CY Tech - 2024/2025

Analyse et programmation orienté objet

- Java -

Rendu 1

DABOUT Mattéo,
DOSSANTOS Emma,
EISH Aicha,
ROBARD Baptiste,
YAZIDI Asma

Première année d'ingénieur
Génie mathématique apprentissage

Analyse et programmation orienté objet
Verin Renaud

Table des matières

1. Introduction

• Contexte du projet

Dans le cadre de notre cursus nous devons ce semestre réaliser un projet en groupe dans la matière analyse et programmation orienté objet. Notre groupe est composé de Mattéo Dabout, Asma Yazidi, Baptiste Robard, Aicha Eish et Emma Dos Santos.

Dans ce projet il nous est demandé de mettre en pratique les notions vues en cours comme la programmation orientée objet, la modélisation UML, l'accès à une base de données et le développement d'une interface graphique.

Le thème de ce projet porte sur la gestion intelligente du tri sélectif des déchets. L'idée est de développer une application simulant le fonctionnement d'un système dans lequel des ménages déposent leurs déchets dans des poubelles connectées. Ces dernières sont capables d'identifier l'utilisateur, de mesurer la quantité et la nature des déchets déposés, de vérifier la conformité du tri, et d'envoyer des alertes au centre de tri lorsqu'elles sont pleines. En contrepartie d'un tri bien fait, les ménages reçoivent des points de fidélité qu'ils peuvent échanger contre des réductions chez des commerces partenaires.

Le projet se déroule en trois rendus :

— Une modélisation UML du système,
— Une implémentation Java en mode texte,
— L'implémentation de l'IHM.

• Objectifs et enjeux du rendu 1

Ce premier rendu concerne la modélisation UML, une étape essentielle dans tout projet de développement. Elle permet de structurer l'ensemble du système, de définir précisément les classes, leurs attributs, leurs méthodes, ainsi que les relations entre les objets. L'UML que nous avons conçu représente la base de notre projet et il va nous aider à structurer notre code epour assurer une meilleure organisation du code.

L'enjeu de ce premier rapport est donc de répondre à la problématique suivante : comment représenter de manière claire, logique et complète le système de tri décrit dans le cahier des charges, à travers un diagramme de classes UML ?

Pour cela, nous avons analysé les acteurs, les objets manipulés et les différentes interactions possibles, afin de produire une modélisation fidèle et exploitable pour les prochaines étapes du projet.

2. Description des classes

• Classe : CentreDeTri

- **Attributs de la classe**

  — `idCentreDeTri` : identifiant unique du centre de tri.
  — `nom` : nom du centre.
  — `adresse` : adresse physique du centre.
  — `partenariats` : liste des contrats ou partenariats établis avec des commerces.
  — `statistique` : lien vers les statistiques associées à l'activité du centre.

- **Méthodes de la classe**

  — `placerPoubelle()` : ajoute une poubelle à la zone gérée par le centre.
  — `retirerPoubelle()` : retire une poubelle de la zone gérée.
  — `collecteDechets()` : déclenche une collecte des déchets dans les poubelles pleines.
  — `getNom()` : retourne le nom du centre.
  — `getAdresse()` : retourne l'adresse du centre.

• Classe : Poubelle

- **Attributs de la classe**

  — `idPoubelle` : identifiant unique de la poubelle.
  — `capaciteMax` : capacité maximale en volume ou poids.
  — `capaciteActuelle` : capacité actuellement utilisée.
  — `typePoubelle` : type de déchets acceptés (plastique, verre, etc.).
  — `emplacement` : information sur l'emplacement exact (quartier, rue...).
  — `adresse` : adresse complète pour la localisation.
  — `estPleine` : booléen indiquant si la poubelle est pleine.

- **Méthodes de la classe**

  — `identifierUtilisateur()` : identifie l'utilisateur lors du dépôt.
  — `verifierAcces()` : vérifie si l'utilisateur a le droit d'utiliser la poubelle.
  — `verifierTypeDechet()` : contrôle si le type de déchet correspond à celui attendu.
  — `calculerQuantiteDechets()` : calcule le volume ou poids des déchets déposés.
  — `attribuerPoint()` : attribue des points de fidélité en fonction du tri.
  — `notifierCentreTri()` : envoie une alerte au centre lorsque la poubelle est pleine.
  — `estPleine()` : indique si la poubelle a atteint sa capacité maximale.

• Classe : Compte

- **Attributs de la classe**

  — `idCompte` : identifiant unique du compte utilisateur.
  — `nom` : nom de l'utilisateur associé au compte.
  — `email` : adresse mail de l'utilisateur.
  — `motDePasse` : mot de passe du compte (à sécuriser dans une application réelle).
  — `pointFidelite` : nombre de points accumulés grâce au tri.

- **Méthodes de la classe**

  — `seConnecter()` : permet à l'utilisateur de se connecter à son compte.
  — `consulterHistoriqueDepots()` : affiche la liste des dépôts effectués.
  — `deposerDechets()` : enregistre un dépôt de déchets effectué par l'utilisateur.
  — `echangerPoints()` : échange les points contre des réductions ou produits.
  — `acheterProduits()` : permet l'achat de produits dans la boutique partenaire.
  — `getNom()` : retourne le nom de l'utilisateur.
  — `getPtsFidelite()` : retourne le nombre de points fidélité.
  — `setPtsFidelite()` : modifie le nombre de points fidélité.

• Classe : Depot

- **Attributs de la classe**

  — `idDepot` : identifiant unique du dépôt.
  — `poids` : poids total des déchets déposés.
  — `quantite` : quantité (nombre d'objets ou volume).
  — `dateDepot` : date à laquelle le dépôt a été effectué.
  — `pointsGagnes` : nombre de points fidélité obtenus.
  — `idPoubelle` : identifiant de la poubelle utilisée.
  — `contenu` : description du contenu du dépôt.

- **Méthodes de la classe**

  — `getTypeDechet()` : retourne le type de déchet déposé.
  — `getPoids()` : retourne le poids du dépôt.
  — `getQuantite()` : retourne la quantité déposée.
  — `getDateDepot()` : retourne la date du dépôt.
  — `getPoints()` : retourne les points gagnés pour ce dépôt.

• Classe : Commerce

- **Attributs de la classe**

  — `idCommerce` : identifiant unique du commerce partenaire.
  — `nom` : nom du commerce.
  — `categoriesProduits` : liste des catégories de produits proposés.
  — `contrat` : référence au contrat de partenariat signé avec le centre de tri.

- **Méthodes de la classe**

  — `supprimerContrat()` : permet de résilier le contrat avec un commerce.
  — `estEnContrat()` : vérifie si un contrat actif est en cours.
  — `appliquerReduction()` : applique une réduction sur les produits.
  — `getProduitsConcernes()` : retourne la liste des produits concernés par la réduction.
  — `getNom()` : retourne le nom du commerce.

- # Classe : Contrat

- **Attributs de la classe**

  — `idContrat` : identifiant unique du contrat.
  — `dateDebut` : date de début du contrat.
  — `dateFin` : date de fin du contrat.
  — `CategoriesConcernees` : catégories de produits concernées par le contrat.

- **Méthodes de la classe**

  — `definirRegleUtilisation()` : définit les règles d'utilisation des réductions.
  — `estActif()` : vérifie si le contrat est toujours valide.
  — `renouvelerContrat()` : prolonge la durée du contrat.
  — `getDateDebut()` : retourne la date de début.
  — `getDateFin()` : retourne la date de fin.

- # Classe : Produit

- **Attributs de la classe**

  — `idProduit` : identifiant unique du produit.
  — `nom` : nom du produit.
  — `prix` : prix unitaire du produit.
  — `categorieProduit` : catégorie à laquelle appartient le produit.

- **Méthodes de la classe**

  — `getNom()` : retourne le nom du produit.
  — `getPrix()` : retourne le prix du produit.
  — `getCategories()` : retourne la catégorie associée.
  — `setPrix()` : modifie le prix du produit.

- # Classe : CategorieProduit

- **Attributs de la classe**

  — `idCategorie` : identifiant unique de la catégorie.
  — `nom` : nom de la catégorie.
  — `tauxReduction` : pourcentage de réduction applicable.
  — `produits` : liste des produits appartenant à cette catégorie.
  — `pointNecessaire` : nombre de points requis pour activer la réduction.

- **Méthodes de la classe**

  — `getNom()` : retourne le nom de la catégorie.
  — `getTauxReduction()` : retourne le taux de réduction.
  — `verifierReduction()` : vérifie si la réduction peut être appliquée.
  — `appliquerReduction()` : applique la réduction au produit.
  — `setTauxReduction()` : modifie le taux de réduction.

- ## Classe : Statistique

- **Attributs de la classe**

  — `historiqueDepot` : enregistrement des dépôts réalisés, servant de base aux statistiques.

- **Méthodes de la classe**

  — `calculerTotalDechets()` : calcule la quantité totale de déchets déposés.
  — `productionMoyenne()` : calcule la moyenne de production de déchets par période ou utilisateur.

3. Description des associations

4. Description des associations

- ## Association : Compte – Produit (`+Echanger`)

- **Type d'association**

Association. Cette relation permet de modéliser l'échange de points fidélité contre des produits.

- **Cardinalité**

  — Un `Compte` peut échanger ses points contre `0..*` `Produit`s.
  — Un `Produit` peut être obtenu via l'échange par `0..*` comptes.

- ## Association : Compte – Depot (`+Deposer`)

- **Type d'association**

Association. Modélise le fait qu'un utilisateur dépose des déchets via un compte.

- **Cardinalité**

  — Un `Compte` peut effectuer `0..*` `Depot`s.
  — Un `Depot` est toujours lié à `1` seul `Compte`.

- ## Association : Depot – Poubelle (`+Dans`)

- **Type d'association**

Association. Chaque dépôt est effectué dans une poubelle spécifique.

- **Cardinalité**

  — Un `Depot` est lié à `1` `Poubelle`.
  — Une `Poubelle` peut contenir plusieurs `Depot`s.

- **Association : Depot – Statistique (+Générer)**

- **Type d'association**

Association. Cette relation indique qu'un dépôt contribue à la génération de statistiques.

- **Cardinalité**

  — Un `Depot` est pris en compte dans `1 Statistique`.
  — Une `Statistique` agrège plusieurs `Depots`.

- **Association : CentreDeTri – Poubelle (+Gérer)**

- **Type d'association**

Composition. Un centre de tri gère plusieurs poubelles. Ces dernières dépendent directement de lui.

- **Cardinalité**

  — Un `CentreDeTri` gère `0..* Poubelles`.
  — Chaque `Poubelle` est liée à `1 CentreDeTri`.

- **Association : CentreDeTri – Statistique (+S'occuper)**

- **Type d'association**

Association. Un centre est responsable des statistiques liées à son activité.

- **Cardinalité**

  — Un `CentreDeTri` est associé à `1` ou plusieurs `Statistique`.
  — Une `Statistique` appartient à un seul `CentreDeTri`.

- **Association : CentreDeTri – Contrat (+Etablir)**

- **Type d'association**

Agrégation. Le centre de tri établit plusieurs contrats avec des commerces.

- **Cardinalité**

  — Un `CentreDeTri` peut établir `0..* Contrats`.
  — Un `Contrat` est toujours lié à `1 CentreDeTri`.

- **Association : Contrat – Commerce**

- **Type d'association**

Association. Chaque contrat est conclu avec un commerce partenaire.

- **Cardinalité**

  — Un `Contrat` concerne `1 Commerce`.
  — Un `Commerce` peut avoir plusieurs `Contrats`.

• Association : Commerce – CategorieProduit (`+Souihaiter`)

- **Type d'association**

Association. Les commerces souhaitent des catégories de produit.

- **Cardinalité**

  — Un `Commerce` propose `0..* CategorieProduits`.
  — Une `CategorieProduit` est proposé par `1..* Commerce`.

• Association : Produit – CategorieProduit (`+Concerner`)

- **Type d'association**

Association. Cette relation permet de catégoriser les produits (par type ou par offre).

- **Cardinalité**

  — Un `Produit` peut appartenir à `1 CategorieProduits`.
  — Une `CategorieProduit` regroupe plusieurs `Produits`.

5. Conclusion

• Bilan de ce premier rendu

Ce premier rendu nous a permis de structurer notre vision du projet. L'élaboration de l'UML nous a permis d'identifier les classes principales, leurs attributs, leurs méthodes, ainsi que les relations entre elles. Ce travail nous a aidés à mieux visualiser le fonctionnement global du système et à poser les premières bases du développement.

Petit à petit, le projet a commencé à prendre forme. On a compris comment les utilisateurs allaient interagir avec le système, comment les dépôts de déchets seraient enregistrés, ou encore comment les points gagnés pourraient être utilisés pour obtenir des produits.

Certains liens entre les classes n'étaient pas évidents à représenter dès le départ, alors on a dû revenir plusieurs fois sur notre schéma, discuter ensemble, tester différentes idées que ce soit pour les attributs, méthodes de classes ou les associations. Les questions les plus débattues furent celles concernant les types d'associations et leurs cardinalités si celles-ci étaient nécessaires. C'est pourquoi on a tenu à définir ces dernières précisément dans le rapport pour expliquer la réflexion finale derrière.

• Suite et perspectives du projet

La prochaine étape sera de passer à l'implémentation des classes en Java, en suivant le schéma qu'on a défini avec l'UML. On commencera à coder petit à petit, en ajoutant des tests unitaires au fur et à mesure pour s'assurer que tout fonctionne correctement. L'objectif, c'est d'avoir à la fin un projet solide, clair et qui répond à la demande du sujet.

6. Annexe

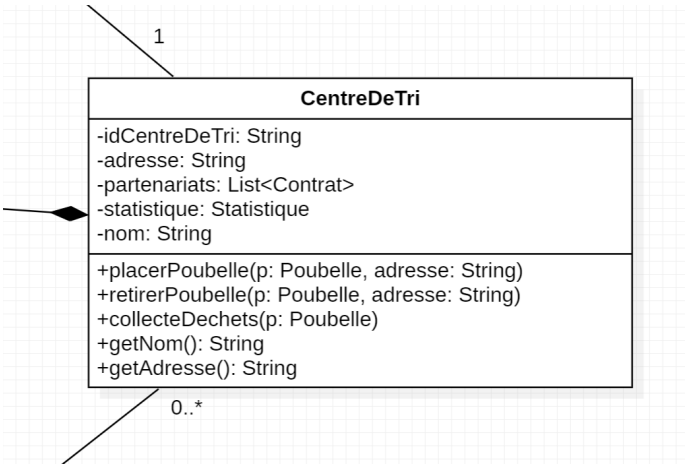

FIGURE 1 – Classe CentreDeTri.

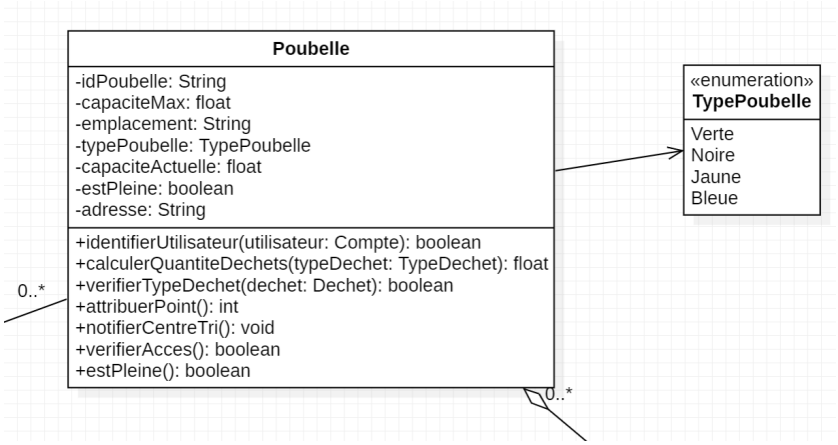

FIGURE 2 – Classe Poubelle.

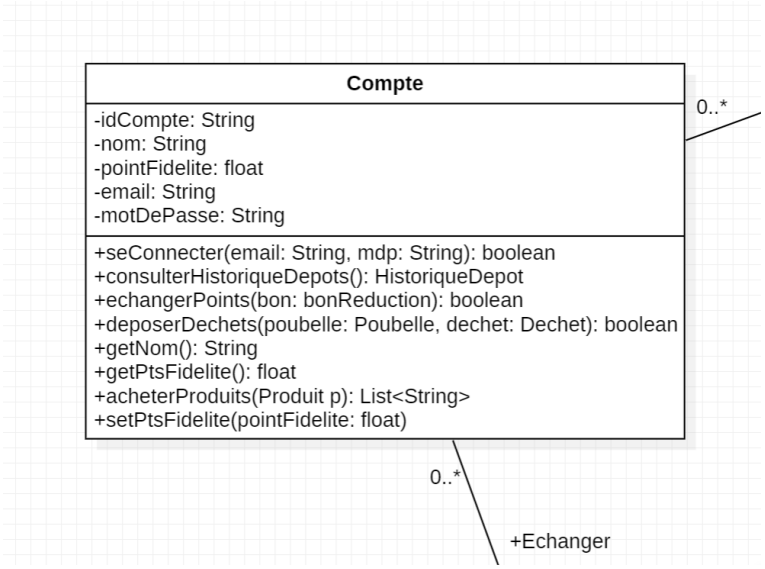

FIGURE 3 – Classe Compte.

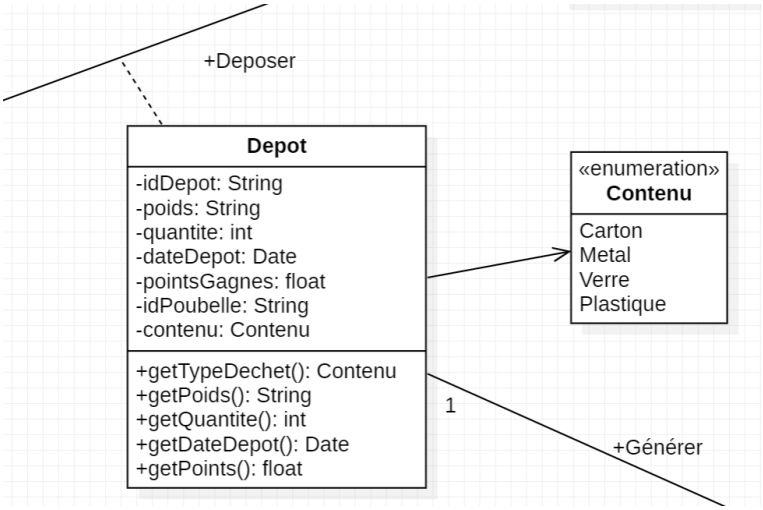

FIGURE 4 – Classe Depot.

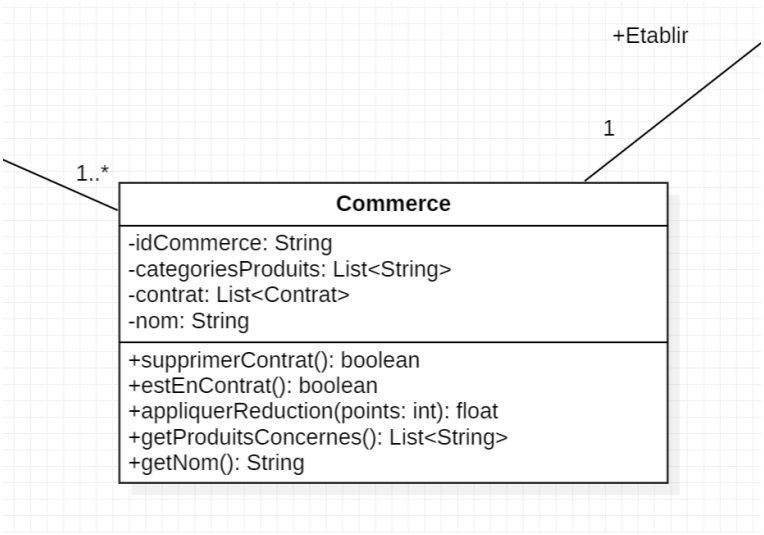

FIGURE 5 – Classe Commerce.

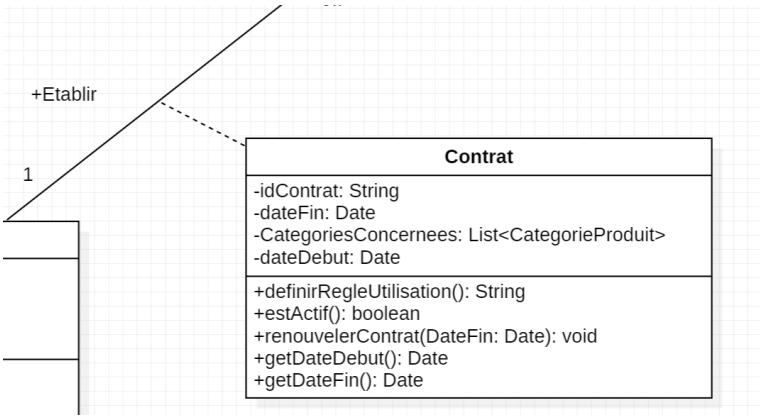

FIGURE 6 – Classe Contrat.

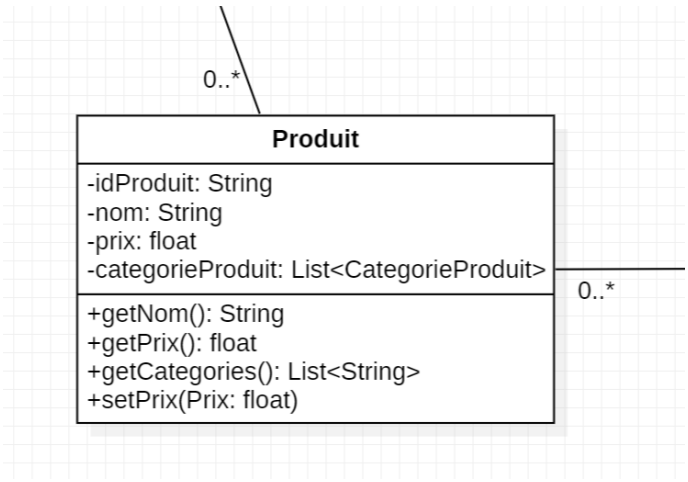

FIGURE 7 – Classe Produit.

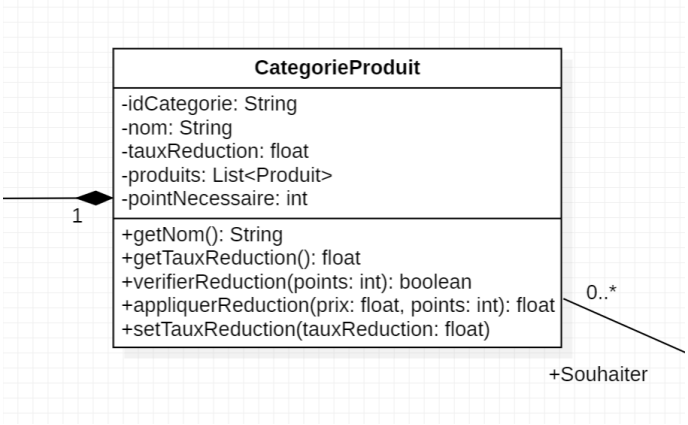

FIGURE 8 – Classe CategorieProduit.

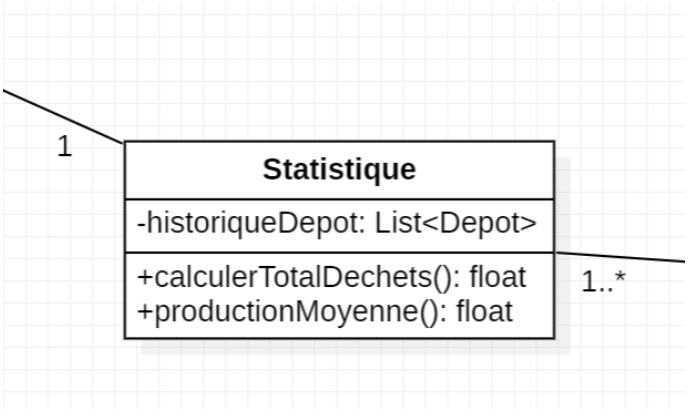

FIGURE 9 – Classe Statistique.

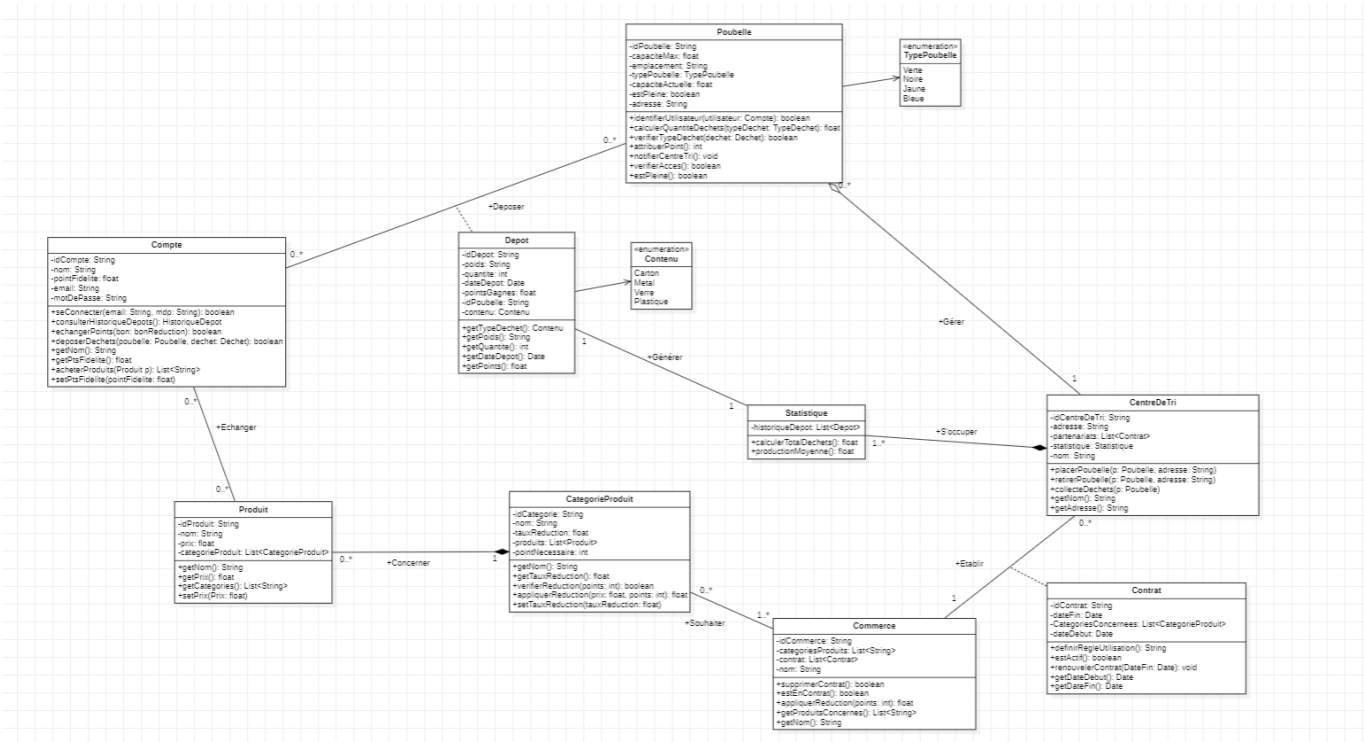

Figure 10 – UML.