

Rapport Projet Final  
Programmation Orientée objet et Java

# Projet Gestion de Tri sélectif



Réalisé par:

- MARIAU Julia
- GARCIA Jeanne
- SMITH William
- GROLLEAU Florian
- JAFFUEL Romain

Encadré par:

- Dr. VÉRIN.Renaud

# Tables des Matières

<b>Introduction.....</b>	<b>3</b>
<b>Partie 1 – le diagramme de classe UML.....</b>	<b>3</b>
Introduction et Enjeux.....	3
Notre UML.....	3
Points de tension.....	4
Premières réflexions.....	4
Distinction entre corbeille et dépôt.....	4
Les entités enumérées.....	5
Accessibilité des attributs.....	5
Les contraintes.....	5
<b>Partie 2– Java Texte.....</b>	<b>5</b>
Introduction et Enjeux.....	5
Coder les classes en Java.....	6
Quelques exemples de nos classes Java.....	10
Notre classe qui nous permet d'avoir un CSV.....	11
Points de tensions.....	11
Passage de l'UML à notre schéma d'IHM et l'implémentation des classes en Java.....	11
Problème avec MainTest.....	12
Transmission des classes.....	12
Difficultés liées au code :.....	12
Avis.....	13
<b>Partie 3 – Conception de l'IHM.....</b>	<b>14</b>
Introduction et enjeux:.....	14
Méthode.....	14
Avant la conception de l'IHM.....	14
Pendant la conception de l'IHM.....	15
Notre IHM.....	17
Points de tension.....	19
<b>Avis Global.....</b>	<b>20</b>

## Introduction

Nous avons pour objectif dans ce projet de modéliser un système de fidélité, initié par un centre de tri et visant à réduire les coûts de traitement des déchets, tout en incitant les ménages à trier correctement leurs déchets. Les points de fidélité sont accordés en fonction de la quantité de déchets triés, et peuvent être retirés si le tri n'a pas été réalisé sérieusement. Ces points de fidélité sont échangeables contre des bons d'achats, ou encore des réductions dans les commerces locaux en partenariat avec le centre de tri. Mais ce système de fidélité est plus qu'un moyen de réduire les coûts de traitement des déchets : il permet aussi de promouvoir une gestion des déchets plus écologique. Nous avons donc pour mission l'implémentation de ce système de fidélité.

Dans un premier temps, nous proposerons un diagramme de classe UML qui permettrait de gérer l'ensemble du cycle de tri sélectif.

## Partie 1 – le diagramme de classe UML

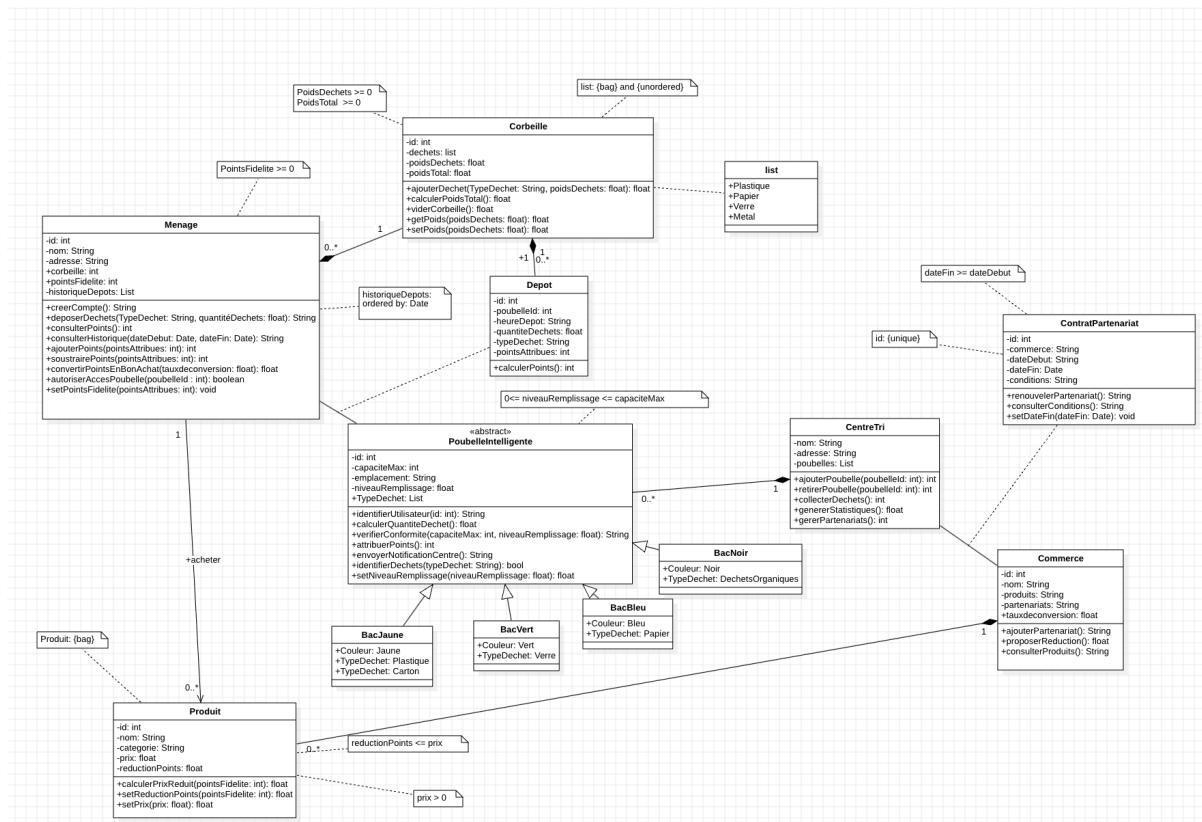
### Introduction et Enjeux

Nous avons commencé par réaliser un diagramme UML (Unified Modeling Language) dans le cadre de notre projet Java car il permet de représenter visuellement les différentes composantes de notre centre de tri, ses relations ainsi que leur fonctionnement avant de passer à l'implémentation réelle du code. Pour commencer, nous avons décidé de lire chacun de notre côté le cahier des charges de ce projet. Puis nous avons divisé le groupe de travail en 2, afin que chaque groupe puisse proposer un diagramme de classe UML. Ce choix a été réalisé car nous pensons que la confrontation de deux versions nous permettrait de tirer de chacune de ces versions les meilleures idées.

Nous avons donc ensuite organisé une réunion afin d'échanger sur nos diagrammes de classe et nous en avons tiré un diagramme de classe UML final.

## Notre UML

Capture d'écran de notre UML représentant notre centre de tri



## Points de tension

### Premières réflexions

En se basant sur le cahier des charges, nous avons répertorié l'ensemble des entités nécessaires comme le Centre de Tri, la Poubelle Intelligente ou encore les Ménages. Puis nous avons ajouté leurs attributs et leurs opérations soit depuis le cahier des charges soit par déduction. En ce qui concerne les liaisons, certaines nécessitent des associations plus complexes comme des dépendances. Ensuite la question s'est posée si certaines entités étaient des compositions ou des agrégations d'autres. Comme les déchets sont triés par la Poubelle Intelligente pour les placer dans chaque poubelle spécifique (verte, grise,...), il faut représenter ces nouvelles entités sous

forme d'héritages de l'entité Poubelle Intelligente. L'un de nos deux groupes avait représenté le stockage des dépôts effectués dans une entité historique Depot. Nous avons ensuite décidé de la représenter comme une opération en List dans l'entité compte pour avoir un UML moins lourd.

### **Distinction entre corbeille et dépôt**

Pendant notre réunion, nous avons eu plusieurs débats sur les différentes solutions qui pouvaient être proposées sur les différentes parties du cahier des charges. Dans un premier temps, nous avons débattu de l'utilité de la distinction entre corbeille et dépôt. En effet, l'un des groupes avait représenté l'entité Corbeille tandis que l'autre non, la question s'est alors posée de savoir si la Corbeille devait être représentée par l'entité Depot ou si c'était une entité à part entière. Nous avons finalement décidé de la représenter de manière indépendante car la corbeille existe en dehors de l'action de faire un dépôt.

### **Les entités enumerate**

Nous utilisons une entité enumerate pour le Type de Déchet, nous savons qu'il existe plusieurs types de déchets, cartons, verres, plastiques et métal. En faisant une énumération nous précisons le différent type de déchets qui ont une valeur finie de quatre. Cela permet d'avoir un UML qui précise les différents types de déchets qui doivent être identifiés lorsqu'elles sont mises dans des poubelles, une chose demandée dans le cahier des charges.

### **Accessibilité des attributs**

En ce qui concerne l'accessibilité des attributs, nous avons mis en privé tous ceux qui sont des informations relatives aux ménages, donc des informations "sensibles". Nous avons en revanche pris soin de mettre des attributs en public, comme les points de fidélité et la corbeille du ménage, le type de déchets, ou encore l'id du déchet jeté, puisque l'on s'en ressert dans d'autres entités, comme pour poubelle intelligente dans le cas de l'id déchet.

### **Les contraintes**

Enfin, nous avons placé les contraintes auxquelles certains des attributs doivent répondre. Par exemple, le prix des produits du commerce doit être supérieur à 0, et la réduction ne doit pas dépasser le prix de vente. Les points de fidélités dans le compte d'un ménage ne doivent pas être inférieurs à 0, et le poids des déchets doit être supérieur à 0. Le contrat qui lie le centre de tri et le commerce doit aussi avoir une date de fin qui est supérieure ou égale à la date de début du partenariat. Aussi, nous avons ajouté une contrainte Produit : {bag} pour spécifier que les consommateurs peuvent acheter plusieurs fois le même produit.

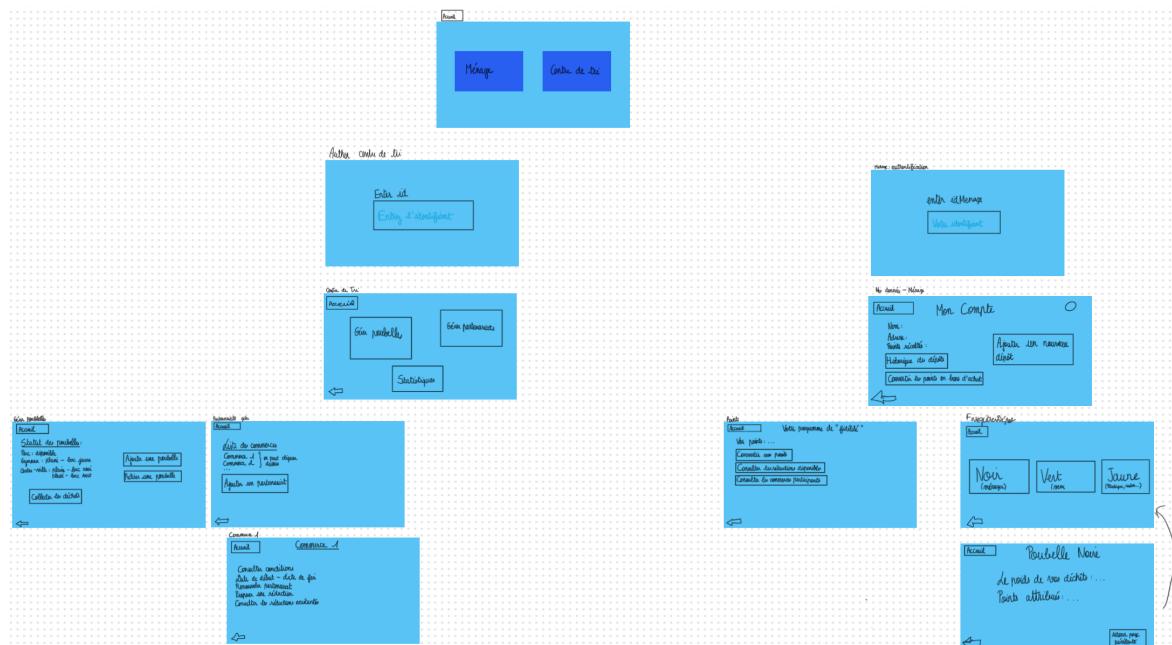
## Partie 2– Java Texte

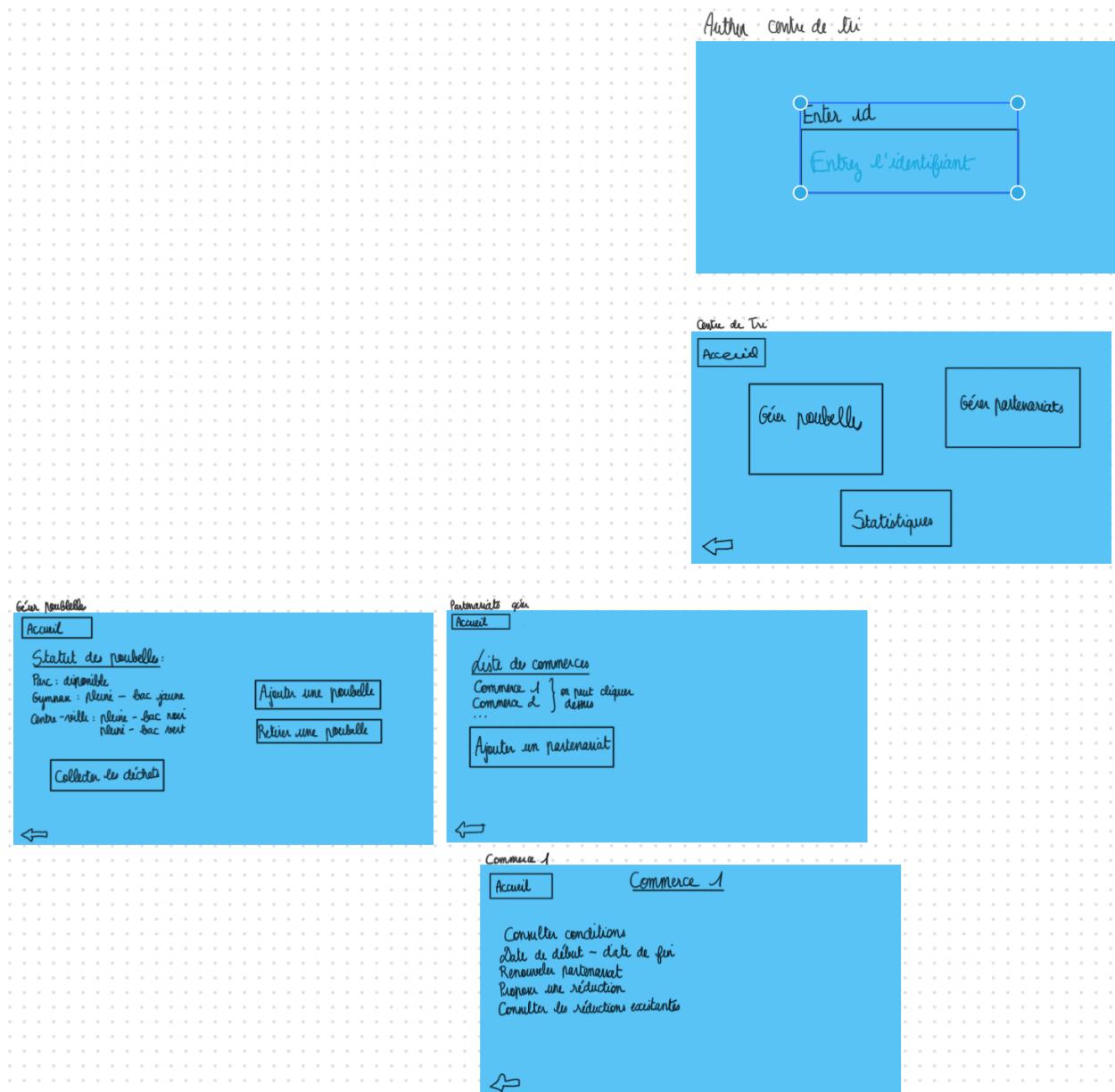
### Introduction et Enjeux

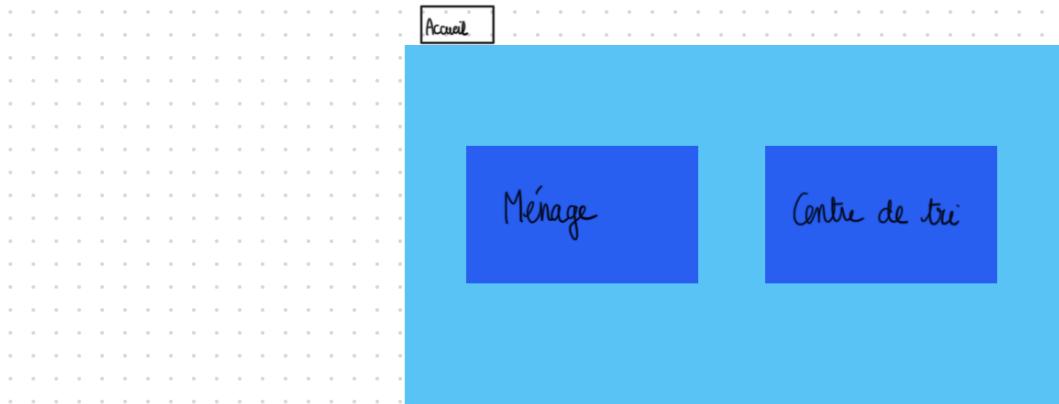
Nous avons pour objectif dans ce projet de modéliser un système de fidélité, initié par un centre de tri et visant à réduire les coûts de traitement des déchets, tout en incitant les ménages à trier correctement leurs déchets. Les points de fidélité sont accordés en fonction de la quantité de déchets triés et peuvent être retirés si le tri n'a pas été réalisé sérieusement. Ces points de fidélité sont échangeables contre des bons d'achat ou encore des réductions dans les commerces locaux en partenariat avec le centre de tri. Mais ce système de fidélité est plus qu'un moyen de réduire les coûts de traitement des déchets : il permet aussi de promouvoir une gestion des déchets plus écologique. Nous avons donc pour mission l'implémentation de ce système de fidélité. Après avoir réalisé un UML pour notre projet, nous avons commencé à produire nos classes créées dans la première partie du projet en java texte dans l'optique d'être utilisables dans la troisième partie pour l'IHM de notre projet Java. Pour la partie java texte, nous ne prenons pas en compte les conditionneurs et les recycleurs.

### Coder les classes en Java

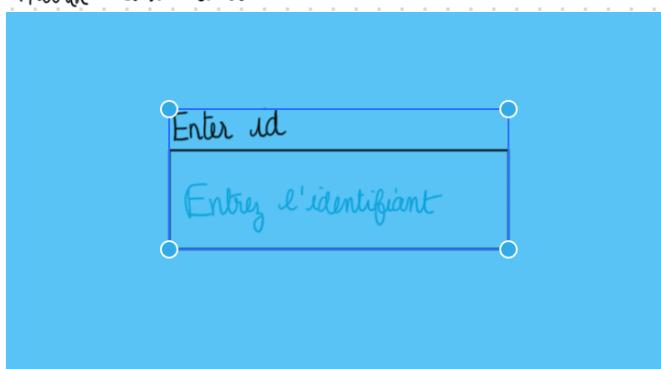
Pour commencer à produire les différentes classes Java, nous avons commencé par réfléchir à la forme de notre IHM. Nous avons donc fait un schéma pour visualiser notre future interface graphique, mais aussi pour créer nos classes Java qui proviennent de notre UML mais qui doivent prendre en compte le fonctionnement de notre IHM.



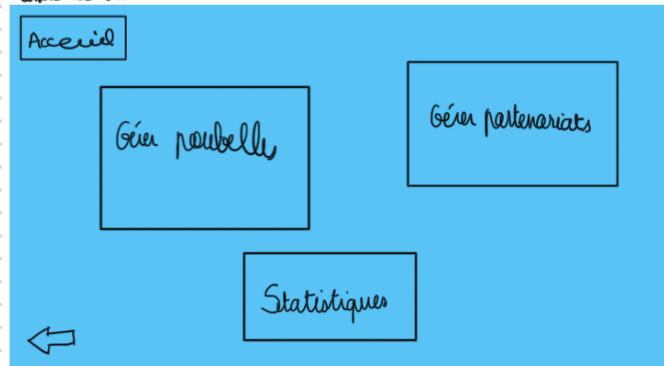


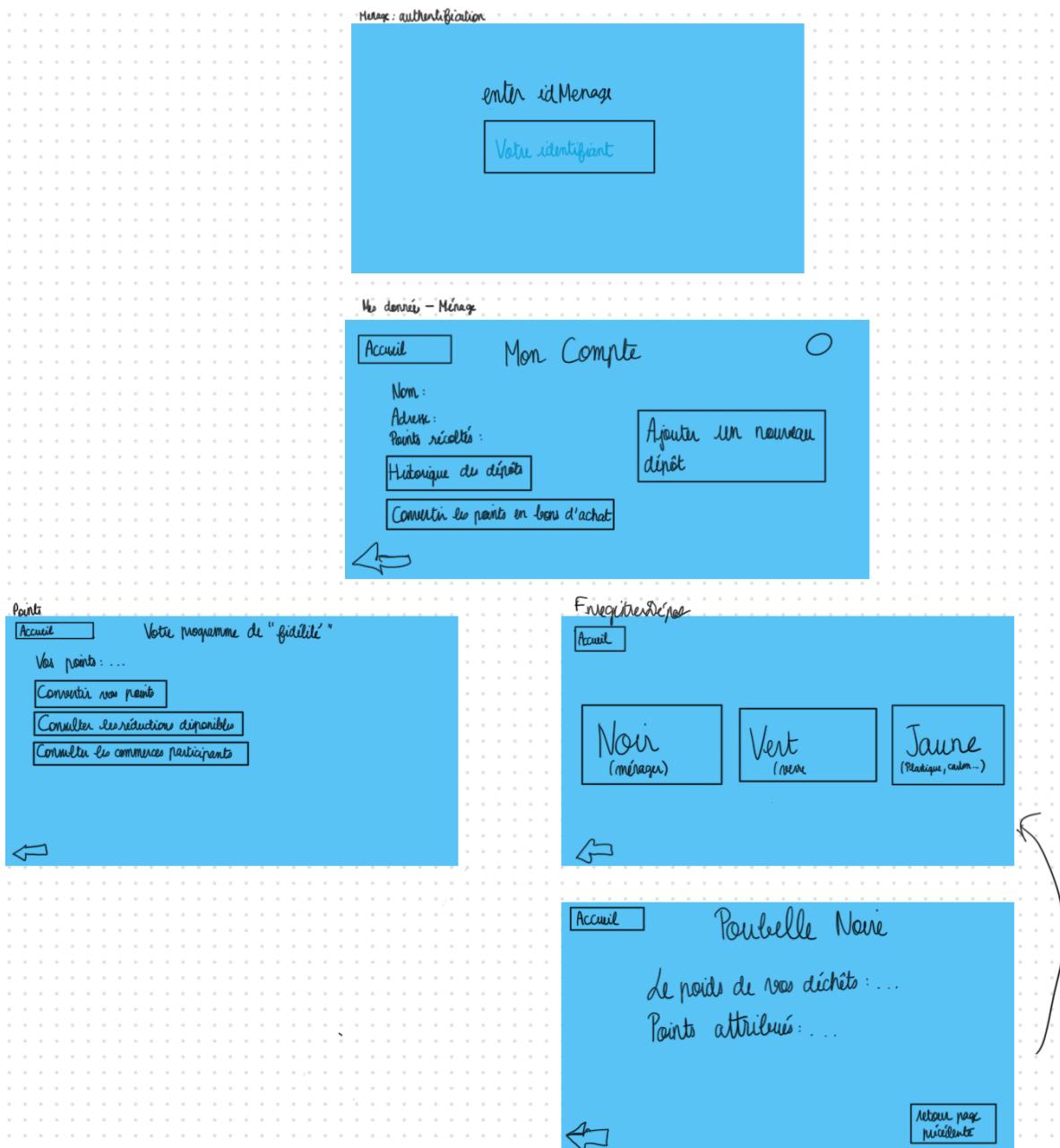


Authen : centre de tri



Centre de Tri





Ci-dessus, le premier schéma de notre IHM qui nous sert de squelette pour produire le vrai IHM, mais surtout pour créer et implémenter les classes et méthodes nécessaires en Java pour que le projet respecte le cahier des charges et soit fonctionnel.

Pour commencer, nous avons réparti les tâches entre les membres du groupe. Chaque personne avait des classes à implémenter, plus des classes de tests à faire, sans oublier de prendre en compte que certaines classes vont s'appeler ou appeler d'autres attributs d'autres classes. Nous gardons une communication importante pour coder les classes java, car malgré la répartition du travail, ces classes sont censées fonctionner dans un projet unique.

## Quelques exemples de nos classes Java

```

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

public class Menage {

    private String idMenage;
    private String nom;
    private String adresse;
    private List<Depot> historiqueDepots;
    private List<Produit> produitsAchetes;
    private int corbeille; // Passé en privé
    private int pointsFidelite;

    // Constructeur
    public Menage(String nom, String adresse, int pointsFidelite) {
        this.nom = nom;
        this.adresse = adresse;
        this.historiqueDepots = new ArrayList<>();
        this.produitsAchetes = new ArrayList<>();
        this.corbeille = 0;
        this.pointsFidelite = pointsFidelite;
        this.idMenage = UUID.randomUUID().toString();
    }

    // Getters et Setters
    public String getIdMenage() {
        return idMenage;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }
}

public class PoubelleIntelligente {

    private int idPoubelle;
    private int capaciteMax;
    private String emplacement;
    private float niveauRemplissage;
    private List<TypeDechet> typesAcceptes;
    public boolean poubellePleine;

    private Bac bacJaune;
    private Bac bacVert;
    private Bac bacNoir;

    // Mes setters
    public PoubelleIntelligente(int id, int capaciteMax, String emplacement, float niveauRemplissage, List<TypeDechet> typesAcceptes) {
        this.idPoubelle = id;
        this.capaciteMax = capaciteMax;
        this.emplacement = emplacement;
        this.niveauRemplissage = niveauRemplissage;
        this.typesAcceptes = typesAcceptes;
        this.poubellePleine = false;
    }

    this.bacJaune = new Bac(this.capaciteMax, this, "jaune");
    this.bacVert = new Bac(this.capaciteMax, this, "vert");
    this.bacNoir = new Bac(this.capaciteMax, this, "noir");
}

// Les multiples getters
public int getIdPoubelle() {
    return idPoubelle;
}

public int getCapaciteMax() {
    return capaciteMax;
}

public String getEmplacement() {
    return emplacement;
}

public float getNiveauRemplissage() {
    return niveauRemplissage;
}

```

Notre classe Poubelle Intelligente représente une poubelle intelligente capable de gérer plusieurs types de déchets. Elle est conçue pour optimiser la gestion du tri sélectif. Elle a un identifiant unique, une capacité maximale de déchets que chaque bac peut contenir, une localisation, un niveau de remplissage et d'autres attributs qui permettent de gérer correctement les déchets.

Notre classe ménage, quant à elle, représente nos foyers, utilisateurs du système de tri sélectif. Elle identifie chaque ménage, avec un identifiant UUID, elle référence nos produits, nos dépôts et nos points de fidélités par ménage.

## Notre classe qui nous permet d'avoir un CSV

```

public class CSVExporter {
    // Méthode pour exporter les ménages dans un fichier CSV
    public static void exporterMenages(String cheminFichier, List<Menage> menages) {

        try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(cheminFichier), "UTF-8"))) {
            // Écrire l'en-tête du CSV
            writer.write("IDDD Menage,Nom,Adresse,Points de fidélité,Historique Dépôts,Produits Achetés\n");

            // Parcourir les ménages et écrire les informations dans le fichier CSV
            for (Menage menage : menages) {
                // Créer une chaîne avec les informations de chaque ménage
                StringBuilder sb = new StringBuilder();
                sb.append(menage.getIdMenage()).append(",");
                sb.append(menage.getNom()).append(",");
                sb.append(menage.getAdresse()).append(",");
                sb.append(menage.getPointsFidelite()).append(",");

                // Historique des dépôts
                String historiqueDepots = "";
                for (Depot d : menage.getHistoriqueDepots()) {
                    historiqueDepots += d.getPointsAttribues() + ";";
                }
                sb.append(historiqueDepots.isEmpty() ? "" : historiqueDepots.substring(0, historiqueDepots.length() - 1));

                // Produits achetés
                String produitsAchetes = "";
                for (Produit p : menage.getProduitsAchetes()) {
                    produitsAchetes += p.getNom() + ",";
                }
                sb.append(produitsAchetes.isEmpty() ? "" : produitsAchetes.substring(0, produitsAchetes.length() - 1));

                // Écrire la ligne dans le fichier
                writer.write(sb.toString() + "\n");
            }

            System.out.println("Données exportées avec succès dans le fichier " + cheminFichier);
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("Erreur lors de l'exportation du fichier : " + e.getMessage());
        }
    }
}

```

La classe **CSVExporter** a pour objectif de faciliter l'exportation de données. Elle permet de prendre nos données, en l'occurrence nos ménages, leurs points de fidélités, leurs dépôts, et de les convertir en un format compatible avec le CSV, afin de les sauvegarder dans un fichier. Cette classe offre ainsi une manière simple et structurée d'enregistrer des informations sous forme tabulaire, facilitant le partage, l'analyse ou l'archivage des données. En automatisant le processus d'exportation, la classe **CSVExporter** permet de gagner du temps et d'assurer la cohérence du formatage des fichiers générés.

## Points de tensions

### Passage de l'UML à notre schéma d'IHM et l'implémentation des classes en Java

Notre première difficulté a été de faire un schéma d'IHM respectant notre UML le plus possible en sachant que toutes les classes ne sont pas forcément utilisées dans notre IHM. Nous avons dû repenser notre UML pour faire un schéma d'IHM plus logique. Re-travailler de la construction de l'UML était obligatoire pour commencer l'implémentation des classes Java pour avoir des

classes qui fonctionnent mieux ensemble. Nous avons dû transformer notre liste de déchets en une classe d'énumération.

```
package centretri;  
  
// c'est plus typeDepot que typeDechet  
public enum TypeDechet {  
    METAL(15F), VERRE(20F), CARTON(20F), PAPIER(10F), MENAGER(20F), PLASTIQUE(20F);
```

Pour le Centre de tri, la méthode "ajouterPoubelle" est devenue un objet de type poubelle alors qu'il était défini comme objet de type "int". Nous avons dû comprendre comment appeler certains attributs dans certaines classes, comme "PoidsDechet" créé dans la classe Corbeille et appelé dans la classe "Dépot".

### Problème avec MainTest

Pour les tests, nous avons rencontré le problème de savoir si nous devions avoir un seul "MainTest" ou plusieurs. L'autre difficulté est de faire des tests proches du réel et qui soient logiques et permettent de tester les classes et leurs exceptions.

### Transmission des classes

Des difficultés à transmettre les classes Java entre nous, à trouver la version la plus récente, ont été rencontrées. Il était parfois ardu de se retrouver dans les différentes versions modifiées. Nous avons réussi à centraliser à peu près la transmission d'informations. La plupart des classes sont interdépendantes, donc pour finaliser cette classe, d'autres classes devaient être faites avant. Occasionnellement, les classes devaient être retravaillées pour mieux correspondre aux attentes.

### Difficultés liées au code :

La gestion de l'identifiant unique était compliquée ainsi que l'authentification. Nous avons longuement réfléchi à la manière dont notre client peut se connecter à notre poubelle intelligente. Nous avons opté pour l'utilisation de UUID afin de générer des identifiants uniques pour chaque ménage. Le dépôt, lié à la corbeille, a également posé des difficultés, en effet, étant lié à la fois à la corbeille, à la poubelle intelligente et au ménage. Il a fallu s'accorder sur nos méthodes et la manière dont procéder. Nous avons par ailleurs rencontré des difficultés quant à l'héritage de notre poubelle intelligente. Nous avons privilégié la composition, car elle correspondait plus à la vision que nous avions d'une poubelle intelligente, c'est-à-dire du fait qu'une poubelle est composée de bacs de couleur.

## Modifications apportées

Suite aux retours et à nos différentes réunions, nous avons modifié certaines relations comme, le lien entre notre entité Déchet et Dépôt. Nous avons également rajouté une relation entre Corbeille et Dépôt. Nous avons également peaufiné quelques détails sur nos énumérations, et nos attributs afin d'être en troisième forme normale.

## Avis

Travailler sur ce projet nous a permis de travailler la programmation orientée objet Java, ce qui est essentiel pour notre recherche de stage, au vu de l'universalité de Java. Ce dernier facilite le développement d'applications capables de gérer plusieurs tâches simultanément, ce qui change de la programmation que nous avions pu faire jusqu'à maintenant. De plus, travailler en groupe nous a permis de travailler notre adaptabilité, notre résilience et notre coordination de projet, ce qui pourrait s'appliquer dans le monde du travail. La gestion de projet, en particulier si on fait face à des défis comme la gestion du temps, l'organisation du code et la résolution de problèmes techniques en un temps imparti assez réduit, rend le travail très dynamique et vraiment fatigant. Nous aurions apprécié peut-être de commencer Java en deuxième année afin d'avoir plus d'expérience dans ce langage de programmation.

## Instructions

Implémentation d'une interface permettant le tri de déchets grâce à des poubelles intelligentes en Java. Pour exécuter notre programme, il faut télécharger tous les fichiers java qui sont dans le dossier zip. Puis il faut créer un projet java et le package centretri, et ajouter les classes à ce package.

## Présentation succincte des composantes du projet :

- Classe Poubelle Intelligente - permet d'instancier les poubelles intelligentes ainsi que leur bac. Elle gère aussi le tri des dépôts, c'est-à-dire qu'elle ajoute les bons déchets dans les bonnes poubelles et informe de la capacité disponible pour chaque bac ainsi que du statut de la poubelle.
- Classe Corbeille – permet d'instancier la corbeille donnée aux ménages qui ont un compte. La classe Corbeille permet de mettre des déchets de certains poids dans un même endroit et de connaître le poids total des déchets dans notre poubelle.
- La classe Menage représente un foyer dans une application de gestion des déchets et des points de fidélité. Chaque ménage est identifié par un ID unique, possède un nom, une adresse, un historique de dépôts, ainsi qu'un solde de points de fidélité. Ces points sont attribués en fonction des dépôts effectués dans des poubelles intelligentes.
- La classe Depot permet au Menage de vider sa Corbeille dans la Poubelle Intelligente. On récupère ainsi l'heure de l'action et le nombre de déchets déposés de chaque type. On calcule ensuite les points obtenu dans l'action en appliquant un coefficient par type de déchet

- La classe Produit permet d'utiliser les points obtenus par le tri des déchets dans des réductions sur des types de produit. Une méthode calcule ensuite le prix du produit après remise.

## Partie 3 – Conception de l'IHM

### Introduction et enjeux:

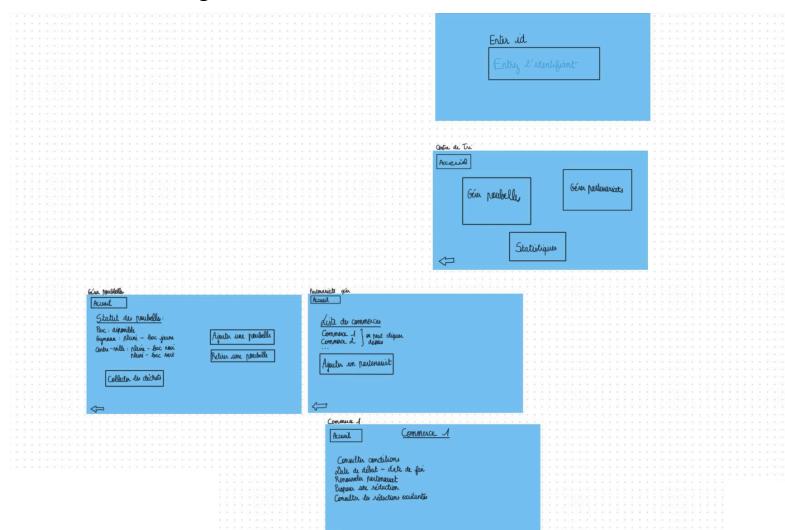
Nous avions pour objectif de réaliser une “Interface Homme Machine” à partir de notre code Java de gestion de notre Poubelle Intelligente, le but étant que nos ménages puissent se connecter à la poubelle grâce à son identifiant, et qu’ils puissent y voir leurs dépôts, leurs points de fidélités, mais également ajouter des dépôts. L'IHM sert de pont entre l'utilisateur et les fonctionnalités du logiciel. Nous avons utilisé JavaFx comme frameworks. Java permet de gérer ces interactions grâce à des listeners et des handlers, qui sont cruciaux pour rendre l'interface interactive. L'objectif principal de la création de cette IHM est de garantir une expérience utilisateur fluide, efficace et agréable, tout en assurant un code bien structuré et facile à maintenir. Nous avons donc repris nos croquis sur nos visualisations de l'IHM pour construire la structure du code.

### Méthode

Nous avons donc suivi la logique et le schéma de fonctionnement dessiné de l'IHM durant la partie de création des classes java textes pour la création de l'IHM avec Scene Builder et JavaFx. La programmation de l'IHM suit un long processus commencé avec la création des classes java.

### Avant la conception de l'IHM

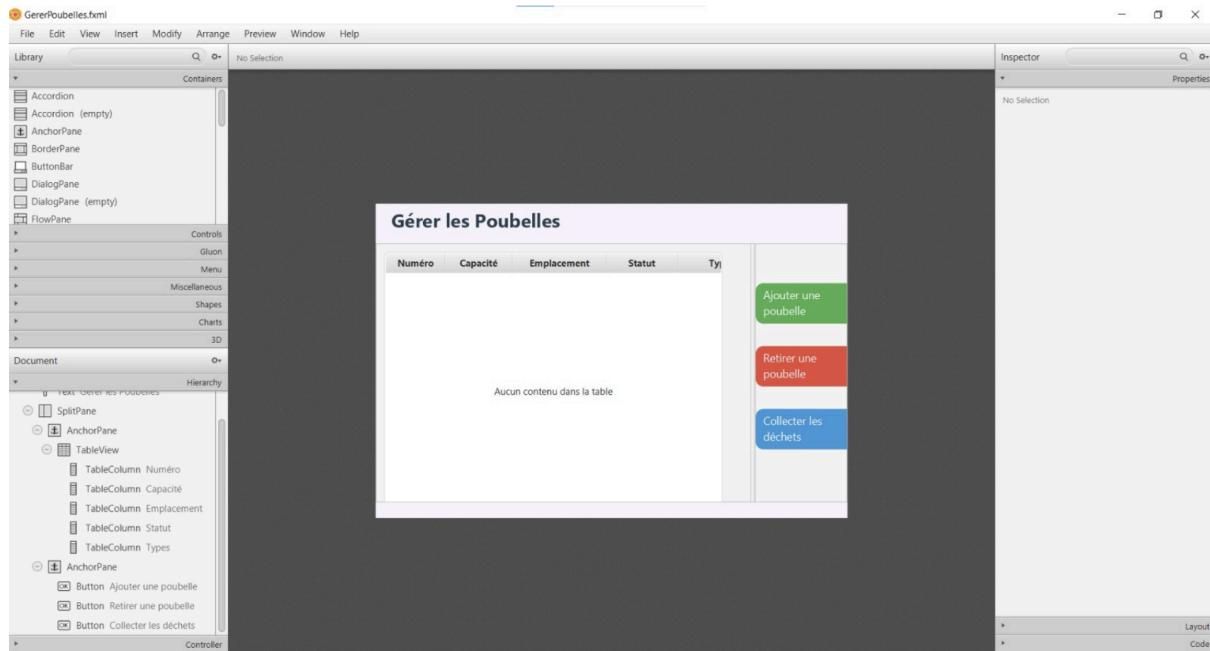
#### Croquis de la visualisation de notre IHM:



## Pendant la conception de l'IHM

L'utilisation de Scene Builder, nous a été indispensable pour la création de notre interface. En effet, cet outil permet de concevoir graphiquement les interfaces utilisateur en JavaFX de manière intuitive, sans avoir à écrire directement le code XML correspondant (FXML). Cela nous a permis de gagner du temps tout en assurant une meilleure organisation et une visualisation en temps réel des modifications apportées à l'interface. On s'est particulièrement servit des panneaux properties, code et layout afin d'ancrer et d'ajouter des identifiants viables à nos composants (boutons, textfield) dans le fichier fxml.

Capture d'écran de notre Scene Buider:



Extrait du code d'un de nos contrôleurs:

```

1 package centreDeTri;
2
3 import javafx.fxml.FXML;
4 import javafx.fxml.FXMLLoader;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Alert;
7 import javafx.scene.control.TextField;
8 import javafx.scene.layout.VBox; // Import nécessaire pour VBox
9 import javafx.stage.Stage;
10 import java.io.IOException;
11
12 public class LoginController {
13
14     @FXML
15     private TextField idMenageField;
16
17     private Menage menageConnecte; // Pour stocker le ménage actuellement connecté
18
19     @FXML
20     public void initialize() {
21         MenageManager.ajouterMenage(new Menage("François", "Paris", 0));
22         MenageManager.ajouterMenage(new Menage("Marie", "Lyon", 100));
23         MenageManager.ajouterMenage(new Menage("Luc", "Marseille", 50));
24     }
25
26     @FXML
27     public void handleLogin() {
28         String idSaisi = idMenageField.getText();
29         Menage menage = MenageManager.getMenageById(idSaisi);
30
31         if (menage != null) {
32             menageConnecte = menage; // Stockez le ménage connecté
33             afficherPageMonCompte(); // Passez à la page "Mon Compte"

```

Le but de notre contrôleur est de servir d'intermédiaire . Il permet de gérer la connexion d'un ménage à partir d'un identifiant saisi par l'utilisateur. La méthode “initialize()” initialise des données de test en ajoutant plusieurs ménages, tandis que *handleLogin()* vérifie l'identifiant saisi, connecte le ménage correspondant et prépare la redirection vers une autre page.

### La fonction handle Retour:

```

        }
    }

    @FXML
    private void handleRetour() {
        try {
            // Charger la page précédente (ex: ChoosePlayer.fxml)
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/centreDeTri/ChoosePlayer.fxml"));
            Parent root = loader.load();

            // Obtenir la scène actuelle et la remplacer par la nouvelle
            Stage stage = (Stage) idMenageField.getScene().getWindow();
            stage.setScene(new Scene(root));
            stage.setTitle("Choose Player");
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

Cette méthode permet de revenir à la page précédente (interface utilisateur) dans notre application. FXMLLoader est utilisé pour charger une nouvelle page définie par un fichier en l'occurrence notre centre de tri. La méthode `getScene().getWindow()` récupère la fenêtre actuelle. La fenêtre est mise à jour avec un nouveau titre "Choose Player". En cas d'erreur lors du chargement du fichier FXML, l'exception IOException est capturée et affichée avec `printStackTrace()`.

### Nos boutons:

```

<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane prefHeight="450.0" prefWidth="600.0" style="-fx-background-color: #f4f4f9;" xmlns="http://javafx.com/javafx/21" xmlns:fx="http://javafx.com/fxml/1"
    <children>
        <!-- Bouton Retour -->
        <Button text="← Retour" layoutX="20" layoutY="20" onAction="#handleRetour"
            style="-fx-background-color: transparent; -fx-text-fill: #4CAF50; -fx-font-size: 14px; -fx-cursor: hand;" />

        <!-- Titre principal -->
        <Text layoutX="100" layoutY="80" text="Connexion pour les ménages"
            style="-fx-font-size: 24px; -fx-font-weight: bold; -fx-fill: #333333;" />

        <!-- Texte pour ID Ménage -->
        <Label layoutX="130" layoutY="160" text="ID Ménage:"
            style="-fx-font-size: 14px; -fx-text-fill: #555555;" />

        <!-- Champ de saisie pour ID -->
        <TextField fx:id="idMenageField" layoutX="230" layoutY="150" prefHeight="40" prefWidth="250"
            style="-fx-background-radius: 8px; -fx-border-color: #dddddd; -fx-border-radius: 8px; -fx-font-size: 14px;" />

        <!-- Bouton Connexion -->
        <Button layoutX="230" layoutY="220" text="Se connecter" onAction="#handleLogin"
            style="-fx-background-color: #4CAF50; -fx-text-fill: white; -fx-font-size: 14px; -fx-background-radius: 8px; -fx-cursor: hand;"
            prefWidth="250" prefHeight="40" />

```

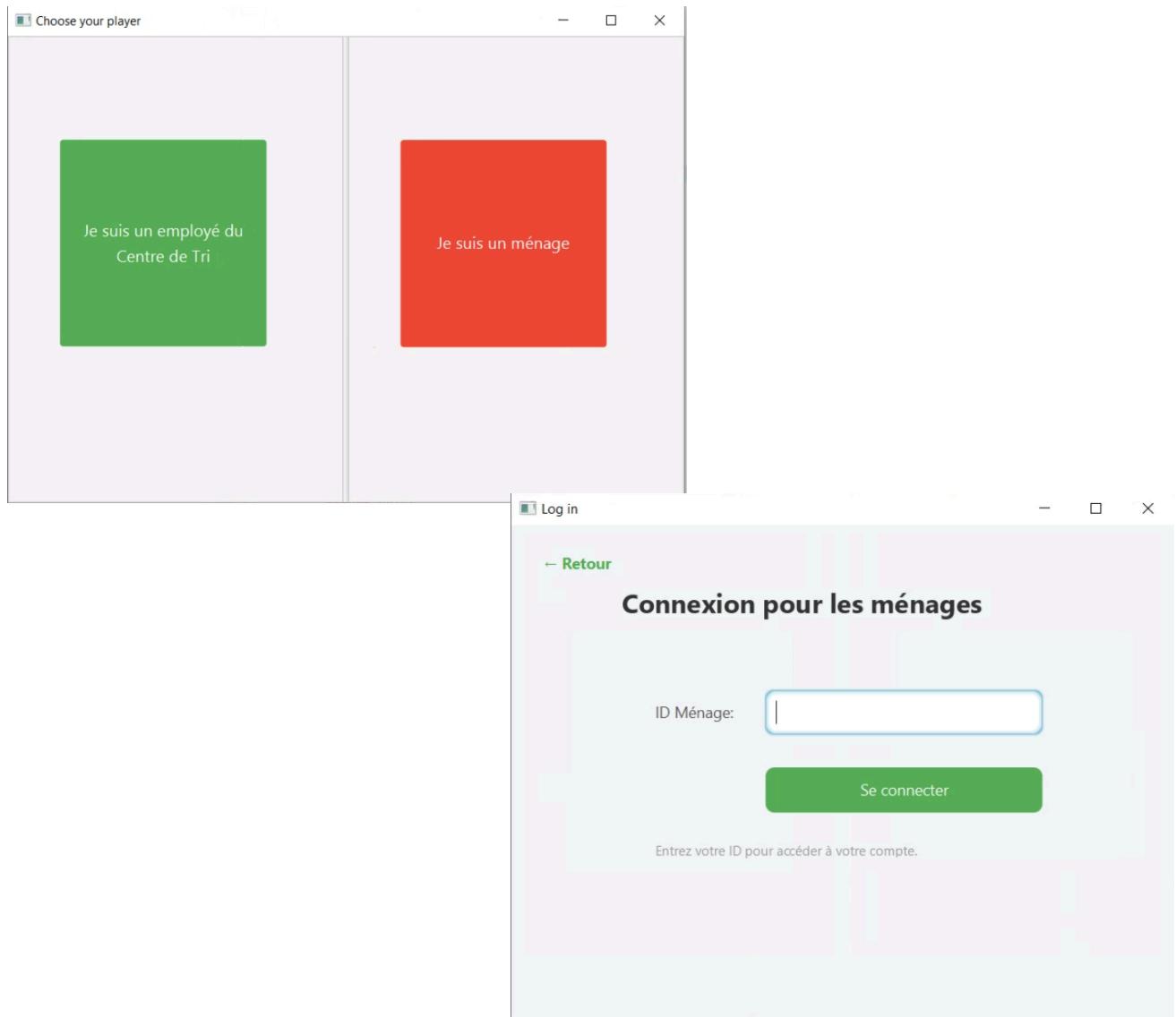
Nous avons ici plusieurs l'interface utilisateur qui est définie (boutons, champs) et relie les éléments visuels aux méthodes Java grâce à `onAction` et `fx:id`. Le bouton retour permet à l'utilisateur de revenir à l'écran précédent lorsqu'il est cliqué. Le bouton Retour est lié à la méthode `handleRetour()`.

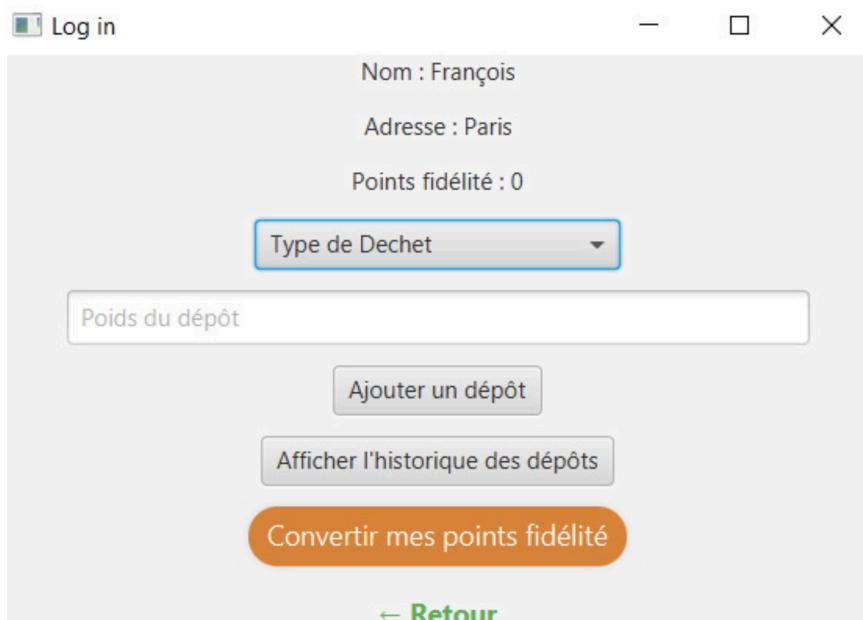
Nous avons également le champ de saisie de l'`idMenageField` qui permet à l'utilisateur de saisir un ID Ménage. L'utilisation du composant `TextField` pour permettre la saisie. Et l'ID `fx:id="idMenageField"` est essentiel pour récupérer sa valeur via `idMenageField.getText()`. Pour finir nous avons, également le Bouton Connexion, le but étant de soumettre les informations saisies pour se connecter à une fonctionnalité liée aux ménages.

## Notre IHM

### Quelques extraits de notre interface graphique

Comme vous pouvez le voir, notre utilisateur choisit s'il est un employé du centre de tri ou un ménage. Le ménage peut se connecter grâce à son identifiant unique.





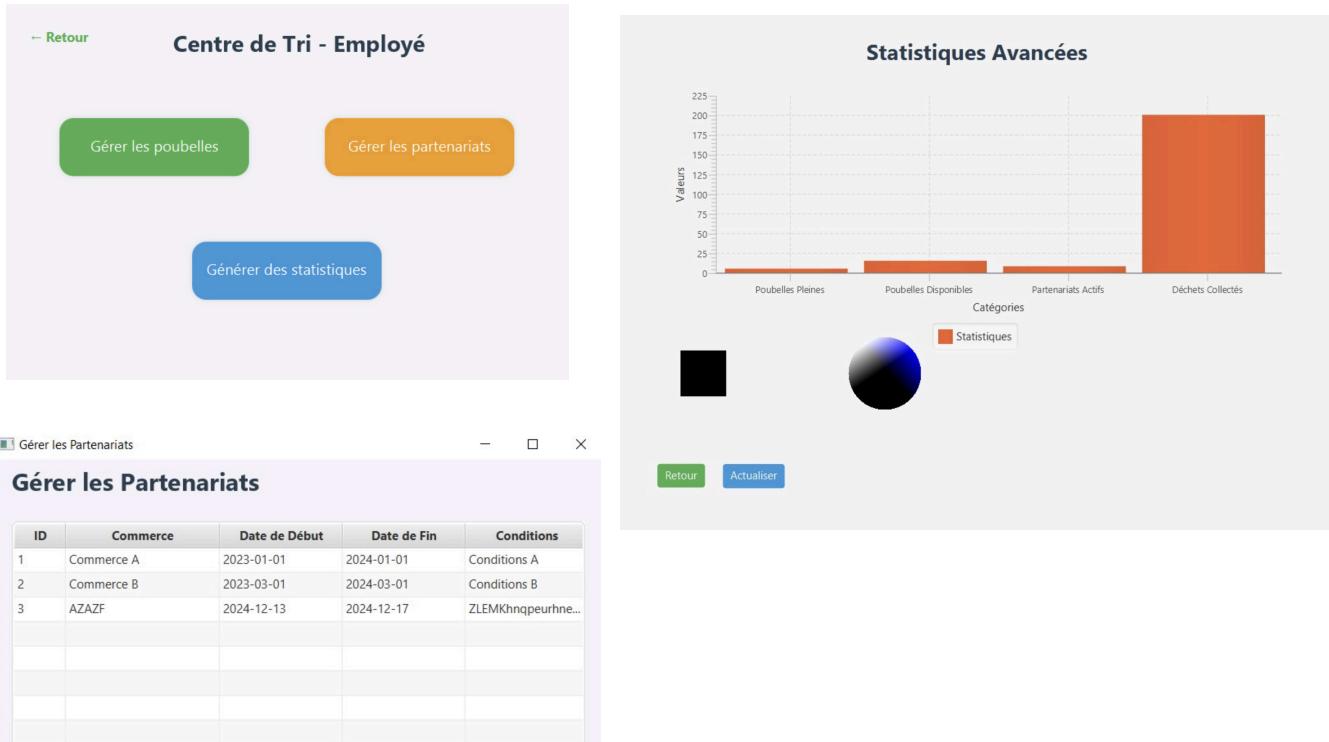
```
Liste des ménages créés :
Menage{idMenage='6e995abe-9fb5-4293-86ff-efaa2a0218f1', nom='François',
adresse='Paris', pointsFidelite=0}
Menage{idMenage='41b73bba-634e-4aed-a3bc-719020f25f9f', nom='Marie',
adresse='Lyon', pointsFidelite=0}
Menage{idMenage='5886bd93-3d63-4861-8adb-1568b6c45a7f', nom='Luc',
adresse='Marseille', pointsFidelite=0}
```

Une fois l'identifiant entré, les données liées au compte de l'utilisateur s'affichent : son nom, son adresse, ses points de fidélité. Il peut aussi consulter son historique de dépôts, et en ajouter un. L'interface nous permet également d'accéder à toutes nos poubelles intelligentes reliées à notre centre de tri, ainsi qu'à leur quantité de déchets. Nous avons une page statistique qui résume les quantités de déchets de nos poubelles.

Le ménage peut également consulter ses points de fidélités, convertir ses points en bon d'achat, et consulter les réductions disponibles avec nos partenaires.



Les employés du centre de tri peuvent gérer les poubelles, les partenariats, et générer des statistiques. En gérant les partenariats, il peut consulter des dates de début et de fin de notre contrat ainsi que les conditions du commerce. Les statistiques permettent de regarder l'état de nos poubelles.



## Points de tension

Nous avons rencontré quelques difficultés que ce soit pour la prise en main de Java Fx, en effet cette interface ne nous est pas intuitive. Et par conséquent, sa manipulation est compliquée. Comprendre son fonctionnement demande énormément de temps. Malgré les TP, utiliser JavaFx reste complexe.

Nous avons également rencontré quelques difficultés lorsque nous avons mis en commun nos fichiers Java Fx, par manque de compatibilité sur certaines versions de JavaFx. L'installation de nouveaux packages était parfois nécessaire pour utiliser les fonctionnalités voulues pour que l'interface face ce qui est attendu. Coordonner notre travail entre nous pour l'IHM fut compliqué car réussir à réutiliser le code d'autrui est souvent complexe, parfois ne fonctionne pas comme prévu donc il faut réussir à l'adapter pour que ces bouts de JavaFx puissent fonctionner ensemble.

## Modifications apportées

Suite aux retours, nous avons ajouté un CSV à notre Java, qui regroupe tous nos ménages ainsi que leurs dépôts, leurs points de fidélité. L'exportation des données utilisateurs et de son dépôt en fichier CSV sont réalisés dans notre classe main grâce à une autre classe la classe CSVExporter qui contient les méthodes pour créer et exporter le csv.

Nous avons également rajouté une méthode de test par classe, et retiré toute saisie de nos classes tests.

## Avis Global

Ce projet, grâce à sa complexité et à son ancrage dans le réel, était très intéressant. En effet, il présente une application des concepts de la programmation orientée objet tout en répondant aux exigences d'un cahier des charges complexes pour la gestion du tri sélectif. Cette approche du tri permet de nous ramener un sujet actuel, le tri, l'écologie avec les nouvelles réglementations de compost en ville. D'un aspect plus technique, ce projet a permis de revoir tout ce que l'on avait fait depuis le début de l'année. Le passage du diagramme UML à l'implémentation en Java a été un véritable défi, mais également un excellent moyen de renforcer nos compétences en architecture logicielle et en développement Java. La gestion du projet en groupe nous a obligés à gérer les versions du code et à résoudre des problèmes techniques en temps réel.

Ce projet nous a par ailleurs permis de mieux comprendre l'importance de la conception de l'IHM pour garantir une expérience fluide et intuitive, notamment avec l'utilisation de JavaFX. Bien que l'apprentissage de ce logiciel ait présenté des défis, particulièrement en termes de prise en main. In fine, ce projet nous a permis d'appliquer nos connaissances techniques tout en abordant des enjeux actuels et environnementaux importants.