Hibernate - persistence

Sérialisation

Sauvegarde de l'état d'un objet sous forme d'octets.

ORM

Object Relational Mapping. permet de mapper des objets avec des tables.

JPA

Java Percistance API. Norme = JPA 2 : propose un modèle standard de persistance à l'aide d'Entity beans.

Initialisation de SessionsFactory

```
    private static Configuration configuration;
    private static SessionFactory sessionFactory;
    private Session s;

    try
    {
        //etape 1
        configuration = new Configuration();

        //etape 2
        sessionFactory = configuration.configure().buildSessionFactory();

        //etape 3
        s = sessionFactory.openSession();
    }
    catch(Throwable ex)
    {
        log.error("Building SessionFactory has failed.", ex);
        throw new ExceptionInInitializerError(ex);
    }
```

Entitée

Classe dont les instances peuvent être persistante. (Spécificité de JPA).

```
//import
import javax.persistance.Entity;

//use
@Entity
```

Exemple:

```
@Entity
public class Book
{
    //@Id = indicateur de clef primaire
    //@GeneratedValue = valeur auto-générée
    @Id @GeneratedValue
    private Long id;

    //Equivalent du NOT NULL en SQL
    @Column (nullable = false)
    private String title;
    private Float price;

    @Column (length = 2000)
    private String description;
    private String isbn;
    private Integer nbOfPage;
}
```

Gestion de la percistance pour hibernate

1. Création & ouverture d'une session hibernate
2. [Debut transaction] - Fortement conseillé
3. Appliquer les opérations de Session pour interagir avec l'environement de persistance
4. [valider (commit()) la transaction]
5. Synchroniser avec la base de données (flush) et fermer la session

---

Annotations

Permet de définir les propriétés

- Clé primaire : *@Id*
- Clé composite : *@EmbeddedId* ou *@IdClass* <— pas recommandé
- Génération automatique de la clé : *@GeneratedValue*
- Orderby *@OrderBy* existe, à éviter cependant
-

Association

- Cardinalité :1-1, 1-n, n-n… ex : *@OneToOne*
- 2 dirrections pour les relations : uni ou bi directionnelles.

Ex annotation OneByOne :

```
public class Person implements Serializable
{
    //...

    //Une personne est liée à un évenement
    @OneToOne
    private Event event;

    //...
}
```

Obtention session

```
session = HibernateUtil.getSessionFactory().openSession();
transaction = session.getTransaction();
transaction.begin();
```

Création personne + stockage en base

```
private void createAndStorePerson(Long i, String firstname, String
lastname, int age, Session session)
{

        Person person = new Person();

        //valeurs
        person.setId(i);
        person.setFirstname(firstname);
        person.setLastname(lastname);
        person.setAge(age);
        person.setEvent(e);

        //sauvegarde en base (avec écrasement en cas de doublon)
        session.saveOrUpdate(person);

        System.out.println(person);

}
```

Suppression

Le piège classique. On ne peut pas supprimer une relation si elle est partagée par un autre objet.

Ex: Personne a une collection d'adresses (relation 1-n). Si une de ces adresses est aussi en relation avec une autre personne il y aura une erreur. Il faut d'abbord tuer les enfants avant de s'occuper des parents.

***Eager* ou *Lazy*?**

Par défaut JPA ne récupère que les entités associées par des associations dont le but est "one" : OneToMany et ManyToOne = *lazy*. Pour les associations dont le but est "many", *eager* va tout retourner.

Hibernate vs JPA

```
//hibernate
saveOrUpdate();

//JPA
entityManager.persits(Object)
```

Objet détaché

Objet qui n'intervient pas dans le contexte de persistance. Objet inconnu du contexte. (Ex. Objet sans id)

---

percistance.xml

Fichier de configuration, qui permet des déclaration de paramètres dont on a besoin pour dialoguer avec la base de données. Il permet aussi de créer la session.

Transactions

Enclance le mode transactionnel. Assure la cohérence du modèle. Si transaction : ok commit si ko : rollback.
