Rapport Projet Java - Partie 2

1. Introduction

Cette partie du rapport présente la structure technique du projet Java, son organisation en packages, la mise en place de deux bases de données distinctes (principal et test), la configuration du projet Maven avec Java 21 et JavaFX, ainsi que l'intégration de tests automatisés via JUnit. L'objectif est de garantir la robustesse, la maintenabilité et la testabilité de l'application dans un contexte réaliste.

2. Organisation du projet

Le projet est structuré en deux packages principaux :

➔ **src/main/java/com.projet.projet_java**

Contient toutes les **classes de l'application principale** :

- Modèles métiers (**Dechet**, **Menage**, **Corbeille**, etc.)
- Utilitaires (**DatabaseConnection**, **SQLExecutor**, etc.)
- Visualisation des bases de données (**DatabaseViewerTri_selectif.java** & **DatabaseViewerTri_selectifTest.java**)
- Fichiers d'interface JavaFX (**HelloApplication**, **HelloController**)
- Fichier d'entrée principal : **Main.java** (vérifie la connexion à la base principale **tri_selectif.sql**)

À noter :
HelloApplication et **HelloController** sont des fichiers générés par défaut lors de la création du projet JavaFX dans IntelliJ. Ils ont été laissés en l'état pour conserver la structure initiale.

➔ **src/test/java/com.projet.projet_java**

- Une classe de test pour chaque entité (**DechetsTest**, **MenageTest**, etc.)
- Un lanceur global : **MainTest.java**

  Exécute tous les tests et affiche le contenu des tables de la base **tri_selectifTest**

Cette séparation permet d'assurer une isolation claire entre le code principal et les scénarios de test.

3. Bases de données : Principal vs Test

Le projet utilise **deux bases MySQL** distinctes pour éviter les conflits :

Base	Usage	Script associé
`tri_selectif`	Base principale	`tri_selectif.sql`
`tri_selectifTest`	Base de test	`tri_selectifTest.sql`

La classe DatabaseConfig.java permet de **basculer dynamiquement** entre les deux environnements grâce à un mode **TEST** ou **PRINCIPAL.**

➔ Lors de l'exécution des tests, la base de test est automatiquement sélectionnée via :

```java
DatabaseConfig.setUseTestDatabase(true);
```

➔ **DatabaseConfig.setTestDatabase(true) :** pour accéder à la base **tri_selectifTest.sql** (la base de données Test)

➔ **DatabaseConfig.setTestDatabase(false) :** pour accéder à la base **tri_selectif.sql** (la base de données Principal)

4. Visualisation des données

Deux fichiers Java permettent de **consulter rapidement le contenu des bases** sans passer par un outil externe :

Fichier	Base ciblée
`DatabaseViewerTri_selectif.java`	Affiche la base `tri_selectif`
`DatabaseViewerTri_selectifTest.java`	Affiche la base `tri_selectifTest`

Chaque table est parcourue et ses données sont affichées proprement dans la console.

```
Poubelle insérée !
BonAchat inséré !
Dechet inséré avec succès dans la base de test !
CentreDeTri inséré !
Commerce inséré !
Corbeille insérée !
HistoriqueDepot inséré !
Menage inséré avec succès !
Categorie insérée !
Contrat inséré !

Affichage du contenu des tables dans tri_selectifTest :
Connexion à la base MySQL [tri_selectifTest] réussie !

Table : Dechet
➤ idDechet=1  typeDechet=Verre  poids=2.5

Table : Menage
➤ idMenage=1  nom=Testeur  adresse=42 rue du Test  email=testeur+2a9367c9@example.com  motDePasse=motdepasse  badgeAccess=BADGE123  pointsFidelity=50
```

5. Tests JUnit & validation automatique

Les tests JUnit vérifient le bon fonctionnement des opérations suivantes :

- Insertion dans la base
- Accesseurs et comportements métier
- Relations entre objets (clé étrangère, poids total, etc.)

Lancement automatisé

MainTest.java :

- Exécute **tous les tests présents dans le projet** via **ConsoleLauncher**
- Affiche les résultats dans le terminal
- Montre ensuite le contenu réel des tables dans **tri_selectifTest**

Cela permet de **valider le comportement de bout en bout** du système, sans toucher à la base principale.

6. Compilation & génération du .jar

Le fichier **pom.xml** configure Maven pour :

- Compiler avec Java 21
- Intégrer toutes les ressources (SQL, FXML, etc.)
- Générer un **.jar** exécutable propre avec les **.class et** les fichiers **.java**

Main.java (fichier de démarrage principal)

- Permet de **vérifier si la connexion à la base de production fonctionne**
- Peut évoluer pour devenir le point d'entrée réel de l'application JavaFX

MainTest.java (dans /test)

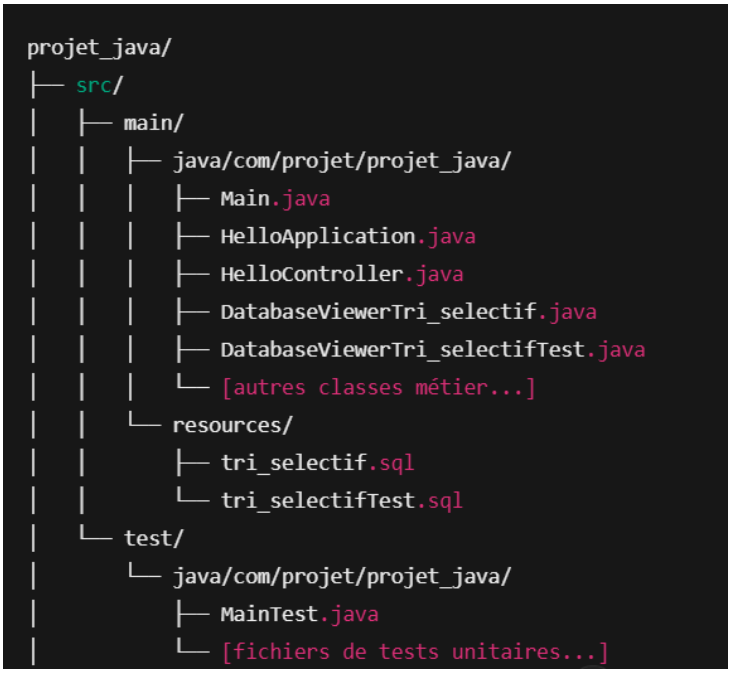
- Exécute tous les tests unitaires
- Ne doit **pas** être confondu avec **Main.java**

Main.java vs MainTest.java

- Main.java est le **point d'entrée de l'application** (principal)
- MainTest.java est le **lanceur des tests automatisés**

Cela permet de produire un **.jar** propre pour l'exécutable principal, tout en isolant les tests.

7. Arborescence finale



8. Conclusion

Cette base technique solide permet une évolution fluide du projet Java :

- Une organisation claire en production/test
- Une base de tests automatisés isolée
- Une possibilité de visualiser les données à tout moment
- Une compilation en **.jar** fonctionnelle avec JavaFX & Java 21
- Une future extensibilité (GUI JavaFX, déploiement, export des données…)

Ce projet montre ainsi la capacité à développer un système complet, modulaire, fiable et extensible, avec de bonnes pratiques professionnelles en Java

Jolan Elyakouti
Bissem Meddour
Julien Mégnoux
Sofiane Ait El Hadj
Clement Rimbeuf