Informàtica

ICB0 Desenvolupament d'aplicacions multiplataforma

M06 Accés a dades

UF2 Persistència en BDR-BDOR-BDOO

Diari d'activitats

Isidre Guixà

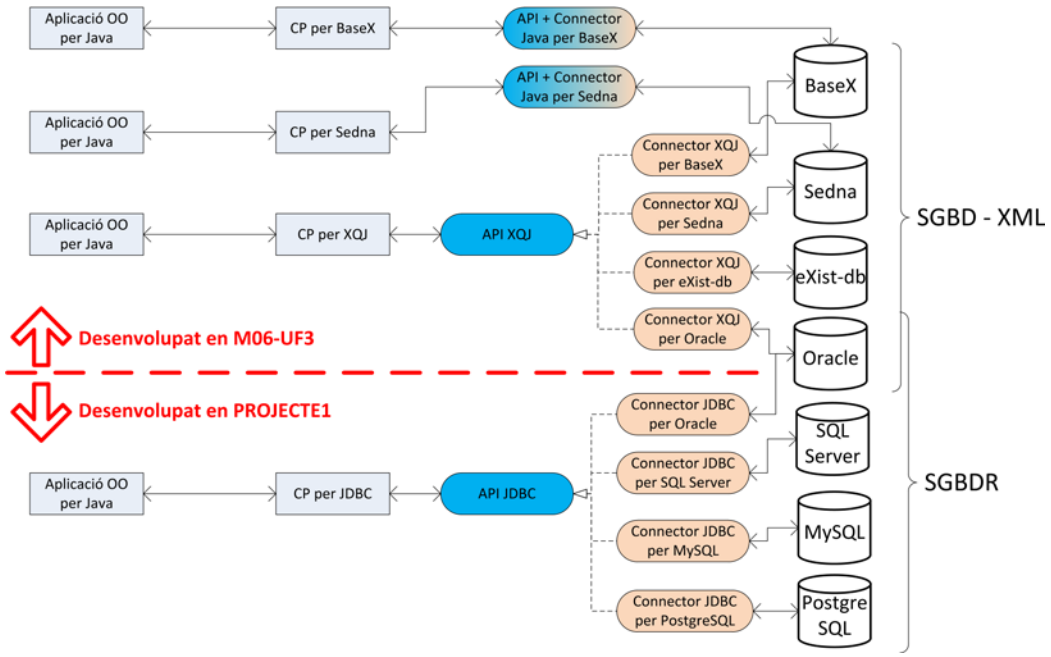
Curs 2022/23

Què sabem fer?	**02-12-22**

En UFs precedents hem practicat **el desenvolupament d'aplicacions OO que gestionen dades enregistrades en un SGBD amb l'objectiu final que l'aplicació OO sigui independent del SGBD** a emprar.

S'ha practicat usant el llenguatge Java, però també es pot efectuar en altres llenguatges pels que existeixi connectors adequats per a cada SGBD, ja sigui facilitats pel propi fabricant del SGBD o per tercers.

Per aconseguir que l'aplicació sigui independent del SGBD, cal desenvolupar una capa de persistència (CP) que contingui un conjunt complet de mètodes que s'encarreguin del diàleg amb el SGBD (consultes i actualitzacions) de manera que l'aplicació pugui ser única i invoqui, quan hagi de dialogar amb el SGBD, els mètodes de la CP que són els encarregats del diàleg. La imatge mostra els tipus de desenvolupaments practicats:



- En M06-UF3 hem desenvolupat:

- CP per a BaseX, utilitzant l'API i connector propietaris de BaseX. Aquesta capa només serveix per a connectar amb SGBD-XML BaseX.

- CP per a Sedna, utilitzant l'API i connector propietaris de Sedna. Aquesta capa només serveix per a connectar amb SGBD-XML Sedna.

- CP per a XQJ, que és una API (definició de funcionalitats) ideada per establir un protocol de connexió amb SGBD-XML i que permeti connectar amb qualsevol SGBD-XML que faciliti connector XQJ.

  Aquesta CP, en la fase d'implementació (escriptura de codi i compilació) només necessita de la llibreria on hi ha la definició de l'API (`xqjapi.jar`). En canvi, en temps d'execució, per connectar amb un SGBD-XML concret, precisa del connector que implementi l'API XQJ específic pel SGBD.

  A diferència de les CP anteriors que només servien per 1 SGBD, aquesta CP serveix per a qualsevol SGBD-XML que faciliti connector XQJ.

- En PROJECTE1 hem desenvolupat:

- CP per a JDBC, que és una API (definició de funcionalitats) ideada per establir un protocol de connexió amb SGBDR i que permeti connectar amb qualsevol SGBDR que faciliti connector JDBC.

  Aquesta CP, en la fase d'implementació (escriptura de codi i compilació) no necessita cap llibreria especial per què la definició de l'API està incorporada en JDK. En canvi, en temps d'execució, per connectar amb un SGBDR concret, precisa del connector que implementi l'API JDBC específic pel SGBD.

  De forma similar a la CP per a XQJ, aquesta CP serveix per a qualsevol SGBDR que faciliti connector JDBC.

Suposem que les diverses CP desenvolupades contenen el mateix conjunt de mètodes. Llavors, la problemàtica inicial a resoldre (aplicació OO independent de qualsevol SGBD) ja està assolida?

NO!!! A la imatge veiem que no hi ha 1 única aplicació, sinó 1 aplicació per a cada CP. La resolució completa l'assolirem a la UF4.

- En M06-UF3, el programa que utilitza la capa de persistència és idèntic en els 3 casos excepte en 3 punts:
  (1) Incorporació de la capa a la zona `import`:
  `import nomCapaPersistencia;`
  (2) Creació de l'objecte de la CP per invocar els mètodes de la capa:
  `nomCapaPersistencia cp = new nomCapaPersistencia(...);`
  (3) Utilització de la classe `Exception` que correspongui segons capa de persistència usada.

  En conseqüència l'aplicació està lligada a la CP; no està lligada al SGBD però de moment continua lligada a la CP. La independència total, també de la CP, l'assolirem a la UF4.

  De la mateixa manera que les CP desenvolupades en M06-UF3 gestionaven dades d'una empresa (classes `Empresa`- `Departament`-`Empleat`) emmagatzemades en SGBD-XML en format XML seguint específics DTD, podríem desenvolupar una CP amb idèntics mètodes gestionant dades emmagatzemades en SGBDR via JDBC. El programa seria el mateix; només caldria efectuar els canvis dels 3 punts anteriors.

- En PROJECTE1, passa el mateix via JDBC. L'aplicació desenvolupada incorpora els 3 punts anteriors.

  De la mateixa manera que s'ha desenvolupat la CP per JDBC, es podria definir un format XML per emmagatzemar les dades de les classe gestionades en PROJECTE1 i es podria desenvolupar:
  - Una CP amb idèntics mètodes gestionant dades emmagatzemades en SGBD-XML, usant l'API XQJ.
  - Una CP amb idèntics mètodes gestionant dades emmagatzemades en BaseX, usant l'API pròpia.
  - Una CP amb idèntics mètodes gestionant dades emmagatzemades en Sedna, usant l'API pròpia.

  L'aplicació seria la mateixa; només caldria efectuar els canvis dels 3 punts anteriors.

La imatge distingeix amb color blau les API (definició de funcionalitats) que ha d'invocar la CP i amb color taronja els connectors (classes concretes que implementen l'API). Els connectors es necessiten en temps d'execució, per establir la connexió amb el SGBD que pertoqui. Les API són necessàries en temps d'implementació.

En el cas de les APIs específiques/propietàries (per exemple BaseX i Sedna) és força comú que API i classes estiguin en una mateixa llibreria, però en les APIs estàndards (XQJ i JDBC) sempre estan separades, doncs per una banda existeix la definició de l'API, elaborada per l'organisme/grup d'experts que pertoqui i per altra banda existeix la implementació concreta de les classes, específica per a cada SGBD.

Què farem a la UF2?	**02/12/22**

El programa oficial de la UF2 incorpora:

- Implementació de capes de persistència per a BDR-BDOR-BDOO
- Eines de mapatge ORM

La programació de la UF2 en el M&F desestima destinar part de la UF2 en la implementació de capes de persistència per a BDR-BDOR-BDOO per dos motius:

- Amb el desenvolupament de capes de persistència efectuat a la UF3 i els coneixements de JDBC i BDOO obtinguts a UF6 de M03, queda coberta la implementació de capes de persistència per a BDR-BDOR-BDOO.
- Les empreses demanden el coneixement d'eines ORM, fet que precisa d'un elevat nombre d'hores.

Per aquest motiu, en el M&F **la UF2 es centra en l'accés a dades via eines ORM**.

Eines de mapatge objecte-relacional (ORM) – Introducció - Dossier de l'IOC (només ORM)	**13-12-22**

2.1. Concepte de mapatge objecte-relacional
2.2. Eines de mapatge
2.2.1. Tècniques de mapatge
2.2.2. Llenguatge de consulta
2.2.3. Tècniques de sincronització (JDO, JPA 2.0 – JSR 317, JPA 2.1/2.2 - JSR 338)
2.3.3. Característiques generals del JPA

Javadocs oficials: JPA 2.0 – JSR 317, JPA 2.1/2.2 – JSR 338.

Actualitat de les versions de JDO – JPA (no instal·leu cap de les versions que indica el dossier)

L'agost de 2017 es publica JPA 2.2 com a revisió de JPA 2.1 dins el mateix JSR 338, del que podem veure un resum de les novetats aquí.

Versions estables que suporten JPA 2.2 existents en el moment de començar aquesta UF en el curs 22/23:
- Versió estable d'Hibernate: 5.6.14 (4/11/22) que compleix estàndard JPA 2.2 i **requereix JDK8/11/17/18**
- Versió estable d'EclipseLink: 2.7.11 (10/08/22) que compleix l'estàndard JPA 2.2 i **requereix JDK8**

L'octubre de 2020 s'ha publicat l'estàndard Jakarta Persistence 3.0 que forma part de Jakarta EE, que és l'evolució *open source* de Java EE, propietat d'Oracle.
- Les versions 3.x i 4.x d'EclipseLink donen suport a Jakarta Persistence 3.0. Cal **JDK11/17**
- Les versions 5.6.x d'Hibernate són compatible amb JPA 2.2 i Jakarta Java 3.0. Cal **JDK8/11/17/18**
- Les versions 6.x d'Hibernate són compatibles amb Jakarta Java 3.1 i 3.0. **Cal JDK11/17/18**

Si comparem la documentació de JPA 2..2 amb la documentació de Jakarta Persistence 3.0, entre altres canvis, observem diferència en els noms dels paquets:

JPA 2.2	Jakarta Persistence 3.0
`javax.persistence`	`jakarta.persistence`
`javax.persistence.criteria`	`jakarta.persistence.criteria`
`javax.persistence.metamodel`	`jakarta.persistence.metamodel`
`javax.persistence.spi`	`jakarta.persistence.spi`

DataNucleus (JPOX) és un producte OpenSource que compleix els 2 estàndards:
- JPA 2.2+ / Jakarta 3.0+
- JDO 3,2+

Tots aquests productes, a més d'ORM, faciliten solucions per altres tipologies de SGBD!!! Més informació a:

- JPA vs Hibernate
- Java Persistence API (JPA)

Productes JPA2.2 dels que utilitzarem en el curs 22/23 el connector JPA:

- Hibernate 5.6.14. S'aconsella crear a NetBeans la biblioteca amb nom
  `Hibernate 5.6.14`
  ja que els projectes solució que es facilitaran en aquest dossier incorporaran una biblioteca amb aquest nom.

  El dossier de l'IOC explica que les llibreries que cal carregar són les que hi ha dins la carpeta `jpa` i `required`. Això era així fins versions 5.1.x. A partir de 5.2, la carpeta `jpa` deixa d'existir.

  La nostra biblioteca ha de contenir les llibreries de la carpeta `lib/required` d'Hibernate 5.6.14.

  En 1/12/2022 l'enllaç anterior (definit a la web d'Hibernate i que porta a *SourceForge*) no funciona i deixa descarregar la versió 5.6.14. No ens podem descarregar tot l'Hibernate (no el necessitem) però com que només necessitem les llibreries requerides, tenim la sort de trobar-les aquí.

- EclipseLink 2.7.11. S'aconsella crear a NetBeans la biblioteca amb nom
  `EclipseLink 2.7.11`
  ja que els projectes solució que es facilitaran en aquest dossier incorporaran una biblioteca amb aquest nom.

  Aquesta biblioteca ha de contenir els jars següents d'EclipseLink 2.7.11:
  - `jlib/jpa/jakarta.persistence_2.2.3.jar`
  - `jlib/jpa/ org.eclipse.persistence.jpa.modelgen_2.7.11.v20220810-51b5b24bf1`
  - `jlib/jpa/ org.eclipse.persistence.jpars_2.7.11.v20220810-51b5b24bf1`
  - `jlib/eclipselink.jar`

  Fixem-nos que EclipseLink ja no proporciona `javax.persistence`... sinó `jakarta.persistence`..., cosa que succeeix des de que es va posar en marxa el projecte Jakarta. Però internament els noms dels paquets encara coincidien amb els de JPA2.2. Ha estat a partir de Jakarta 3 que s'ha canviat els noms dels paquets.

Compte! NetBeans incorpora algunes llibreries de JPA que no es corresponen amb les versions que utilitzarem en aquest curs. Per tant, cal descarregar els productes indicats i utilitzar els `jars` indicats.

En aquestes llibreries, introduir com a javadoc, el del projecte JPA 2.1/2.2 – JSR 338.

Peces bàsiques de JPA - Dossier de l'IOC (només ORM)	**15/12/22**
2.4.1. Entitats **2.4.2. El gestor d'entitats** **2.4.3. Unitats de persistència**	

Definició d'Unitat de Persistencia – Creació d'`EntityManager` — 15/03/22

- La UP ha d'englobar el conjunt de propietats necessàries per configurar la connexió amb el SGBD i la funcionalitat de la persistència.
- La UP ha d'existir dins un fitxer dins un fitxer de nom `persistence.xml` ubicat dins la carpeta `META-INF` a l'arrel del projecte. Aquest fitxer (que és únic) pot contenir més d'una unitat de persistència, cadascuna amb un nom diferent. Aquest nom és el què s'indica en el mètode per crear l'`EntityManagerFactory`:

  ```
  EntityManagerFactory emf = Persistence.createEntityManagerFactory ("nomUP")
  ```

  I dins la UP del fitxer `persistence.xml` hi ha les propietats necessàries per aconseguir la connexió amb el SGBD i també altres propietats referents al funcionament de la persistència.
- Les propietats de la UP definides dins el fitxer `persistence.xml` poden ser substituïdes/completades en la creació de l'`EntityManagerFactory`, utilitzant un mètode alternatiu de creació:

  ```
  EntityManagerFactory emf = Persistence.createEntityManagerFactory("nomUP",props)
  ```

  on `props` és un conjunt de propietats que substitueix/completa les existents en el `persistence.xml`.

  En algun dels projectes que desenvoluparem utilitzarem aquesta possibilitat.

Una vegada obtingut l'`EntityManagerFactory` cal obtenir l'`EntityManager` via `createEntityManager`.

Els entorns de desenvolupament, com per exemple NetBeans, acostumen a facilitar la possibilitat de crear l'arxiu de persistència.

Exemple en NetBeans curs 22/23:

- Crear projecte Java nou i seleccionar el seu nom.
- `File | New File | Persistence | PersistenceUnit`

Observem que la versió que tenim de NetBeans en permet seleccionar EclipseLink (JPA 2.1) que no és el què ens interessa. Possiblement es podria configurar NetBeans per a JPA2.2...

Creació manual (nostre cas):

- Crear carpeta `META-INF` a l'arrel del projecte
- Crear dins un arxiu `persistence.xml` amb capçalera següent segons versió:

Capçalera del fitxer `persistence.xml` per JPA 2.0:

```
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
```

Capçalera del fitxer `persistence.xml` per JPA 2.1:

```
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
```

Capçalera del fitxer `persistence.xml` per JPA 2.2: **Capítol 8 de doc. oficial de JPA2.2 (8.2.1)**

```
<persistence version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
```

Cal finalitzar el fitxer `persistence.xml` amb `</persistence>`. L'element `<persistence>` ha de contenir en el seu interior les unitats de persistència que calgui.

Compte amb NetBeans!!! En obrir un arxiu `persistence.xml`, NetBeans presenta un formulari per emplenar-lo... Però és per EclipseLink (JPA 2.1) i no ens serveix. Cal seleccionar la pestanya superior *Source* per veure el codi del fitxer i gestionar nosaltres directament el seu contingut.

⇨ El fitxer `persistence.xml` ha de validar l'esquema `persistence_2_2.xsd` adjunt.

Cada unitat de persistència és un node similar a:

```
<persistence-unit name="nom up" transaction-type="RESOURCE_LOCAL">
  <provider>classe JPA del fabricant</provider>
  <properties>
    <property name="javax.persistence.jdbc.url"
              value="URL jdbc per connectar amb SGBD"/>
    <property name="javax.persistence.jdbc.user" value="usuari"/>
    <property name="javax.persistence.jdbc.password" value="contrasenya"/>
    <property name="javax.persistence.jdbc.driver"
              value="classe JDBC"/>
    <!-- més propietats property -->
  </properties>
</persistence-unit>
```

Element `provider`

L'element `provider` dins la unitat de persistència ha de contenir el nom de la classe principal que implementa JPA, facilitada pel corresponent fabricant:

- Hibernate <= 5.1.x: `org.hibernate.ejb.HibernatePersistence`
- Hibernate >= 5.2.x i posteriors: `org.hibernate.jpa.HibernatePersistenceProvider`
- EclipseLink : `org.eclipse.persistence.jpa.PersistenceProvider`

Element `properties`

- Propietats que defineixen la connexió (driver JDBC, url JDBC, usuari, contrasenya,...)

- En NetBeans curs 21-22, si `provider` és EclipseLink, en generar una `property` i escriure `name=`, mostra les propietats estàndards de JPA i les pròpies d'EclipseLink. En Hibernate, només JPA ☹!

- Dades per connectar amb Oracle (port habitual: 1521)
  Driver: `ojdbc8.jar` (ubicat a `pathOnEsOracle\dbhomeXE\jdbc\lib`)
  URL: `jdbc:oracle:thin:@//IP:PORT/nomBD`
  Classe JDBC: `oracle.jdbc.driver.OracleDriver`

- Dades per connectar amb PostgreSQL (port habitual: 5432)
  Driver: `postgresql-42.2.5.jar` (aula del Classroom)
  URL: `jdbc:postgresql://IP:PORT/nomBD`
  Classe JDBC: `org.postgresql.Driver`

- Dades per connectar amb MySQL8 (aules Milà) (port habitual: 3306)
  Driver: `mysql-connector-java-8.0.13` (aula del Classroom)
  URL: `jdbc:mysql://IP:PORT/nomBD`
  Classe JDBC: `com.mysql.cj.jdbc.Driver`

- Dades per connectar amb MySQL5 (servidor núvol) (port habitual: 3306)
  Driver: `mysql-connector-java-5.1.41-bin_JDJ8-7-6` (aula del Classroom)
  URL: `jdbc:mysql://IP:PORT/nomBD`
  Classe JDBC: `com.mysql.jdbc.Driver`

- **Compte en MySQL8**: Si apareix error `java.sql.SQLException: The server timezone value … is unrecognized or represents more than one timezone` completar la URL com segueix:
  `jdbc:mysql://IP:PORT/nomBD?serverTimezone=UTC`

Els projectes solució que es facilitaran en aquest dossier incorporaran biblioteques amb noms:

- **`JDBC Oracle`**, amb la llibreria adequada JDBC per connectar amb l'Oracle que correspongui
- **`JDBC PostgreSQL`**, amb la llibreria adequada JDBC per connectar amb el PostgreSQL que correspongui
- **`JDBC MySQL`**, amb la llibreria adequada JDBC per connectar amb MySQL que correspongui

- Propietats específiques de cada proveïdor, com:
- `hibernate.dialect`, per indicar el llenguatge de comunicació amb el SGBDR; si un projecte no té aquesta propietat i hi ha algun problema que no permet la connexió (per exemple contrasenya errònia), Hibernate no informa correctament de la causa del problema per què no té el dialecte informat. A més a més, en cas que Hibernate hagi de procedir a la creació o modificació de taules (ho veurem més endavant), si el dialecte no és l'adequat, pot no generar adequadament les sentències.
  Per veure com emplenar aquesta variable, segons el SGBD a accedir, cal consultar la documentació (javadocs) incorporada en la versió d'Hibernate descarregada i cercar dins el paquet `org.hibernate.dialect` quina classe cal indicar com a valor de la propietat. Exemple:

  - Oracle21c: `org.hibernate.dialect.Oracle12cDialect`
  - PostgreSQL9.6 (servidor núvol): `org.hibernate.dialect.PostgreSQL96Dialect`
  - PostgreSQL10 (aules Milà): `org.hibernate.dialect.PostgreSQL10Dialect`
  - MySQL5 (servidor núvol): `org.hibernate.dialect.MySQL57Dialect`
  - MySQL8 (aules Milà): `org.hibernate.dialect.MySQL8Dialect`

- Altres que anirem incorporant a mida que es necessitin

Informació referent a les Unitats de Persistència dels projectes adjunts:

El fitxer `persistence.xml` incorpora tres UP:

- `UP-Oracle`, per connectar amb un Oracle de la VM
- `UP-PostgreSQL`, per connectar amb un PostgreSQL 10 (aules del Milà)
- `UP-MySQL`, per connectar amb un MySQL8 (aules del Milà)

Si voleu connectar amb un PostgreSQL i/o MySQL diferents, tingueu en compte dades a canviar.

Canvieu també, evidentment, dades de connexió com IP, BD, usuari i contrasenya.

Exemples de projectes amb diverses configuracions:

A continuació es presenten 6 projectes per fer diverses comprovacions. Tots ells contenen 3 configuracions d'execució, que invoquen el mateix programa passant per paràmetre el nom d'una unitat de persistència concreta (UP-Oracle, UP-MySQL, UP-PostgreSQL).

Proveu-los, com a mínim, en Oracle i en un PostgreSQL (el del núvol o un que hagueu instal·lat en M10). Penseu en adequar les dades de connexió de les unitats de persistència.

Comprovació 1: Projectes
- `221216_1_JPA_HibernateNomesLibJPA`
- `221216_2_JPA_EclipseLinkNomesLibJPA`

El programa només intenta crear l'`EntityManagerFactory`.

Els dos projectes només incorporen la llibreria JPA 2.2, és a dir, NO la implementació d'Hibernate o d'EclipseLink.

- En el cas d'Hibernate, la llibreria **`Hibernate 5.6.14 (Only JPA2.2)`** constituïda pel `javax.persistence-api-2.2.jar` de la carpeta `lib/required` d'Hibernate 5.6.14.
- En el cas d'EclipseLink, la llibreria **`EclipseLink 2.7.11 (Only JPA2.2)`** constituïda pel `jakarta.persistence_2.2.3.jar` de la carpeta `jlib/jpa` d'EclipseLink 2.7.11.

En ambdós casos, els programes es compilen, però a l'hora d'executar-los, en qualsevol SGBD, peten per què no tenen accés a la llibreria del corresponent proveïdor ORM (Hibernate / EclipseLink).

Comprovació 2: Projectes
- `221216_3_JPA_HibernateJPAComplet` amb llibreria **`Hibernate 5.6.14`**.
- `221216_4_JPA_EclipseLinkJPAComplet` amb llibreria **`EclipseLink 2.7.11`**

Mateix programa que els projectes anteriors, que només intenta crear `EntityManagerFactory`.

Els dos projectes incorporen les llibreries del corresponent proveïdor ORM (Hibernate / EclipseLink). En teoria, la seva execució hauria de crear la factoria (`EntityManagerFactory`) doncs ja es té accés al proveïdor.
- En Hibernate no té lloc la situació teòrica... En crear la factoria, peta amb error:

  Unable to load class [oracle.jdbc.driver.OracleDriver]

  L'error diu que no troba la classe JDBC per connectar. Però si li afegim la llibreria, continuarà petant fins que no aconsegueix la connexió. És a dir, Hibernate estableix la connexió en el moment de crear la factoria.
- En EclipseLink sí té lloc la situació teòrica i la factoria es crea, sense intentar establir connexió.

Comprovació 3: Projectes
- `221216_5_JPA_HibernateJPA+JDBC`
- `221216_6_JPA_EclipseLinkJPA+JDBC`

El programa, a més de crear la factoria, també crea el gestor d'entitats. En aquest moment EclipseLink és quan estableix la connexió.

Els dos projectes incorporen els connectors JDBC per als 3 SGBDR i els programes ja s'executen en qualsevol dels 3 SGBD sempre i quan puguin establir la connexió amb el corresponent SGBD segons la informació subministrada en la UP.

Si tot va bé, apareixeran els missatges:

```
EntityManagerFactory creada
EntityManager creat
EntityManager tancat
EntityManagerFactory tancada
```

Hibernate dona molts missatges informatius en vermell, que també apareixen per la consola. Normal. Feu una ullada al contingut.

EclipseLink també treu informació, en menor quantitat i en negre.

La quantitat d'informació que donen aquestes eines acostuma a ser configurable via propietats específiques de les eines.

Com funciona JPA?	**20//12/22**

A l'apartat 2.2.1 del dossier s'explica la tècnica que utilitzen les diferents tipologies d'eines ORM. En qualsevol cas es tracta de poder informar a l'eina ORM de les classes que son persistents, és a dir, les classes de les que podem tenir objectes que interessi emmagatzemar en un SGBDR.

Per tant, en una aplicació conviuran:
- Classes no persistents: cap dels seus objectes necessita ser emmagatzemat en una BD
- Classes persistents: algun dels seus objectes (no tots) necessiten ser emmagatzemats en una BD

En el cas de JPA:
- Per informar de les classes que són persistents, JPA facilita dos mecanismes:
  - "Marcar" les classes persistents amb anotacions (similar al marcatge de JAXB)
  - Introduir les "marques" en arxiu `xml`, sense tocar el codi de la classe

  Poden conviure els dos mecanismes i, si per una classe existeix mapatge XML i mapatge via anotacions, el

mapatge XML preval sobre les anotacions. Això permet que les anotacions JPA que pugui contenir una classe en el seu codi siguin sobreescrites per marques diferents via XML
- Per informar quins objectes d'una classe persistent han de passar a ser "entitats" gestionades per JPA (és a dir, objectes emmagatzemats a la BD), el programador disposa de mètodes en l'`EntityManager`.

Independentment del tipus de mapatge (anotacions o XML), hem de tenir present que l'eina ORM fa la traducció:

objecte de classe ⇔ fila de taula

tot i que en alguna ocasió, les dades d'un objecte pot quedar emmagatzemades en taules diferents:

objecte de classe ⇔ files de taules

i respecte les taules de la BD hem de saber que l'eina ORM pot estar configurada per a que, en posar-se en marxa, respecte les classes "marcades" com a persistents:
- Creï les taules corresponents si encara no existeixin.
- Comprovi si les taules corresponents coincideixen amb el "marcatge"
  - Si no coincideixen, executi `alter table` per aconseguir la coincidència
  - Si no coincideixen, generi excepció.
- Elimini i creï de nou les taules

No totes les eines JPA faciliten les mateixes possibilitats. L'estàndard JPA marca unes possibilitats concretes (que veurem) però cada eina JPA les "aplica" a la seva manera i no hi ha uniformitat.

Per tant, en tots els projectes que desenvolupem al llarg de la UF, haurem de tenir en compte com ens interessa configurar l'eina (Hibernate o EclipseLink), segons vulguem que mantingui o no les dades i l'estructura de les taules a la BD.

Requeriments inicials per la classe - Capítol 2.1 de documentació oficial de JPA2.2	**20/12/22**

- Obligatorietat de constructor sense paràmetres, públic o protegit
- `Serializable`
- No pot ser final
- Ha de tenir obligatòriament un(s) camp(s) que la identifiquin i han de ser **IMMUTABLES**.

Configuració mínima a les classes per començar a fer proves - Vídeo	**20/12/22**

- Primeres anotacions obligatòries
  - Nivell de classe: `@Entity,` per indicar que és una classe persistent
  - Nivell de dades: `@Id,` per indicar camp PK (més endavant ja s'explicarà com fer en claus compostes)

- Desenvolupem una primera classe que contingui un camp de cada tipus de dada habitual en Java per veure com els diversos proveïdors JPA els mapen en els diversos SGBD.

  Projectes amb la classe `ProvaTipusDades` amb configuració mínima:
  `221220_1_ProvesTraduccióTipusDades_Hibernate`
  `221220_2_ProvesTraduccióTipusDades_EclipseLink`
  La versió per EclipseLink necessita la `property eclipselink.canonicalmodel.subpackage` (situació estranya)

- La carpeta `META-INF` amb el fitxer `persistence.xml` ha de residir a l'arrel de l'aplicació que invoca la persistència (no en el projecte que contingui les classes).

- Incorporem a la UP les classes que estan marcades, per a que el proveïdor de persistència les tingui en compte, en element `<class>` (en ubicació segons indiqui `persistence_2_2.xsd`)

  És possible que algun proveïdor de persistència, sense tenir la informació de les classes marcades, incorpori directament aquelles que contenen alguna marca.

- Les UP subministrades contenen les propietats següents que cal que inicialment deixem comentades:
  De JPA: `javax.persistence.schema-generation.database.action`
  Pròpia d'Hibernate: `hibernate.hbm2ddl.auto`
  Pròpia d'EclipseLink: `eclipselink.ddl-generation`

- **Comprovació** 1: Executem programa de proves amb les propietats anteriors comentades.
  Mirem la BD. No veiem cap taula creada. Motiu: NO hem informat a la UP quina acció ha de dur a terme.

- **Comprovació 2**: Activem la `property`:
  `javax.persistence.schema-generation.database.action`
  que permet opcions: `none, create, drop-and-create, drop`
  (cercar aquesta propietat en PDF de documentació oficial JPA2.2)

  Executem el programa de prova comprovant què passa segons proveu amb `create`, `drop-and-create`,...
  Mirem la taula que es crea a la BD.

  Alerta!!! Perfecte l'opció `create` en la primera ocasió si volem que creï les taules i `drop-and-create` en següents, mentre s'afina la definició de les anotacions, però en explotació: none!!!

  La majoria d'ocasions, la creació de la BD no l'efectua l'aplicació sinó que s'efectua prèviament via guions.

  Alguns proveïdors de persistència faciliten propietat alternativa amb més possibilitats:

  - Hibernate: `hibernate.hbm2ddl.auto` amb els valors:
  ✓ `validate`: validate the schema, makes no changes to the database (en explotació)
  ✓ `update`: update the schema (en desenvolupament) **=> Per treballar a classe**
  ✓ `create`: creates the schema, destroying previous data.
  ✓ `create-drop`: drop the schema at the end of the session.

  - EclipseLink: `eclipselink.ddl-generation` amb els valors:
  ✓ `create-tables`
  ✓ `create-or-extend-tables`: Similar a `update` d'Hibernate **=> Per treballar a classe**
  ✓ `drop-and-create-tables`
  ✓ `none`

- **Comprovació 3**: Desactivar la `property`
  `javax.persistence.schema-generation.database.action`
  i activar la `property` `hibernate.hbm2ddl.auto` en Hibernate
  o la `property` `eclipselink.ddl-generation` en EclipseLink.

  Executem el programa prèvia eliminació de la taula a la BD, per veure què passa segons el valor de la propietat.

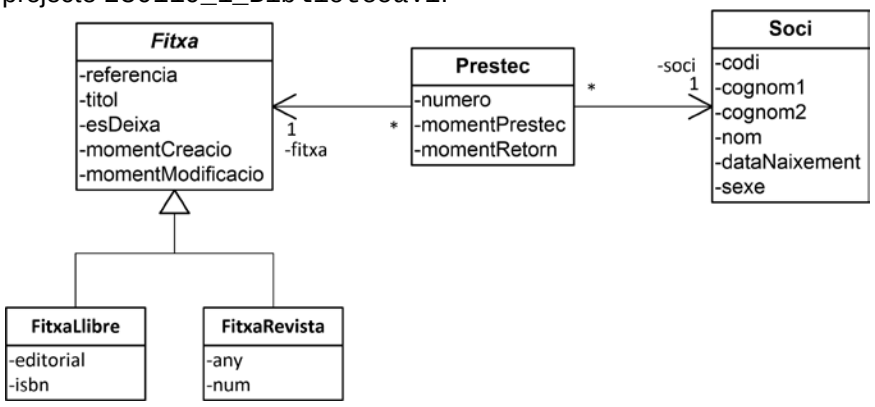
- En l'execució dels projectes anteriors, deixant que el proveïdor JPA creï la taula, observem:

Tipus de dada	Hibernate			EclipseLink		
	Oracle	PostgreSQL	MySQL	Oracle	PostgreSQL	MySQL
Integer	NUMBER(10)	Integer	int(11)	NUMBER(10)	Integer	int(11)
BigDecimal	NUMBER(19,2)	numeric(19,2)	decimal(19,2)	NUMBER(38)	numeric(38,0)	decimal(38,0)
BigInteger	NUMBER(19,2)	numeric(19,2)	decimal(19,2)	NUMBER(38)	Bigint	bigint(20)
Boolean	NUMBER(1)	Boolean	bit(1)	NUMBER(1)	Boolean	tinyint(1)
Byte	NUMBER(3)	Smallint	tinyint(4)	NUMBER(3)	Smallint	tinyint(4)
Character	CHAR(1 CHAR)	character(1)	char(1)	CHAR(1)	character(1)	char(1)
Double	FLOAT(126)	double precision	double	NUMBER(19,4)	double precision	Double
Float	FLOAT(126)	Real	Float	NUMBER(19,4)	double precision	Float
Long	NUMBER(19)	Bigint	bigint(20)	NUMBER(19)	Bigint	bigint(20)
Short	NUMBER(5)	Smallint	smallint(6)	NUMBER(5)	Smallint	smallint(6)
String	VARCHAR2(255 CHAR)	character varying(255)	varchar(255)	VARCHAR2(255)	character varying(255)	varchar(255)

Alerta: Donada la diversitat de tipus de dades que generen els proveïdors JPA en els diversos SGBD, la creació i/o modificació d'una BD en explotació s'acostuma a fer amb guions que incorporen les instruccions de creació i/o modificació adequades, amb possibles processos de preparació prèvia de dades i/o modificació posterior de dades.

Pràctica	**10/01/23**

Es vol assolir i gestionar la persistència de les classes del següent UML, de les que ja es facilita una implementació en el projecte `230110_1_BibliotecaV1`.



Pràctiques – Nomenclatura	**12/01/23**

Efectuarem l'aprenentatge a partir d'un projecte inicial `230110_1_BibliotecaV1`, que anirà evolucionant.

Desenvoluparem 2 versions en simultani:

- Primer, amb anotacions a les pròpies classes, en projecte `BibliotecaV1A`
  que utilitzarem en projectes:
  `BibliotecaV1A_Hibernate`, utilitzant Hibernate com a proveïdor JPA
  `BibliotecaV1A_EclipseLink`, utilitzant EclipseLink com a proveïdor JPA
- Segon, amb marques XML en fitxers externs, que utilitzarem en projectes:
  `BibliotecaV1X_Hibernate`, utilitzant Hibernate com a proveïdor JPA
  `BibliotecaV1X_EclipseLink`, utilitzant EclipseLink com a proveïdor JPA
  Aquests projectes sempre utilitzaran les classes de `BibliotecaV1`,

Si en algun moment es decideix fer algun canvi a les classes de `Biblioteca`, passarem a `BibliotecaV2` i així successivament.

Primeres anotacions – Primers mètodes per gestionar els objectes – Classe `Soci`	**12/01/23** **17/01/23**

- Anotacions a incorporar
  - Nivell de classe: `@Table`
  - Nivell de dades: `@Basic` | `@Column`
- **Diferència entre anotacions optional=false i nullable=false**

Segons wikibooks:

A `Basic` attribute can be `optional` if its value is allowed to be null. By default everything is assumed to be optional, except for an `Id`, which can not be optional. Optional is basically only a hint that applies to database schema generation, if the persistence provider is configured to generate the schema. It adds a `NOT NULL` constraint to the column if `false`. Some JPA providers also perform validation of the object for optional attributes, and will throw a validation error before writing to the database, but this is not required by the JPA specification. Optional is defined through the `optional` attribute of the `Basic` annotation or element.

Per altra banda, resposta d'un "gurú" de JPA a Stack Overflow:

The difference between `optional` and `nullable` is the scope at which they are evaluated. The definition of '`optional`' talks about property and field values and suggests that this feature should be evaluated within the runtime. '`nullable`' is only in reference to database columns.

If an implementation chooses to implement `optional` then those properties should be evaluated in memory by the Persistence Provider and an exception raised before SQL is sent to the database otherwise when using 'updatable=false' 'optional' violations would never be reported.

Per últim, per Hibernate:

The Hibernate JPA implementation treats both options the same way in any case, so you may as well use only one of the annotations for this purpose.

- Primers mètodes per gestionar dades persistents:
  - `em.persist(obj)`
    Marca l'objecte per fer-lo persistent (alta) i passa a ser controlat per l'Entity Manager. No passa a la BD. Si l'Entity Manager ja està controlant un objecte amb mateix ID, hauria de petar amb una `EntityExistsException`. Però la documentació JPA2.2 diu textualment:
    If the entity already exists, the `EntityExistsException` may be thrown when the persist operation is invoked, or the `EntityExistsException` or another `PersistenceException` may be thrown at flush or commit time.
    I aquí podem topar amb diferent funcionament segons proveïdor de persistència:
    - ✓ Hibernate 5.6.14: Genera excepció en enregistrar els canvis a la BD (`flush` o `commit`).
    - ✓ EclipseLink 2.7.11: Genera excepció en enregistrar els canvis a la BD (`flush` o `commit`)
  - `em.flush()`
    Enregistra els canvis a la BD (dins una transacció, sense fer `commit`; podríem fer `rollback`)
  - `em.getTransaction.begin()`
    Inicia transacció
  - `em.getTransaction.commit()`
    Tanca transacció activa validant canvis pendents a la BD. Peta si no transacció activa.
  - `em.getTransaction.rollback()`
    Tanca transacció activa sense validar canvis pendents a la BD. Peta si no transacció activa.
  - `em.getTransaction.isActive()`
    Informa si tenim transacció activa.
  - `em.find(NomClasse.class,valorDeCampId)`
    Recupera objecte persistent de la BD. Recupera `null` si no existeix.
  - `em.createQuery` (llenguatge JPQL)
    Per executar consultes similars a SQL, recuperant objectes!!!

  Què passa quan es produeix alguna `PersistenceException`? Doncs que la transacció activa, si n'hi ha, queda marcada com a transacció de "només rollback" i si s'intenta fer `commit` es produirà una excepció. És responsabilitat del programa invocar `rollback`; del contrari, les modificacions passades a la BD pendents de validar (s'hagin fet via `flush`) poden ser validades => Compte!!! Veure següent apartat sobre actuació JDBC.

  - `em.getTransaction.getRollbackOnly()`
    Informa si la transacció activa està marcada de "només rollback". Peta si no transacció activa.
  - `em.getTransaction.setRollbackOnly()`
    Marca la transacció activa de "només rollback". Peta si no transacció activa.

  En EclipseLink, els mètodes `commit`, `rollback`, `getRollbackOnly` i `setRollbackOnly` peten si no hi ha transacció activa (com marca JPA). Hibernate, però, no es queixa. Cal actuar com diu JPA i estar segurs de que hi ha transacció activa.

- **Projectes desenvolupats:**

  `230112_1_BibliotecaV1` (projecte "net" sense cap anotació i amb els requeriments a nivell de classe per a `Soci`: `Serializable`, constructor sense paràmetres, NO `final`, camps identificadors immutables)
  `230112_2_BibliotecaV1A` (projecte anterior amb les anotacions)
  Observar que necessita, per compilar, el jar de JPA 2.2 (indiferent d'Hibernate o EclipseLink)
  `230112_3_BibliotecaV1A_Hibernate` i `230112_4_BibliotecaV1A_EclipseLink`, de proves amb marcatge via anotacions
  Observar que necessiten incorporar llibreries JDBC (segons SGBD), el projecte `BibliotecaV1A` que conté les anotacions i llibreria Hibernate o EclipseLink (segons correspongui)

`230117_1_BibliotecaV1X_Hibernate` i `230117_2_BibliotecaV1A_EclipseLink`, de proves amb marcatge via anotacions

Observar que la carpeta META-INF està SEMPRE en els projectes que creen `EntityManager`.
En el cas de marcatge XML, els fitxers podrien estar fora de META-INF, però acostumen a estar allí.

⇨ El fitxer XML de marcatge de cada classe ha de validar l'esquema `orm_2_2.xsd` adjunt.

Capçalera del fitxer XML que conté el marcatge d'una classe per JPA 2.0:

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm/orm_2_0.xsd" version="2.0">
```

Capçalera del fitxer XML que conté el marcatge d'una classe per JPA 2.1:

```
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd" version="2.1">
```

Capçalera del fitxer XML de mapatge per JPA 2.2: **Capítol 12 de doc. oficial de JPA2.2 (12.3)**

```
<entity-mappings xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
http://xmlns.jcp.org/xml/ns/persistence/orm_2_2.xsd" version="2.2">
```

⇨ És fonamental que `persistence.xml` i els fitxers de mapatge siguin vàlids i si usem NetBeans per a la validació, en ocasions NetBeans informa que la validació ha finalitzat però amb un error previ. Teniu la solució en aquest vídeo explicatiu sobre la validació dels fitxers XML (persistence.xml i fitxer de mapatge)

En aquest primer projecte, estem utilitzant TOTS els SGBD i els 2 proveïdors JPA.
Això és molt costós. Ho hem fet aquí per comprovar-ne el funcionament.

A partir d'ara, practicarem amb Hibernate en Oracle, doncs hem de centrar els esforços en JPA i no pas en les particularitats que ens podem trobar en diversos SGBD i diversos fabricants ORM.

Vídeo del desenvolupament

ATENCIÓ!!! Actuació de JDBC en tancar connexió amb transacció oberta — 17/01/23

- L'API JDBC defineix:

  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.

  És a dir, si es tanca la connexió amb transaccions actives, que succeeixi commit o rollback depèn de l'actuació del connector JDBC emprat.

- Oracle, en la documentació del seu connector `ojdbc8.jar`, diu:

  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.

- SQLServer, en la documentació diu:

  Calling the close method in the middle of a transaction causes the transaction to be rolled back.

- PostgreSQL & MySQL, via comprovació (no trobat en documentació), executen `rollback` en tancar connexió

Davant el fet que no tots els SGBD actuen de la mateixa manera, és altament recomanable que abans de tancar una connexió (tancar `EntityManager` en JPA), es revisi si hi ha una transacció activa i, en cas afirmatiu, sembla lògic forçar un `rollback`:

```
if (em.getTransaction().isActive()) {
    em.getTransaction().rollback();
}
```

Exercici per dijous, 19 de gener - Continuació pràctica – Classe `Fitxa`	**19/01/23**

- Retocar la classe `Fitxa` per a que no sigui abstracta
  (oblideu-vos de `FitxaRevista` i `FitxaLlibre`)
- Fer la classe persistent, via anotacions i via marcatge XML
- Requeriments a nivell de taula:
  - Un índex per títol
  - Les fitxes s'identifiquen per la seva referència
  - Llargada màxima de títol: 60 caràcters
- Comprovar creació de taula en ambdues vies
- Fer alguns programes que juguin amb objectes `Fitxa` (per això no pot ser abstracta)