METHODES DE CONCEPTION

TP 3

UNIVERSITÉ CAEN NORMANDIE

Cette séance de TP permet de faire des révisions sur la notion d'interface, et permet de faire un premier pas vers la réification d'actions (un pattern consistant à créer des objets qui font des actions plutôt que de faire directement ces actions, ce qui permet notamment de revenir en arrière).

Elle permet par ailleurs de mettre en œuvre les diagrammes UML vus en cours.

En revanche, pour des raisons de temps, la modélisation est déjà intégrée au sujet.

Exposé du problème :

Nous désirons créer une application permettant de gérer des comptes bancaires.

Un compte bancaire appartient à une et une seule banque. Une banque possède bien sûr un certain nombre de comptes bancaires.

On peut agir sur un compte bancaire en invoquant sur ce dernier une opération bancaire. Un exemple type d'opération bancaire est un ajout ou un retrait d'argent. Un autre exemple est l'application d'intérêts (dans le cas d'un compte d'épargne).

Nous voulons par ailleurs que la banque enregistre toutes les opérations effectuées sur ses comptes, et puisse, à la demande, annuler toutes les dernières opérations jusqu'à une date donnée.

1. Travail préliminaire : gestion des dates

Créer la classe Date, qui permet donc de dater une opération bancaire.

Elle utilise trois attributs, respectivement l'année, le mois et le jour du mois.

Par ailleurs, elle doit permettre de comparer entre elles deux dates, pour savoir si l'une est antérieure à l'autre (et laquelle), ou si elles correspondent exactement au même jour. Pour ce faire, lui faire implémenter l'interface Comparable<T> (qui existe déjà en Java). Celle-ci déclare une seule méthode :

public int compareTo(T o)

La valeur de retour de instanceA.compareTo(instanceB) doit être de 1 si instanceA est supérieure à instanceB, -1 dans le cas inverse, et enfin 0 en cas d'égalité. Dans notre cas, cette relation d'ordre correspondra à la postériorité temporelle (par ex. 1 si instanceA est une date ultérieure à instanceB, etc.).

La tester sur quelques cas, par exemple :

```
System.out.println(new Date(24,9,2018).compareTo(new Date(31,8,2018))==1); //true
System.out.println(new Date(4,9,2018).compareTo(new Date(1,7,2019)) == -1); // true
```

2. Modélisation, Diagramme de classes

Faire un diagramme de classes intégrant les types suivants :

La classe **Banque** possède deux attributs : une liste de comptes, et une liste d'opérations. Son constructeur par défaut crée les listes vides.

La classe **CompteBancaire** possède trois attributs : un identifiant, une banque et un solde.

L'interface **OperationBancaire** délare 4 méthodes :

```
void setCompteBancaire(CompteBancaire cb);
Date getDate();
void appliquer();
void desappliquer();
```

Les classes **Ajout** et **Retrait** implémentent OperationBancaire, et permettent respectivement de créditer ou de débiter un compte. Afin d'éviter toute redondance de code, ces deux classes étant très proches, nous les ferons hériter de la classe **AjoutOuRetrait**.

3. Développement

La liste d'opérations permet d'enregistrer toutes les opérations (de tous les comptes) depuis la création de la banque. En conséquence, un compte doit « prévenir » sa banque lorsqu'on lui applique une OperationBancaire.

Cette liste d'opérations est une véritable mémoire exhaustive de la banque, et permet notamment de revenir dans le temps en annulant toutes les opérations de la dernière en date jusqu'à une date quelconque. Ce sera l'objet d'une méthode annuleOperationsJusquau(Date d). Son fonctionnement est assez simple : partir de la dernière opération vers la première. Dès qu'une opération possède une date antérieure à la date limite de remontée dans le temps (la date d passée en paramètre), sortir de la méthode. Dans le cas contraire, appliquer la méthode desappliquer() de cette opération.

4. main() et test...

On testera le programme avec le main() suivant :

```
Banque b=new Banque();
CompteBancaire c1=b.creationCompte("129023434");
CompteBancaire c2=b.creationCompte("543458349");
c1.operation(new Ajout(500, new Date(01,01,2017)));
c2.operation(new Ajout(750, new Date(02,03,2017)));
c1.operation(new Retrait(32, new Date(05,03,2017)));
c1.operation(new Ajout(33, new Date(15,01,2018)));
b.annuleOperationsJusquau(new Date(04,03,2017));
```

La trace d'exécution correspondante est :

```
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=500.0
Compte n°1 Id=543458349 Solde=0.0
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=500.0
Compte n°1 Id=543458349 Solde=750.0
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=468.0
Compte n°1 Id=543458349 Solde=750.0
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=501.0
Compte n°1 Id=543458349 Solde=750.0
annulation de l'operation n°3
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=468.0
Compte n°1 Id=543458349 Solde=750.0
annulation de l'operation n°2
========== Descriptif des comptes ===========
Compte n°0 Id=129023434 Solde=500.0
Compte n°1 Id=543458349 Solde=750.0
```