**Explication visibilité de chaque attributs et méthodes**

Pour cette modélisation UML, nous avons choisi de faire un diagramme composé de 9 classes UML :

* CentreDeTri
* Poubelle
* Utilisateur
* Dépôt
* Bon de commande
* Commerce
* ContratPartenariat
* CategorieProduit
* Produit

**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

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

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

Et comme méthode (publique, +) :

* + + identifierUtilisateur() -> vérifie l'accès des utilisateurs avec les identifiants
  + + calculerQuantiteDechets() -> calcul le nombre de déchets jetés
  + + 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

La poubelle est assez indépendante pour vérifier si le tri est bien respecté et envoi une alerte dans le cas ou 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

Et comme méthode (publique, +) nous avons 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. Produit**

Cette classe comporte comme attributs (privés, -) :

* + – nom : String
  + – id : int
  + – pointsNecessaires : int -> nombre de points nécessaires pour l’achat du produit
  + – categorieProd : List<CategorieProduit> -> les catégories à laquelle les produits peuvent appartenir

Et comme méthode (publique, +) :

* + + getNom() -> getter
  + + getPrixEnPoints() -> getter
  + + GetCategories() -> getter
  + + AssignCategorie() -> catégorie attribuée au produit
  + + ListerProduitsDisponibles()

Les méthodes de lecture getType() et getPoids() sont publiques (+) pour que d’autres classes puissent lire ces informations sans pouvoir les modifier directement.

**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.
  + – commandes : List<BonDeCommande>
  + – contrat : contratPartenariat

Et comme méthode (publique, +) :

* + + echnagerPoints() -> Convertit les points fidélité en réductions ou bons d’achat.
  + + getCategorieProduits() -> getter
  + + VerifierConditionsContrat(ContratParteneriat contrat)
  + + AccepterCommande(BonDeCommande commande)

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>

Et comme méthode (publique, +) :

* + + definirReglesUtilisation() -> Détermine les conditions d'utilisation des points fidélité.
  + + getCategorie() -> getter

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. CategorieProduit**

Cette classe comporte comme attributs (privés, -) :

* + – id :int
  + – nom : String -> nom de la catégorie du produit
  + – tauxConversion : int -> taux de conversion des points obtenus avec les déchets déposés
  + – produits :List<Produit>

Et comme méthode (publique, +) :

* + + GetNom() -> getter
  + + getTauxConversion() -> getter
  + + AssocierProduit(Produit p) -> associe chaque produit à une catégorie
  + + VerifierProduit(Produit p) -> vérifie si le produit est en stock ou existe

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.

**9. BonDeCommande**

Cette classe comporte comme attributs (privés, -) :

* + – id :int
  + – utilisateur : Utilisateur
  + – produits : List<Produit>
  + – montantTotal : double
  + – etatCommande : String
  + – commerce : Commerce
  + – dateCommande : LocalDate

Et comme méthode (publique, +) :

* + + validerCommande()
  + + UtiliserPoints()
  + + VerifierSoldeUtilisateur()

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.