Documentation

Notion JDBC :

- Connexion à une base de données : Pour se connecter à une base de données en utilisant JDBC il est nécessaire d'installer un driver qui est disponible sur internet. Ensuite nous chargerons ce driver et nous utiliserons l'interface connexion présente dans le package java.sql et nous établirons la connexion avec la base de données en renseignant l'adresse du serveur sur lequel est hébergé la base de données et l'identifiant et le mot de passe pour y accéder, le tout entouré d'un bloc try/catch afin de récupérer les erreurs.

Voici comment nous avons procéder dans notre code :

```java
public class DataBase {
    private Connection connexion;

    public Connection getConnection() {
        return connexion;
    }
    /*
     * initialise une connexion en tant que admin du serveur mysql et creé la bdd
     * signaturedesign si elle n'existe pas
     *
     * @param nom  = le nom de l'admin
     * @param pass = le mot de passe de l'admin
     */
    public boolean DataBase(String nom, String pass) {
        try {
            connexion = DriverManager.getConnection("jdbc:mysql://localhost:3306/", nom, pass);
            Statement requete = connexion.createStatement();
            if (!bddExiste()) {
                requete.executeUpdate("CREATE DATABASE IF NOT EXISTS SignatureDesign;");
                requete.execute("Use SignatureDesign;");
                createSchemaBdd();
            } else
                requete.execute("Use SignatureDesign;");
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
```

- PreparedStatements : Avant de parler de l'interface preparedStatement, parlons des Statement (déclaration en français), un statement nous permet de déclarer une requête SQL que nous allons ensuite stocker dans un Resultset pour pouvoir manipuler son résultat.
  Lorsque l'on crée un PreparedStatement l'objectif est d'optimiser les temps de chargement en réutilisant cette même requête déjà chargée mais en ne modifiant uniquement les attributs que l'on associe à cette requête. Pour cela on créer un PreparedStatement auquel on associe

une requête SQL via la connexion à la base de données. On remplace les attributs par des points d'interrogation qui prendront la valeur des paramètres indiqués selon leur index. Enfin on exécute la requête , le tout entouré d'un bloc try/catch.

Voici un exemple dans notre classe article pour l'ajout d'article :

```
public void modifierArticle(String nom, double prix, String marque, String cat) {
    try {
        PreparedStatement ps = App.db.getConnection().prepareStatement("UPDATE ARTICLES SET Nom_Article = ?,"
                + " Prix_Article = ?, Marque_Article = ? , Categorie_Article = ? WHERE Id_Article = ?");
        ps.setString(1,nom);
        ps.setDouble(2,prix);
        ps.setString(3,marque);
        ps.setString(4,cat);
        ps.setInt(5,this.getId());
        ps.executeUpdate();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

- Stored Procedures : Les Stored Procedures reposent sur le même principe que les PreparedStatement mais sont encore plus efficaces puisqu'elles stocks directement dans la base de données des requêtes SQL auquel on pourra insérer des paramètres via l'interface callableStatement.

  Voici un exemple :

```
  public void createAjoutArticleProc() {
      String drop = "DROP PROCEDURE IF EXISTS AJOUT_ARTICLE";
      String createProcedure = " create procedure AJOUT_ARTICLE(IN vnom_article varchar(45), IN vprix_article double,"
              + " IN vmarque_article varchar(45)," + " IN vcategorie_article varchar(45) " + ")" + "begin "
              + "INSERT INTO ARTICLES ( nom_article ,  prix_article , marque_article , categorie_article ) "
              + "VALUES ( vnom_article ,  vprix_article , vmarque_article , vcategorie_article)" + "; " + "end  ";
      // createProcedure
      try (Statement stmt = connexion.createStatement()) {
          stmt.execute(drop);
          stmt.executeUpdate(createProcedure);
      } catch (SQLException e) {
          e.printStackTrace();
      }
  }
```

Notion JavaFX : Le point d'entrée pour les applications JavaFX est la classe Application. JavaFX exécute dans l'ordre, chaque fois qu'une application est lancée, les actions suivantes :

- Construit une instance de la classe Application spécifiée
- Appelle la méthode init ()
- Appelle la méthode start (javafx.stage.Stage)
- Attend l'achèvement de l'application,
- Appelle la méthode stop ()

Notez que la méthode de démarrage (start) est abstraite et doit être surchargée. Les méthodes d'initialisation (init) et d'arrêt (stop) ont des implémentations concrètes qui ne font rien.

Un layout ou gestionnaire de mise en page est un nœud graphique qui hérite de la classe javafx.scene.layout.Pane. Il s'agit d'une entité qui contient d'autres nœuds et qui est chargée de les déplacer, de les disposer, voire de les redimensionner de manière à changer la présentation de cet ensemble de nœuds et à les rendre utilisables dans une interface graphique.

Pour la mise en page et la création des différents composants de notre application nous avons utilisé SceneBuilder qui est téléchargeable depuis internet sur le site Gluon. Après la création d'une page Fxml, on ouvre cette page avec SceneBuilder qui nous permet d'y ajouter des composants via de simples glisser/déposer et qui va éditer le code associé à notre place, nous avons également associer un fichier css à cet page afin d'y ajouter quelques éléments graphiques . Il ne reste plus qu'a associer les différents composants aux méthodes créée dans notre programme.