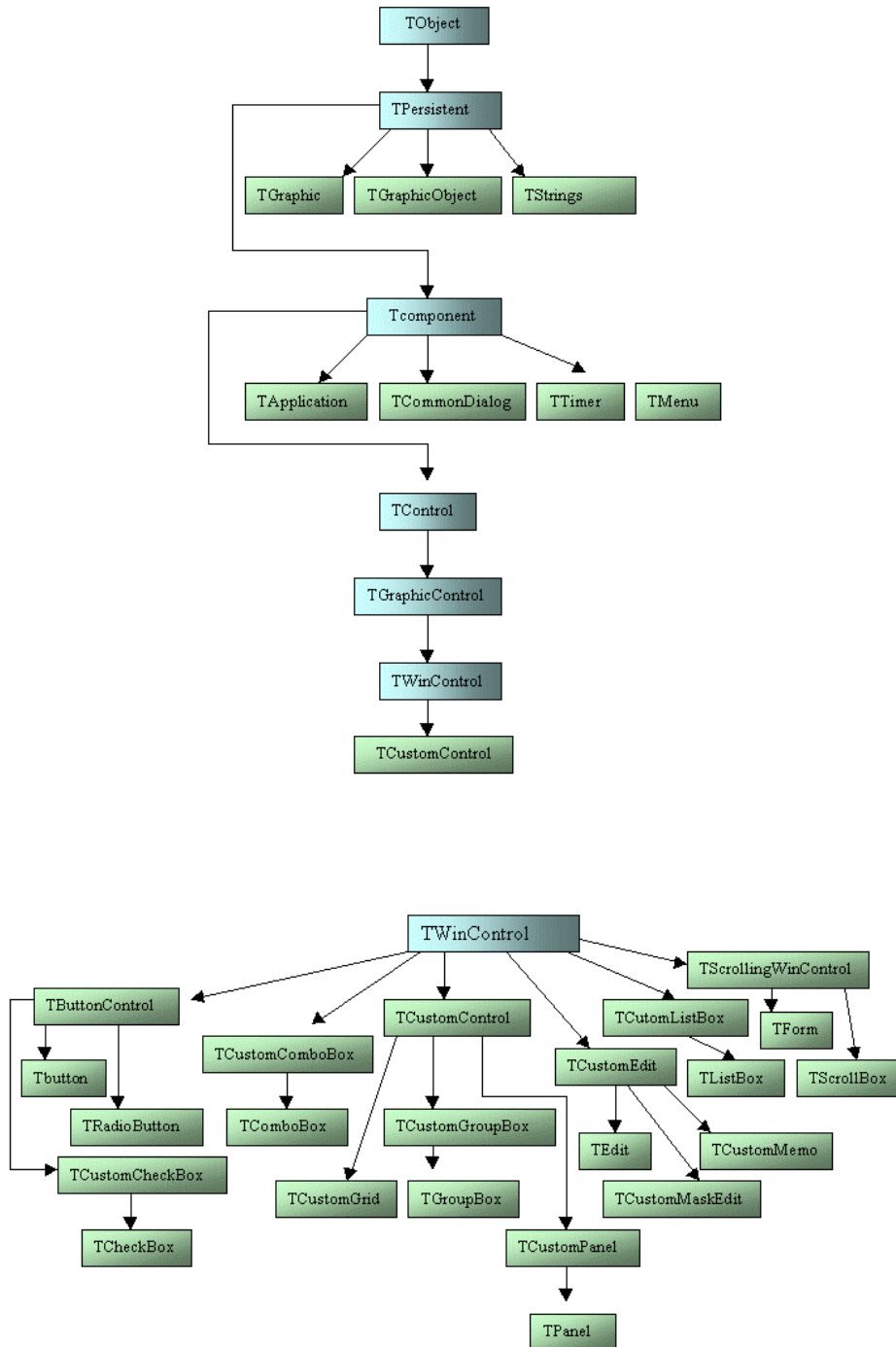
ContainsClause -> CONTAINS IdentList... ';'

IdentList -> Ident/','...

```

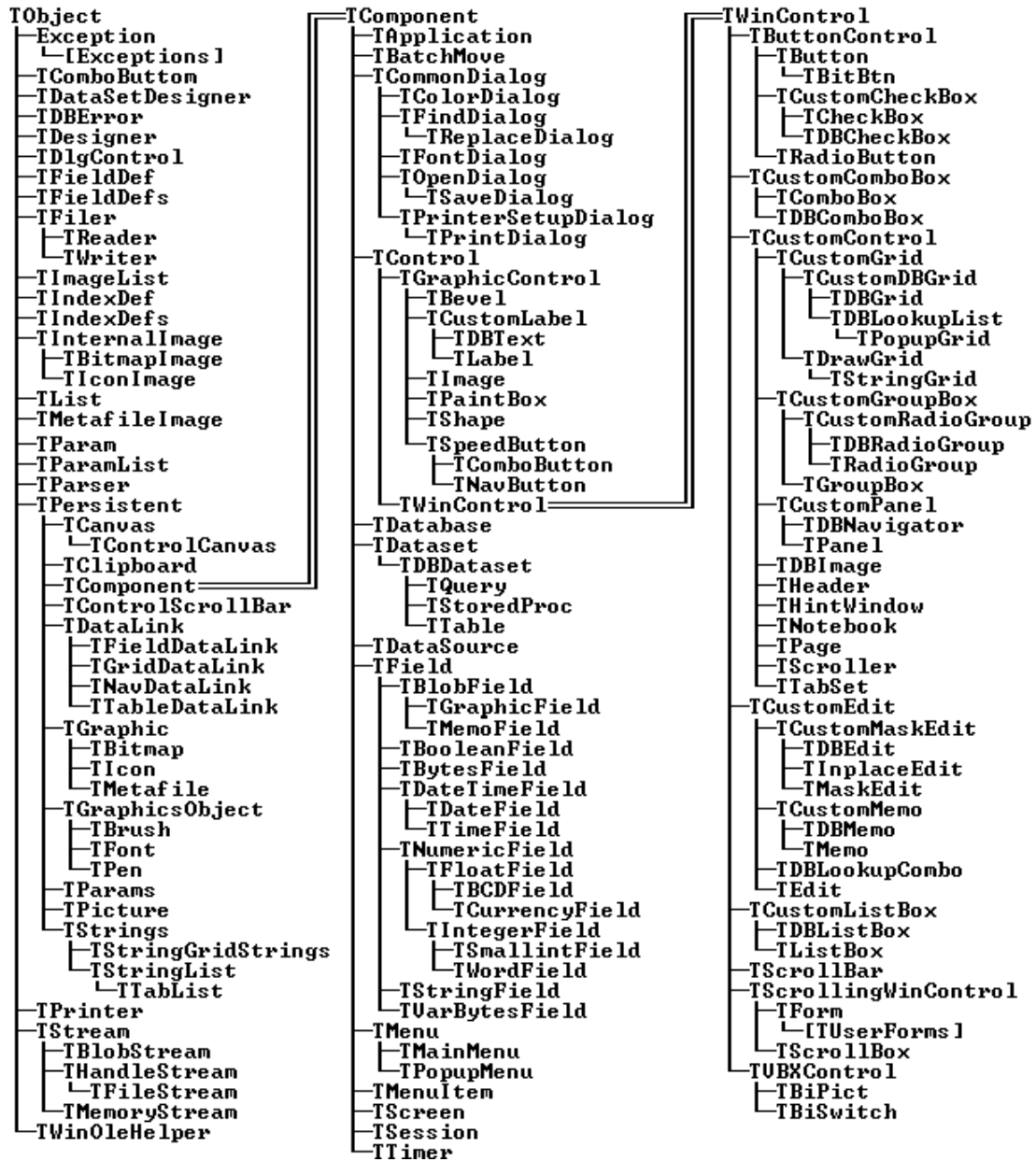
**QualId** -> [UnitId '.'] Ident  
**TypeId** -> [UnitId '.'] <type-identifier>  
**Ident** -> <identifier>  
**ConstExpr** -> <constant-expression>  
**UnitId** -> <unit-identifier>  
**LabelId** -> <label-identifier>  
**Number** -> <number>  
**String** -> <string>

# Apèndix IV. Taxonomia de classes a Delphi



La Llibreria de Components Visuals de Delphi, o VCL, és una jerarquia d'objectes que comença d'una classe base TObject i s'especialitza arribant des de controls estàndard de Windows com botons, formularis, menús i fitxers .ini fins a objectes no visuals més sofisticats com taules de bases de dades, peticions SQL i diagrames. La VCL conté un centenar de components que poden ser acoblades ràpidament per construir aplicacions complexes.

Delphi Object Tree Reference - by Jim Gunkel: 70711,2020



## Apèndix V. Delphi a Internet

### Borland

<http://www.borland.com>

*Borland Inprise Internacional*

<http://www.borland.es>

*Borland Inprise Espanya*

### En castellà

<http://www.clubdelphi.com/>

*Club Delphi*

### En anglès

<http://delphi.icm.edu.pl/>

*Delphi Super Page* és un enorme magatzem de components per Delphi.

<http://www.torry.ru/>

*Torry's Delphi Pages* és un enorme magatzem de components i recursos per Delphi.

<http://www.marcocantu.com/links/components.htm>

La pàgina de Marco Cantú conté enllaços a pàgines de nous components per Delphi.

[http://software-tools.com/tools\\_delphi/](http://software-tools.com/tools_delphi/)

*Software Tools* conté una llista molt completa d'eines per desenvolupament amb Delphi.

<http://buglist.jrsoftware.org/>

*Delphi Bugs List* conté la llista més completa d'errors a Delphi.

<http://www.delphi-jedi.org/>

*Delphi Jedi* és una comunitat internacional de desenvolupadors de software per Delphi. A les seves pàgines trobareu recursos, interessants projectes on participar i un munt de programes de codi obert.

<http://www.efg2.com/>

*Efg's Computer Lab and Reference Library* dona accés a una gran col·lecció de tòpics tècnics, bibliografia i projectes per Delphi.

### Compiladors gratuïts

<http://www.freepascal.org>

Compilador de Object Pascal multiplataforma

### Tutorials

<http://www.delphi-dolphin.com/>

*Delphi Dolphin*

### Grups de notícies

<news://es.comp.lenguajes.delphi>

Grup de notícies de Delphi en castellà