SERIALISATION ET PERSISTANCE

Serialisation et Persistance

- Concernent les données d'une application
- Ces données doivent être
  - Conservées
  - Sauvegardées
  - A jour
  - Sans erreur…
-

La sérialisation

Sérialisation

- La sérialisation permet de rendre un objet persistant :
  - Pour stocker l'objet après la fermeture de l'application
  - Pour échanger des informations entre applications
  - …
- Le framework Java gère la persistance par le biais de l'interface Serializable

Sérialisation d'un objet

- Nécessaire d'implémenter l'interface Serializable
- Les types standards sont sérialisables
- Les types complexes ne sont pas sérialisables et doivent être définis comme transient (ignoré lors de la sérialisation)

Identifiant de la classe

- Nécessité de reconstruire l'objet à partir de la classe qui l'a sérialisé :
  - Numéro de version de la classe

- Lorsque le numéro est omis, la JVM en génère un (non conseillé)
- Si la classe a été modifiée entre la sérialisation et la désérialisation, l'Exception java.io.InvalidClassException sera déclenchée

Numéro de version

- Attribut à ajouter à la classe :
  - Private
  - Static
  - Final
  - Type : long
  - Nom : serialVersionUID
  - Valeur choisie par le programmeur ou générée par l'IDE

Sérialisation personnalisée

- Une classe peut surcharger les méthodes utilisée pour sérialiser ou charger un objet afin de personnaliser la sérialisation

- Une classe peut implémenter Externalizable à la place de Serializable et redéfinir les méthodes readExternal et writeExternal (peu utilisé)

La persistance en base de données

Persistance des données

- Problème de compatibilité entre
  - le modèle relationnel de stockage des données
  - les classes d'objets dans les programmes

Persistance des données

- Particularités des représentations

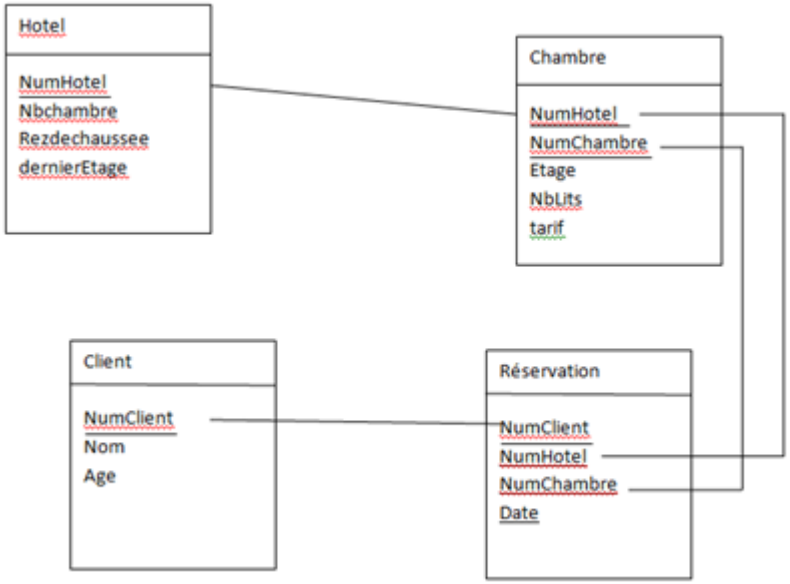
Relationnel	Objet
Identifiant pour chaque enregistrement	Héritage
	Polymorphisme

Persistance des données

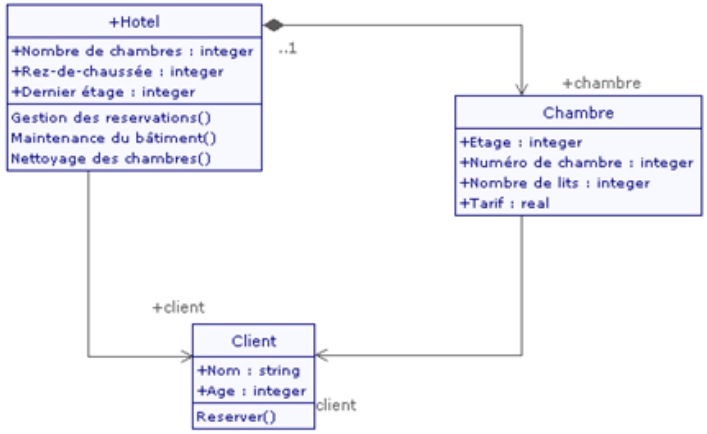
- Nécessite la mise en place d'un mapping objet relationnel
  - Description de la correspondance entre les classes des programmes et les tables de la base de données

Mapping objet-relationnel

Base de données



Classes



Accès classique aux données de la BD

- Utilisation d'un driver JDBC ou ODBC

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
  this.dbConnect = DriverManager.getConnection("jdbc:mysql:" + this.dbURL,
    this.user, this.password);
this.dbStatement = this.dbConnect.createStatement();

String requete = "delete from Formation where idForm = " + id;
ResultSet res = bd.mySQLexec(requete);
if (res != null) {
        return true;
}
```

La persistance

- Gestion des données automatique (requêtes générées à partir du mapping objet-relationnel)
  - Ajout d'un objet = insert des enregistrements
  - Modification d'un objet = update
  - Suppression d'un objet = delete

Mise en place de la sérialisation

Exemple

- On souhaite rendre les classes Heros, HerosTerre, HerosFeu, HerosMer sérialisable
  - Elle doivent implémenter l'interface Serializable
  - Le numéro de version sera 1, 10, 11, 12

Rendre un objet persistant

- Un objet peut être sérialiser dans un fichier, une base de données
- Créer un sous-répertoire serialise dans le projet pour ranger le fichier de sérialisation
- Utilisation de la classe ObjectOutputStream
  - Utilisation de la méthode writeObject(Object o)

Exemple

- Sérialiser un objet HerosTerre
  - Dans le main
  - Configurer le fichier de sérialisation
  - Créer un objet ObjectOutputStream associé au fichier
  - Créer un objet HerosTerre
  - Appeler la méthode writeObject pour sérialiser le héros
  - Fermer le flux de sortie

Charger un objet

- Pour construire un objet à partir d'un fichier ou d'une base de données

- Utilisation de la classe ObjectInputStream
  - Utilisation de la méthode readObject()

Exemple

- Récupérer l'objet héros stocké dans le fichier
  - Dans le main
  - Configurer le fichier de sérialisation
  - Créer un objet ObjectInputStream associé au fichier
  - Appeler la méthode readObject qui retourne l'objet stocké dans le fichier (ajouter un cast)
  - Afficher les informations du héros terre
  - Fermer le flux d'entrée

Exercice

- On souhaite enregistrer dans un fichier les héros créés dans l'application fenêtrée
- Il faudra loader les objets au démarrage de l'application
- Et sérialiser les objets à la fermeture de l'application

Création des fonctions de sérialisation

- Dans la classe Infos, créer 2 méthode statiques
  - serialiseHeros pour sérialiser tous les médicaments de la liste lesHeros dans un fichier heros.ser
  - chargeHeros pour charger tous les médicaments depuis le fichier heros.ser dans la liste lesHeros

Appel des fonctions de sérialisation

- Configurer l'application pour que la méthode chargeHeros soit appelée au démarrage de l'application
- Configurer l'application pour que la méthode serialiseHeros soit appelée à la fermeture de l'application

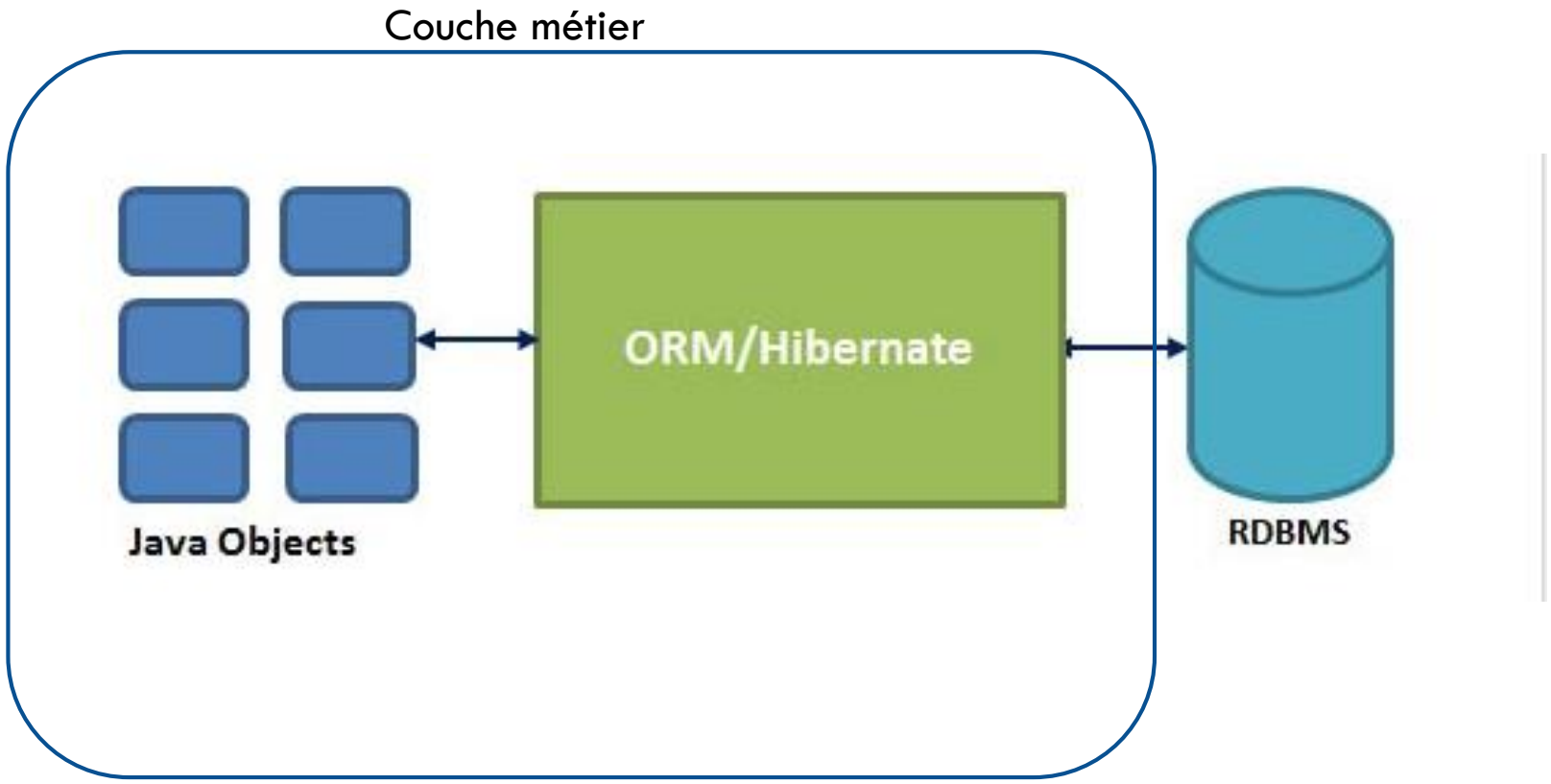
Mise en place de la persistance

Le framework Hibernate

Hibernate

- Framework open source
- Permet de gérer la persistance
- Exemples de BD supportées
  - SQL Server
  - MySQL
  - Oracle
  - PostGreSQL
  - DB2…

Mapping objet-relationnel



Mise en place de la persistance avec Hibernate

- Créer la BD
- Configurer l'accès à la BD
- Créer les classes contenant les données
- Création du fichier de correspondance
- Création du programme
  - Instanciation des classes contenant les données
  - Gestion d'objets de session
  - Gestion des transactions

Classes contenant les données

- Classes qui seront manipulées par l'application
- Doivent avoir
  - Un constructeur par défaut
  - Les propriétés private
  - Des accesseurs pour atteindre les propriétés
    - Getter
    - Setter

Fichier de correspondance

- Format XML
- Correspondance entre la classe et la table de la BD
  - la classe qui va encapsuler les données
  - l'identifiant dans la base de données
  - le mode de génération de l'identifiant
  - le mapping entre les propriétés de classe et les champs de la base de données
  - les relations

Exemple

```
01. <?xml version="1.0"?><!DOCTYPE hibernate-mapping
02.   PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
03.   "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd"><hibernate-mapping>
04.   <class name="Personnes" table="personnes">
05.     <id name="idPersonne" type="int" column="idpersonne">
06.       <generator class="native"/>
07.     </id>
08.     <property name="nomPersonne" type="string" not-null="true" />
09.     <property name="prenomPersonne" type="string" not-null="true" />
10.     <property name="datenaissPersonne" type="date">
11.       <meta attribute="field-description">date de naissance</meta>
12.     </property>
13.   </class>
14. </hibernate-mapping>
```

Balise class

Nom	Obligatoire	Rôle
Name	Oui	Nom de la classe
Table	Oui	Nom de la table dans la BD
Dynamic-update	Non	booléen qui indique de ne mettre à jour que les champs dont la valeur a été modifiée (false par défaut)
Dynamic-insert	Non	booléen qui indique de ne générer un ordre insert que pour les champs dont la valeur est non nulle (false par défaut)
mutable	non	booléen qui indique si les occurrences peuvent être mises à jour (true par défaut)

Balise id

Nom	Obligatoire	Rôle
Name	Non	nom de la propriété dans la classe
Table	Non	le type Hibernate
Column	Non	le nom du champ dans la base de données (par défaut le nom de la propriété)
Unsaved-value	Non	permet de préciser la valeur de l'identifiant pour une instance non encore enregistrée dans la base de données. Les valeurs possibles sont : any, none, null ou une valeur fournie. Null est la valeur par défaut.

Balise generator

- Obligatoire
- Précise la classe qui va assurer la génération de la valeur d'un nouvel identifiant
- Plusieurs classes disponibles dans Hibernate
  - native : utilise la meilleure solution de la BD
  - identity : identifiant auto-incrémenté
  - assigned : valeur donnée par l'application

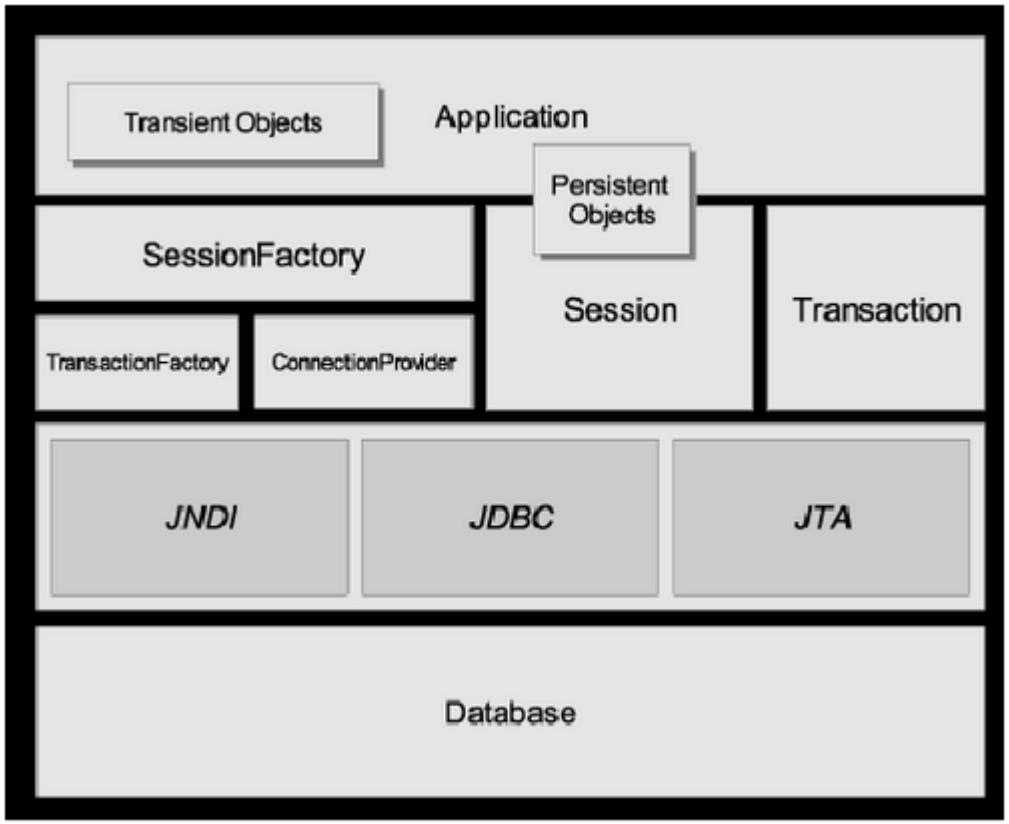
Balise property

Nom	Obligatoire	Rôle
name	oui	précise le nom de la propriété
type	non	précise le type
column	non	précise le nom du champ dans la base de données (par défaut le nom de la propriété)
update	non	précise si le champ est mis à jour lors d'une opération SQL de type update (par défaut true)
insert	non	précise si le champ est mis à jour lors d'une opération SQL de type insert (par défaut true)

Gestion de la persistance

- Créer des objets à partir de la BD (load)
- Enregistrer les données des objets dans la BD
  - Création
  - Modification
- Supprimer des données dans la BD

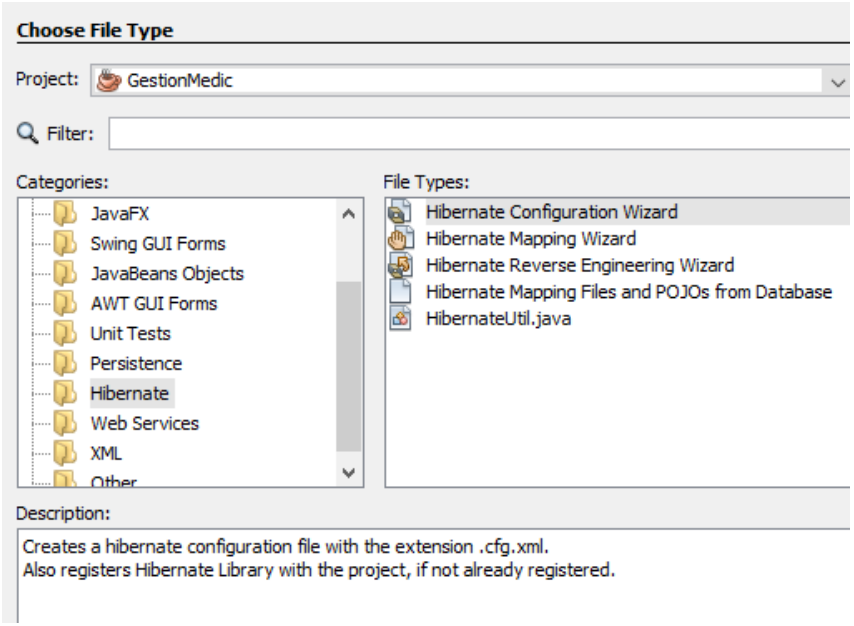
Classes d'Hibernate

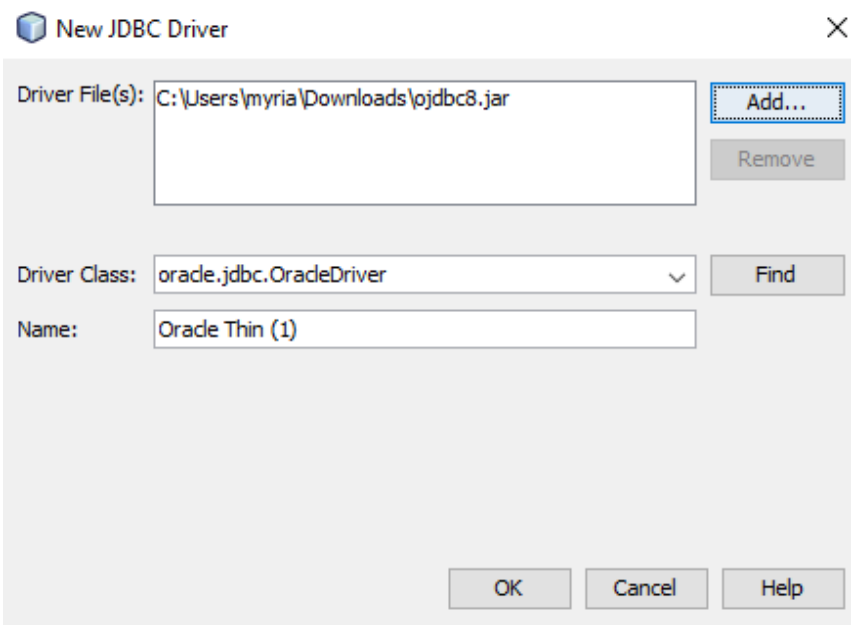


Création d'objets avec Hibernate

- Objet Configuration indiquant le fichier de mapping
- Objet Session dépendant d'un objet SessionFactory permettant de se connecter à la BD
- Objet Transaction permettant de réaliser plusieurs opérations

Création du fichier de configuration





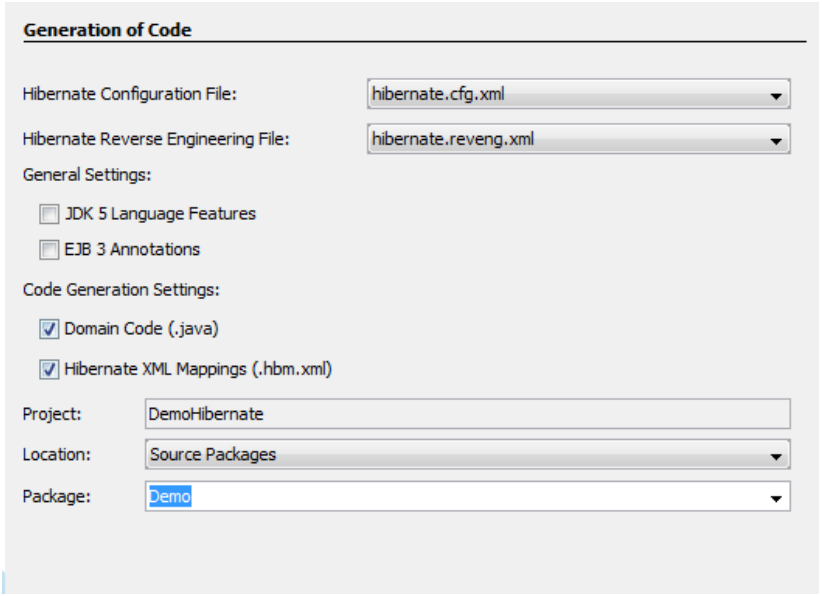
Reverse Engineering

- On créé une structure de classe à partir de la table
- Hibernate → Reverse Engineering Wizard

Création de la classe

- Hibernate Mapping Files and POJO

- Création d'une classe à partir de la table et son fichier de mapping



Fichiers créés

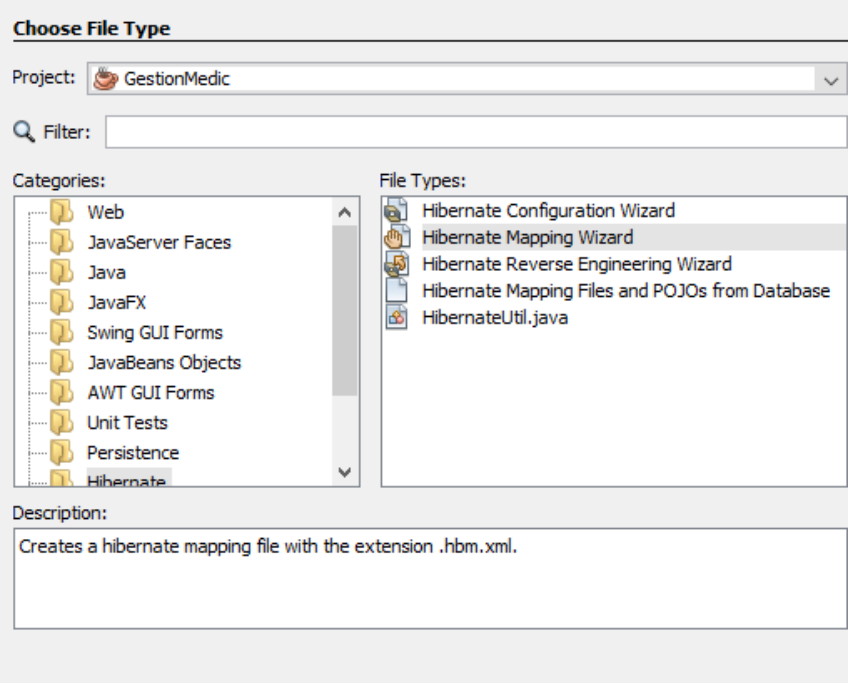
Classe

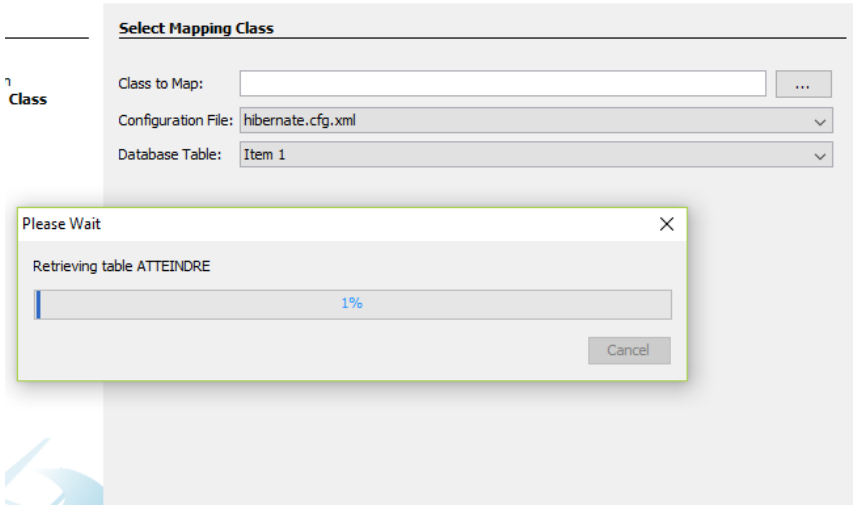
```
public class Personne  implements java.io.Serializable {


    private int id;
    private String nom;
    private String prenom;
    private String mail;

   public Personne() {
   }
```

Mapping

```
<hibernate-mapping>
    <class name="Demo.Personne" table="PERSONNE" schema="MFORT" optimistic-loc
        <id name="id" type="big_decimal">
            <column name="numero" precision="22" scale="0" />
            <generator class="assigned" />
        </id>
        <property name="nom" type="string">
            <column name="NOMPers" length="25" />
        </property>
        <property name="prenom" type="string">
            <column name="PRENOMPers" length="25" />
        </property>
        <property name="mail" type="string">
```

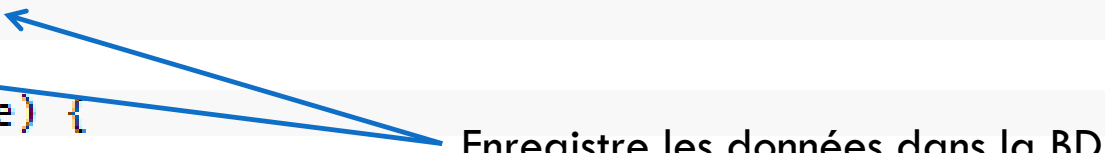
Création du fichier de mapping





Création d'un objet persistant

```
Transaction tx = null;
try {
  tx = session.beginTransaction();
  Personnes personne = new Personnes("nom3", "prenom3" );
  session.save(personne);
  session.flush() ;
  tx.commit();
} catch (Exception e) {
  if (tx != null) {
    tx.rollback();
  }
  throw e;
```



Load

- Chargement d'un enregistrement dans un objet

```
// TODO Auto-generated method stub
EntityManager manager = Persistence.createEntityManagerFactory(
                "Hibernate Tutozone").createEntityManager();

manager.getTransaction().begin();
//get inserted data
List foods = manager.createQuery("from Food").getResultList();
System.out.print(foods.toString());

manager.close();
```

Gestion des transactions

- Une interaction avec la BD est réalisée dans le cadre d'un objet de la classe Session
- Méthodes de la classe Session

Transaction beginTransaction()	Démarre une transaction
Close()	Ferme une transaction
Delete(Object object)	Supprime un objet à l'intérieur d'une transaction

Langage de requete d'Hibernate

- Langage HQL
- Similaire au SQL
- Utilise les propriétés et les objets plutôt que les tables et les champs
- Indépendant de la BD (traduction en SQL compatible avec la BD géré par le framework)

Syntaxe HQL

Clause	Description	Syntaxe	Exemple
from	précise la classe d'objets dont les occurrences doivent être retrouvées. Il est possible de définir un alias pour un objet en utilisant le mot clé alias	from object [as objectalias]	from Personne as pers (retourne toutes les occurrences de type Personne)
select	précise les propriétés à renvoyer. Doit être utilisé avec une clause from		select pers.nom from Personne as pers (retourne le nom de toutes les personnes)
where	précise une condition qui permet de filtrer les occurrences retournées. Doit être utilisé avec une clause select et/ou from	where condition	from Personne as pers where pers.nom = "Dupond" (retourne toutes les personnes dont le nom est Dupond.
order by	précise un ordre de tri sur une ou plusieurs propriétés. L'ordre par défaut est ascendant	order by propriete [asc|desc] [, propriete] ...;	select pers.nom, pers.prenom from Personne as pers order by pers.nom asc, pers.prenom desc
group by	précise un critère de regroupement pour les résultats retournés. Doit être utilisé avec une clause select et/ou from	group by propriete [, propriete] ...	