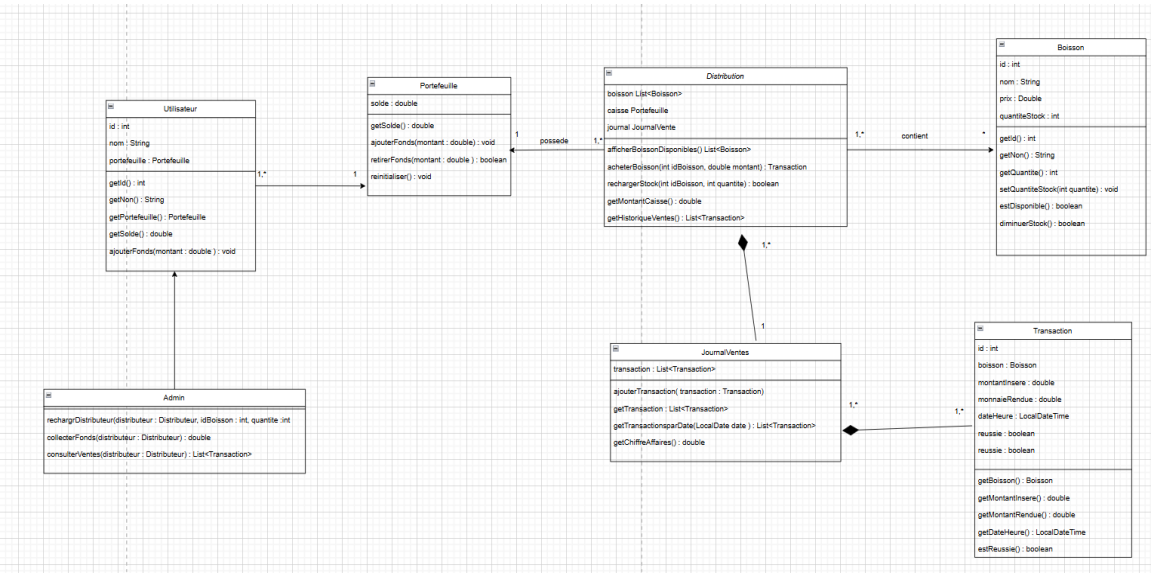
Projet : Geston d’un distributeur automatque de boissons

1. Diagramme de classes



Description des classes

--Distributeur

Cette classe représente le distributeur automatique de boissons. Elle gère la liste des boissons disponibles, les transactions d'achat et le rechargement du stock.

--Boisson

Cette classe représente une boisson disponible dans le distributeur. Elle contient les informations sur le nom, le prix et la quantité en stock de la boisson.

--Transaction

Cette classe représente une transaction d'achat. Elle enregistre les détails de la transaction, comme la boisson achetée, le montant inséré, la monnaie rendue et la date/heure de la transaction.

--Portefeuille

Cette classe gère les montants d'argent. Elle est utilisée à la fois pour la caisse du distributeur et pour le portefeuille de l'utilisateur.

--JournalVentes

Cette classe enregistre toutes les transactions effectuées par le distributeur. Elle permet de consulter l'historique des ventes et de calculer le chiffre d'affaires.

--Utilisateur

Cette classe représente un utilisateur du distributeur. Elle contient les informations sur l'utilisateur et son portefeuille.

--classe Admin

Cette classe représente un administrateur du système. Elle hérite de la classe Utilisateur et ajoute des fonctionnalités spécifiques comme le rechargement du distributeur et la collecte des fonds.

2. Tests unitaires

Tests Unitaires

1. **testAfficherListeBoissonsNonVide** : affiche la liste des boissons lorsque le stock contient au moins un élément.
2. **testAchatBoissonAvecMontantSuffisant** : achat réussi quand le montant inséré ≥ prix (quantité diminue, monnaie rendue correcte).
3. **testAchatBoissonExactMontant** : achat réussi avec montant exactement égal au prix (quantité diminue, monnaie = 0).
4. **testAchatBoissonMontantInsuffisant** : échec d'achat si le montant inséré < prix (quantité inchangée, exception MontantInsuffisantException).
5. **testAchatBoissonRuptureStock** : échec d'achat si la boisson est en rupture (quantité = 0, exception BoissonEnRuptureException).
6. **testAchatBoissonNonTrouvee** : échec d'achat si la boisson n'existe pas (exception BoissonInexistanteException).
7. **testRechargeStockBoissonExistante** : recharge d'une boisson existante augmente la quantité correctement.

8. **testRechargeStockBoissonNonExistante** : échec de recharge si la boisson n'existe pas (exception BoissonInexistanteException).
9. **testRechargeStockQuantiteNegative** : échec de recharge avec quantité négative (exception QuantiteInvalideException).
10. **testCalculerMonnaieRendueExact** : calcul correct de la monnaie rendue quand le surplus est important.
11. **testPortefeuilleMontantInitial** : portefeuille du distributeur initialisé à 0 FCFA.
12. **testPortefeuilleApresAchat** : portefeuille augmente du montant exact de la boisson vendue après achat.
13. **testTransactionJournalisee** : chaque achat réussi est enregistré dans le journal des ventes.
14. **testJournalVideAuDemarrage** : journal des ventes vide immédiatement après création du distributeur.
15. **testGetBoissonParNomCasNormal** : recherche d'une boisson par nom renvoie l'objet attendu.
16. **testGetBoissonParNomInexistant** : recherche d'un nom inexistant lève BoissonInexistanteException.
17. **testConstructeurBoisson** : constructeur de la classe Boisson initialise correctement nom, prix et quantité.
18. **testSetterQuantiteBoisson** : setQuantite(int) met à jour la quantité dans le cas nominal.
19. **testSetterQuantiteBoissonNegative** : setQuantite(-x) lève QuantiteInvalideException et n'impacte pas la quantité.
20. **testConstructeurTransaction** : constructeur de Transaction stocke correctement boisson, montant et date.
21. **testHistoriqueTransactionsOrdreChronologique** : le journal renvoie les transactions dans l'ordre chronologique.
22. **testAjoutMultipleBoissonsDifferentes** : ajout de boissons distinctes crée plusieurs entrées dans le stock.
23. **testAjoutBoissonDejaExistante** : ajout d'une boisson existante augmente uniquement sa quantité.
24. **testConstructeurPortefeuilleInvalidInitial** : création d'un Portefeuille avec un montant initial négatif lève MontantInvalideException.
25. **testAjouterMontantPortefeuilleValide** : ajouter un montant positif au portefeuille fonctionne correctement.
26. **testAjouterMontantPortefeuilleNegatif** : ajouter un montant négatif lève MontantInvalideException et n'affecte pas le solde.

27. **testRetirerMontantPortefeuilleValide** : retirer un montant quand le solde est suffisant diminue correctement le portefeuille.
28. **testRetirerMontantPortefeuilleInsuffisant** : retirer un montant supérieur au solde lève SoldeInsuffisantException.
29. **testToStringBoisson** : toString() de Boisson renvoie une chaîne formatée correcte (nom – prix – quantité).
30. **testToStringTransaction** : toString() de Transaction renvoie une chaîne formatée correcte (date – boisson – montant).

3. Tests d'acceptance

Scénario 1 : Achat réussi

L'utilisateur sélectionne une boisson existante, insère un montant ≥ prix, reçoit la boisson, la monnaie rendue correcte, et le journal est mis à jour.

Scénario 2 : Montant insuffisant

L'utilisateur insère un montant < prix pour une boisson en stock, l'achat est refusé, le stock reste inchangé, la monnaie est rendue et un message d'erreur s'affiche.

Scénario 3 : Rupture de stock

L'utilisateur tente d'acheter une boisson dont la quantité est à zéro, l'achat est immédiatement refusé, la somme est rendue et un message « produit en rupture » apparaît.

Scénario 4 : Boisson inexistante

L'utilisateur saisit le nom d'une boisson absente, insère de l'argent, l'achat est refusé, la monnaie est rendue et un message « produit non trouvé » s'affiche.

Scénario 5 : Recharge valide (admin)

L'administrateur authentifié sélectionne une boisson existante, ajoute une quantité positive, et le stock est mis à jour avec un message de confirmation.

Scénario 6 : Recharge boisson absente (admin)

L'administrateur tente de recharger une boisson inexistante, l'opération échoue, le stock reste inchangé et un message « produit non trouvé » apparaît.

Scénario 7 : Achat avec rendu de monnaie

L’utilisateur sélectionne une boisson, insère un montant > prix, reçoit la boisson, la monnaie rendue correcte, et le journal est mis à jour.

Scénario 8 : Consultation du journal (admin)

L’administrateur authentifié choisit « Afficher journal » et voit la liste chronologique des transactions (date, boisson, montant).

Scénario 9 : Achat multiple

L’utilisateur commande plusieurs unités d’une même boisson, insère le montant total ≥ (prix * quantité), et la transaction réussit avec déduction du stock, rendu de la monnaie et enregistrement au journal.

Scénario 10 : Montant négatif

L’utilisateur insère un montant négatif pour une boisson, l’achat est rejeté, rien ne change et un message « montant invalide » s’affiche.

Scénario 11 : Hors service

Le distributeur est en mode hors service, toute tentative d’achat est refusée, la monnaie est rendue et un message « hors service » apparaît.

Scénario 12 : Modifier prix valide (admin)

L’administrateur sélectionne une boisson, saisit un nouveau prix > 0, et le prix est mis à jour avec un message de confirmation.

Scénario 13 : Modifier prix négatif (admin)

L’administrateur saisit un prix < 0 pour une boisson, la modification est rejetée, le prix reste inchangé et un message « prix invalide » s’affiche.

Scénario 14 : Consultation du stock complet (admin)

L’administrateur choisit « Afficher stock » et voit pour chaque boisson son nom, son prix et sa quantité.

Scénario 15 : Réinitialiser distributeur (admin)

L’administrateur confirme « Réinitialiser », le stock et le journal sont vidés, et le message « Distributeur réinitialisé » apparaît.

MEMBRES :

Aly THIOBANE

Maimouna DIA

BIENVENU MENDY