TP de PPO en JAVA

Polytech Lille GIS2A3

Créer un répertoire `tp4` puis un répertoire de travail de nom `banque`.

Copier le répertoire `~aetien/public/PPO/tp4/source`. Il contient un squelette d'une classe `Banque` et une application (`main`) `Guichet`.

Placer ces deux classes directement à la racine de votre répertoire `banque`.

1 Banque (version 1)

Une banque gère des comptes représentés par la classe `Compte` programmée lors d'un TP précédent. Compléter cette classe en ajoutant une méthode `void virerVers(double x, Compte dest)` qui permet d'effectuer un virement d'une somme `x` vers le compte `dest`.

Une banque est représentée par la classe `Banque.` Pour simplifier, la liste des comptes est rangée dans une `ArrayList` nommée `comptes`. Une variable d'instance `nbComptes` permet de connaître le nombre de comptes dans la banque. On confondra le numéro de compte avec son indice de rangement.

Programmer une première version de la classe `Banque` :

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

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

Classe d'application

La classe `Guichet` est une application qui crée une banque et offre un menu permettant à l'utilisateur (typiquement un agent de la banque) d'effectuer des opérations. L'affichage de l'état de la banque permet d'identifier les comptes pour les opérations où un numéro est requis. Testez cette application.

2 Banque (version 2)

La banque gère maintenant également des `CompteEpargne` (reprendre la classe programmée lors d'un TP précédent).

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 :

- ajouter dans la classe `Banque` une méthode `ouvrirCompteEpargne()` qui crée un nouveau `CompteEpargne` et l'ajoute dans la liste de comptes
- ajouter dans la classe d'application `Guichet` l'option correspondante dans le menu.

Observer en particulier la liaison dynamique sur les redéfinitions de `toString()` et `debiter(double)`.

3 Banque (version 3)

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

— `interets(i)` qui calcule les intérêts du compte i.

— `echeance(i)` qui échéance le compte i.

On affichera un message d'erreur `''operation non valide''` si `i` ne correspond pas à un compte rémunéré [1].

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

4 Exceptions

1. Dans la classe `Banque`, reconsidérer les cas d'erreur `''compte inexistant''` et `''operation non valide''` en provoquant les exceptions suivantes, plutôt que par affichage :
   - `CompteInexistantException` : provoquée par les méthodes paramétrées par un numéro de compte qui n'existe pas.
   - `OperationNonValideException` : provoquée lors d'opérations non applicables (cf. 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. Ainsi dans l'application `Guichet` (terminal texte) 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 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.`

5 Tester votre travail

Afin de savoir si votre code est de qualité et correspond à ce qui vous est demandé, il vous est possible de le tester. Pour cela, copier sur votre compte dans le répertoire `tp4` le répertoire `test` :

```
cp -r ~aetien/public/PPO/tp4/test .
```

Mettez vous dans votre répertoire `tp4/test` et :

— exécutez la commande suivante pour tester la première version de la banque : `./runTest.sh`

— Exécutez la commande suivante pour tester la deuxième version : `./runV2Test.sh`

— Exécutez la commande suivante pour tester la troisième version : `./runV3Test.sh`

Si votre code vérifie tous les critères de qualité évalués, vous aurez un message du genre `OK (24 tests)` pour la première version de la banque, `OK (29 tests)`, pour la deuxième version et `OK (33 tests)` pour la partie 3. Sinon, vous aurez des messages vous indiquant les erreurs qui peuvent exister dans votre code.

La version avec les exceptions ne peut pas être testée de cette façon.

1. Rappel : l'expression `x instanceof CompteEpargne` permet de tester si `x` est de type dynamique `CompteEpargne`.