



Informàtica

ICB0 Desenvolupament d'aplicacions multiplataforma

M06 Accés a dades

UF4 Components d'accés a dades

Diari d'activitats

Isidre Guixà

Curs 2024/25

INTRODUCCIÓ

Els **components de programari** són aquelles unitats executables i independents que componen una aplicació, les quals tenen una funcionalitat i una forma d'interactuar perfectament definides, de manera que el seu assemblatge amb la resta de components es pugui fer sense haver de modificar el codi intern de cap d'ells i, a més, en qualsevol moment sigui possible la substitució per un altre component equivalent (amb una funcionalitat definida de forma idèntica).

Exemples:

- **Plugins.** Són unitats executables i independents que poden formar part d'una aplicació amb la qual interaccionen. La seva incorporació no requereix cap modificació del codi de l'aplicació ni de la resta de components. La instal·lació és molt fàcil de dur a terme i en qualsevol moment podem substituir el *plugin* per un altre de més eficient o per una versió superior del mateix.

Els *plugins* són components molt independents que afegixen funcionalitat a l'aplicació, però que no són necessaris per a l'execució de la resta de funcionalitats. Hi ha, però, *components* que formen part del nucli funcional de l'aplicació i, per tant, el seu rendiment afectarà l'execució de tota l'aplicació. No és possible executar l'aplicació sense ells i si s'eliminen, s'han de substituir per d'altres equivalents.

- **Connectors ODBC i JDBC per accés a SGBDR.** Són components de baix nivell a partir dels quals construïm altres components més específics de cada aplicació, però són components al cap i a la fi que compleixen tots els requisits exposats. Són unitats executables independents que formen part de les aplicacions, que responen a una funcionalitat perfectament definida i es poden assemblar i intercanviar fàcilment per altres d'equivalents.

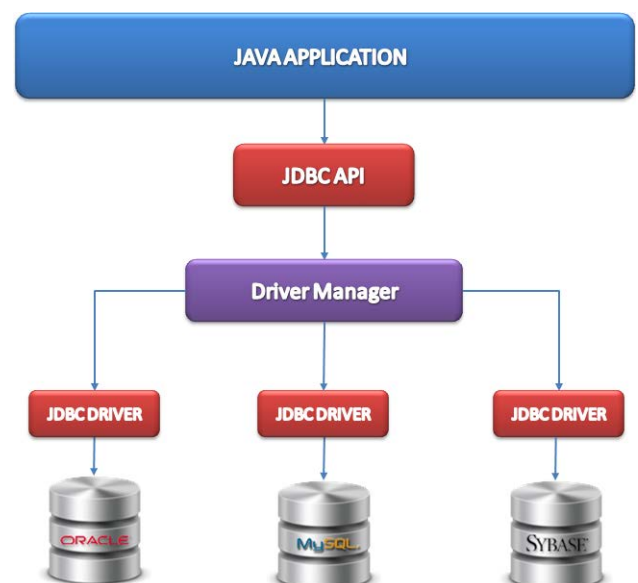
Si decidim desenvolupar una aplicació que accedeix a BD via ODBC o JDBC, podem canviar el SGBD sense alterar en absolut l'aplicació, amb la simple utilització del connector ODBC o JDBC que correspongui i, possiblement, amb una petita configuració.

Així, podem fer una aplicació que utilitzi SQL estàndard per gestionar una BDR, via JDBC, de manera que el SGBD pugui ser MySQL, Oracle, PostgreSQL, SQLServer, Access... sempre i quan tinguem el driver JDBC (i ODBC si utilitzem JDBC via ODBC) que correspongui al SGBD amb el que es vol connectar.

Com a exemple d'utilització, tenim els programes que desenvolupem en el M03-UF6, via JDBC, que poden treballar indistintament amb qualsevol SGBDR pel que disposem de connector JDBC.

Recordem com fem una aplicació Java que connecta amb qualsevol SGBDR que faciliti connector JDBC en els projectes:

a) Intro_1_ConnexioJDBCSenseConnectorsJDBC





Fixem-nos que no incorpora cap llibreria, a banda del JDK. Per establir connexió, s'invoca el mètode estàtic `getConnection` de la classe `DriverManager` que ve incorporada en JDK. Consultar l'ajuda de `DriverManager` per comprovar que és una classe. En canvi, `Connection` és una interfície, també incorporada en JDK i, per tant, amb la referència `Connection con` es pot fer referència a qualsevol objecte d'una classe que implementi `Connection`.

Observem que el programa és únic i que abans d'establir connexió necessita recuperar d'un fitxer de propietats les dades necessàries per establir la connexió. Aquest muntatge permet tenir un únic programa per connectar amb diversos SGBDR. Aquest projecte incorpora tres programes llençadora (`ProvaMySQL`, `ProvaOracle`, `ProvaPostgreSQL`) que invoquen el programa únic `Connexio` passant-li el fitxer de configuració adequat.

Què passa en intentar executar qualsevol dels programes? Doncs que NO troben el connector JDBC i el programa finalitza informant del problema.

b) Intro_2_ConnexioJDBCAmbConnectorsJDBC

És idèntic al projecte anterior però incorpora els connectors JDBC per a MySQL, Oracle i PostgreSQL i, per tant, el programa `Connexio` pot intentar establir la connexió, que serà existosa o no segons les dades de connexió dels fitxers de propietats siguin correctes i els corresponents SGBDR estiguin engegats i siguin accessibles per la xarxa.

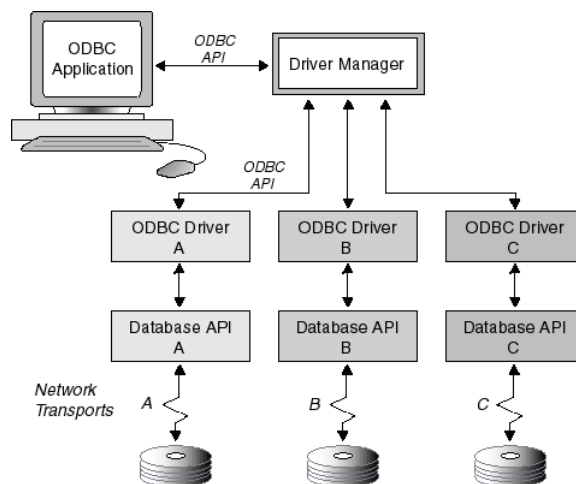
El programa `Connexio`, en cas d'establir connexió, informa de la classe a la que pertany l'objecte apuntat per la referència `Connection con`. Fixem-nos que:

- MySQL8: `com.mysql.cj.jdbc.ConnectionImp`
- Oracle: `oracle.jdbc.driver.T4CConnection`
- PostgreSQL: `org.postgresql.jdbc.PgConnection`

Observem com cada fabricant ha implementat la seva versió de la interfície `Connection` en el seu connector JDBC. **Cada connector JDBC és un component que és substituïble en una aplicació.** En el projecte exemple, hi ha els tres components JDBC per poder provar les 3 connexions, però en una implantació d'una aplicació, una vegada decidit quin és el SGBD, només s'instal·la el connector JDBC corresponent.

Un altre exemple són els connectors ODBC que també permeten l'accés a SGBDR. En màquines Windows, els connectors ODBC s'administren amb l'**Administrador d'origen de dades ODBC** que ve incorporat en el S.O. i des d'on es pot consultar els connectors instal·lats i definir els DSN (Data Source Name – configuració que conté la informació de connexió amb el SGBDR).

L'aplicació que usa ODBC per connectar, utilitza el DSN i el connector ODBC que correspongui (similar al fitxer de propietats i connector JDBC usats en els projectes Java anteriors).



- **Implementacions XQJ i XML-DB per accés a SGBD-XML.**

Equivalent a les implementacions JDBC-ODBC

Si fem una aplicació que accedeixi a SGBD-XML utilitzant qualsevol de les dues API (XQJ o XML-DB), hauríem de poder treballar amb qualsevol SGBD-XML pel que disposessim del corresponent connector (implementació).

Així, podem fer una aplicació que utilitzi qualsevol d'aquestes dues API, de manera que el SGBD-XML pugui ser eXist-db, BaseX, Oracle (només XQJ) o Sedna.

Com a exemple d'utilització, tenim els programes que desenvolupem en el M06-UF3, via XQJ, que poden treballar indistintament amb qualsevol SGBD-XML pel que disposem de connector XQJ.

Recordem com fem una aplicació Java que connecta amb qualsevol SGBD-XML que faciliti connector XQJ en els projectes:

c) Intro_3_ConnexioXQJSenseConnectorsXQJ

Fixem-nos que a diferència dels projectes JDBC:

- o Cal incorporar la llibreria `xqjapi.jar` que defineix l'API XQJ, no incorporada en el JDK.
- o Cal carregar dinamicament la classe que implementa l'API XQJ adequada per connectar amb el SGBD-XML via `Class.forName(className).newInstance()` que retorna un objecte `XQDataSource xqs`. Això en JDBC era necessari pels connectors JDBC inferiors a 4.0.

Observem que el programa és únic i que abans d'establir connexió amb `xqs.getConnection()` necessita recuperar d'un fitxer de propietats les dades necessàries per establir la connexió, amb les quals s'informa l'objecte `xqs`. Aquest muntatge permet tenir un únic programa per connectar amb diversos SGBD-XML. Aquest projecte incorpora quatre programes llençadora (`ProvaBaseX`, `ProvaExist`, `ProvaOracle` i `ProvaSedna`) que invoquen el programa únic `Connexio` passant-li el fitxer de configuració adequat.

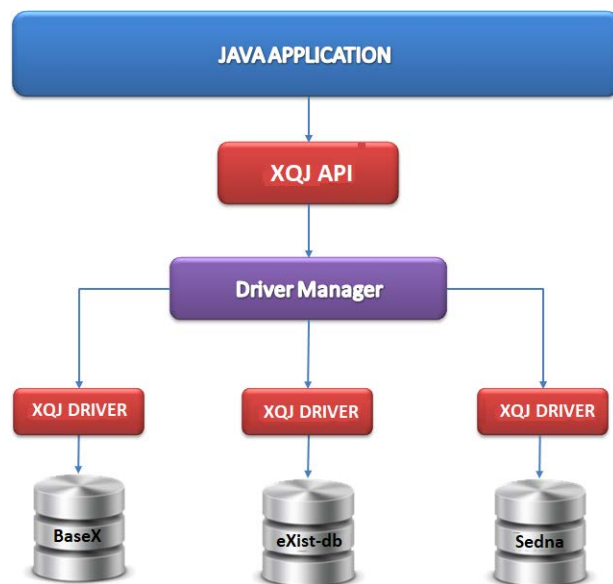
Què passa en intentar executar qualsevol dels programes? Doncs que NO troben el connector XQJ i el programa finalitza informant del problema.

d) Intro_4_ConnexioXQJAmbConnectorsXQJ

És idèntic al projecte anterior però incorpora els connectors XQJ per a BaseX, eXist-db, Oracle i Sedna i, per tant, el programa `Connexio` pot intentar establir la connexió, que serà existosa o no segons les dades de connexió dels fitxers de propietats siguin correctes i els corresponents SGBDR estiguin engegats i siguin accessibles per la xarxa.

El programa `Connexio`, en cas d'establir connexió, informa de la classe a la que pertany l'objecte apuntat per la referència `XQConnection xq`. Fixem-nos que:

- o BaseX: `net.xqj.basex.bin.c`
- o eXist-db: `net.xqj.exist.bin.bm`
- o Oracle: `oracle.xml.xquery.xqjimpl.OXQDConnection`
- o Sedna: `net.xqj.sedna.bin.bm`



Observem com cada fabricant ha implementat la seva versió de classe `XQConnection` en el seu connector XQJ. **Cada connector XQJ és un component que és substituïble en una aplicació.** En el projecte exemple, hi ha els quatre components XQJ per poder provar les 4 connexions, però en una implantació d'una aplicació, una vegada decidit quin és el SGBD, només s'instal·la el connector XQJ corresponent.

Un altre exemple seria l'ús de l'API XML-DB (predecessora de l'API XQJ).

- **Implementacions JPA per persistència sobre SGBDR.** Equivalents als connectors anteriors, però per gestionar la persistència.

Com a exemple d'utilització, tenim les capes de persistència desenvolupades en el M06-UF2, via JPA, on utilitzem la implementació facilitada per Hibernate i/o EclipseLink que podem substituir per altres proveïdors JPA.

Què tenen en comú XQJ, XML-DB, JDBC o JPA?

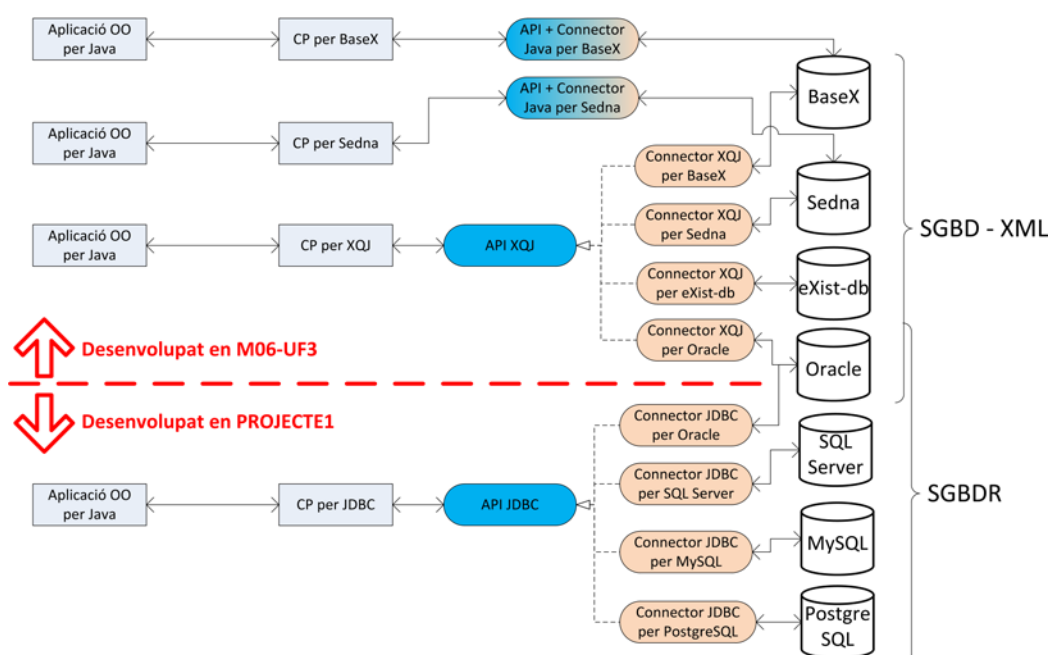
- Els tres fan referència a components d'accés a dades existents i que nosaltres ja estem acostumats a utilitzar en les nostres aplicacions.
- Si fem una ullada a la seva documentació, observem que es tracta d'una definició d'interfícies que defineixen el "contracte" que han de verificar els diversos connectors.

Objectiu de la UF4

Ser capaços de dissenyar una capa de persistència que pugui ser usada en una aplicació, totalment independent del tipus de l'origen de dades.

Exemple: Suposem que hem de desenvolupar una aplicació OO per gestionar una empresa.

- En UF3 hem après a fer muntatges com els de la part superior de la imatge següent, on l'aplicació OO seria gairebé la mateixa però... no és única, per què en el seu codi havíem incorporat `!import` de la CP a usar...
- En el projecte1 hem estat més "intel·ligents" que en UF3, doncs vàrem definir una "interfície de persistència" i vàrem desenvolupar la CP per JDBC implementant la interfície, de manera que l'aplicació OO no tenia `!import` d'una CP determinada, sinó que tenia `!import` de la interfície i podria usar qualsevol altra CP que implementés la interfície. Bravo!!!





On està la feina/ tasca d'aquesta UF? Aprofundir en la tècnica emprada en el desenvolupament del projecte 1, de manera que l'aplicació OO pugui ser única i pugui usar indistintament diverses CP. És a dir, les capes diverses CP que puguem desenvolupar han de ser intercanviables: **components!!!**

Càrrega dinàmica de components i invocació del seu constructor

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 `Intro_5_ComCrearObjecteDeComponent` mostra com cal actuar.



Exercici 1 – Convertir capes de persistència amb “mètodes similars” en components

Considerar la BD `Empresa` dins els diversos SGBD XML - SGBDR utilitzats a UF3: BaseX / Sedna / eXist-db modelada pel projecte `Empresa01` (UF3).

- En els SGBD-XML, les dades resideixen en un fitxer XML (similar a `empresa.xml`) validat per `empresa.dtd`, ubicat en base de dades/col·lecció/nom indicats en fitxers de configuració.

Fitxers `empresa.xml` i `empresa.dtd` es poden trobar dins projecte `Empresa01` de la UF3.

Partir des projectes desenvolupats en UF3, en carpeta `Ex1_Empresa_Capes_App_Materials`:

- 241118_1_EPBBaseX en UF3
Capa de persistència via API BaseX per a SGBD-XML BaseX
- 241118_1_EPBBaseXApp en UF3
Programa de proves per provar la capa en BaseX
- 241125_1_EPSedna en UF3
Capa de persistència via API Sedna per a SGBD-XML Sedna
- 241125_1_EPSednaApp en UF3
Programes de proves per provar la capa en Sedna
- 241209_1_EPXQJ en UF3
Capa de persistència via API XQJ per qualsevol SGBD-XML que faciliti connector XQJ
- 241209_1_EPXQJApp en UF3
Programes de proves per provar la capa en BaseX – Sedna – eXist-db

Conclusió: Es disposa de 3 capes de persistència gairebé idèntiques.

Què canvia?

El nom de la classe que facilita la persistència (`EPBaseX`, `EPSedna`, `EPXQJ`) i, en conseqüència, per utilitzar-les, ens cal tenir aplicacions diferents, ja que cada aplicació ha de declarar la importació de la corresponent classe.

Objectiu:

Convertir les 3 versions de `EPsGBD` en 3 **components** que puguin ser utilitzats indistintament per un programa (per exemple, els programes de proves que teníem a cadascuna de les APIs dissenyades a la UF3). Aquest programa, ara, no podrà estar lligat a cap de les APIs (no pot tenir cap `import` que faci referència a cap de les APIs i en temps d'execució ha de carregar la classe que correspongui).

Solució dins carpeta `Ex1_Empresa_Capes_App`:

1. Crear interfície `IGestorEmpresa` amb tots els mètodes (excepte constructors, doncs les interfícies no tenen constructors) desenvolupats a les diverses classes (per sort, varem dissenyar-los idènticament quan varem definir les classes).

Però els mètodes de les capes generen excepcions no comunes (`EPBaseXException`, `EPSednaException`, `EPXQJException` i `EPJPAException`) i això no pot ser així, ja que l'aplicació que utilitzi qualsevol de les capes, ha de poder capturar/gestionar les excepcions d'una única classe, independent de la capa. Per tant, cal definir una classe `exception` comuna, per exemple de nom `GestorEmpresaException` i substituir les excepcions generades per cada capa per aquesta classe.

Aquesta classe `GestorEmpresaException` acompanya sempre la definició de la interfície.

Projecte: `GestorEmpresa`



2. Evolucionar les 3 capes de persistència fent que implementin aquesta nova interfície. D'aquesta manera, una aplicació que vulgui utilitzar qualsevol de les capes de persistència, incorporará la interfície (`import GestorEmpresa`) i podrà invocar qualsevol dels seus mètodes i en temps d'execució, carregarà la classe de la capa que interressi utilitzar.

Projectes: EPBaseX, EPSedna i EPXQJ

En afegir EPBaseX implements `IGestorEmpresa`, topem amb els mètodes `commit()` i `rollback()` que no existien, doncs BaseX no gestiona transaccions. Ens cal afegir-los buits.

3. Aplicació/programa de proves

Ja tenim les 3 capes convertides a components que poden ser utilitzades indistintament per una aplicació que implementi la interfície. Recuperem programa de prova dels projectes originals i el retoquem pert a que pugui usar qualsevol dels 3 components de persistència desenvolupats:

- Eliminem qualsevol `import` específic de les classes que donen la persistència
- L'objecte `cp` que manté la connectivitat amb la BD el redeclarem com a `GestorEmpresa`
- No podem invocar cap constructor específic de les classes dels components a utilitzar i, per tant, hem d'efectuar una càrrega dinàmica de la classe que interressi, via `Class.forName(...)`.
- Cal incorporar a inici de programa mecanisme per poder-lo informar amb el nom de la capa a usar i el nom del fitxer de configuració amb les dades de connexió corresponents a la capa.

Projecte ProvaCapes...

4. Detall dels projectes desenvolupats i de les llibreries necessàries en cada projecte

Projectes	Llibreries/Projectes necessàries per compilar	Observacions
ModelEmpresa		Incorpora les classes del model (Empresa, Departament, Empleat,...)
GestorEmpresa	ModelEmpresa	Incorpora la interfície <code>IGestorEmpresa</code> i la classe <code>GestorEmpresaException</code>
EPBaseX	ModelEmpresa GestorEmpresa	
	JDom...	Llibreria que utilitzem en el codi desenvolupat
	BaseX...	L'API facilitada per BaseX per connectar amb els seus SGBD
EPSedna	ModelEmpresa GestorEmpresa	
	JDom...	Llibreria que utilitzem en el codi desenvolupat
	Sedna...	L'API facilitada per Sedna per connectar amb els seus SGBD
EPXQJ	ModelEmpresa GestorEmpresa	
	JDom...	Llibreria que utilitzem en el codi desenvolupat
	XQJ	La definició de l'API XQJ (<code>xqjapi.jar</code>) En aquest projecte no es necessiten les APIs específiques per als diversos SGBD amb els que volguem connectar via XQJ, doncs aquest projecte únicament incorpora la implementació de la interfície i no incorpora cap programa executable.
ProvaCapes	ModelEmpresa GestorEmpresa	

5. Llibreries necessàries per ProvaCapes en temps d'execució:

- Les que ja incorpora per poder compilar
- La llibreria de la capa que vulgui usar
- Les llibreries que necessita la capa a usar
- En el cas de XQJ, les llibreries específiques pel SGBD-XML amb el què es vol connectar

Comprovació de funcionament, en següent exercici

Exercici 2: Simular execució d'aplicació amb components

Comprovació de funcionament de ProvaCapes per les diferents capes:

- Projecte ProvaCapaBaseX, que executi ProvaCapes usant EPBaseX
- Projecte ProvaCapaSedna, que executi ProvaCapes usant EPSedna
- Projecte ProvaCapaXQJ-BaseX, que executi ProvaCapes usant EPXQJ per BaseX
- Projecte ProvaCapaXQJ-Sedna, que executi ProvaCapes usant EPXQJ per Sedna
- Projecte ProvaCapaXQJ-eXist, que executi ProvaCapes usant EPXQJ per eXist-db

Per comprovar les capes en els diferents SGBD podríem incorporar 5 llançadores en el mateix projecte ProvaCapes, però això obligaria a incorporar totes les llibreries i potser no quedaria prou clar quines són les llibreries imprescindibles per cada execució.

Per això es proposa desenvolupar 5 projectes independents, cadascun dels quals ha de tenir les llibreries imprescindibles.

Solució dins carpeta Ex2_Empresa_Execució:

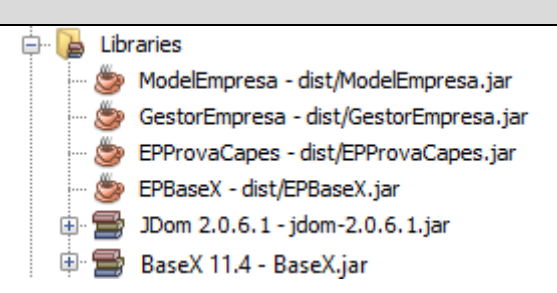
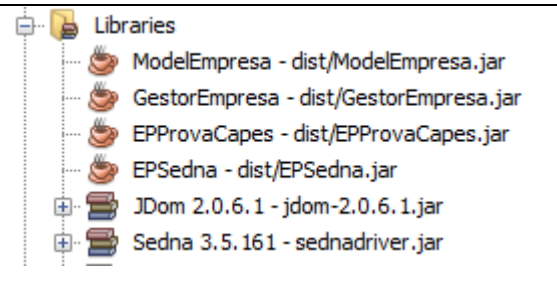
Les capes a comprovar resideixen a la carpeta Ex1_Empresa_Capes_App

La solució consisteix en crear, per a cada situació, un projecte amb una classe Main (o nom que vulgueu) que contingui:

```
public static void main(String[] args) {
    String t[] = {"nomFitxerConfiguracióQueCorrespongui"};
    ProvaCapes.main(t);
}
```

El main d'aquest projecte ha d'invocar el main de ProvaCapes indicant nom del fitxer de configuració que conté nomCapa i nomFitxerConfiguracioCapa per què main de ProvaCapes així ho exigeix. Si haguéssim desenvolupat main de ProvaCapes de manera que no obligués a rebre nom del fitxer de configuració i cerqués per defecte un fitxer de nom prefixat (per exemple infoCapa.properties), podríem estalviar-nos la declaració de la taula "t" i invocar directament ProvaCapes.main(args).

La imatge següent mostra el muntatge.

Component SGBD	Arxius incorporats	Llibreries i arxius incorporats
Projecte		
EPBaseX BaseX ProvaCapaBaseX	<p>infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPBaseX nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa: host=... port=1984 user=admin password=admin path=doc('BD/economia/empresa.xml')</p>	 <p>Libraries</p> <ul style="list-style-type: none"> ModelEmpresa - dist/ModelEmpresa.jar GestorEmpresa - dist/GestorEmpresa.jar EPProvaCapes - dist/EPProvaCapes.jar EPBaseX - dist/EPBaseX.jar JDom 2.0.6.1 - jdom-2.0.6.1.jar BaseX 11.4 - BaseX.jar
EPSedna Sedna ProvaCapaSedna	<p>infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPSedna nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa: host=... port=5050 bd=BD user=SYSTEM password=MANAGER path=doc('empresa.xml', 'economia')</p>	 <p>Libraries</p> <ul style="list-style-type: none"> ModelEmpresa - dist/ModelEmpresa.jar GestorEmpresa - dist/GestorEmpresa.jar EPProvaCapes - dist/EPProvaCapes.jar EPSedna - dist/EPsedna.jar JDom 2.0.6.1 - jdom-2.0.6.1.jar Sedna 3.5.161 - sednadrivers.jar



Component SGBD	Arxius incorporats	Llibreries i arxius incorporats
Projecte		
EPXQJ BaseX ProvaCapaXQJ-BaseX	<p>infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPXQJ nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa: className=net.xqj.basex.BaseXXQDataSource path=doc('BD/economia/empresa.xml') updateVersion=XQUF transactional=N # Propietats següents per connectar via XQJ serverName=... databaseName=BD port=1984 user=admin password=admin</p>	<p>Libraries</p> <ul style="list-style-type: none"> ModelEmpresa - dist/ModelEmpresa.jar GestorEmpresa - dist/GestorEmpresa.jar EPProvaCapes - dist/EPProvaCapes.jar EPXQJ - dist/EPXQJ.jar JDome 2.0.6.1 - jdome-2.0.6.1.jar XQJ - xqjapi.jar XQJ BaseX - basex-xqj-1.4.0.jar XQJ BaseX - xqj2-0.2.0.jar Apache Xerces 2.12.2
EPXQJ Sedna ProvaCapaXQJ-Sedna	<p>infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPXQJ nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa: className=net.xqj.sedna.SednaXQDataSource path=doc('empresa.xml','economia') updateVersion=PL transactional=Y # Propietats següents per connectar via XQJ serverName=... databaseName=BD port=5050 user=SYSTEM password=MANAGER</p>	<p>Libraries</p> <ul style="list-style-type: none"> ModelEmpresa - dist/ModelEmpresa.jar GestorEmpresa - dist/GestorEmpresa.jar EPProvaCapes - dist/EPProvaCapes.jar EPXQJ - dist/EPXQJ.jar JDome 2.0.6.1 - jdome-2.0.6.1.jar XQJ - xqjapi.jar XQJ Sedna - sedna-xqj-1.0.0.jar XQJ Sedna - xqj2-0.0.1.jar
EPXQJ eXist-db ProvaCapaXQJ-eXist	<p>infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPXQJ nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa: className=net.xqj.exist.ExistXQDataSource path=doc('/db/economia/empresa.xml') updateVersion=PL transactional=N # Propietats següents per connectar via XQJ serverName=... port=8080 user=admin password=admin</p>	<p>Libraries</p> <ul style="list-style-type: none"> ModelEmpresa - dist/ModelEmpresa.jar GestorEmpresa - dist/GestorEmpresa.jar EPProvaCapes - dist/EPProvaCapes.jar EPXQJ - dist/EPXQJ.jar JDome 2.0.6.1 - jdome-2.0.6.1.jar XQJ - xqjapi.jar XQJ eXist-db - exist-xqj-1.0.1.jar XQJ eXist-db - xqj2-0.0.1.jar

Com posar en marxa l'aplicació des de consola, sense NetBeans

En una instal·lació real, els usuaris no disposarien del NetBeans i ho executarien des de consola. El mateix NetBeans informa, en finalitzar el *Clean and Build* de l'ordre d'execució:

```
java -cp ModelEmpresa.jar;GestorEmpresa.jar;EPProvaCapes.jar org.milaifontanals.EPProvaCapes
```

Proveu de crear una carpeta en el vostre SO, ubicar-hi els 3 arxius jar implicats i des d'una consola de sistema executar l'anterior instrucció.

- Tant si executeu des de la consola com si ho feu des del projecte EPProvaCapes del NetBeans, apareix error relatiu a que no troba el fitxer de configuració infoCapa.properties.
- Incorporem el fitxer infoCapa.properties correctament emplenat (propietats nomCapa i nomFitxerConfiguracioCapa). Executem. Apareix error relatiu a que no troba la capa indicada dins el fitxer. Evident!



Cal afegir a la carpeta el jar corresponent a la capa i en la instrucció d'execució incorporar el corresponent jar en la clàusula `-cp`. Possibilitats d'execució:

- Si tots els jar es troben a la mateixa carpeta:

```
java -cp ./* org.milaifontanals.Main
```

- És habitual crear una carpeta `lib` amb els jar necessaris excepte el jar que conté la classe i:

```
java -cp ProvaCapaXXX.jar;lib/* org.milaifontanals.Main
```

- També es podria posar el jar de la classe dins la carpeta `lib` (no habitual) i executar:

```
java -cp lib/* org.milaifontanals.Main
```

En qualsevol cas, el(s) fitxer(s) de configuració han d'estar a la carpeta des d'on s'executa l'aplicació, a no ser que la documentació de la classe informés que han de residir en una carpeta concreta.

- c) Incorporem el fitxer indicat en el paràmetre `nomFitxerConfiguracioCapa` de `infoCapa.properties`.
- d) Suposem que la capa a emprar és `EPBaseX` i l'afegim. Executem. Error! Manca `JDom...`
- e) Afegim `JDom....` Executem. Error! Manca el fitxer de propietats a usar per la capa.
- f) Afegim el fitxer `EPBaseX.properties`. Executem. Error! Manca el connector `BaseX` que usa la capa.
- g) Afegim `BaseX....` Executem i si les dades de configuració són correctes i tenim connectivitat i el SGBD està engegat i existeix la BD i... etc... aconseguim executar l'aplicació.

És a dir... per poder executar l'aplicació `EPProvaCapes` es necessita un conjunt de llibreries i arxius de configuració que depenen de la capa a usar.

Com s'ha dit més amunt, en una instal·lació real s'instal·laria l'aplicació i tots els arxius jar i de configuració necessaris (segons component i SGBD) en una determinada ubicació i es configuraria la línia d'execució corresponent. Hi ha utilitats de tercers que permeten generar un executable que executa l'aplicació i engloba tots els elements necessaris.

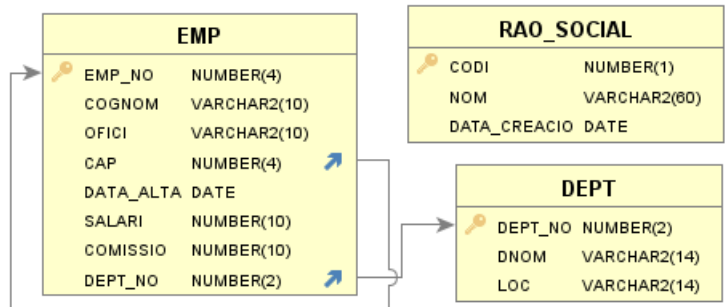


Exercici 3 – Desenvolupar capa JDBC

Considerar la versió relacional de la BD Empresa (disseny MR de la imatge) usada en exercicis anteriors i modelada pel projecte ModelEmpresa.

Es facilita els guions:

- empresaOracleMREstructura.sql
- empresaOracleMRDades.sql



Desenvolupar capa de persistència EPJDBC per a que pugui ser usada per l'aplicació EProvaCapes ubicada a la carpeta Ex1_Empresa_Capes_App de la mateixa manera que l'aplicació ja pot usar les capes EPBaseX, EPSedna i EPXQJ desenvolupades en exercici 1.

Desenvolupar projecte ProvaCapaJDBC-Oracle que comprovi funcionament d'EProvaCapes.

Recordeu com crear un usuari en Oracle:

```
DROP USER NOM_USUARI CASCADE;
/
CREATE USER NOM_USUARI IDENTIFIED BY CONTRASENYA;
/
GRANT CONNECT TO NOM_USUARI;
/
GRANT RESOURCE TO NOM_USUARI;
/
GRANT DROP ANY VIEW TO NOM_USUARI;
/
GRANT CREATE ANY VIEW TO NOM_USUARI;
/
GRANT UNLIMITED TABLESPACE TO NOM_USUARI;
/
```






Consideracions a tenir en compte abans d'implementar la solució:

- Excepte la consulta per cercar l'empresa, que sempre és igual, la resta d'instruccions haurien de ser parametritzades per guanyar en eficiència.
- Per fer més llegible el codi, totes les sentències parametritzades es poden crear en el constructor, una vegada establerta la connexió. D'aquesta manera, dins la resta de mètodes no cal anar consultant si està creada.
- Quan es crida un PreparedStatement, el ResultSet de l'anterior execució queda tancat. Això cal tenir-ho present en el mètode getEmpleat, on es fa una crida recursiva per cercar el cap i, per tant, abans de tornar-lo a invocar cal haver recollit totes les dades del RecordSet actual.
- En els casos en que cal executar varies instruccions d'actualització i es vol tenir la seguretat de que s'ha executat totes o cap, es pot crear un punt de salvaguarda (savepoint) abans de la primera instrucció i en cas de fallar, executar la recuperació (rollback) fins el punt de salvaguarda.

Solució en carpeta Ex3_Empresa_Capa_JDBC

La imatge següent mostra el muntatge desenvolupat:



Component SGBD	Arxius incorporats	Llibreries i arxius incorporats
Projecte		
EPJDBC Oracle ProvaCapaJDBC- Oracle	infoCapa.properties demanat per l'aplicació: nomCapa=org.milaifontanals.persistence.EPJDBC nomFitxerConfiguracioCapa=xxx.properties xxx.properties demanat per la capa: url=jdbc:oracle:thin:@//IP:1521/XEPDB1 user=... password=...	Classpath  ModelEmpresa - dist/ModelEmpresa.jar  GestorEmpresa - dist/GestorEmpresa.jar  EPProvaCapes - dist/EPProvaCapes.jar  EPJDBC - dist/EPJDBC.jar  JDBC Oracle

Exercici 4 – Desenvolupar capa JPA

Desenvolupar la capa EPJPA per gestionar les dades la versió relacional de la BD Empresa (exercici anterior) i modelada pel projecte ModelEmpresa.

Desenvolupar projectes següents per comprovar funcionament d'EPProvaCapes:

- ProvaCapaJPA-Hibernate-Oracle
- ProvaCapaJPA-EclipseLink-Oracle

Consideracions a tenir en compte abans d'implementar la solució:

- **Respecte la gestió de la classe Empresa (mètode `getEmpresa` i altres possibles)**

La classe Empresa no pot ser controlada per JPA, doncs per ser-ho, hauria de contenir un camp que es correspongués amb el camp codi de la taula RAO_SOCIAL i que estaria marcat amb @Id, i la classe Empresa no té aquest camp. Per tant, els pocs mètodes que calgui desenvolupar dins la capa, relatius a l'objecte Empresa, els desenvolupem amb consultes natives, sense usar JPA.

- **Retocs bàsics en les classes del ModelEmpresa (Departament i Empleat) per usar JPA.**

- a) Classes han de ser Serializable
- b) Classes han de tenir constructor per defecte protected
- c) Mètodes setter/getter no poden ser final.
- d) Camps que corresponen a claus primàries a la BD han de ser immutables

Habitualment, en el disseny de les classes, ja s'incorporen aquests requeriments, per poder-les usar en tecnologies habituals (JPA, JAXB,...). El nostre ModelEmpresa inicial no les incorporava...

Els requeriments *a-b-c-d* els incorporem en el ModelEmpresa de Ex1_Empresa_Capes_App per estalviar-nos una nova versió de ModelEmpresa i usar el mateix ModelEmpresa en totes les capes (desenvolupades i per desenvolupar). En principi no haurien d'afectar les capes desenvolupades ni els programes que les utilitzen (comprovem-ho obrint des de NetBeans tots els projectes desenvolupats que usen ModelEmpresa).

- **Ubicació de META-INF/persistence.xml => En projecte de capa o en projecte aplicació?**

Estaria molt bé poder-lo ubicar en el projecte de capa (EPJPA), però... hi ha uns continguts que canvien:

- segons el proveïdor JPA, en element <provider>
- segons les propietats de connexió, en elements:

```
<properties>
  <property name="jakarta.persistence.jdbc.url"
    value="???" />
  <property name="jakarta.persistence.jdbc.user" value="???" />
  <property name="jakarta.persistence.jdbc.password" value="???" />
  <property name="jakarta.persistence.jdbc.driver" value="???" />
</properties>
```

Tenim solucions:

- L'element <provider> NO és necessari. Si no s'especifica utilitza el primer proveïdor JPA que trobi entre les llibreries accessibles via classPath. Així doncs, si no el posem, usarà *Hibernate* si troba llibreries d'*Hibernate* o usarà *EclipseLink* si troba les seves llibreries o...
- I respecte el contingut de <properties>, en crear l'EntityManagerFactory es pot indicar un conjunt de propietats. Així doncs, podem obligar que el constructor de la capa de persistència carregui aquestes propietats des d'un fitxer de configuració.

Quina tècnica de marcatge usar?













Evidentment amb marcatge XML, doncs ModelEmpresa no pot contenir anotacions JPA en ser usat per altres capes.

Mínim marcatge de classes Departament i Empleat per a que hi hagi la correspondència adequada entre:

- Nom de la classe amb nom de la taula
- Nom dels camps de la classe amb nom de columna de la taula
- Correcta definició dels camps relacionals

Solució en carpeta Ex4_Empresa_Capa_JPA

La imatge següent mostra el muntatge desenvolupat, tenint en compte:

Component SGBD	Arxius incorporats	Llibreries i arxius incorporats
Projecte		
EPJPA Hibernate Oracle ProvaCapaJPA- Hibernate-Oracle	<p>infoCapa.properties demanat per l'aplicació:</p> <p>nomCapa=org.milaifontanals.persistence.EPJPA nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa, amb les variables:</p> <p>jakarta.persistence.jdbc.url= jakarta.persistence.jdbc.user= jakarta.persistence.jdbc.password= jakarta.persistence.jdbc.driver=</p>	<p>Classpath</p> <p> ModelEmpresa - dist/ModelEmpresa.jar  GestorEmpresa - dist/GestorEmpresa.jar  EPProvaCapes - dist/EPProvaCapes.jar  EPJPA - dist/EPJPA.jar  Hibernate 6.6.3  JDBC Oracle</p>
EPJPA EclipseLink Oracle ProvaCapaJPA- EclipseLink-Oracle	<p>infoCapa.properties demanat per l'aplicació:</p> <p>nomCapa=org.milaifontanals.persistence.EPJPA nomFitxerConfiguracioCapa=xxx.properties</p> <p>xxx.properties demanat per la capa, amb les variables:</p> <p>jakarta.persistence.jdbc.url= jakarta.persistence.jdbc.user= jakarta.persistence.jdbc.password= jakarta.persistence.jdbc.driver=</p>	<p>Classpath</p> <p> ModelEmpresa - dist/ModelEmpresa.jar  GestorEmpresa - dist/GestorEmpresa.jar  EPProvaCapes - dist/EPProvaCapes.jar  JDBC Oracle  EPJPA - dist/EPJPA.jar  EclipseLink 4.0.4</p>

Els fitxers XML que contenen el marcatge així com un fitxer persistence.xml sense <provider> ni <properties> els ubiquem en carpeta META-INF dins la capa EPJPA.

Exercici 5 – Incorporar factoria per capes que implementen GestorEmpresa

Considerem els components EPxxx generats en els exercicis 1-3-4, que implementen la interfície GestorEmpresa també generada a l'exercici 1.

Recordem que tenim els components EPSedna, EPBaseX, EPXQJ, EPJDBC i EPJPA.

Un programa que en vulgui utilitzar un, per exemple el main del programa EPProvaCapes, ha d'executar codi similar a:

```
GestorEmpresa cp = null;
try {
    if (nomFitxerConfiguracioCapa == null || nomFitxerConfiguracioCapa.equals("")) {
        cp = (GestorEmpresa) Class.forName(nomCapa).newInstance();
    } else {
        Class classe = Class.forName(nomCapa);
        Constructor con = classe.getConstructor(String.class);
        cp = (GestorEmpresa) con.newInstance(nomFitxerPropietatsCapa);
    }
    System.out.println("Capa de persistència creada!");
}
catch (Exception ex) {...}
```

Fixem-nos que el programador de l'aplicació ha de conèixer que ha d'invocar constructor sense paràmetres o ha de cercar un constructor de la classe corresponent a la capa de persistència, al qual se l'hi ha de passar el nom del fitxer de propietats corresponent a la classe...

Hi ha casos en que pot ser més complicat... En aquest cas tenim la "sort" que la capa de persistència té el constructor dissenyat amb un paràmetre per on rep el nom del fitxer de propietats i ell s'encarrega de cercar en el seu interior les dades que necessita per establir la connexió. Però... Suposem que el constructor fos del tipus:

```
nomCapa (String host, int port, String user, String password, ...)
```

Uf... El programador, per invocar la capa de persistència, hauria d'invocar:

```
getConstructor(String.class, Integer.class, String.class, String.class,...)
```

i... això és una mica feixuc...

Seria més fàcil si es disposés d'una factoria que s'encarregués de retornar l'objecte que correspongui que implementi GestorEmpresa. És a dir, poder fer:

```
GestorEmpresa cp = null;
try {
    obj = Factoria.getGestorGomponent(nomClasseComponent, ...paràmetres...);
    System.out.println("Capa de persistència creada");
} catch (Exception ex) {...}
```

Cal desenvolupar la factoria EPFactoria i el lògic és incorporar-la en el projecte on hi ha la interfície definida (sempre acompanyant la interfície).

Solució: Dins la carpeta **Ex5_Empresa_AccésCapesViaFactoria**.

- Nova versió de projecte GestorEmpresa amb incorporació de la factoria.
- Nova versió de projecte ProvaCapes amb retoc en el programa invocant la factoria.
- Còpia de totes les capes, actualitzant la versió del projecte GestorEmpresa.
- Còpia de tots els projectes de prova desenvolupats en exercicis 2-3-4, canviant les versions dels projectes GestorEmpresa i ProvaCapes incorporats.



Exercici 6 – Incorporar singleton per capes que implementen GestorEmpresa (Ex4)

Factory o Singleton?

La nostra factoria, cada vegada que s'invoca, genera un nou objecte de la classe component i, per tant, si el programador és distret i dins l'aplicació s'invoca més d'una vegada la factoria, doncs... tindrem establertes diverses connexions.

Podem evitar aquesta situació convertint la factoria en singleton (instància única) o mantenint la factoria i incorporant un versió singleton, (EPSingleton).

És convenient tenir només opció singleton?

No sempre té per què ser bona opció, ja que ens podria interessar, en un programa, treballar amb dues connexions diferents a diferents servidors o BD o utilitzant diferents usuaris i això l'opció singleton no ho permetria.

Projectes solució dins carpeta **Ex6_Empresa_AccésCapesViaSingleton**:

- Nova versió de projecte GestorEmpresa amb incorporació del singleton.
- Nova versió de projecte ProvaCapes amb retoc en el programa invocant el singleton.
- Còpia de la resta de projectes.

Atenció!

La utilització del singleton ha d'assegurar que facilita el component que ja existeix però... i si la capa que gestiona el component s'havia tancat? En aquest cas, que retorni el component que ja existia no interessa, ja que la capa està tancada...

Per solucionar això, ens interessa "descobrir" si la connexió de la capa està tancada... I per aconseguir-ho necessitem que les capes incorporin un mètode isClosed() que no havíem implementat. De fet, això hagués estat útil des de que vàrem implementar la primera capa EPBaseX a la UF3. Mai és tard...

Per tant:

- Incorporarem el mètode isClosed() a la interfície IGestorEmpresa.
- Implementem el mètode isClosed() a totes les classes.
- Utilitzem el mètode isClosed() dins el singleton.



Exercici 7 – Disseny de components des de zero

L'exercici 1 ha servit per convertir unes capes de persistència "similars" en components substituïbles en una aplicació. En aquell cas hem partit d'una capes ja fetes amb la "casualitat" que contenen gairebé els mateixos mètodes i per això ha estat força fàcil el muntatge.

En l'exercici 2 hem comprovat el correcte funcionament dels components en una mateixa aplicació.

En els exercicis 3 i 4 hem creat noves capes (JDBC i JPA) que mantenen compatibilitat amb les capes de l'exercici 1 i, per tant, la mateixa aplicació `EPProvaCapes` pot usar qualsevol dels 5 components.

El fet de tenir capes (UF3) i posteriorment passar-les a components, no és usual. Quan es desenvolupa una capa de persistència ja s'ha de pensar en que sigui un component doncs en un "futur" pot ser que es necessiti una nova capa "compatible" usant una altra tecnologia o SGBD.

En aquest cas, anem a dissenyar una utilitat, que encara no està desenvolupada, usant diverses tècniques/vies de manera que es comportin com a components, és a dir, que siguin substituïbles en una aplicació.

Però en aquest cas, encara no tenim res dissenyat... Per tant, com que sabem que pretenem tenir diversos components, començarem amb el procés adequat:

1. Dissenyar interfície amb mètodes a contenir i amb classe `Exception/RuntimeException` que correspongui.
En aquest exercici volem que la classe sigui `Exception`. En l'exercici 1 va ser `RuntimeException`.
2. Dissenyar cada component, usant les tècniques/vies que correspongui, implementant ja d'entrada la interfície.
3. Fer l'aplicació que pugui usar qualsevol dels components.

Ok? Doncs... què se'ns demana? **Utilitat que permeti carregar un arxiu XML en una BD XML.**

Us heu adonat que ni a la UF3 ni a la UF4, en tots els programes Java que hem fet, hem vist com introduir un arxiu XML en una BD i que sempre ho hem fet via la interfície gràfica que ens ha facilitat Sedna/BaseX/eXist-db?

Doncs això s'ha acabat. Volem una utilitat Java que ens ho faci i, volem fer-ho usant 3 camins diferents (per tant, 3 components):

- A) Usant l'API específica de Sedna, que només ens servirà per incorporar XML en una BD de Sedna.
- B) Usant l'API específica de BaseX, que només ens servirà per incorporar XML en una BD de BaseX.
- C) Usant l'API XQJ, que ens servirà per incorporar XML en una BD/XML que faciliti connector XQJ.

La utilitat, que podríem denominar `load` (bastant apropiat, no?), ha de permetre:

- Indicar el nom del fitxer a carregar
- Indicar el nom que ha de tenir dins la BD
- Indicar la col·lecció o jerarquia de col·leccions (p.e. `col1/col2/col3`) on ha de residir, de manera que:
 - ✓ Si no s'indica col·lecció, ha d'anar a parar a l'arrel.
 - ✓ Si s'indica col·lecció i no existeix, l'ha de crear i ubicar-hi l'arxiu.

Com es carrega un arxiu XML en una BD XML usant les APIs BaseX, Sedna, XQJ ???

Aquesta informació cal obtenir-la a partir de la documentació de les corresponents APIs. Com que en aquesta UF no es tracta de destinar temps a investigar en les APIs BaseX – Sedna – XQJ, teniu el codi que necessitareu usar per carregar el fitxer XML dins la carpeta `Ex7_UtilitatLoadXML_Materials`.

A títol informatiu:



- Les APIs específiques de Sedna i de BaseX faciliten mètodes per afegir un document dins la BD.
 - a) BaseX: Veure els mètodes que facilita la classe `ClientSession`.
 - b) Sedna: Veure els mètodes que facilita la interfície `SednaStatement`.
- L'API XQJ, no ho contempla, però Charles Foster ha començat a desenvolupar l'API XQJ2, extensió de l'API XQJ oficial, que sí incorpora aquesta facilitat i per la que es pot trobar la documentació i els corresponents jars a <http://xqj.net/maven/com/xqj2/xqj2/>. De fet, les implementacions d'XQJ desenvolupades per Charles Foster per a eXist-db, BaseX i Sedna, incorporen aquesta facilitat. Baixeu-vos `xqj2-0.2.0-javadoc.jar`, desempaqueteu-lo i feu una ullada a la documentació de `XQConnection2`.

Requeriments:

- Les capes han de generar excepcions de classe `Exception` (no `RuntimeException`)
- Faciliten una factoria.

Dissenyau també un programa que pugui invocar qualsevol de les 3 capes per carregar un XML.

Solució: Dins la carpeta **Ex7_UtilitatLoadXML**.

- Interfície + Classe Exception: `UtilsBDXml`
- Component per BaseX: `UtilsBDXmlBaseX`
 Necessita del paquet que defineix la interfície i de la llibreria específica de BaseX
- Component per Sedna: `UtilsBDXmlSedna`
 Necessita del paquet que defineix la interfície i de la llibreria específica de Sedna
- Component per XQJ: `UtilsBDXmlXQJ`
 Necessita del paquet que defineix la interfície i la llibreria que defineix XQJ i la que afegeix XQJ2 (versió més recent incorporada a XQJ de BaseX)
- Projecte de proves: `UtilsBDXmlApp`

Per poder comprovar l'aplicació en diverses capes, cal passar-li el nom de la capa (obligatori) i un possible nom de fitxer de configuració per la capa a usar. Això es pot fer de dues maneres:

- ✓ Passant la informació en un fitxer de propietats (com en exercicis 1-6 anteriors)
- ✓ Passant la informació via arguments (optem per aquesta opció)

Per la compilació només necessita el projecte `UtilsBDXml`

Si executéssim directament aquest programa, necessitaríem la llibreria específica de la capa a usar + fitxer de configuració per la capa, però com que invocarem aquest programa des de altres programes llançadora, no necessitem res més.

- Projectes llançadora per executar el programa de proves usant diverses capes:

✓ ProvaCapaBaseX	✓ ProvaCapaXQJ-Sedna
✓ ProvaCapaSedna	✓ ProvaCapaXQJ-eXist
✓ ProvaCapaXQJ-BaseX	

Per la seva execució, tots necessiten incorporar:

- ✓ `UtilsBDXml`
- ✓ `UtilsBDXmlApp`
- ✓ Projecte de la capa a usar (`UtilsBDXmlBaseX` o `UtilsBDXmlSedna` o `UtilsBDXmlXQJ`)
- ✓ Fitxer de propietats de cada capa per la connexió al corresponent SGBD
- ✓ Llibreries específiques (drivers connexió) per l'API en la que es basa la capa a usar

- ALERTA:



- a) Si intentem carregar diverses vegades el mateix arxiu, els diversos SGBD tenen resposta desigual:
 - a. BaseX i eXist-db no es queixen i substitueixen l'arxiu existent per la nova versió.
 - b. Sedna es queixa i genera excepció indicant que ja existeix un fitxer amb igual nom
- b) Què passa si el fitxer XML a carregar té en el seu interior etiqueta DOCTYPE?
 - a. Les APIs BaseX i Sedna no es queixen i no fan cas del contingut de DOCTYPE.
 - b. XQJ es queixa si no troba el corresponent fitxer dtd. Solució: comentar la línia DOCTYPE.



Exercici 8 – Components per executar un guió en SGBDR

Volem disposar d'utilitat `execute(File guió)` que permeti executar en un SGBDR un guió d'instruccions SQL codificat amb UTF sense BOM.

Donat que coneixem dues API per treballar amb SGBDR (JPA i JDBC), es demana dissenyar la utilitat en dos components (un via JPA i l'altre via JDBC) que siguin compatibles en qualsevol aplicació que necessiti usar la utilitat.

La utilitat ha d'ometre les sentències `SELECT` que pugui contenir el guió.

Requeriments:

- Les capes han de generar excepcions de classe `Exception` (no `RuntimeException`)
- Faciliteu una factoria.

Desenvolpeu també aplicació per comprovar el correcte funcionament dels 2 components executant qualsevol guió SQL sobre algun SGBDR (Oracle, PostgreSQL, MySQL,...)

Com executar un guió en una BDR usant les APIs JDBC, JPA ???

No hauria de ser difícil dissenyar el procés per assolir-ho. Com que en aquesta UF no es tracta de destinar temps a investigar-ho, teniu el codi que necessitareu usar per executar un guió (omitint sentència `SELECT`) dins la carpeta `Ex8_UtilitatExecutarGuioSGBDR_Materials`.

Solució dins carpeta **Ex8_UtilitatExecutarGuioSGBDR**:

- Interfície+Exception+Factoria: `UtilsBDR`
- Component JDBC: `UtilsBDRvJDBC`
 Necessita el projecte on està definida la interfície i la classe `Exception`.
 No necessita de cap llibreria JDBC, ja que l'API JDBC ja està incorporada en JDK.
- Component JPA: `UtilsBDRvJPA`
 Necessita el projecte on està definida la interfície i la classe `Exception`.
 Incorpora la llibreria que defineix JPA (Only Persistence JPA2.2), sense incorporar cap llibreria específica de cap fabricant.
- Aplicació proves: `UtilsBDRApp`
 Incorpora la classe `org.milaifontanals.proves.ProvaExecucioGuio` amb un `main` preparat per executar un guió via un component amb un determinat fitxer de configuració.
 Les tres dades (guió, component i fitxer de configuració del component) es reben per arguments en el moment d'efectuar la crida.
- Projectes llançadora per executar el programa de proves usant diverses capes:
 - ✓ `ProvaCapaJDBC-Oracle`
 - ✓ `ProvaCapaJPAEclipseLink-Oracle`
 - ✓ `ProvaCapaHibernate-Oracle`

Per la seva execució, tots necessiten incorporar:

- ✓ `UtilsBDR`
- ✓ `UtilsBDRApp`
- ✓ Projecte de la capa a usar (`UtilsBDRvJDBC` o `UtilsBDRvJPA`)
- ✓ Fitxers de configuració de cada capa per la connexió al corresponent SGBD
- ✓ Llibreries específiques (drivers connexió) per l'API en la que es basa la capa a usar

Tinguis present que JPA connecta amb SGBD via JDBC i cal el corresponent driver pel SGBDR.