



Informàtica

**ICB0 Desenvolupament
d'aplicacions multiplataforma**

**ICC0 Desenvolupament
d'aplicacions web**

M0486 Accés a dades

AEA1 Accés a BDR

Diari d'activitats

Isidre Guixà

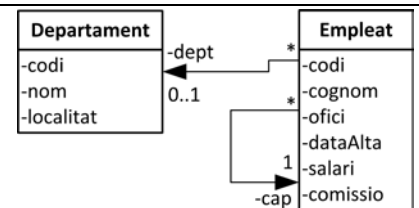
Curs 2025/26

	15-09-2025
➤ Què és JDBC?	
Establim connexions amb JDBC	19-09-2025
<p>➤ JDBC forma part de JDK. No cal instal·lar cap llibreria per fer programes i compilar-los.</p> <p>➤ Però en executar-los necessitem tenir incorporat el connector JDBC adequat al SGBD i a la versió de Java</p> <p>➤ Programes que hem fet:</p> <p>250918_1_ConnexioJDBCOracle</p> <p>250918_1_ConnexioJDBCOracle</p> <p>Els 2 programes estableixen connexió, mostren la classe del SGBD que implementa <code>Connection</code>, miren l'estat de l'<code>autocommit</code>, el desactiven i tanquen connexió.</p> <p>➤ Hem vist com retocar/configurar el PostgreSQL que va instal·lar Odoo, per poder-lo usar aquí (veure <code>empresa_PostgreSQL_llegime.txt</code> de <code>GuionsEmpresaDiversosSGBD.zip</code> publicat a la zona de treball del Classroom). Hem creat un usuari <code>dam2</code> amb contrasenya <code>dam2</code> i una bd <code>empresa</code> propietat de <code>dam2</code></p>	
Exercici	23-09-2025
<p>De l'anterior classe, tenim 2 programes "idèntics" per connectar amb 2 SGBD diferents. Això no interessa. Aconseguir un únic programa que pugui servir per connectar amb qualsevol SGBD.</p> <p>Pistes:</p> <ul style="list-style-type: none"> ✓ Les dades de connexió, enlloc de residir en el codi, han de residir en un fitxer de configuració. ✓ Aconsellable usar fitxers <code>.properties</code> o <code>.xml</code> que puguin ser gestionats per la classe <code>Properties</code> de Java. Si no la coneixeu... cerqueu informació. ✓ El programa reculli el nom del fitxer de configuració via paràmetre en el moment de la seva execució i, recuperi les dades de connexió i les usi per establir la connexió. <p>Solució:</p> <p>Projecte <code>250923_1_ConnexioJDBC</code>, que conté 2 arxius de configuració diferents (un per connectar amb un Oracle i un per connectar amb un PostgreSQL).</p> <p>Per a que es pugui executar, en el moment d'execució cal tenir accés al connector JDBC que correspongui. El projecte porta incorporats els 2 connectors. En un cas real, en una organització, només caldria tenir el connector adequat al SGBD amb el que calgui connectar.</p>	
Exercici	26-09-2025
<p>Ampliar el programa anterior amb la recuperació de les dades de tots els empleats i les mostri per pantalla.</p> <p>En el fitxer <code>GuionsEmpresaDiversosSGBD.zip</code> publicat a la zona de treball del Classroom, trobarem:</p> <ul style="list-style-type: none"> ➤ Guió per instal·lar "empresa" en PostgreSQL tenint en compte el corresponent <code>llegime</code>. ➤ Guions per instal·lar "empresa" en Oracle. Podeu crear escriptori d'Oracle en Isard. La plantilla actual de l'Isard té esquema <code>empresa</code> amb dades corruptes en els caràcters especials. Si segueix les instruccions del <code>llegime</code> que us proporciono, disposareu de dades correctes. Ara mateix hi ha un problema a l'Isard que no em permet arreglar la plantilla. Possiblement, en breus dies, per arreglar la plantilla hauré d'eliminar tots els escriptors Isard de l'Oracle. Per tant... no us mateu configurant-lo. Simplement, si voleu, executeu guió per tenir dades "empresa" correctes. <p>Solució:</p> <p>Projecte <code>250926_1_MostrarEmpleats</code>, que conté 2 arxius de configuració diferents (un per connectar amb un Oracle i un per connectar amb un PostgreSQL).</p>	
Desenvolupament capa persistència – Estructura – Constructor(s) – Mètodes bàsics	30-09-2025
<p>Objectiu IMPORTANT en desenvolupar una aplicació que gestioni dades, és que la seva gestió sigui independent del lloc on s'ubiquin les dades (BDR, BDOR, BDOO,...) i això s'aconsegueix amb capes de persistència.</p> <p>Per tant, ara que estem en BDR, ens interessa:</p> <ul style="list-style-type: none"> ➤ Desenvolupar una capa de persistència per BDR pensada per emmagatzemar les dades de departaments i empleats definits en les classes <code>Departament</code> i <code>Empleat</code> ja facilitades, en amb taules també facilitades. <p>L'aplicació, per ser totalment independent de la tecnologia on s'emmagatzemen les dades, mai ha de connectar amb l'origen de les dades, sinó que tots els accessos han de ser via una capa de persistència. Çper tant, la capa ha de facilitar el conjunt complet de mètodes que pugui necessitar l'aplicació.</p>	



El projecte Empresa01 facilitat conté les classes, que verifiquen l'UML:

Alerta! Observem que Empleat pot no tenir Departament assignat i això està en contradicció amb el disseny de la taula EMP facilitada. Fem-ho tot coherent i retoquem el disseny de la taula EMP permetent que un empleat pugui no tenir departament.



- En Oracle: `alter table emp modify (dept_no null);`
- En PostgreSQL: `ALTER TABLE public.emp
ALTER COLUMN dept_no DROP NOT NULL;`

- Desenvolupar un programa que permeti comprovar el funcionament dels mètodes desenvolupats a la capa.

Solució:

- Projecte 250930_1_CpJdbcEmpresa, que conté una primera fase de la capa de persistència per gestionar les dades d'empresa (Departaments i Empleats) en una BDR.

Hem incorporat la classe CpJdbcEmpresa que serà la capa de persistència i la classe CpJdbcEmpresaException per controlar les excepcions que es puguin produir.

Una capa de persistència MAI manté interacció amb l'usuari!!! Tots els errors via excepcions!!!

La capa de persistència ha d'incorporar com a mínim un constructor. En el nostre cas hem decidit incorporar 2 constructors. El(s) constructor(s) s'encarreguen d'establir la connexió amb l'origen de dades.

Tot ha d'estar documentat!!! La classe i tots els mètodes!!!

- Projecte 250930_1_AplicacióEmpresa, que conté un programa per anar comprovant el correcte funcionament de la capa de persistència.

Aquest programa ha de crear un objecte de la capa, que mantindrà la connexió amb l'origen de dades i usarem per invocar els diversos mètodes facilitats per la capa de persistència.

Exercici per divendres, 3/octubre:

- Ara mateix, els 2 projectes 250930_* no estan finalitzats.
- El constructor de CpJdbcEmpresa no està finalitzat. Acabar-lo!
- Programar mètode existeixDepartament(int codi)
- El programa del projecte AplicacióEmpresa ha de funcionar i comprovar el correcte funcionament del mètode existDepartament.

Solució:

- Projecte 251003_1_CpJdbcEmpresa, que conté:
 - ✓ Constructors
 - ✓ Mètode tancarCapa, que tanca la connexió => Veure [això](#) referent a Connection close().
 - ✓ Mètodes per gestionar departaments (n'hi podria haver més...):

```

existeixDepartament(int codi)
obtenirDepartament(int codi)
inserirDepartament(Departament d) => Veure això referent a errors DML en PostgreSQL

```
 - ✓ Mètodes imprescindibles en tota capa (hem vist la seva necessitat en desenvolupar mètodes que incorporen instruccions DML):

```

validarCanvis(), que fa commit
desferCanvis(), que fa rollback

```
- Projecte 251003_1_AplicacióEmpresa, amb programa per comprovar el funcionament de la capa.

Incorpora un fitxer de propietats per comprovar-la en Oracle i un altre per comprovar-la en PostgreSQL.

El programa està preparat per:

 - ✓ invocar la capa amb el fitxer de propietats que s'indiqui via arguments en la crida al programa
 - ✓ invocar la capa sense fitxer de propietats si no s'indica cap fitxer de propietats en la crida al programa

Incorpora 2 programes en *Test Packages* per invocar programa per Oracle i per PostgreSQL.



Disseny de mètode <code>Connection close()</code> en JDBC => Implicacions!!!	03-10-2025
<ul style="list-style-type: none"> L'API JDBC defineix: <i>It is strongly recommended that an application explicitly commits or rolls back an active transaction prior to calling the close method. If the close method is called and there is an active transaction, the results are implementation-defined.</i> És a dir, si es tanca la connexió amb transaccions actives, que succeeixi <code>commit</code> o <code>rollback</code> depèn de l'actuació del connector JDBC emprat. Oracle, en la documentació del seu connector <code>ojdbc8.jar</code>, diu: <i>If the auto-commit mode is disabled and you close the connection without explicitly committing or rolling back your last changes, then an implicit COMMIT operation is run.</i> SQLServer, en la documentació diu: <i>Calling the close method in the middle of a transaction causes the transaction to be rolled back.</i> PostgreSQL & MySQL, via comprovació (no trobat en documentació), executen <code>rollback</code> en tancar connexió <p>Davant el fet que no tots els SGBD actuen de la mateixa manera, és altament recomanable que abans de tancar una connexió s'efectuï un <code>rollback</code>: i per això està incorporat dins el mètode <code>tancarCapa()</code>.</p>	
Error PostgreSQL: current transaction is aborted, commands ignored until end of transaction block	03-10-2025
<p>PostgreSQL, quan es produeix un error en instrucció DML (per exemple, per què es viola alguna restricció de PK, FK, UN, CK,...), no executa altres instruccions fins que no s'executa un <code>rollback</code>, com a mínim fins el punt abans de la instrucció que ha provocat l'error i informa amb l'error: <pre>current transaction is aborted, commands ignored until end of transaction block</pre></p> <p>Aquest funcionament NO és habitual en SGBD. Si la nostra capa de persistència volem que pugui ser usada en PostgreSQL, cal tenir-ho en compte i preveure un punt de salvaguarda (Savepoint) just abans de qualsevol instrucció DML i, en cas d'error, retrocedir (<code>rollback</code>) fins el punt de salvaguarda (no pas un <code>rollback</code> total!!!).</p> <p>Aquest mecanisme ha estat implementat en el mètode <code>inserirDepartament</code>; ens soluciona el problema en PostgreSQL i no afecta als altres SGBD que no tenen aquest funcionament "estrany".</p>	
Com garantir no tenir varies instàncies d'un mateix objecte en memòria	07-10-2025
<p>Abans de continuar, fixem-nos en certes problemàtiques que no hem tingut en compte.</p> <p>Cada vegada que invoquem <code>obtenirDepartament</code>, en cas d'existir a la BD, s'obté un objecte <code>Departament</code> i, per tant, si el programador no és conscient que per un determinat codi de departament ja l'ha invocat abans, en memòria hi podrà haver diversos objectes referents al mateix departament i això és una font de problemes (modificacions que només es fan en un dels objectes,...). Per tant, sembla que hauríem d'intentar garantir que no hi hagués en memòria diferents objectes referents a una mateixa dada en la BD.</p> <p>Un altre cas pel que interessaria unicitat en memòria d'un objecte. Pensem en el mètode <code>obtenirEmpleat</code> (que farem en breu) que permetrà obtenir un objecte <code>Empleat</code> i tenir-lo en memòria. Apareix el mateix problema plantejat amb <code>obtenirDepartament</code> però, a més, tenir un objecte <code>Empleat</code> en memòria implica tenir també els corresponents objectes <code>Departament</code> i <code>Cap</code>, si s'escau, en memòria i, llavors, si un mateix <code>Departament</code> és apuntat per varis empleats, és adequat tenir-lo "repetit" tantes vegades com empleats tingui? No pas, tornem a arribar a la conclusió que hauríem d'intentar garantir que no hi hagués en memòria diferents objectes referents a una mateixa dada en la BD.</p> <p>Les eines ORM (AEA5) tenen mecanismes que ho gestionen i no són gens simples. Nosaltres en farem una aproximació consistent en que la capa sigui conscient de quins són els objectes que té en memòria i això un farem amb una col·lecció que ens permeti una ràpida consulta: <pre>HashMap<Integer, Departament> hmDept</pre> per anar guardant els departaments que tenim en memòria; <pre>HashMap<Integer, Empleat> hmEmp</pre> per anar guardant els empleats que tenim en memòria; que gestionarem adequadament en els diferents mètodes de la capa.</p> <p>Així, el mètode <code>obtenirDepartament</code> s'haurà de preocupar d'incorporar el departament dins <code>hmDept</code> després d'haver-lo recuperat de la BD.</p> <p>I, el mètode <code>inserirDepartament</code> també s'ha de preocupar d'incorporar el departament dins <code>hmDept</code> després d'haver-lo inserit en la BD??? Sembla que SÍ, però la resposta no és tant fàcil.</p>	



Què passarà si `inserirDepartament` incorpora el departament dins `hmDept` i posteriorment el programador invoca el mètode `desferCanvis`? Doncs que `hmDept` contindrà un objecte que no és a la BD...

És a dir, `hmDept` hauria de contenir els objectes que es troben definitivament a la BD i els que estan pendents d'un commit els haurem d'incorporar en una altra col·lecció, com `hmDeptTemp`, de manera que el mètode `inserirDepartament` incorpora el nou departament dins `hmDeptTemp` enlloc de dins `hmDept`.

En tal situació:

- El mètode `desferCanvis` s'ha de preocupar de buidar `hmDeptTemp`
- El mètode `validarCanvis` s'ha de preocupar de traspasar els objectes de `hmDeptTemp` a `hmDept` deixant buit `hmDeptTemp`.

I, llavors, els mètodes `existeixDepartament` i `obtenirDepartament`, abans d'accedir a la BD, han de comprovar el contingut de `hmDept` i `hmDeptTemp`.

Aquestes solucions estan implementades en els projectes:

- 251007_1_CpJdbcEmpresa
- 251007_1_AplicacioEmpresa

I si les nostres classes estiguessin connectades bidireccionalment?

Imaginem que la relació entre `Departament` i `Empleat` o entre `Empleat` i `Empleat`, que segons model `Empresa01` que estem seguint, són unidireccionals fossin bidireccional. Això implicaria que a la classe `Departament` tindríem accés a la col·lecció dels seus objectes `Empleat` i a la classe `Empleat` tindríem accés a la col·lecció dels seus objectes `Empleat` subordinat i, per tant:

- En carregar un `Departament` en memòria ens veuríem obligats a carregar tots els seus objectes `Empleat` i per cada `Empleat`, els seus objectes `Empleat` subordinat i per cada subordinat, que a la vegada pot tenir subordinants, els seus objectes `Empleat` subordinat i...
- En carregar un `Empleat` en memòria ens veuríem obligats a carregar el seu `Departament` i tots els seus objectes `Empleat` subordinat...

IMPOSSIBLE per nosaltres!!!!

Les eines ORM (AEA5) ens donen la solució, que passa per treballar amb "objectes incomplets" que no carreguin certes dades fins que les necessitem.

executeQuery sobre un PreparedStatement tanca el ResultSet previ generat pel mateix PS.

08-10-2025

Compte, compte, compte!

Fixeu-vos en el programa del projecte 251008_1_ProvaPreparedStatement.

En ell tenim un `PreparedStatement` que s'executa una vegada recollint resultat en el `ResultSet rs7500` que podem consultar i, en el que recuperem el codi de l'empleat de la primera fila i que es torna a executar recollint resultat en el `ResultSet rs9999` que també podem consultar... Però si volem continuar consultant `rs7500` ja no es pot fer per què està tancat, sense que el programa hagi explicitat el seu tancament.

És a dir, l'execució d'un `PreparedStatement` tanca el `ResultSet` previ generat pel mateix `PreparedStatement`. O dit d'altra manera, només hi pot haver un `ResultSet` actiu a partir d'un `PreparedStatement`.

El projecte exemple facilitat té dos programes de test: un per Oracle i un per PostgreSQL.

En els dos, el missatge que facilita és similar. En ocasions, Oracle només informa amb `findColumn`, degut a que no pot trobar la columna demanada (degut a que el `ResultSet` està tancat).

Igual topeu amb aquest problema en algun dels mètodes del següent exercici.

Si és així, cerqueu com solucionar el problema!

Exercici

10-10-2025

Ampliar capa amb mètodes:

```
existeixEmpleat (int codi)
obtenirEmpleat (int codi)
inserirEmpleat (Empleat e)
```



Solució:

Tenim 2 versions, degudes a donar diferent solució a un problema que es presenta en el mètode `obtenirEmpleat`.

Aquest mètode és recursiu, doncs en recuperar un empleat, també hem de provocar la recuperació del seu cap, en cas que en tingui i, si el cap no el tenim en memòria (no és a `hmEmp` ni a `hmEmpTemp`), s'executarà un `executeQuery` sobre el `PreparedStatement` `psObtenirEmp`.

I... un `executeQuery` sobre un `PreparedStatement` tanca automàticament el `ResultSet` que pogués existir d'un `executeQuery` sobre el mateix `PreparedStatement`.

Per tant, en cas que el `PreparedStatement` només es creï una vegada i sigui compartit per les diferents crides del mètode `obtenirEmpleat`, caldrà assegurar que la crida recursiva s'efectuï un cop hagi finalitzat el processament sobre el `ResultSet` actual. És l'opció implementada en els projectes:

251010_1_CpJdbcEmpresa
251010_1_AplicacioEmpresa

Una altra opció, per no haver de tenir en compte quan s'efectua la crida recursiva, és que el `PreparedStatement` es creï cada vegada que s'hagi d'usar. Encara que la referència `psObtenirEmp` sigui global, si es crea cada vegada que calgui usar-lo, desapareix el problema i no cal que la crida recursiva s'efectuï quan hagi finalitzat el processament del `ResultSet` actual. És l'opció implementada en els projectes:

251010_2_CpJdbcEmpresa
251010_2_AplicacioEmpresa

on es pot observar que la crida recursiva s'efectua en mig del processament del `ResultSet` actual.

Objectiu del RA6: La nostra capa sigui un component d'accés a dades

10-10-2025

La nostra capa de persistència pot ser usada en qualsevol SGBDR que faciliti connector JDBC i l'aplicació de proves que hem estat desenvolupant, ha permès provar la nostra capa, en diversos SGBDR. En el nostre cas, l'aplicació ha estat per efectuar proves, però podria ser una completa aplicació (interfície gràfica,...) per gestionar departament i empleats (altes, baixes, consultes, modificacions) i elaborar informes.

Suposem que ens trobem amb la necessitat de que les dades (departaments i empleats) enlloc de residir en una BDR, resideixin en altre tecnologia de BD:

- BDOR com podria ser Oracle Objecte Relacional – AEA3
- BDOO com podria ser Matisse – AEA3
- BDXML com podria ser BaseX, Sedna, eXist-db via API XQJ – AEA4
- BDR via ORM – AEA5

És clar que per cada cas hauríem de desenvolupar la corresponent capa de persistència. Suposem que les tenim:

- `CpOracleOOEmpresa`
- `CpBDOOMatisse`
- `CpXQJ`
- `CpORM`

L'objectiu és que la nostra aplicació pugui usar qualsevol capa sense haver de tocar el seu codi.

Com ho podem aconseguir?

- En primer lloc, és evident que totes les capes han de facilitar els mateixos mètodes (excepte els constructors, que no poden ser iguals per què el seu nom ha de coincidir amb el nom de la capa).

Suposem que totes les capes ho verifiquen.

- Què més hem de fer per assolir el nostre objectiu?

Respecte l'aplicació que usi una capa:

- ✓ No pot contenir en el seu interior cap declaració `!import nomCapa`.
- ✓ No pot invocar el constructor de la capa

És a dir, ha de poder crear un objecte de la capa (sense invocar el constructor via el seu nom) i poder invocar els mètodes de la capa segons els necessiti.



Com s'aconsegueix això?

- ✓ Creant una interfície que declari tots els mètodes de la capa
- ✓ Totes les capes hagin d'implementar aquesta interfície
- ✓ L'aplicació ha de contenir `!import nomInterfície` enlloc de `!import nomCapa`
D'aquesta manera, l'aplicació ja pot invocar els mètodes de la capa, per què estan declarats a la interfície.
- ✓ L'aplicació ha d'aconseguir crear l'objecte de la capa que correspongui (ja veurem com fer-ho)

➤ Com s'organitza?

- ✓ El model de les dades en un projecte `M` (és el nostre `Empresa01`)
- ✓ La interfície en un projecte `I`, que ha d'incorporar el projecte `M`.
- ✓ Cada capa de persistència (ara només tenim `CpJdbcEmpresa`) en un projecte `CP`, i com que la capa ha d'implementar la interfície, aquest projecte ha d'incorporar el projecte on és la interfície.
Evidentment, cada capa de persistència també ha d'incorporar el projecte `M` amb el model de dades.
- ✓ L'aplicació en un projecte `AP` que ha d'incorporar el projecte `I` amb la interfície i el projecte `M` amb el model de dades.

En temps d'execució:

- ✓ El projecte `AP` ha de tenir accés al projecte de la capa a usar i a les APIs i/o connectors que aquesta capa necessiti (en el cas de la capa `CpJdbcEmpresa`, el connector `JDBC` que correspongui).

Normalment, en un projecte real on calgui poder tenir diverses capes de persistència, es segueix l'ordre:

1. Definir interfície
2. Crear capes

En el nostre cas, tenim la capa sense haver definit la interfície. *Vàrem començar la casa per la teulada!*

Podem crear la interfície de forma manual o, aprofitant-nos de l'opció de refactorització que facilita NetBeans per obtenir una interfície a partir d'una classe ja existent. Aquest procés:

- ✓ Crea la interfície en el mateix paquet on resideix la classe.
- ✓ Retoca la classe incorporant que implementa la interfície.

Ens podem aprofitar del procés de refactorització que té NetBeans i posteriorment moure la interfície a un projecte nou, doncs capa i interfície han de residir en projectes diferents.

En efectuar el procés de refactorització i generar la interfície (l'anomenem `InterCpEmpresa`, observem que els mètodes de la interfície fan referència a la classe `CpJdbcException`. No és adequat.

La interfície sempre ha d'anar acompanyada de la classe `Exception` que generin els seus mètodes. El nom de la nostra classe és `CpJdbcException` per què la vàrem crear en implementar la classe `CPJdbcException`, però ens cal canviar-li el nom adequadament (p.e. `InterCPEmpresaException`).

Així tenim:

- Projecte `251010_3_InterCpEmpresa`, amb interfície i corresponent classe `Exception`.
- Projecte `251010_3_CpJdbcEmpresa`, que incorpora el projecte anterior
- Projecte `251010_3_AplicacioEmpresa`, **pendent de retocar**:
 - ✓ El seu codi ha de fer referència a la interfície i no a una capa en concret
 - ✓ La declaració `CpJdbcEmpresa cp` s'ha de substituir per `InterCpEmpresa cp`
 - ✓ Com substituir la crida al constructor `CpJdbcEmpresa???`

Càrrega dinàmica de components i invocació del seu constructor

14-10-2025

Per a que una aplicació pugui usar diferents components, que tindran diferent nom, no podem incorporar dins el codi cap `import` amb el nom del component. Cal poder efectuar una càrrega dinàmica de la classe que correspongui el component.

Els components, per poder ser usats indistintament en una mateixa aplicació, han d'implementar una interfície i l'aplicació es desenvolupa invocant els mètodes definits a la interfície.

Però en temps d'execució, caldrà crear l'objecte del component a usar... i en el codi no es pot invocar el constructor per què les interfícies no en tenen.

El projecte `251014_1_ComCrearObjecteDeComponent` mostra com cal actuar.

Com aconseguir que una aplicació pugui usar diferents components (capes)	14-10-2025
<p>➤ L'aplicació no pot tenir cap declaració <code>import</code> referent a un component en concret. Ha de tenir:</p> <pre>import nomInterfície import nomClasseExceptionQueGenerinElsMètodes</pre> <p>➤ La declaració</p> <pre>nomCapa cp</pre> <p>s'ha de substituir per</p> <pre>nomInterfície cp</pre> <p>➤ La creació de l'objecte capa no pot ser invocant estàticament el constructor de la classe, doncs no tenim <code>import nomClasse</code> dins l'aplicació. Cal usar càrrega dinàmica de components i invocació de constructor.</p> <p>Així tenim:</p> <p>➤ Projecte 251014_2_InterCpEmpresa, amb interfície i corresponent classe <code>Exception</code>. Necessita el projecte <code>Empresa01</code>.</p> <p>➤ Projecte 251014_2_CpJdbcEmpresa Necessita els projectes <code>Empresa01</code> i <code>InterCpEmpresa</code>.</p> <p>➤ Projecte 251014_2_AplicacioEmpresa, amb la implementació de les consideracions anteriors. Necessita els projectes <code>Empresa01</code> i <code>InterCpEmpresa</code>. En temps d'execució, necessita també: La capa i les llibreries que pugui necessitar la capa a usar. Així, en els programes de test de 251014_2_AplicacioEmpresa, ha calgut afegir:</p> <ul style="list-style-type: none"> ✓ Projecte <code>CpJcbceEmpresa</code>, tant en la prova en Oracle com en PostgreSQL ✓ Connector JDBC Oracle per la prova en Oracle ✓ Connector JDBC PostgreSQL per la prova en PostgreSQL 	
Exercicis per practicar	
<p>➤ Ampliar capa amb mètodes:</p> <ul style="list-style-type: none"> ✓ <code>void eliminarDepartament (int codiDepartament);</code> Ha d'intentar eliminar el departament el departament indicat, de manera que els empleats que el puguin tenir, passin a quedar sense departament assignat. Qualsevol error ha de generar excepció correctament informada. ✓ <code>void eliminarDepartament (int codiDepartament, int nouDepartament);</code> Ha d'intentar eliminar el departament el departament indicat, de manera que els empleats que el puguin tenir, passin a quedar assignats al nou departament. Abans, però, ha de comprovar l'existència del nou departament, generant excepció correctament informada en cas que no existeixi. Qualsevol error ha de generar excepció correctament informada. ✓ <code>void actualitzarDepartament (Departament d);</code> Ha d'intentar actualitzar el departament passat per paràmetre. Prèviament ha de comprovar l'existència d'aquest departament, generant excepció correctament informada en cas que no existeixi. Qualsevol error ha de generar excepció correctament informada. ✓ I qualsevol mètode que se us pugui passar pel cap... <p>➤ Comprovar el correcte funcionament dels nous mètodes, amb un joc complet de proves.</p> <p>➤ La nostra aplicació de proves, en aquest moment, obliga a ser executada amb 1 o 2 arguments:</p> <p>1r. Nom de la capa a usar (obligatori)</p> <p>2n. Nom del fitxer de propietats per la capa (optatiu)</p> <p>Fer nova versió de l'aplicació de proves de manera que obligui a ser executada amb 0 o 1 argument, de manera que:</p> <ul style="list-style-type: none"> ➤ Sense argument, ha de cercar un fitxer de propietats amb un nom per defecte (el què decidiu) ➤ Amb 1 argument, ha de cercar un fitxer de propietats amb el nom d'aquest argument. <p>Aquest fitxer de propietats ha de contenir:</p> <ul style="list-style-type: none"> ➤ Propietat <code>nomCapa</code> amb el nom de la capa a usar ➤ Propietats necessàries per la capa. <p>No s'ha de fer cap modificació a la capa! Només cal retocar l'aplicació per a què pugui instanciar la capa.</p>	