Plan

1

Le modèle DAO

☞ **Inconvénients de l'architecture classique**

- Liaison forte entre le code responsable des traitements métier au code responsable du stockage des données
- il est impossible de mettre en place des tests unitaires :
  - ✓ impossible de tester le code métier de l'application sans faire intervenir le stockage (BDD, etc.) ;
  - ✓ impossible de ne tester que le code relatif au stockage des données, obligation de lancer le code métier.

2

Le modèle DAO

☞ **Inconvénients de l'architecture classique**

- Il est impossible de changer de mode de stockage.
  - ✓ Vers un autre SGBD,
  - ✓ Vers un système complètement différent d'une base de données (XML, etc.)

➔ une réécriture complète de tout le modèle, car le code métier est mêlé avec et dépendant du code assurant le stockage.

Le modèle DAO

☞ **Isoler le stockage des données**

- ~~Communiquer directement nos objets métier avec la base de données~~

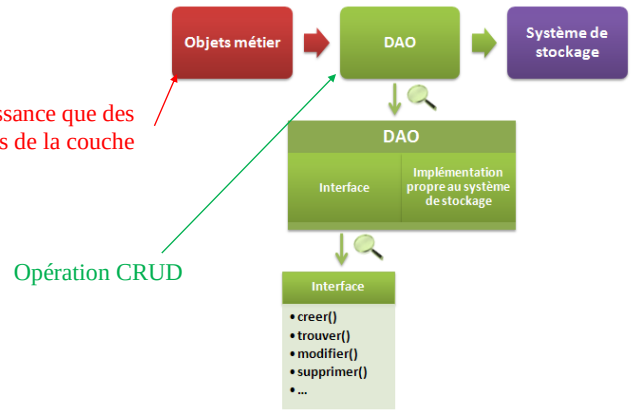
- Communiquer avec une couche DAO qui va ensuite de son côté communiquer avec le système de stockage.

Isolement pur et simple du code
responsable du stockage des données.

Le modèle DAO

☞ **Isoler le stockage des données**

La couche métier n'a connaissance que des interfaces décrivant les objets de la couche données.



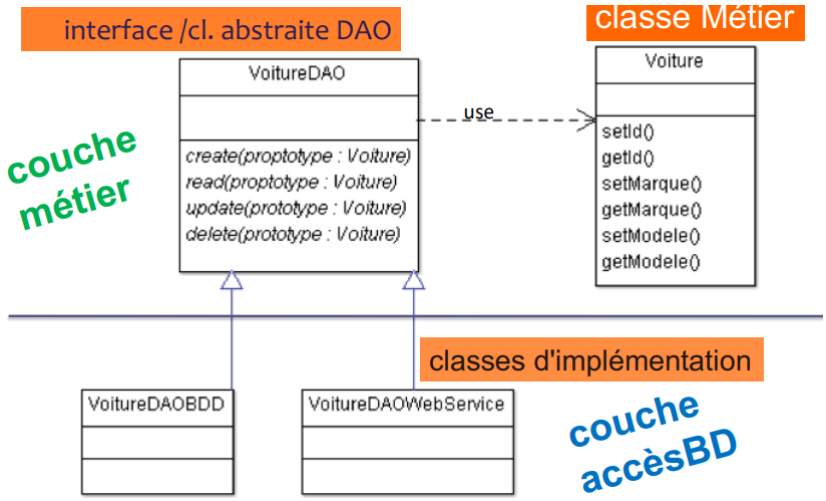
Opération CRUD

Le modèle DAO

☞ **Composants à implémenter**

- Les JavaBeans
- Création des exceptions DAO
- Création d'une Factory: instanciation des différents DAO de l'application:
  - ✓ lire les informations de configuration depuis le fichier properties ;
  - ✓ charger le driver JDBC du SGBD utilisé ;
  - ✓ fournir une connexion à la base de données.
  - ✓ Renvoi les composants DAO
- Création des interfaces DAO et leur implémentation
- Création de la servlet

Le modèle DAO



7

Le modèle DAO

☞ **L'interface d'objet DAO**

- Expose les **fonctionnalités CRUD** indépendantes de l'implémentation
- Aucune des méthodes spécifiées ne doit contenir de requête SQL en paramètre
- Doit proposer une gestion personnalisée des exceptions
  - les exceptions standards sont attrapées et redirigées vers une ou des exceptions particulières
  - gérées par une ou plusieurs **classes d'exceptions indépendantes du support** de persistance

8

Le modèle DAO

☞ **L'implémentation d'objet DAO**

- Un seul objet DAO créé par classe métier
  - il y a donc autant d'objets DAO que de classes métier
- L'application ne manipule **que** les objets métier
  - seuls les objets métier utilisent les services de **leur DAO**
- Création des objets DAO
  - les objets DAO dépendent de la source de données
  - il existe donc une famille d'objets DAO par type de source de données

Le modèle DAO

☞ **Exemple**

Le Bean (l'entité)

```
public class Produit implements Serializable {
private String nom;
private int qte;

public String getNom() {
return nom;
}
public void setNom(String nom) {
this.nom = nom;
}
public int getQte() {
return qte;
}
public void setQte(int qte) {
this.qte = qte;
}


}
```

Le modèle DAO

☞ **Exemple**

La DAO Factory (**DaoFactory.java**) permet d'initialiser le DAO en chargeant les drivers nécessaires et se connecte à la base de données

La factory DAO

```
public class DaoFactory {
    private String url;
    private String username;
    private String password;

    DaoFactory(String url, String username, String password) {
        this.url = url;
        this.username = username;
        this.password = password;
    }
```


Le modèle DAO

☞ **Exemple** La factory DAO

```
public static DaoFactory getInstance() {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch (ClassNotFoundException e) {

    }

    DaoFactory instance = new DaoFactory(
            "jdbc:mysql://localhost:3306/javaee", "root", "");
    return instance;
}
```

Le modèle DAO

☞ **Exemple**

La factory DAO

```
public Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url, username, password);
    }

    // Récupération du Dao
    public ProduitDao getUtilisateurDao() {
        return new ProduitDaoImpl(this);
    }
}
```


Le modèle DAO

☞ **Exemple**

L'interface DAO

```
public interface ProduitDao {
    void creer( Produit p ) ;
    Produit trouver( String nom ) ;
    List<Produit> lister();

}
```

Le modèle DAO

☞ Exemple

L'implémentation de la couche DAO

```
public class ProduitDaoImpl implements ProduitDao {
 private DaoFactory          daoFactory;
public ProduitDaoImpl(DaoFactory daoFactory) {
this.daoFactory = daoFactory;
}

@Override
public void creer(Produit prod) {
        Connection connexion = null;
        PreparedStatement preparedStatement = null;

        try {
            connexion = daoFactory.getConnection();
            preparedStatement = connexion.prepareStatement("INSERT INTO
Produit(nom, qte) VALUES(?, ?);");
            preparedStatement.setString(1, prod.getNom());
            preparedStatement.setLong(2, prod.getQte());

            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }}
```


Le modèle DAO

☞ Exemple

```
public List<Produit> lister() {
   List<Produit> ListProd = new ArrayList<Produit>();
        Connection connexion = null;
        Statement statement = null;
        ResultSet resultat = null;

        try {
            connexion = daoFactory.getConnection();
            statement = connexion.createStatement();
            resultat = statement.executeQuery("SELECT * FROM Produit");

            while (resultat.next()) {
                String nom = resultat.getString("nom");
                int qte = resultat.getInt("qte");
                Produit prod = new Produit();
                prod.setNom(nom);
                prod.setQte(qte);
                ListProd.add(prod);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ListProd;    }
```

Le modèle DAO

☞ **Exemple d'implémentation**