TP de PPO en JAVA

Polytech Lille GIS4

2018-2019

Objectifs

Ce sujet est à faire en 2 séances. Il s'agit de programmer une banque gérant des comptes et accessible par un guichet. Les objectifs sont :

- Partie I : tableaux polymorphes, réutilisation de code
- Partie II : exceptions

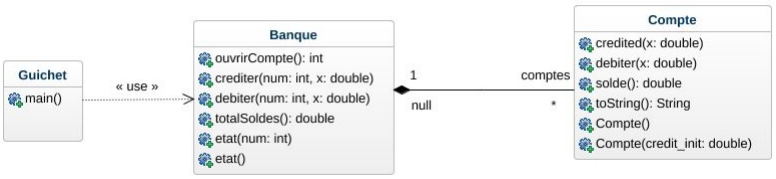
Lire une première fois le sujet en entier pour comprendre ses objectifs puis y répondre en suivant attentivement l'ordre des questions.

I Tableaux polymorphes, réutilisation de code

Consigne : ne pas tenir compte des exceptions pour dans cette 1ère partie.

I.1 Comptes

Une banque (classe `Banque`) gère des comptes tels que programmés dans un TP précédent (classe `Compte`), selon le schéma UML de principe suivant :



Pour simplifier les comptes gérés par une banque (association de rôle `comptes` entre `Banque` et `Compte`) sont rangés dans la classe `Banque` dans un tableau `comptes` surdimensionné de `Compte` (taille `MAX` = 1000 par exemple) muni d'un compteur (nombre de comptes créés) et on confond le numéro d'un compte avec son indice de rangement dans ce tableau.

Une banque offre les opérations suivantes :

- `int ouvrirCompte()` : crée un nouveau compte, l'ajoute en bout de la liste des comptes et renvoie son numéro
- `crediter(int num, double x)` : crédite le compte d'indice num du montant x
- `debiter(int num, double x)` : débite le compte d'indice i du montant x
- `totalSoldes()` : renvoie le total des soldes de tous les comptes
- `etat(int num)` : fournit l'état du compte d'indice num par appel à son `toString()`
- `etat()` : affiche l'état de tous les comptes de la banque avec leur numéro (indice), puis le total des soldes.

Pour les opérations dépendant d'un numéro de compte, elles doivent vérifier que celui-ci existe, sinon afficher un message d'erreur : ``compte inexistant''.

Copier le répertoire ~bcarre/public/tpBanque. Il contient un squelette de la classe Banque à programmer et une application complète (main) Guichet qui utilisent la classe Compte que vous avez programmée dans un TP précédent (récupérer cette classe). La classe Guichet crée une banque et offre un menu permettant à l'utilisateur (typiquement un agent de la banque) d'effectuer des opérations sur un compte. Programmer la classe Banque et tester en exécutant Guichet.

I.2 Comptes épargne

La banque gère maintenant aussi des CompteEpargne, sous-classe de Compte (récupérer cette classe).

Dans un premier temps, sans s'intéresser aux opérations spécifiques de ce nouveau type (interets() et echeance()), vérifier que l'application fonctionne toujours moyennant la possibilité de créer de tels comptes. Pour cela :

- ajouter dans la classe Banque une méthode int ouvrirCompteEpargne() qui crée un nouveau CompteEpargne, l'ajoute en fin de la liste de comptes et renvoie son numéro
- ajouter dans la classe d'application Guichet l'option correspondante dans le menu
- recompiler Banque.java et Guichet.java, tester en exécutant Guichet.

Observer en particulier la liaison dynamique sur les redéfinitions de toString() et debiter(double) quand vous sélectionnez un compte épargne.

I.3 Opérations spécifiques aux comptes épargne

Ajouter à la classe Banque les méthodes suivantes pour le traitement des opérations spécifiques sur les comptes épargnes :

- double interets(num) qui renvoie les intérêts du compte épargne numéro num
- echeance(num) qui échéance le compte épargne numéro num

On affichera un message d'erreur ``operation invalide'' si num ne correspond pas à un compte épargne. Rappel : l'expression x instanceof T permet de tester si x est de type dynamique T.

Modifier la classe Guichet pour offrir ces nouvelles options dans le menu et tester.

I.4 Virement

On ajoute la possibilité d'effectuer un virement d'un compte vers un autre :

- ajouter à la hiérarchie des classes de comptes une méthode void virerVers(double x, Compte dest) qui vire une somme x vers le compte dest. Attention au comportement spécifique des comptes épargne qui ne peuvent pas être débiteur.
- ajouter dans la classe Banque une méthode :
  void virement(int numSrc, int numDest, double x)
  qui permet de virer x euros du compte numéronumSrc vers le compte numDest
- ajouter cette option au menu de Guichet et tester.

II Exceptions

1. Dans la classe Banque, reconsidérer les cas d'erreur ``compte inexistant'' et ``operation invalide'' en provoquant les exceptions suivantes, plutôt que par affichage :
   - CompteInexistant : provoquée par les méthodes paramétrées par un numéro de compte qui n'existe pas.
   - OperationInvalide : provoquée lors d'opérations non applicables (I.3).
   Cela libère la classe Banque de tout affichage et la rend plus réutilisable (indépendante

de tout environnement d'entrée/sortie). Charge aux applications de traiter ces exceptions à leur façon : affichage sur un terminal texte par `System.out` dans notre cas, mais cela pourrait être une alerte graphique, ou tout autre traitement.

2. Dans l'application `Guichet` capturer ces exceptions et les traiter de la façon suivante :
   - afficher le message d'erreur
   - compter le nombre de tentatives d'opérations non valides.
3. Dans la classe `Banque`, remplacer les tests de type `(instanceof)` que vous avez dû utiliser à la question I.3 par des captures de l'exception `ClassCastException`. Tester avec `Guichet` (inchangé).
4. Dans la classe `Banque` ajouter une opération `echeance()` qui échéance tous les comptes épargnes (c'est la fin de l'année...). Filtrer ces comptes par capture de `ClassCastException`. Tester dans `Guichet.`

Complément

L'ensemble de ce code doit pouvoir fonctionner avec l'implémentation de base des comptes ou dans leur version avec historique des opérations (dernière question du TP 2).

Vérifier.