IUT de LYON, ASPE semestre 2
2018/2019

Java Avancé – TP5

Sérialisation, JDBC, DAO

Durée : 5h

Il s'agit maintenant d'assurer la persistance des objets de **notre interface Gestion de Produits** par 2 bases de données, Oracle et MySQL.

Partie 1 – Création de la connexion à la BD Oracle

1- Vous allez créer une BD Oracle pour les articles.

1-1. *Construire le schéma de la base* à l'aide de SQLDeveloppper : combien de champs sont nécessaires ? Comme pour beaucoup d'objets de la vie réelle, il n'y a pas d'identifiant. Vous allez les gérer de façon automatique, à chaque insertion en BD. Préciser donc une clef primaire (entier en auto incrément), et définir quels champs peuvent être `null` ou pas. Pour le libellé et les catégories, utiliser des `varchar(20)`, et pour les valeurs numériques `number` ou `float`.

1-2. Alimenter la base avec un jeu d'essai avec SQLDeveloppper.

2- Mettre en place et tester la connexion à la BD.

Nous allons créer une classe de connexion à l'aide d'un DataSource et d'un fichier de propriétés.
Q1 ➔ Pourquoi vaut-il mieux procéder ainsi plutôt que d'écrire les éléments de connexion directement dans le code (comme c'est le cas avec `DriverManager`) ?

2-1. Récupérer le driver JDBC Oracle : **`ojdbc8.jar*`** et l'ajouter à votre bibliothèque de projet. Ensuite importer la classe : `oracle.jdbc.pool.OracleDataSource;`

> *Pour ceux qui le souhaitent, vous pouvez intégrer un driver JDBC depuis MAVEN : http://www.mkyong.com/maven/how-to-add-oracle-jdbc-driver-in-your-maven-local-repository/*

*Chercher « Oracle JDBC Drivers » sur votre moteur préféré. Il faut créer un compte Oracle, sinon vous le trouverez ici : https://box.univ-lyon1.fr/p/3cceb7

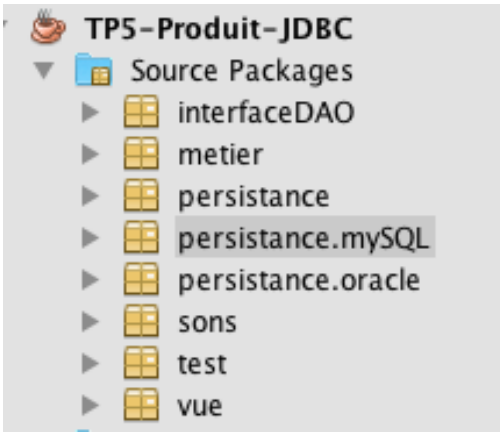
2-2. Créer un package **`persistance`** avec sous-package **`oracle`**, dans lequel vous créez une classe : **`MonOracleDataSource`** et un fichier de propriétés **`connOracle`**

`.properties` (new… properties file) qui contiendra les informations requises pour accéder à la BD Oracle de l'IUT :

```
# FICHIER de propriétés décrivant la connexion à une BD au serveur Oracle de l'IUT
# jamais de guillemet
port=1521
service=orcl
serveur=iutdoua-oracle.univ-lyon1.fr
pilote=thin
user=votre_login_oracle
pwd=votre_pwd_oracle
```

Q2 ➔ Quelle limite voyez-vous à ce mécanisme de connexion ? Nous y reviendrons en Partie 3.

A terme votre projet comportera les packages suivants :



Les packages imbriqués sont représentés séparés par un '.', mais ils sont bien imbriqués

2-3. La classe **`MonOracleDataSource`** hérite de la classe **`OracleDataSource`** et est un **`singleton`** : on aura besoin d'une seule instance lors de l'exécution. Le constructeur est donc privé et ne s'exécute qu'avec l'appel d'une méthode publique, **`getOracleDataSource()`**, qui va vérifier s'il n'existe pas déjà une instance de la classe (statique). Créer donc un champ de classe privé du type attendu, par ex. :
`MonOracleDataSource ods ;`

Écrire ensuite la méthode qui permet de récupérer l'instance unique, la source de données :

```
public static MonOracleDataSource getOracleDataSource() {

    if (ods == null) {

        FileInputStream fichier = null;
        Properties props = new Properties();

        try {
            fichier = new FileInputStream("src/persistance/oracle/connOracle.properties");
```

```
    } catch (FileNotFoundException ex1) {
       System.out.println("Fichier de proprietes Oracle non trouvé");
    }
    try {
       props.load(fichier);
    } catch (IOException ex) {
       System.out.println("Erreur lors du chargement du fichier de proprietes Oracle");

    } finally {
       try {
          fichier.close();
       } catch (IOException ex) {
          System.out.print("Problème d'entree/sortie" + ex.getMessage());
       }
    }
    try {
       ods = new MonOracleDataSource();          // on peut instancier à vide
    } catch (SQLException ex) {     // … OracleDataSource()  peut lever cette exception
       System.out.println("*** Erreur de CONNEXION ORACLE ...");
     }

    // on construit la source de données :
    ods.setDriverType(props.getProperty("pilote"));
    ods.setPortNumber( new Integer( props.getProperty("port") ) );
    ods.setServiceName( props.getProperty("service") );
    ods.setServerName( props.getProperty("Serveur") );
    ods.setUser( props.getProperty("user") );
    ods.setPassword( props.getProperty("pwd") );
  }
  // sinon, une instance de source de données existe deja, on la renvoie :
  return ods;
```

2-4. Si ce n'est déjà fait au TP4, créer une **classe Métier `Article`** avec tous les champs nécessaires du type de ceux choisis en BD (dont un ID), les `get/set`, et surcharger `toString()` pour que soient affichées les données de l'article.

2-5. Ensuite dans un package **`interfaceDAO`**, créer la classe **`IArticleDAO`**, l'interface décrivant ce qu'on va pouvoir faire avec les produits :

```
public interface IArticleDAO {

   public List<Article> getLesArticles();
   public int           setLesArticles(List<Article> lesArticles);
   public void          insertArticle(Article a);
   public boolean       supprArticle(int refArticle);
```

On imagine pouvoir changer dynamiquement de source de données ou de connexion ; on ajoute donc les 2 méthodes :

```
public void     setDataSource(DataSource ds);
public void     setConnection(Connection connexionBD);
```

2-6. Nous allons créer une classe *d'implémentation* de l'interface **`IArticleDAO`** avec le langage SQL, qu'on utilisera pour Oracle et MySQL. Cette classe sera nommée **`ArticleDAO`**.

Q3 ➔ Dans quel package allez-vous placer cette classe ?

La classe **`ArticleDAO`** possède 2 champs statiques, qui permettront de récupérer dynamiquement la source de données et/ou de connexion à la BD :

```
private static DataSource ds;
private static Connection connexionBD;
```

Elle possède un seul constructeur, qui a besoin d'un `Datasource` et d'une `Connection` (du coup, donnez le constructeur par défaut, vide).

Implémentez dans un premier temps uniquement la méthode **`public List<Article> getLesArticles()`** qui va lire les articles triés par catégorie, et les placer par exemple dans un `ArrayList<Article>`.

Pour cela :

- créer les objets nécessaires pour effectuer une transaction de type `SELECT`
- écrire la requête `SELECT` qui lit tous les articles de votre BD, triés par catégorie
- parcourir le `ResultSet` et instancier un article pour chaque ligne, que vous ajoutez à la liste d'articles
- fermer les objets de la transaction
- retourner la liste créée

2-7. Testez votre connexion.

Pour cela, placez-vous dans le package **`test`**. Créer une fonction `main` dans une classe **`TestConnexion`**, qui possède 3 champs statiques :

```
private static IArticleDAO articleDAO;
private static DataSource dataSourceDAO;
private static Connection connexionBD;
```

Dans le `main()`, créer la source de données (ici Oracle) et la connexion :

```
dataSourceDAO  = MonOracleDataSource.getOracleDataSource();
connexionBD    = dataSourceDAO.getConnection();
```

Puis créer une instance de DAO Article avec la classe précédente :

```
articleDAO = new ArticleDAO( dataSourceDAO, connexionBD );
```

Pour bien voir les erreurs s'il y en a, exploitez tous les messages d'erreur de `SQLException`. Une idée est de créer une méthode chargée de traiter toutes les erreurs possibles :

```
public static void printSQLException (SQLException ex) {

    for (Throwable e : ex) {
         if (e instanceof SQLException) {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " +
                                        ((SQLException) e).getSQLState());
                System.err.println("Error Code: " +
                                        ((SQLException)e).getErrorCode());
                System.err.println("Message: " + e.getMessage());
                Throwable t = ex.getCause();
                while (t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
         }
    }
} // de méth printSQLException
```

Partie 2 – Relier la BD Oracle à l'interface

1- Quelques modifications de l'interface sont nécessaires : il faut **ajouter une colonne à la table pour l'identifiant** (attention au décalage des indices avec le code existant relatif au modèle…).

2- De nouveaux champs de classe sont nécessaires, on aura notamment :

```
private static IArticleDAO articleDAO;
private static DataSource dataSourceDAO;
private static Connection connexionBD;
```

Ainsi que la liste des articles :

```
private List<Article> lesArticles;
```

Dans le constructeur, ajoutez les éléments nécessaires pour faire le lien avec la BD (cf code de `TestConnexion`).

3- Nous avons mis en place l'export Excel en fichier texte, nous allons maintenant **gérer la sauvegarde / le chargement de la BD dans la table**. Ajouter d'abord les *items* de menu dans `Fichier` : charger / sauvegarder, avec des raccourcis clavier (short-cut keys).

4- On va **modifier le code du bouton Supprimer** pour que la suppression s'effectue non seulement sur l'interface, mais aussi en BD. Créer une méthode `supprimer(Article a)` dans votre classe d'accès à Oracle, qui cherche si l'article se trouve en BD et le supprime si c'est le cas.

*Q4- Ici on choisit d'effectuer immédiatement les actions de l'utilisateur en BD, c'est une application en mode '**synchrone**'. Il existe un autre mode : quel est-il, et quels sont les avantages / inconvénients de chaque mode ?*

Partie 3 – Poursuivre le code de la classe `ArticleDAO`

1- Écrire le code des différentes méthodes de l'interface **`IArticleDAO`**, avec les bonnes requêtes SQL.
2- Ajouter tous les éléments pour permettre une **connexion à une BD MySQL** (locale ou du serveur IUT). SI vous n'avez pas le temps d'implémenter le code, mentionnez à minima les éléments qu'il faudrait développer.

Facultatif

1- **Modifier le code du menu Quitter** pour qu'on enregistre avant de quitter au cas où une modification a été effectuée depuis la dernière sauvegarde.
2- Ajouter une JDialog `JDAuthentificationBD` fenêtre pour la **saisie du nom de l'utilisateur et du mot de passe** : dans quel package la placer ?

Dans la JDialog, il faut importer la classe : `java.net.PasswordAuthentication`.
➔ Pourquoi est-ce mieux de procéder ainsi ?