CY Tech - 2024/2025

Analyse et programmation orienté objet

- Java -

Rendu 2

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.
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 2

Pour ce deuxième rendu il s'agissait cette fois de la transformer en véritable application Java, en mode console.

Nous avons tout d'abord adapté notre UML pour plus de coherence puis il s'agissait d'implémenter en Java l'ensemble des classes définies dans notre modélisation UML, de mettre en place une base de données MySQL, et de réaliser la connexion entre les deux grâce aux classes DAO.

2. Organisation du Projet

Le projet est organisé en plusieurs packages pour une meilleure structuration du code :

• Package `classe`

Ce package contient toutes les classes qui correspondent directement aux entités qu'on a pu définir dans le diagramme UML. Il sert à définir les attributs et les méthodes de chaque entité tout en prenant en compte les relations entre eux.

• Package `DAO`

Ce package rassemble toutes les classes DAO (Data Access Object) qui servent de pont entre les objets Java et la base de données MySQL `tri_selectif` . On y implémente les

opérations CRUD (Create, Read, Update, Delete) pour chaque entité avec chaque classe qui est dédiée à une table de la base.

- **Package `classeMain`**

Ce package contient les classes principales utilisées pour lancer des tests globaux sur toutes les classes avec et sans DAO.

- **Package `classeTest`**

Ce package contient au moins un test spécifique pour chaque classe métier afin de valider leur comportement logique mais également un test DAO pour chaque classe qui s'assure que les opérations de chaque entité sur la base de données MySQL se passent bien. Ceci permet donc de tester séparément : la logique métier sans dépendre de la base de données et la logique en utilisant la base de données via les DAO.

3. Classes Implémentées

Classe BonReduction

Responsabilité : Représenter un bon échangeable contre une réduction pour un produit donné, lié à une catégorie de produit.

Méthodes principales : - `utiliserBon()` : Marque le bon comme utilisé.

Test 1 et 2 : Vérification de la création d'un bon sans catégorie / avec catégorie liée. (Sortie : false)
Test 3 : Test de l'utilisation du bon `utiliserBon()`.(Sortie : true)

Classe CategorieProduit

Responsabilité : Regrouper des produits similaires sous une même catégorie.

Méthodes principales : - `verifierReduction(int points)` : Vérifie si un compte a assez de points pour bénéficier d'une réduction. - `appliquerReduction(float prix, int points)` : Applique une réduction sur un prix si les points sont suffisants.
- `ajouterProduit(Produit produit)` : Ajoute un produit à la liste de la catégorie.

Test 1 et 2 : Vérification que la réduction est accordée avec `verifierReduction()` si le nombre de points est suffisant / insuffisants. (Sortie true car 15>10 et false car 5<10)

Test 3 et 4 : Application d'une réduction avec `appliquerReduction()` si les conditions sont remplies / ne sont pas remplies. (Sortie 80 avec prix de 100 et réduction de 20% et 100)

Classe CentreDeTri

Responsabilité : Gérer l'ensemble des opérations autour des poubelles et des contrats avec les commerces.

Méthodes principales : - `placerPoubelle(Poubelle p, String adresse)` : Ajoute une poubelle à une adresse spécifique dans le centre de tri. - `retirerPoubelle(Poubelle p, String adresse)` : Retire une poubelle placée à une adresse précise. Si aucune poubelle ne reste à l'adresse, la suppression de l'adresse est effectuée. - `collecteDechets(Poubelle p)` : Vide le contenu de la poubelle (déclenche la méthode vider de la poubelle). - `ajouterPartenariat(Contrat contrat)` : Enregistre un contrat de partenariat entre le centre de tri et un commerce.

Test 1 : Appel de `placerPoubelle()` pour placer deux poubelles à la même adresse. (Sortie : Poubelle placée à l'adresse + adresse)

Test 2 : Appel de `retirerPoubelle()` pour retirer une poubelle de l'adresse. (Sortie : Poubelle retirée de l'adresse + adresse)

Test 3 : Appel de `collecteDechets()` pour vider une poubelle restante. (Sortie : Poubelle vidée / Déchets collectés pour la poubelle : 102)

Classe Commerce

Responsabilité : Représenter un commerce pouvant proposer différentes catégories de produits et établir des contrats avec des centres de tri.

Méthodes principales : - `supprimerContrat()` : Supprime le dernier contrat lié au commerce. - `estEnContrat()` : Vérifie si un contrat actif est associé au commerce. - `appliquerReduction(int points)` : Cherche le meilleur taux de réduction applicable en fonction des contrats et du nombre de points de fidélité. - `ajouterContrat(Contrat contrat)` : Ajoute un nouveau contrat au commerce. - `ajouterCategorieProduit(String categorie)` : Ajoute une nouvelle catégorie de produits proposée par le commerce.

Test 1 : Appel de `estEnContrat()` pour vérifier si un commerce possède un contrat actif. (Sortie : true)

Test 2 : Appel de `appliquerReduction()` pour vérifier si le commerce applique correctement une réduction selon 8 points de fidélité. (Sortie : 0.10 car moins de 10 points = catégorie "Electromenager").

Test 3 : Appel de `appliquerReduction()` pour vérifier l'application de la meilleure réduction disponible avec 12 points. (Sortie 0.20 car plus de 10 points = catégorie "High-Tech").

Test 4 : Appel de `supprimerContrat()` pour supprimer le contrat existant. (Sortie : true)

Test 5 : Appel de `estEnContrat()` après suppression pour vérifier qu'aucun contrat n'est actif. (Sortie : false)

Classe Compte

Responsabilité : Représenter un compte utilisateur (ménage) pouvant déposer des déchets, utiliser des bons de réduction et acheter des produits.

Méthodes principales : - `seConnecter(String email, String mdp)` : Vérifie l'identité du compte via l'email et le mot de passe. - `deposerDechets(Poubelle poubelle, Dechet dechet)` : Permet au compte de déposer un déchet dans une poubelle autorisée. - `echangerPoints(BonReduction bon)` : Échange des points contre un bon de réduction.

- `acheterProduits(Produit p)` : Permet l'achat d'un produit en utilisant les points de fidélité. - `consulterHistoriqueDepots()` : Retourne l'historique des dépôts réalisés.

Test 1 et 2 : Appel de `seConnecter()` avec les bonnes informations / avec un mauvais mot de passe. (Sortie : true et false)

Test 3 : Appel de `deposerDechets()` pour déposer un déchet plastique dans une poubelle autorisée. (Sortie : true + Points de fidélité après depot : 12.5)

Test 4 : Appel de `echangerPoints()` pour échanger un bon contre des points de fidélité. (Sortie : true + Points restants : 7.5)

Test 5 : Appel de `acheterProduits()` pour acheter un produit avec le compte. (Sortie : Produit acheté : Écocup + Points restants : 7.5)

Classe Contrat

Responsabilité : Représenter un partenariat entre un CentreDeTri et un Commerce.

Méthodes principales : - `definirRegleUtilisation()` : Génère une description textuelle du contrat. - `getCategoriesProduits()` : Affiche toutes les catégories de produits concernées par le contrat. - `getTaux()` : Affiche le taux de conversion associé au contrat. - `estActif()` : Vérifie si le contrat est actuellement actif en fonction des dates. - `renouvelerContrat(Date nouvelleDateFin)` : Modifie la date de fin du contrat pour le prolonger. - `ajouterCategorie(CategorieProduit categorie)` : Ajoute une nouvelle catégorie de produit concernée par le contrat.

Test 1 : Appel de `definirRegleUtilisation()` pour générer la description du contrat.

Test 2 : Appel de `getCategoriesProduits()` pour afficher la liste des catégories concernées par le contrat. (Sortie : Électroménager + High-tech)

Test 3 : Appel de `getTaux()` pour afficher le taux de conversion défini pour le contrat. (Sortie : 20%)

Test 4 : Appel de `estActif()` pour vérifier si le contrat est actif à la date actuelle. (Sortie : false a cause des dates)

Test 5 : Appel de `renouvelerContrat()` pour changer la date de fin du contrat.

Classe Dechet

Responsabilité : Représenter un déchet qui va être déposé dans une poubelle.

Méthodes principales : - `afficherDescription()` : Retourne une description textuelle du déchet (nom, contenu, masse).

Test 1 et 2 : Création d'un déchet avec `afficherDescription()` pour afficher les informations du déchet avec masse manquante / déchet complet. (Sortie : Bouteille,Plastique et masse par défaut 1 kg et Feuille A4, Papier, 0.025 kg)

Classe Depot

Responsabilité : Représenter un dépôt de déchet réalisé par un compte d'un ménage dans une poubelle.

Méthodes principales : - `afficherDescription()` : Retourne une description du dépôt (type de déchet, quantité, etc.). - `calculerValeurDepot()` : Calcule le nombre de points de fidélité gagnés en fonction du type de déchet et de la quantité déposée.

Test 1 et 2 : Création d'un dépôt avec un déchet plastique / second dépôt avec un déchet en verre puis appel de `afficherDescription()`.

Test 3 : Vérification des points de fidélité générés automatiquement.

Test 4 : Modification du déchet associé à un dépôt et recalcul des points (`calculerValeurDepot()`).

Classe HistoriqueDepot

Responsabilité : Regrouper et gérer l'ensemble des dépôts réalisés par un compte.

Méthodes principales : - `ajouterDepot(Depot depot)` : Ajoute un nouveau dépôt à l'historique. - `afficherHistorique()` : Affiche tous les dépôts enregistrés. - `totalDeposes()` : Calcule la quantité totale de déchets déposés. - `filtrerParContenu(Contenu contenu)` : Récupère uniquement les dépôts d'un certain type de contenu. - `moyenneDepot()` : Calcule la moyenne des quantités déposées par opération.

Test 1 : Appel de `afficherHistorique()` pour afficher tous les dépôts enregistrés.

Test 2 : Appel de `totalDeposes()` pour calculer la somme totale de la masse de tous les dépôts. (Sortie : 9.5 kg).

Test 3 : Appel de `moyenneDepot()` pour calculer la moyenne de la quantité déposée. (Sortie : 3.1667 kg.)

Test 4 : Appel de `filtrerParContenu(Contenu.PLASTIQUE)` pour ne récupérer que les dépôts en plastique.

Test 5 : Appel de `totalParContenu()` pour afficher la répartition de la quantité déposée par type de contenu. (Sortie : Verre : 2.5 kg / Plastique : 7.0 kg)

Classe Poubelle

Responsabilité : Représenter une poubelle connectée gérée par un centre de tri, capable de stocker des déchets.

Méthodes principales : - `identifierUtilisateur (Compte utilisateur)` : Verifie si l'utilisateur peut avoir acces a la poubelle. - `gererQuantiteDechet(Depot depot)` : Ajoute un dépôt de déchet et met à jour la capacité de la poubelle. - `verifierTypeDechet(Dechet dechet)` : Vérifie si le déchet correspond au type accepté par la poubelle. - `attribuerPoint()` : Calcule et attribue des points en fonction du dernier dépôt effectué. - `verifierPleine()` : Vérifie si la poubelle est pleine. - `notifierCentreTri()` : Envoie une notification si la poubelle est pleine. - `vider()` : Vide entièrement la poubelle.

Test 1 : Appel de `identifierUtilisateur()` pour vérifier si un utilisateur domicilié à la même adresse que la poubelle peut accéder. (Sortie : true)

Test 2 et 3 : Appel de `gererQuantiteDechet()` pour déposer un déchet plastique compatible avec la poubelle / déchet incompatible qui doit être rejeté. (Sortie : capacité restante 40.0 kg et true)

Test 4 : Appel de `attribuerPoint()` pour calculer les points de fidélité liés au dernier dépôt réalisé. (Sortie : 20)

Test 5 : La poubelle est détectée comme pleine (Sortie : true)

Test 6 : Appel de `notifierCentreTri()` pour simuler l'envoi d'une notification quand la poubelle est pleine. (Sortie : Notification envoyée au centre de tri : poubelle pleine + Poubelle vidée)

Classe Produit

Responsabilité : Représenter un produit pouvant appartenir à une ou plusieurs catégories de produit.

Méthodes principales : - `afficherDescription()` : Retourne une description complète du produit (nom, prix, date d'achat et catégories associées). - `ajouterCategorie(CategorieProduit categorie)` : Ajoute une catégorie au produit.

Test 1 et 2 : Création d'un produit sans catégorie associée / Ajout de deux catégories `ajouterCategorie()` au produit.

Classe Statistique

Responsabilité : Regrouper des informations statistiques sur les dépôts enregistrés par un centre de tri.

Méthodes principales : - `enregistrerDepot(Depot d)` : Ajoute un dépôt à l'historique et retourne sa quantité. - `calculerTotalDechets()` : Calcule le poids total de tous les dépôts. - `productionMoyenne()` : Calcule la moyenne de la quantité déposée par dépôt. - `afficherHistorique()` : Affiche une liste résumant tous les dépôts enregistrés.

Test 1 : Appel de `enregistrerDepot(Depot d)` pour enregistrer trois dépôts différents dans l'historique.

Test 2 : Appel de `calculerTotalDechets()` pour calculer la quantité totale déposée. (Sortie : 10.0 kg)

Test 3 : Appel de `productionMoyenne()` pour obtenir la moyenne de la quantité par dépôt. (Sortie : 3.3333 kg)

Test 4 : Appel de `afficherHistorique()` pour afficher la liste de tous les dépôts enregistrés, chacun avec sa quantité et son contenu associé.

Énumérations

- **Contenu** : Représente les types de déchets (VERRE, BIODECHET, METAUX, PAPIER, PLASTIQUE).
- **TypePoubelle** : Représente les types de poubelles.

4. Gestion de la base de données et DAO

Le projet repose sur une base de données MySQL appelée `tri_selectif`. Les différentes entités du projet sont liées à des tables dans cette base. Chaque entité dispose d'une classe

DAO dédiée, chargée d'assurer les opérations grâce à JDBC. L'ensemble des tests de ces DAO est centralisé dans la classe `MainDAOTest` et dans `ClasseTestDAO`, qui lancent des tests globaux pour vérifier l'insertion, la modification, la suppression et la récupération d'objets.

5. Création de la base de données

Afin de connecter notre application aux données, nous avons créé manuellement la base de données `tri_selectif` ainsi que l'ensemble des tables nécessaires. Chaque table correspond directement aux entités du projet, en respectant les relations prévues dans notre modélisation.

6. Conclusion

• Bilan de ce deuxième rendu

Ce second rendu a été beaucoup plus concret et technique que le premier. Après avoir posé les bases avec la modélisation UML, nous sommes cette fois rentrés dans la mise en œuvre réelle du projet, en développant en Java, et en ajoutant l'accès à la base de données MySQL.

Pour s'organiser nous avons séparé notre travail en deux parties : dans un premier temps, nous avons testé toutes nos classes sans base de données, uniquement en Java, pour vérifier que ça fonctionnait bien. Et ensuite nous avons ajouté la partie DAO pour connecter notre application à MySQL.

Nous avons utiliser Eclipse pour le développement Java, et MySQL Workbench pour la base de données. Travailler ensemble n'a pas toujours été simple : contrairement à d'autres projets où on peut utiliser des plateformes de partage comme GitHub, ici nous avons dû échanger nos fichiers manuellement, ce qui a parfois ralenti nos avancements. De plus, chaque membre du groupe travaillait sur sa propre base de données en local, ce qui nous a obligés à rester attentifs pour garder une cohérence entre toutes nos versions. À cela s'est ajoutée une autre difficulté : lorsque nous avons commencé à vraiment coder, nous nous sommes rendu compte que certaines parties de notre UML initial n'étaient pas correctes. Nous avons donc dû le reprendre et l'adapter pour qu'il corresponde à notre projet.

Malgré toutes ces difficultés, ce projet nous a permis de progresser, notamment sur l'organisation du code en packages, l'utilisation de bases de données en Java et la gestion d'une logique métier complexe.

• Suite et perspectives du projet

Pour la suite, nous allons devoir nous concentrer sur le développement de l'interface graphique (IHM). Cette dernière étape visera à rendre l'application utilisable par les utilisateurs, en leur permettant de réaliser toutes les actions prévues.

7. Annexe

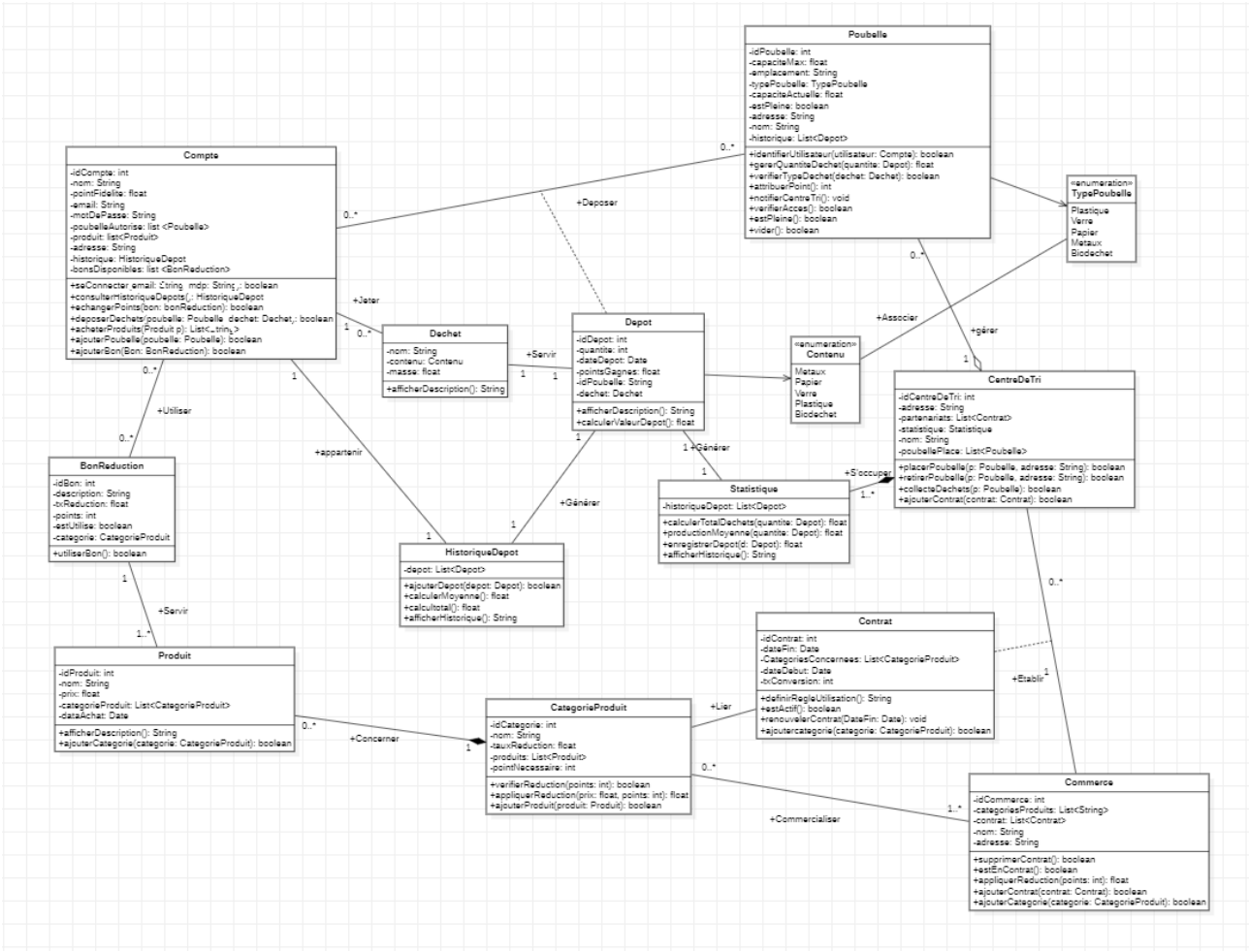

Figure 1 – UML.