Rapport TP2

Étapes du TP :

1. Conception de la hiérarchie de classes :

- **Tâche 1 : Créer une classe abstraite Transaction.**
- Attributs :
  - protected String transactionID : L'ID de la transaction.
  - protected String clientID : L'ID du client concerné par la transaction.
- Méthodes :
  - **Méthode abstraite public abstract void execute() :** à implémenter dans les sous-classes pour exécuter la transaction.
  - **Méthode non abstraite public void showDetails() :** Affiche les informations de base sur la transaction. Exemple de sortie : (Transaction ID: T001 | Client ID: C001)

```java
package TP2;

public abstract class Transaction {  no usages  2 inheritors
    protected String transactionID;  // ID de la transaction  3 usages
    protected String clientID;       // ID du client  3 usages

    // Constructeur pour initialiser les IDs
    public Transaction(String transactionID, String clientID) {  no usages
        this.transactionID = transactionID;
        this.clientID = clientID;
    }

    // Méthode abstraite que chaque sous-classe doit implémenter
    public abstract void execute();  no usages  2 implementations

    // Méthode pour afficher les détails de la transaction
    public void showDetails() {  no usages
        System.out.println("Transaction ID: " + transactionID + ",Client ID: " + clientID);
    }

    // Getters pour accéder aux attributs dans d'autres classes
    public String getTransactionID() {  no usages
        return transactionID;
    }

    public String getClientID() {  no usages
        return clientID;
    }
}
```

- **Tâche 2 : Créer les sous-classes de Transaction pour différents types de transactions.**
- **Sous-classe InsertTransaction :**

  - Implementez la méthode execute() pour afficher un message indiquant l'insertion d'un nouveau client.
  - Exemple de sortie : (Insertion d'un nouveau client avec ID: C001)

```java
package TP2;

// Sous-classe représentant l'insertion d'un nouveau client
public class InsertTransaction extends Transaction {

    public InsertTransaction(String transactionID, String clientID) {
        super(transactionID, clientID);
    }

    // Implémentation de la méthode execute() pour l'insertion

    public void execute() {
        System.out.println("Insertion d'un nouveau client avec ID: " + clientID);
    }
}
```

- **Sous-classe UpdateTransaction :**
  - Implementez la méthode execute() pour afficher un message indiquant la mise à jour des informations d'un client.
  - Exemple de sortie :Mise à jour des informations du client avec ID: C002

```java
package TP2;

public class UpdateTransaction extends Transaction {
    public UpdateTransaction(String transactionID, String clientID){
        super(transactionID, clientID);
    }
    public void execute() {
        System.out.println("Mise à jour des informations du client avec ID:" + clientID);
    }
}
```

- **Sous-classe DeleteTransaction :**
  - Implementez la méthode execute() pour afficher un message indiquant la suppression d'un client.
  - Exemple de sortie : Suppression du client avec ID: C003

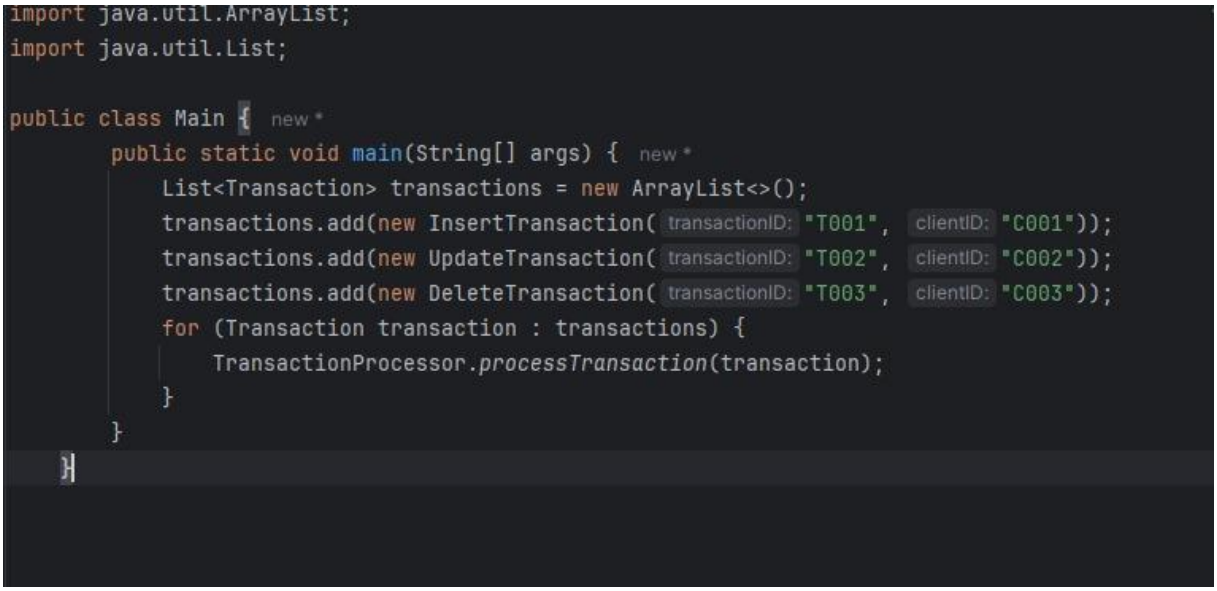
```java
package TP2;

/** Sous-classe représentant la suppression d'un client
public class DeleteTransaction extends Transaction {  no usages

    public DeleteTransaction(String transactionID, String clientID) {  no usages
        super(transactionID, clientID);
    }

    // Implémentation de la méthode execute() pour la suppression

    public void execute() {  no usages
        System.out.println("Suppression du client avec ID: " + clientID);
    }
}
```

2. Implémentation du polymorphisme :

- **Tâche 3 : Créer une méthode statique processTransaction(Transaction t) dans une classe TransactionProcessor.**
  - Cette méthode doit prendre en paramètre un objet de type Transaction et appeler sa méthode execute().
  - Cela permet d'utiliser le polymorphisme pour traiter de manière uniforme différents types de transactions.

- **Tâche 4 : Créer une méthode pour tester le polymorphisme.**
  - Créez une liste d'objets Transaction contenant différents types de transactions (InsertTransaction, UpdateTransaction, DeleteTransaction).
  - Utilisez une boucle pour parcourir la liste et appelez processTransaction() pour chaque transaction.
  - Exemple de sortie attendue :

Transaction ID: T001 | Client ID: C001 | Insertion d'un nouveau client.
Transaction ID: T002 | Client ID: C002 | Mise à jour des informations du client.
Transaction ID: T003 | Client ID: C003 | Suppression du client.

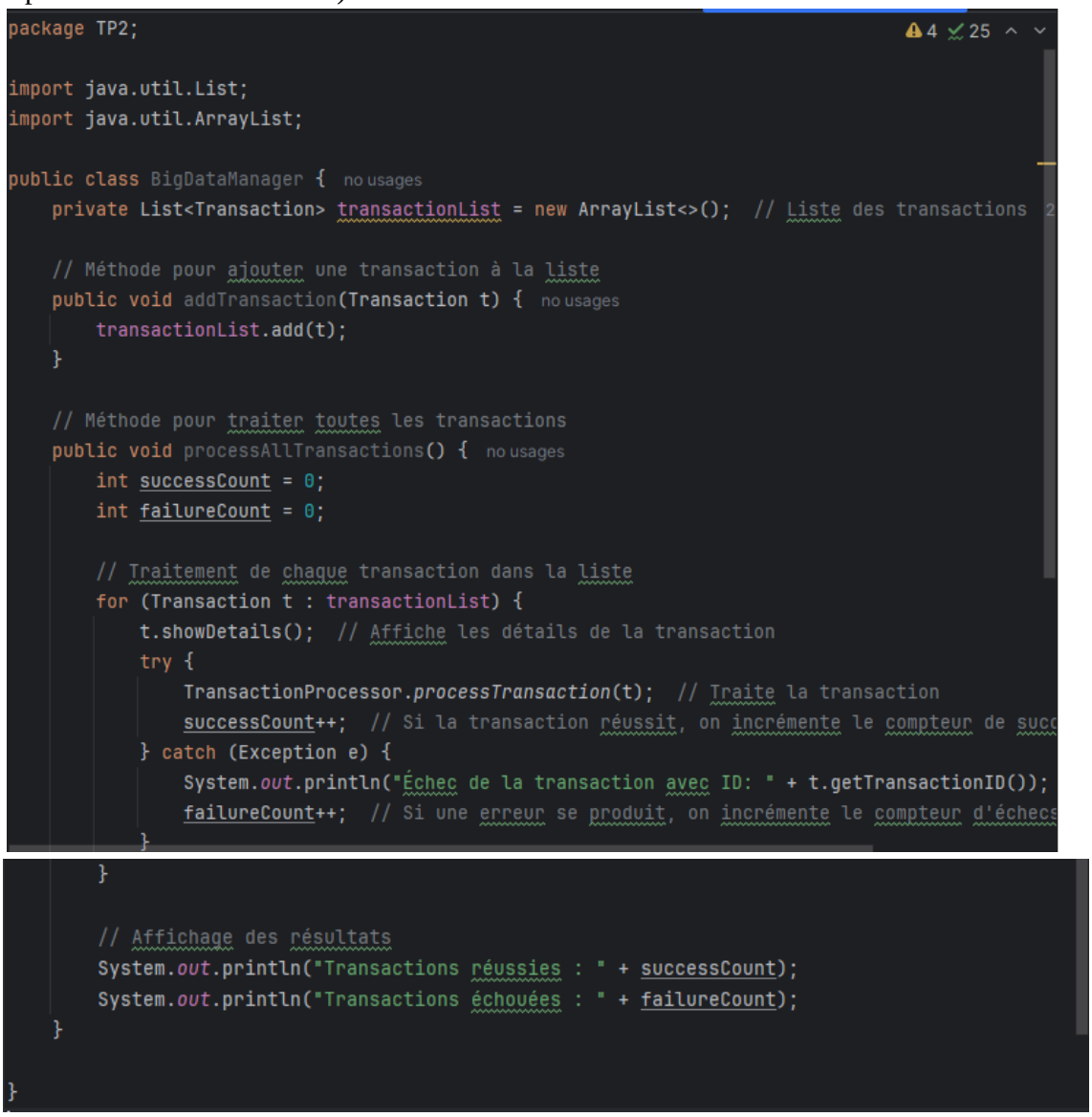


```java
import java.util.ArrayList;
import java.util.List;

public class Main {  new *
        public static void main(String[] args) {  new *
            List<Transaction> transactions = new ArrayList<>();
            transactions.add(new InsertTransaction( transactionID: "T001",  clientID: "C001"));
            transactions.add(new UpdateTransaction( transactionID: "T002",  clientID: "C002"));
            transactions.add(new DeleteTransaction( transactionID: "T003",  clientID: "C003"));
            for (Transaction transaction : transactions) {
                TransactionProcessor.processTransaction(transaction);
            }
        }
    }
```

3. Gestion des transactions à grande échelle :

- **Tâche 5 : Créer une classe BigDataManager.**
    - Cette classe doit contenir une liste de transactions et offrir des méthodes pour :
    - Ajouter des transactions à la liste.
    - Traiter toutes les transactions de la liste en utilisant TransactionProcessor.processTransaction().
    -

- Compter le nombre de transactions réussies et échouées (simulez des erreurs aléatoires pour certaines transactions).



```java
package TP2;

import java.util.List;
import java.util.ArrayList;

public class BigDataManager {
    private List<Transaction> transactionList = new ArrayList<>();  // Liste des transactions

    // Méthode pour ajouter une transaction à la liste
    public void addTransaction(Transaction t) {
        transactionList.add(t);
    }

    // Méthode pour traiter toutes les transactions
    public void processAllTransactions() {
        int successCount = 0;
        int failureCount = 0;

        // Traitement de chaque transaction dans la liste
        for (Transaction t : transactionList) {
            t.showDetails();  // Affiche les détails de la transaction
            try {
                TransactionProcessor.processTransaction(t);  // Traite la transaction
                successCount++;  // Si la transaction réussit, on incrémente le compteur de succ
            } catch (Exception e) {
                System.out.println("Échec de la transaction avec ID: " + t.getTransactionID());
                failureCount++;  // Si une erreur se produit, on incrémente le compteur d'échecs
            }
        }

        // Affichage des résultats
        System.out.println("Transactions réussies : " + successCount);
        System.out.println("Transactions échouées : " + failureCount);
    }

}
```