**PROJET QUME 2024 — Distributeur Automatique de Boissons :**

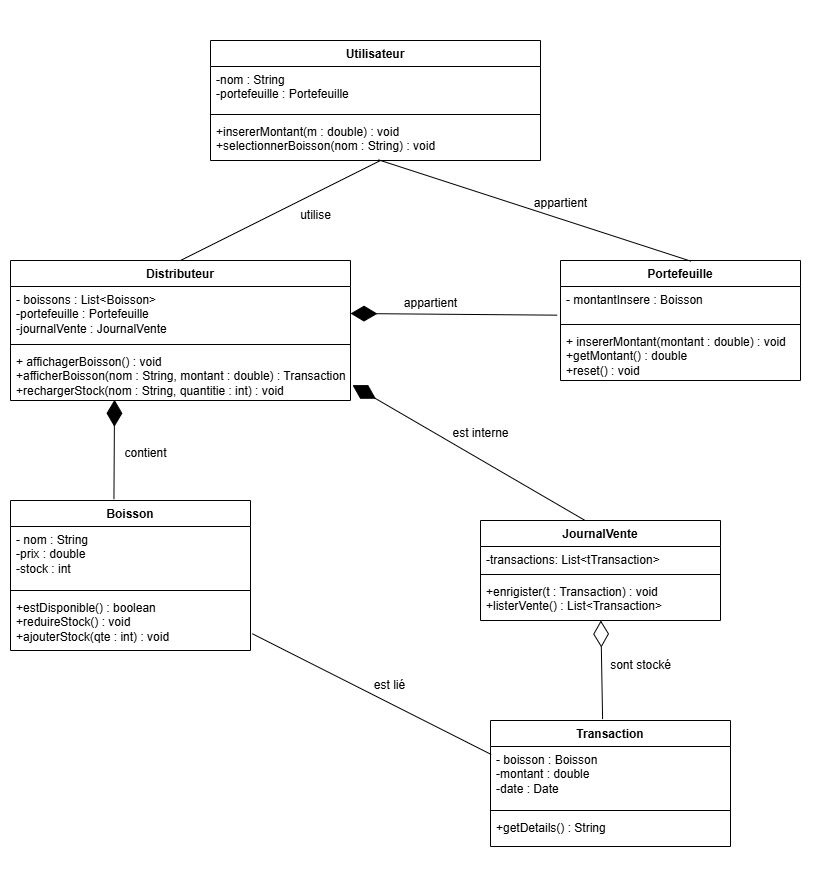
Projet réalisé dans le cadre de l’UE : Mesure de Qualité et Performance Logicielle  
Cycle : Licence 3 Informatique — Semestre 6

**Objet :** Livrable du projet — « Projet QUME 2024 »

📅 **Date de rendu :** 03/06/2024  
👨‍💻 **Encadrant :** Docteur Mansour Diouf

👥 **Membres du groupe :**

* Mohamet Lamine Ba
* Mouhamadou Al Bachir Ba
  1. **Diagramme UML du distributeur automatique :**

****

Le diagramme représente les entités suivantes :

* **Distributeur** : point d'entrée principal du système
* **Boisson :** chaque produit avec son prix et son stock
* **Transaction :** historique des achats
* **JournalVente :** enregistre toutes les transactions
* **Portefeuille :** gestion des montants insérés
* **Utilisateur :** acteur simulant les achats
  1. **Structure du Projet :**

Projet\_QUME\_L3/

├── src/

└── main/

└──java/

├──org.example/

├──models/ contient les fichier.java des classes.

├──Boisson

├──Distributeur

├──JournalVente

├──Portefeuille

├──Transaction

├──Utilisateur

└── resources/

└── features/ ← Contient les fichiers .feature avec les scénarios en langage Gherkin

└── test/

├──java

├──runner/ ←Contient la classe qui lance les tests d’acception

├──steps/ ← Contient les StepDefinitions (logique des scénarios)

├──Juste en bas se trouve tous les fichiers des tests unitaires

**C . Liste avec plus de 30 tests unitaires proposés :**

Les tests suivants vérifient le bon fonctionnement du système :

**Classe BoissonTest :**

* + testCreationBoisson()
  + testEstDisponibleTrue()
  + testEstDisponibleFalse()
  + testReducStock()
  + testAjouterStock()
  + testPrixBoisson()

**Classe PortefeuilleTest :**

* + testInitialMontant()
  + testAAssezPourTrue()
  + testAAssezPourFalse()
  + testInsererMontant()
  + testReset()
  + testInsererMontantNegatif()

**Classe DistributeurTest :**

* testChercherBoissonExistante()
* testChercherBoissonInexistante()
* testAchatReussi()
* testAchatMontantInsuffisant()
* testAchatBoissonInexistante()
* testAchatBoissonRupture()
* testRechargeStock()
* testRechargeStockBoissonInconnue()

**Classe TransactionTest**:

* tesTransactionData()
* testTransactionDetailFormat()
  + testTransactionAvecMontantNegatif()
  + testTransactionHorodatageNonNull()
  + testTransactionBoissonNonNull()

**Classe** **JournalVenteTest :**

* testJournalVideAuDebut()
* testEnrigistrerTransaction()
* testListerTransactionsContientTransaction()

**Classe** **UtilisateurTest :**

* testCreationUtilisateur()
* testInsererMontantDansPortefeuille()
* testSelectionnerBoissonReussie()
* testSelectionnerBoissonEchecMontantInsuffisant()

**D. Liste des 15 scénarios en langage Gherkin :**

Scenario: 1 - Affichage des boissons disponibles

Given le distributeur contient les boissons suivantes

When l'utilisateur consulte la liste des boissons

Then il voit la boisson "Coca"

Scenario: 2 - Achat réussi d'une boisson

Given le distributeur contient une boisson "Fanta" à 400 FCFA avec 5 en stock

When l'utilisateur insère 500 FCFA et achète "Fanta"

Then l'achat est réussi et le stock diminue de 1

Scenario: 3 - Achat avec montant insuffisant

Given le distributeur contient une boisson "Fanta" à 400 FCFA avec 5 en stock

When l'utilisateur insère 100 FCFA et tente d'acheter "Fanta"

Then l'achat échoue avec message "Montant insuffisant."

Scenario: 4 - Achat d'une boisson en rupture de stock

Given le distributeur contient une boisson "Sprite" à 300 FCFA avec 0 en stock

When l'utilisateur insère 500 FCFA et tente d'acheter "Sprite"

Then l'achat échoue avec message "Boisson en rupture de stock."

Scenario: 5 - Recharge de stock

Given le stock de "Eau" est à 0

When le personnel recharge "Eau" avec 5 unités

Then le stock de "Eau" est maintenant à 5

Scenario: 6 - Achat d'une boisson inexistante

When l'utilisateur tente d'acheter "Thé"

Then l'achat échoue avec message "Boisson introuvable."

Scenario: 7 - Transaction enregistrée après achat

Given une boisson "Coca" est achetée

Then une transaction est enregistrée dans le journal

Scenario: 8 - Montant inséré visible dans le portefeuille

When l'utilisateur insère 1000 FCFA

Then le portefeuille contient 1000 FCFA

Scenario: 9 - Réinitialisation du portefeuille après achat

Given le portefeuille contient 500 FCFA

When un achat est effectué

Then le portefeuille est réinitialisé à 0

Scenario: 10 - Affichage du journal de vente

Given deux boissons sont achetées

When l'utilisateur consulte le journal des ventes

Then il voit 2 transactions enregistrées

Scenario: 11 - Recharge avec quantité négative

When le personnel tente de recharger "Coca" avec -5 unités

Then le stock ne change pas

Scenario: 12 - Création d’une nouvelle boisson

When le personnel ajoute une nouvelle boisson "Thé" à 300 FCFA

Then "Thé" apparaît dans la liste des boissons

Scenario: 13 - Tentative d'achat avec portefeuille vide

Given le portefeuille est vide

When l'utilisateur tente d'acheter "Fanta"

Then l'achat échoue avec message "Montant insuffisant."

Scenario: 14 - Recharge d’une boisson inexistante

When le personnel tente de recharger "Jus d’ananas"

Then le système affiche "Boisson non trouvée."

Scenario: 15 - Affichage formaté des boissons

Given le distributeur contient "Fanta" à 400 FCFA avec 3 en stock

When l'utilisateur consulte la liste

Then il voit "Fanta - 400 FCFA (3 en stock)"

**Conclusion**

Ce projet nous a permis de mettre en œuvre une architecture orientée objet, des tests unitaires robustes avec JUnit, et des tests d'acceptance via Cucumber.  
Il illustre l'importance de la validation continue et du découpage fonctionnel dans le développement logiciel.

**Technologies utilisées :**

* Java 17
* Maven
* JUnit 5
* Cucumber 7.14.0
* IntelliJ IDEA