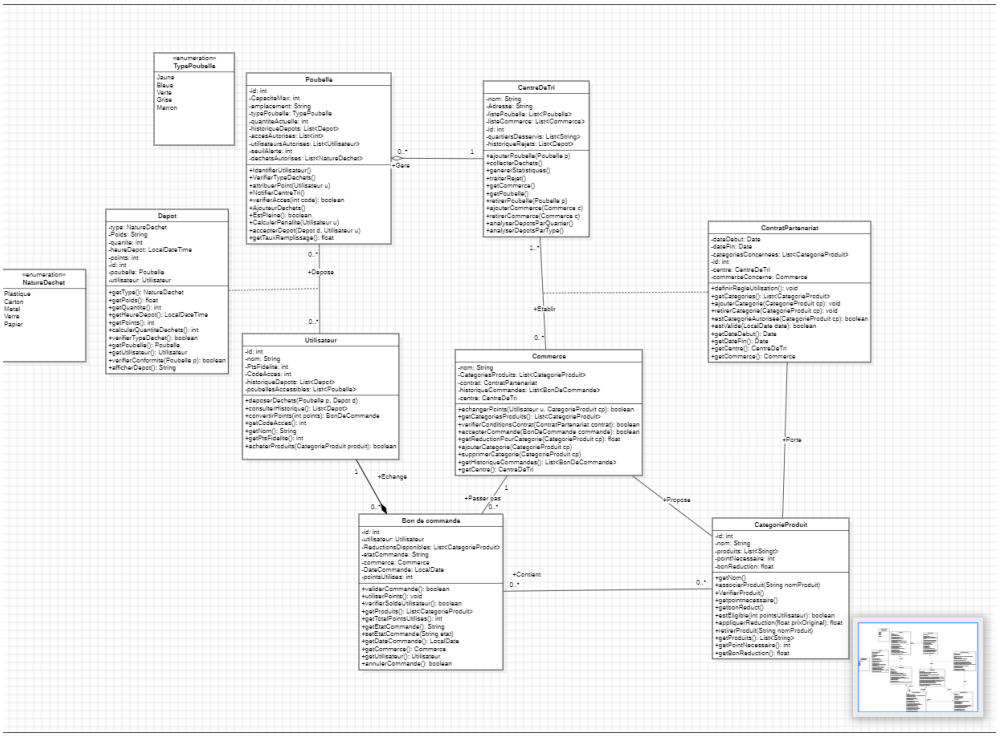
Explication visibilité de chaque attributs et méthodes

Pour cette modélisation UML, nous avons choisi de faire un diagramme composé de 8 classes UML :

- CentreDeTri
- Poubelle
- Utilisateur
- Dépôt
- Bon de commande
- Commerce
- ContratPartenariat
- CategorieProduit

Visibilité des attributs et méthodes pour chaque classe UML



1. CentreDeTri

Cette classe comporte comme attributs (privés, -) :

- - nom : String -> identifie les centres de tri
- - adresse : String -> indique l'emplacement du centre pour organiser la logistique
- – listePoubelle : List<Poubelle> -> identifie toutes les poubelles dans le centre de tri
- -listeCommerce : List<Commerce> -> identifie tous les commerces qui interagissent avec les utilisateurs
- -quartiersDesservis : List<String> -> identifie tous les quartiers qui interagissent avec le centre de tri
- -historiqueRejets: List<Depot> -> affiche la liste des rejets

Et comme méthode (publique, +) :

- + GererPoubelle() -> permet de gérer les poubelles dans la ville.
- + collecterDechets() -> organise la collecte des déchets par les camions.
- + genererStatistiques() -> analyse les données pour optimiser le tri et la collecte.
- + traiterRejet() -> brûle les déchets non utilisés
- + GetCommerce () -> getter
- + GetPoubelle() -> getter
- + analyserDepotsParQuartier()
- + analyserDepotsParType()
- + ajouterPoubelle(Poubelle P)
- + retirerPoubelle(Poubelle P)
- + ajouterCommerce(Commerce C)
- + retirerCommerce(Commerce C)

Le centre de tri est responsable de la gestion des poubelles, du tri et de l'analyse des données pour améliorer le recyclage.

Les attributs sont privés (-) pour garantir l'encapsulation et éviter toute modification directe en dehors de la classe.

Les méthodes sont publiques (+) car elles représentent les actions du centre de tri accessibles par d'autres classes.

2. Poubelle

Cette classe comporte comme attributs (privés, -) :

- − id : int -> chaque poubelle a un identifiant unique.
- − capaciteMax : int -> définit la capacité maximale pour éviter les débordements.
- − emplacement : String -> permet de savoir où la poubelle est installée.
- − TypePoubelle : String -> identifie le type de la poubelle (verre, plastique,…)
- − QuantiteActuelle : int -> identifie la quantité actuelle de déchets dans la poubelle
- - historiqueDepots : List<Depot>
- - accesAutorises : List<int>
- - utilisateursAutorises : List<Utilisateur>
- - seuilAlerte : int
- - dechetsAutorises : List<NatureDechet>

Et comme méthode (publique, +) :

- + identifierUtilisateur() -> vérifie l'accès des utilisateurs avec les identifiants
- + verifierTypeDechets() -> vérifie que le tri est bien respecté
- + attribuerPoints() -> attribue des points de fidélité en fonction du tri
- + notifierCentreTri() -> envoi d'une alerte si la poubelle est pleine ou si le tri n'est pas respecté
- + VerifierAcces() -> vérifie l'accès de l'utilisateur
- + ajouterDechets() -> ajoute des déchets dans la poubelle
- + EstPleine(): booleen
- + CalculerPenalite(Utilisateur u)
- + accepterDepot(Depot d,Utilisateur u)
- + getTauxRemplissage(): float

La poubelle est assez indépendante pour vérifier si le tri est bien respecté et envoi une alerte dans le cas où ce n'est pas respecté ou si la poubelle est pleine.

Les attributs sont privés (-) pour empêcher leur modification directe (par exemple : empêcher de changer l'ID d'une poubelle).

Les méthodes sont publiques (+) car elles doivent être appelées par d'autres classes, comme Utilisateur ou CentreDeTri.

3. Utilisateur

Cette classe comporte comme attributs (privés, -) :

- - id : int -> Identifiant unique de chaque utilisateur
- - nom : String -> Nom de l'utilisateur pour la gestion du compte
- - pointsFidelite : int -> Stocke les points gagnés grâce au bon tri
- - CodeAcces : int -> le code pour accéder aux poubelles
- - ProduitsAchetes : List<Produit> -> les produits échangés avec les points
- - historiqueDepots : List<Depot> -> l'historique de tous les déchets déposés par l'utilisateur

Et comme méthode (publique, +) :

- + deposerDechets() -> Permet à l'utilisateur de jeter ses déchets dans une poubelle connectée
- + consulterHistorique() -> Affiche les dépôts de déchets et les points gagnés
- + convertirPoints() -> Permet d'échanger les points en bons d'achat
- + GetCodeAcces() -> getter
- + GetListProduit() -> getter
- + GetNom() -> getter
- + GetPointsFidelite() -> getter
- + AcheterProduits(Produit p) -> échange les points avec les produits

L'utilisateur est le principal acteur du tri et doit pouvoir suivre ses actions et bénéficier du système de récompenses.

Les attributs sont privés (-) pour protéger les informations personnelles et éviter la modification directe du nombre de points de fidélité.

Les méthodes sont publiques (+) car elles permettent à l'utilisateur d'interagir avec le système.

4. Dépot

Cette classe associative comporte comme attributs (privés, -) :

- − type : String -> type de dépôt
- − Poids : String -> poids du déchet
- − Quantite : int -> quantité de déchets déposés
- − heureDepot : Date -> heure à laquelle le dépot à été fait
- − Points : int -> nombre de points associés au dépot
- - id: int
- - poubelle: Poubelle
- - utilisateur: Utilisateur

Et comme méthode (publique, +):

- + calculerQuantitésDéchets() : int
- + verifierTypeDechets(): booleen
- + verifierConformite(Poubelle p): booleen
- + afficherDepot(); string

Le reste est uniquement des getters. En effet, Depot étant une classe associative, toutes les méthodes sont déjà présentes dans les deux classes associées à celle-ci.

5. CategorieProduit

Cette classe comporte comme attributs (privés, -) :

- − nom : String
- − id : int
- − pointsNecessaires : int -> nombre de points nécessaires pour l'achat du produit
- - pointNecessaire : int
- − bonReduction: float

Et comme méthode (publique, +) :

- + getNom() -> getter
- + associerProduit(Sring nomProduit)
- + verifierProduit ()
- + getpointnecessaire () -> getter
- + getBonReduct () -> getter
- + estEligible(int pointsUtilisateur): booleen
- + appliquerReduction (float prixOriginal): float
- + retirerProduit (string nomProduit)
- + getProduits(): List<Sring>
- + getPointNecessaire(): int
- + getBonReduction(): float

Cette classe permet la catégorisation des produits afin de faciliter leur recherche lors du commerce.

Les attributs sont privés (-) pour protéger les informations des catégories.

La méthode est publique (+) pour que la catégorisation puisse être faite.

6. Commerce

Cette classe comporte comme attributs (privés, -) :

- – nom : String -> Identifie le commerce.
- – categoriesProduit : List<String> -> Liste des types de produits qui acceptent les points fidélité.
- – centre: CentreDeTri
- – contrat : contratPartenariat
- - historiqueCommande: List<BonDeCommande>

Et comme méthode (publique, +) :

- + echangerPoints() -> Convertit les points fidélité en réductions ou bons d'achat.
- + getCategorieProduits() -> getter
- + VerifierConditionsContrat(ContratParteneriat contrat)
- + AccepterCommande(BonDeCommande commande)
- + getReductionPourCategorie(CategorieProduit cp)
- + ajouterCategorie(CategorieProduit cp)
- + supprimerCategorie(CategorieProduit cp)
- + getHistoriqueCommandes(): List<BonDeCommande>
- + getCentre():CentreDeTri

Les commerces partenaires permettent aux utilisateurs d'utiliser leurs points pour des réductions sur des produits.

Les attributs sont privés (-) pour éviter des modifications non contrôlées (exemple : éviter que n'importe quelle classe ne modifie les catégories de produits).

La méthode est publique (+) pour permettre aux utilisateurs d'échanger leurs points contre des réductions.

7. ContratPartenariat

Cette classe comporte comme attributs (privés, -) :

- − dateDebut : Date -> Indique la date de début du contrat.
- − dateFin : Date -> Indique la date de fin du contrat.
- − CategoriesConcernes : List<CategorieProduit>
- - id: int
- - centre: CentreDeTri
- - commerceConcernee: Commerce

Et comme méthode (publique, +) :

- + definirReglesUtilisation() -> Détermine les conditions d'utilisation des points fidélité.
- + getCategorie() -> getter
- + ajouterCategorie(CategorieProduit cp): void
- + retirerCategorie(CategorieProduit cp): void
- + estCategorieAutorises(CategorieProduit cp): booleen
- + estValide(LocalDate date); booleen
- + getDateDebut(): Date
- + getDateFin(): Date
- + getCentre(); CentreDeTri
- + getCommerce(): Commerce

Chaque commerce a un contrat avec le centre de tri, qui définit la durée et les conditions d'utilisation des points.

Les attributs sont privés (-) pour protéger les informations du contrat.

La méthode est publique (+) pour que le commerce puisse définir les règles d'utilisation.

8. BonDeCommande

Cette classe comporte comme attributs (privés, -) :

- - id :int
- - utilisateur : Utilisateur
- - ReductionsDisponibles: List<CategorieProduit>
- - pointsUtilises : int
- - etatCommande : String
- - commerce : Commerce
- - dateCommande : LocalDate

Et comme méthode (publique, +) :

- + validerCommande()
- + UtiliserPoints()
- + VerifierSoldeUtilisateur()
- + getProduits(): List<CategorieProduit>
- + getTotalPointsUtilises(): int
- + getEtatCommande(): String
- + getEtatCommande(String etat)
- + VerifierSoldeUtilisateur()
- + getDateCommande():LocalDate
- + getCommerce():Commerce
- + getUtilisateur():Utilisateur
- + annulerCommande():booleen

Cette classe permet la modélisation d'une transaction entre un utilisateur et un commerce.

Les attributs sont privés (-) pour protéger les informations des bons de commande.

La méthode est publique (+) afin de sécuriser, valider et exécuter les commandes passées par les utilisateurs auprès des commerces.

Explication cardinalité et choix d’association

CentreDeTri → gère plusieurs Poubelles
Le type d’association est une agrégation (◇). Sa cardinalité est 1 (CentreDeTri) → 0..* (Poubelle). Un centre de tri possède plusieurs poubelles, mais celles-ci peuvent exister indépendamment du centre de tri.

Utilisateur → dépose dans un Depot
Il s’agit d’une association simple (—). Sa cardinalité est de 1 (Utilisateur) → 0..* (Depot). Un utilisateur peut déposer ses déchets dans plusieurs dépôts, et un dépôt est utilisé par plusieurs utilisateurs.

Depot → est déposé dans une Poubelle
Il s'agit d'une association simple (—). Sa cardinalité est de 0..* (Depot) → 1 (Poubelle). Un dépôt est toujours déposé dans une seule poubelle.

Commerce → passe un ContratPartenariat avec CentreDeTri
Il s’agit d’une association simple (—). Sa cardinalité est de 1 (Commerce) → 1 (ContratPartenariat) → 1 (CentreDeTri). Chaque commerce définit un contrat avec un centre de tri, un contrat lie un seul commerce et un seul centre de tri.

Commerce → traite des BonDeCommande
Il s’agit d’une association simple (—). Sa cardinalité est de 1 (Commerce) → 0..* (BonDeCommande). Un commerce peut traiter plusieurs bons de commande.

Produit → appartient à une CatégorieProduit
Le type d’association est une agrégation (◇). Sa cardinalité est de 0..* (Produit) → 1 (CatégorieProduit). Un produit appartient à une seule catégorie, mais une catégorie peut contenir plusieurs produits.

Base de données associée à l’UML

On réalise ensuite la base de données associée à l’UML en créant une table différente pour chaque classe. On utilise aussi des tables associatives pour représenter les différentes listes.

Implémentation de l’IHM

L'application développée sous JavaFX (version 21) offre les fonctionnalités suivantes :

- **Gestion des utilisateurs :** Création et administration des comptes pour les foyers.
- **Sécurité d'accès :** Authentification sécurisée pour l'utilisation des bornes de collecte connectées.

- **Gestion des déchets :** Enregistrement et catégorisation des dépôts de déchets.
- **Suivi et historique :** Visualisation des points de fidélité accumulés et consultation de l'historique des transactions.
- **Offres partenaires :** Consultation des bons d'achat et réductions proposés par les commerces affiliés.
- **Supervision centralisée :** Outils de gestion et de supervision des bornes de collecte pour le centre de tri.

L'Interface Homme-Machine (IHM) est structurée autour des fenêtres/scènes principales suivantes :

a) Connexion / Inscription

- **Fonctionnalités :** Permet aux utilisateurs de s'authentifier via la saisie d'un code d'accès existant ou de créer un nouveau compte.
- **Processus :** Suite à l'inscription, un identifiant de badge unique est automatiquement attribué à l'utilisateur.

```java
package ihm;

import ...

public class InscriptionController {

    @FXML
    private TextField nomField, codeAccesField;

    @FXML
    private ComboBox<String> centreComboBox;

    @FXML
    private Label errorLabel;

    private Map<String, Integer> centreMap = new HashMap<>(); // nom du centre -> id du centre  2 usages

    @FXML
    public void initialize() { chargerCentres(); }
```

```java
public class InscriptionController {                                    ⚠ 6  ✓ 24

    private void chargerCentres() {  1 usage
        try (Connection conn = DatabaseConnection.getConnection()) {
            String sql = "SELECT id, nom FROM CentreDeTri";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                ResultSet rs = stmt.executeQuery();
                while (rs.next()) {
                    String nomCentre = rs.getString( columnLabel: "nom");
                    int idCentre = rs.getInt( columnLabel: "id");
                    centreComboBox.getItems().add(nomCentre);
                    centreMap.put(nomCentre, idCentre);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            errorLabel.setText("Erreur chargement centres.");
        }
    }
```

```java
    @FXML
    private void handleInscription(ActionEvent event) {
        try {
            String nom = nomField.getText();
            String codeAccesText = codeAccesField.getText();
            String centreSelectionne = centreComboBox.getValue();

            if (nom.isEmpty() || codeAccesText.isEmpty() || centreSelectionne == null) {
                errorLabel.setText("Veuillez remplir tous les champs.");
                return;
            }

            if (!codeAccesText.matches( regex: "\\d{4}")) {
                errorLabel.setText("Code d'accès : 4 chiffres obligatoires.");
                return;
            }

            int codeAcces = Integer.parseInt(codeAccesText);
```

```java
            int codeAcces = Integer.parseInt(codeAccesText);
            int centreId = centreMap.get(centreSelectionne);

            Utilisateur utilisateur = new Utilisateur( id: 0, nom, codeAcces, centreId);
            utilisateur.setCentreId(centreId); // on affecte le centre choisi

            UtilisateurDAO utilisateurDAO = new UtilisateurDAO(DatabaseConnection.getConnection());
            utilisateurDAO.insert(utilisateur);

            // Redirection vers la page de connexion
            FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/views/hello-view.fxml"));
            Parent root = loader.load();
            Stage stage = (Stage) nomField.getScene().getWindow();
            stage.setScene(new Scene(root,  v: 500,  v1: 400));
            stage.show();
        } catch (NumberFormatException e) {
            errorLabel.setText("Veuillez entrer un code valide (nombre).");
```

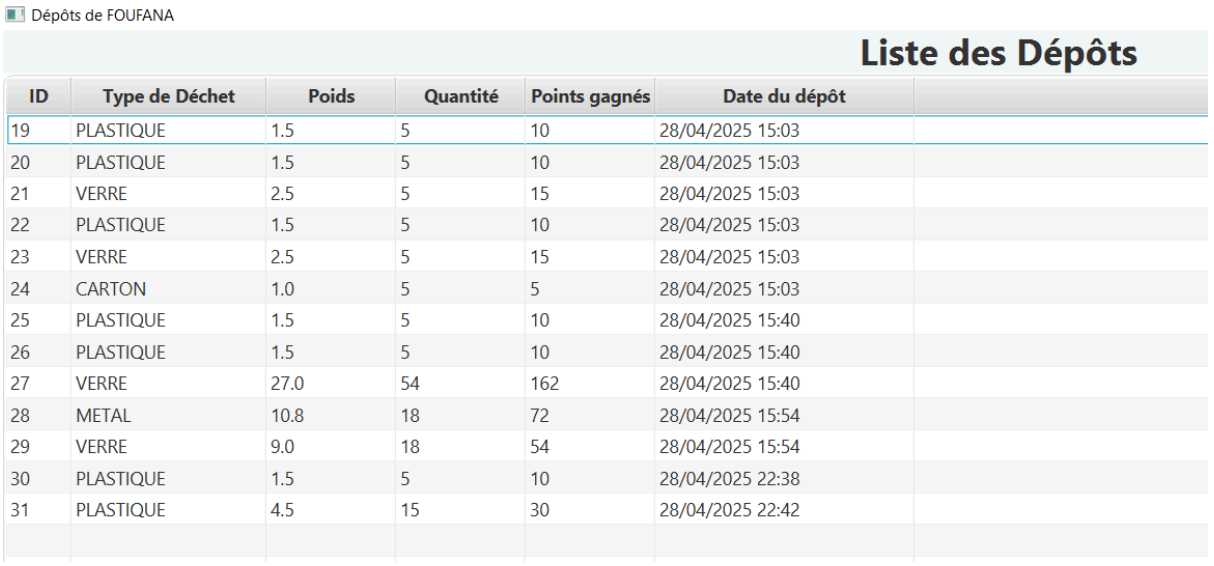
```
        } catch (Exception e) {
            e.printStackTrace();
            errorLabel.setText("Erreur lors de l'inscription.");
        }
    }

    @FXML
    private void handleGoToConnexion(ActionEvent event) {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/views/hello-view.fxml"));
            Parent root = loader.load();
            Stage stage = (Stage) nomField.getScene().getWindow();
            stage.setScene(new Scene(root,  v: 500,  v1: 400));
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
            errorLabel.setText("Erreur retour connexion.");
        }
    }
}
```

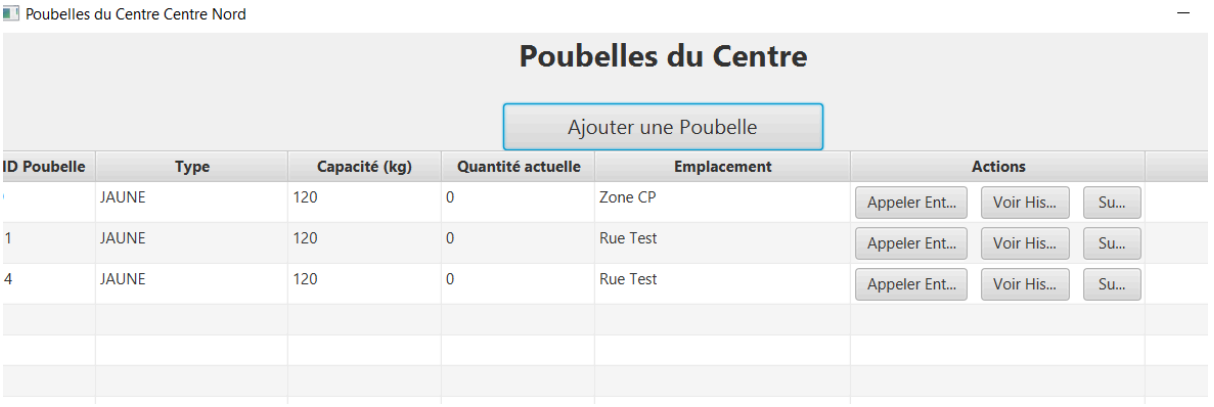
b) Tableau de bord utilisateur

- **Affichage des points :** Présentation en temps réel du solde de points de fidélité de l'utilisateur.
- **Historique des dépôts :** Affichage chronologique des dépôts effectués par l'utilisateur.
- **Consultation des offres :** Accès à la liste des bons d'achat et réductions disponibles.

c) Gestion des dépôts

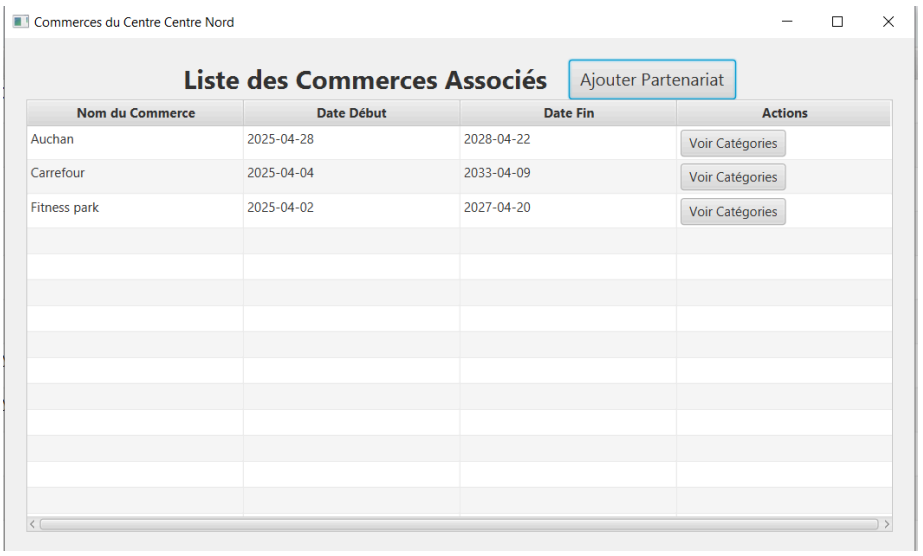
- **Sélection de la borne :** L'utilisateur sélectionne la poubelle connectée utilisée pour le dépôt.
- **Saisie des quantités :** L'utilisateur indique les quantités de déchets déposés, classées par type.
- **Système de points :** Attribution automatique de points de fidélité en fonction du dépôt, avec possibilité de pénalités en cas de non-respect des consignes de tri.

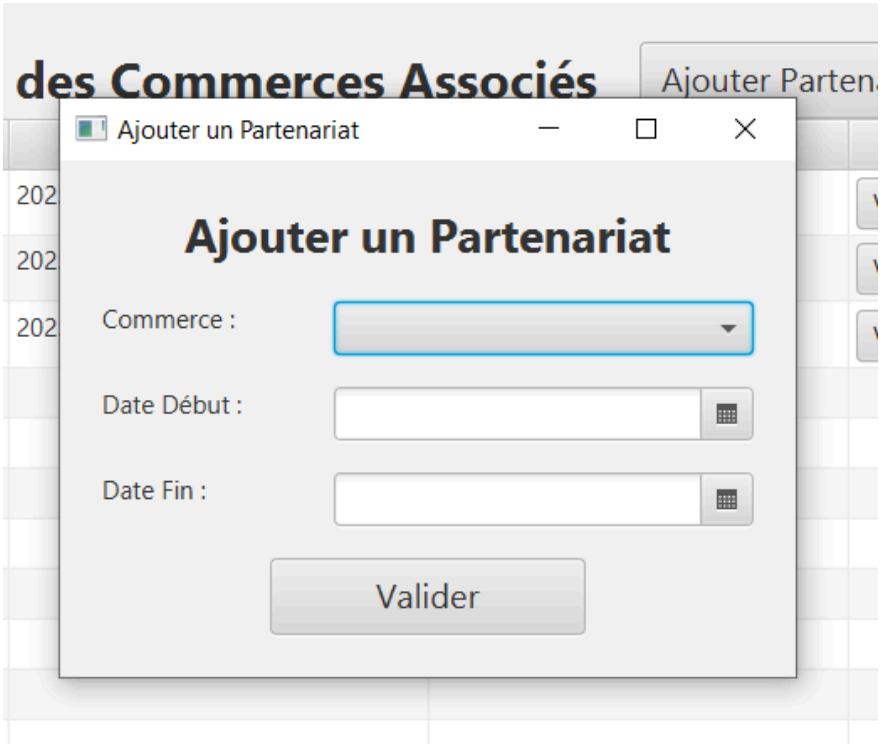




d) Interface administrateur (Centre de tri)

- **Surveillance des bornes :** Consultation en temps réel du niveau de remplissage des différentes poubelles connectées.
- **Analyse des données :** Génération de statistiques sur les dépôts, segmentées par zone géographique ou par catégorie de déchets.
- **Gestion du parc de bornes :** Fonctionnalités d'ajout et de retrait des poubelles connectées du système.
- **Gestion des partenariats :** Outils pour l'administration et le suivi des contrats avec les commerces partenaires.


Commerces du Centre Centre Nord
Liste des Commerces Associés
Ajouter Partenariat
Nom du Commerce
Date Début
Date Fin
Actions
Auchan
2025-04-28
2028-04-22
Voir Catégories
Carrefour
2025-04-04
2033-04-09
Voir Catégories
Fitness park
2025-04-02
2027-04-20
Voir Catégories

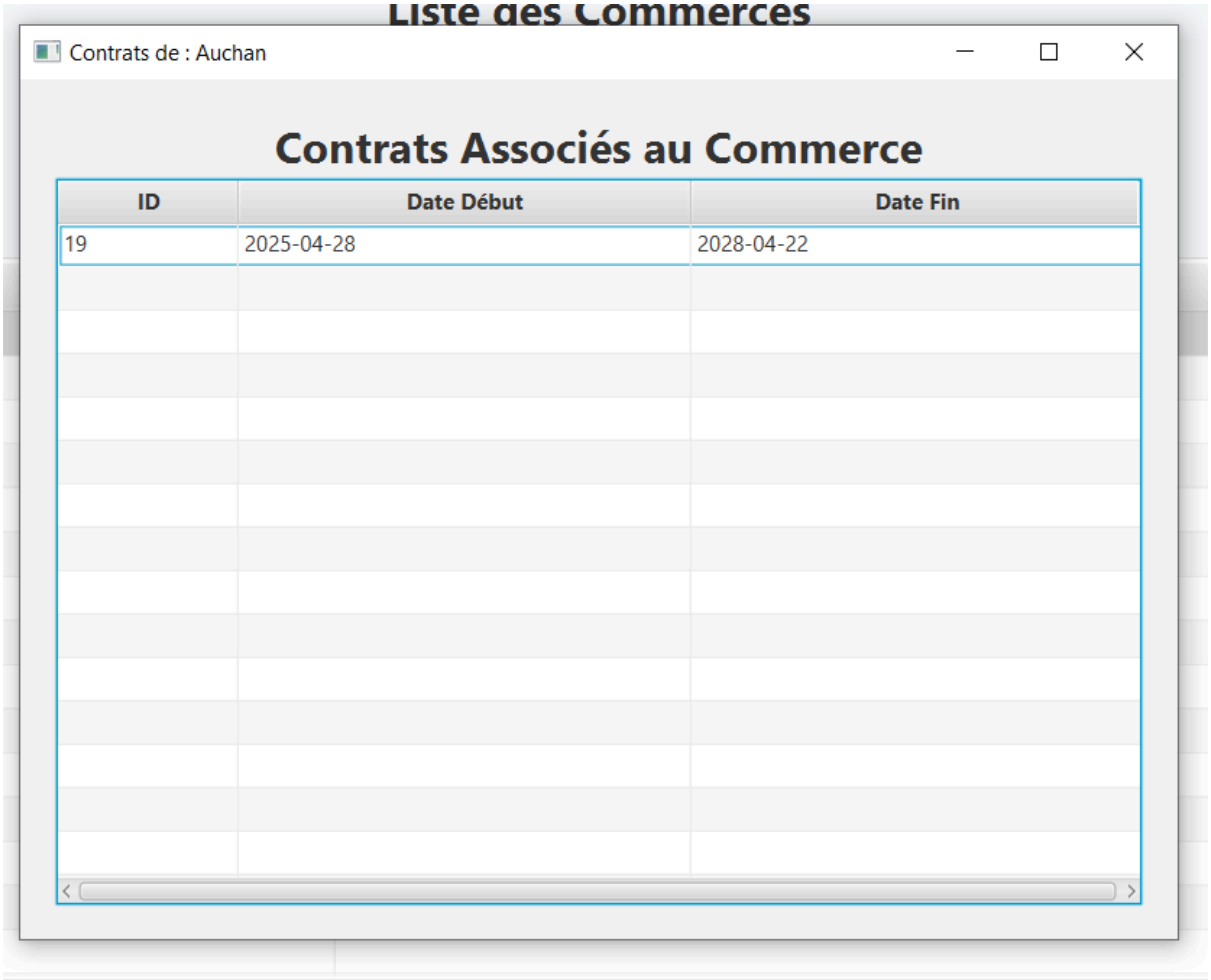

des Commerces Associés
Ajouter Parten
Ajouter un Partenariat
Ajouter un Partenariat
Commerce :
Date Début :
Date Fin :
Valider

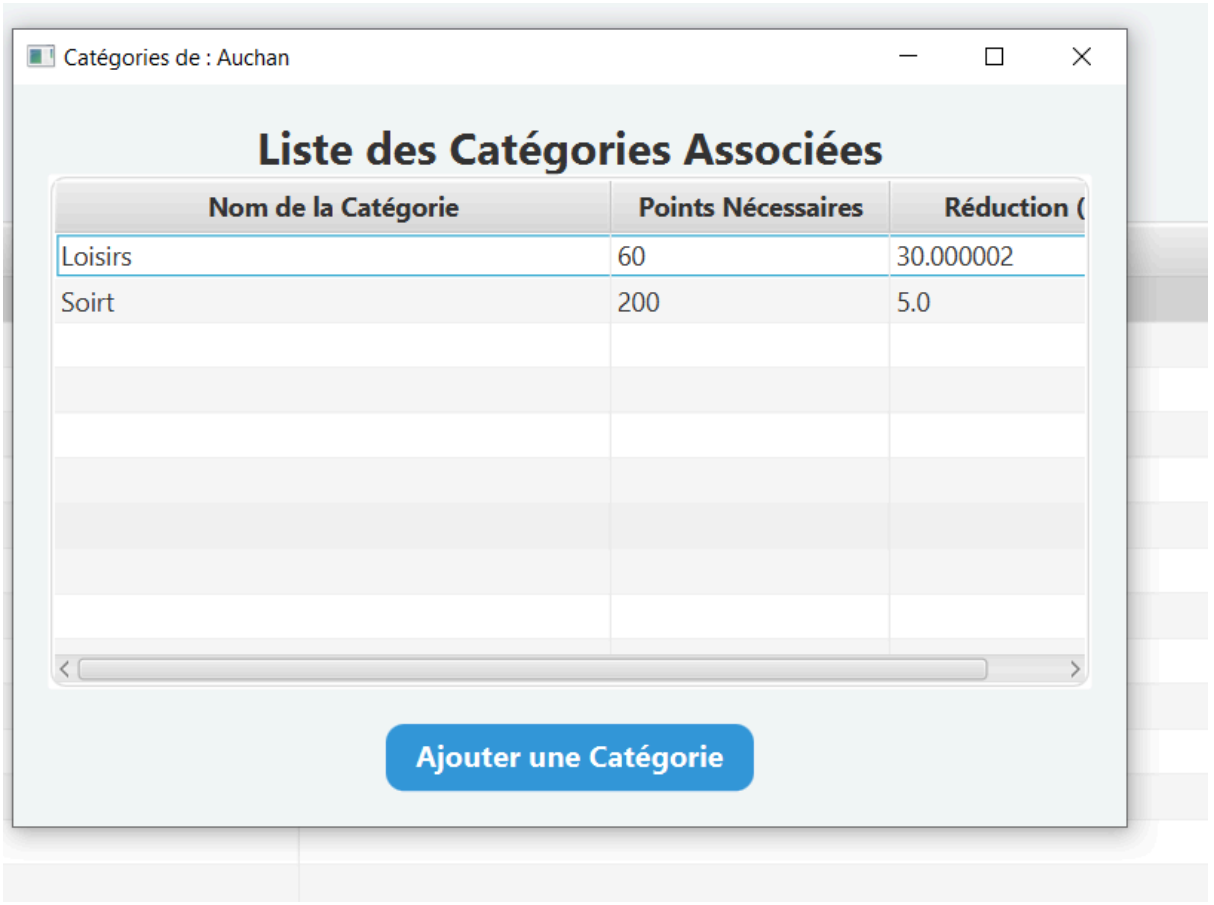
Retour

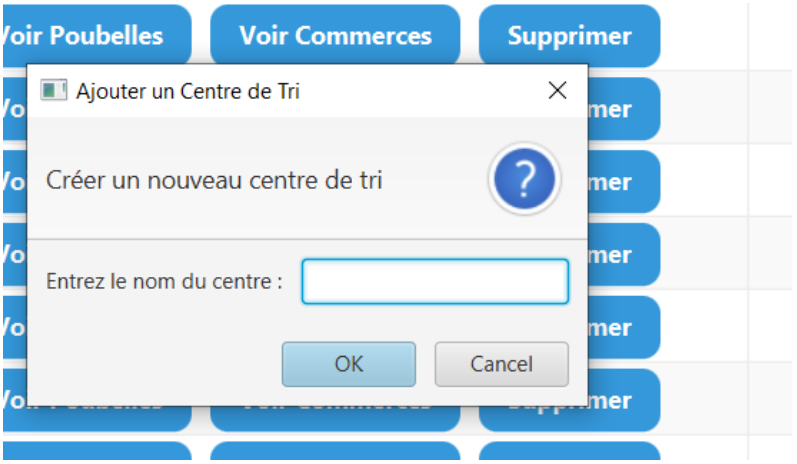
Liste des Commerces

ID	Nom du Commerce	
1	Auchan	
2	Commerce Test	
3	Carrefour	
4	Boulanger	
6	Carrefour	
7	Carrefour	
8	EcoMarket	
9	EcoMarket	
10	Decathlon	
11	Decathlon	
12	Decathlon	
13	Fnac	
14	EcoMarket	
15	EcoMarket	
16	Carrefour	

Voir Contrats Associés Voir Catégories Ajouter un Commerce

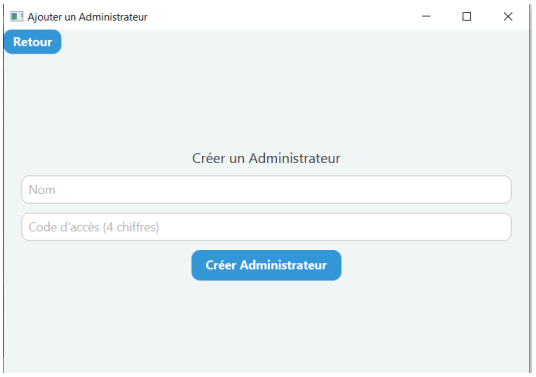

Liste des Commerces
Contrats de : Auchan
Contrats Associés au Commerce
ID
Date Début
Date Fin
19
2025-04-28
2028-04-22


Catégories de : Auchan
Liste des Catégories Associées
Nom de la Catégorie
Points Nécessaires
Réduction (
Loisirs
60
30.000002
Soirt
200
5.0
Ajouter une Catégorie


oir Poubelles
Voir Commerces
Supprimer
Ajouter un Centre de Tri
Créer un nouveau centre de tri
Entrez le nom du centre :
OK
Cancel

Interface

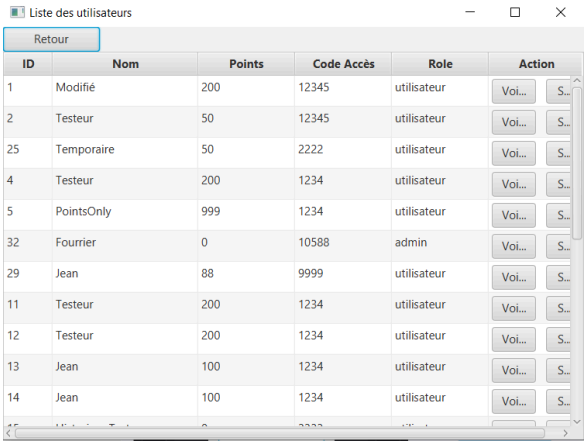

Dépôt de déchets
Interface Administrateur
Voir la liste des utilisateurs
Ajouter un nouvel utilisateur
Voir les centres de tri
Voir les commerces
Se déconnecter

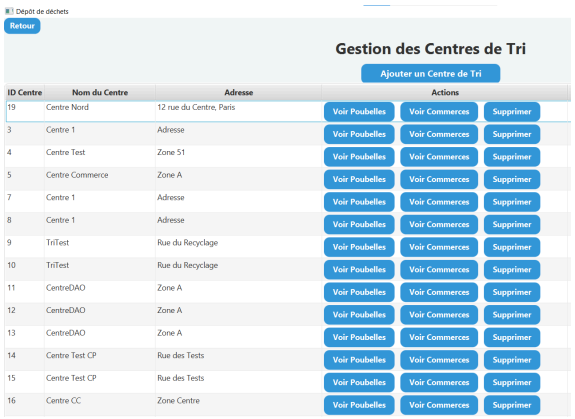

Ajouter un Administrateur
Retour
Créer un Administrateur
Nom
Code d'accès (4 chiffres)
Créer Administrateur

L’interface permet aux utilisateurs de se connecter à leur compte.

Si l'utilisateur est de type simple, il sera dirigé vers la page centre de tri où il pourra déposer des déchets. Chaque utilisateur étant affilié à un centre de tri, il peut choisir dans quelle poubelle de ce centre il souhaite jeter son déchet. L’utilisateur a accès à son historique, voir ses bons de commande ainsi qu’échanger ses points avec les différents commerces.

Si l’utilisateur est de type administrateur, il sera dirigé vers une autre page où il aura accès à la liste de tous les utilisateurs et à la liste des dépôts que chacun d’entre eux a effectué. L’admin peut également supprimer chaque utilisateur simple si il le souhaite. Il peut aussi ajouter un nouvel utilisateur. L’admin a l’accès à tous les centres de tri et peut voir la liste des poubelles associées à chaque centre ainsi que leur type, capacité maximale, la quantité actuelle, leur emplacement. Il est également possible d’ajouter un centre de tri ou les supprimer et de même pour les poubelles associées à un centre. Pour chaque poubelle, l’admin peut voir l’historique des dépôts associés à chaque utilisateur. Enfin, il est possible d’appeler une entreprise extérieure afin de vider la poubelle.






```java
package ihm;

import ...

public class AjouterAdminController {  1 usage

    @FXML
    private TextField nomField;

    @FXML
    private PasswordField motDePasseField;

    @FXML
    private void handleCreerAdmin() {
        String nom = nomField.getText();
        String codeAccesStr = motDePasseField.getText(); // en réalité ici on récupère le code d'accès (mot d

        if (nom.isEmpty() || codeAccesStr.isEmpty()) {
            showAlert(Alert.AlertType.ERROR, title: "Erreur", message: "Tous les champs doivent être remplis.");
            return;
```

```
        try {
            int codeAcces = Integer.parseInt(codeAccesStr);

            Utilisateur nouvelAdmin = new Utilisateur( id: 0, nom, codeAcces,  role: "admin",  centreId: 0);

            UtilisateurDAO utilisateurDAO = new UtilisateurDAO(utils.DatabaseConnection.getConnection());
            utilisateurDAO.insert(nouvelAdmin);

            showAlert(Alert.AlertType.INFORMATION,  title: "Succès",  message: "Administrateur ajouté avec succès
            nomField.clear();
            motDePasseField.clear();

        } catch (Exception e) {
            e.printStackTrace();
            showAlert(Alert.AlertType.ERROR,  title: "Erreur",  message: "Erreur lors de l'ajout de l'administrateu
        }
    }
```

```
    private void showAlert(Alert.AlertType alertType, String title, String message) {  4 usages
        Alert alert = new Alert(alertType);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }
}
```

Conclusion

Ce projet a constitué une opportunité significative pour l'application pratique du modèle architectural MVC (Modèle-Vue-Contrôleur), la mise en œuvre d'interfaces utilisateur ergonomiques via JavaFX, ainsi que l'établissement d'une connexion à une base de données pour la gestion dynamique des informations.

Forts de cette expérience, nous sommes désormais en mesure de proposer une solution tangible visant à optimiser la gestion du tri sélectif, en s'appuyant sur des outils modernes et interactifs.

https://github.com/Kantw1/Projet_gestion_de_tri_java