Programmation Objet

Examen

3 heures - documents écrits autorisés
jeudi 30 janvier 2003

- il est possible d'utiliser les réponses à une question non traitée pour résoudre les autres questions.

On s'intéresse à la gestion d'une banque, de ses comptes et de ses clients.

Exercice 1 : Dates

Une date est définie par un numéro de jour (ou quantième) : un entier, un mois de type `Mois` et une année : un entier. Une fois une date définie, ces trois propriétés ne doivent plus pouvoir varier pour cette date.

Q 1 . La classe `Mois` (voir code donné en annexe).

Q 1.1. Expliquez la fonction et le comportement de l'attribut `cpt` de la classe `Mois`.

Q 1.2. Donnez le code de la méthode `compareTo` de cette classe.

Q 1.3. Pourquoi n'est il pas nécessaire de définir une méthode `equals(Object o)` pour cette classe ?

Q 2 . Définissez les attributs de la classe `Date` ainsi que son constructeur initialisant ces attributs. Cette classe sera définie dans un paquetage `util.date`.

Q 3 . Codez une méthode privée `isBissextile` qui indique si l'année de la date est ou non bissextile. (rappel : c'est le cas si `annee` est divisible par 400, ou, par 4 mais pas par 100).

Q 4 . Donnez le code d'une méthode de la classe `Date` :

```
public int differenceEnJours(Date d)
```

dont le résultat est la différence en nombre de jours entre `this` et le paramètre `d`.

Exercice 2 : Identités

Q 1 . Donnez un code correspondant aux entités définies par le graphe UML de la figure 1. Celles-ci doivent se trouver dans un paquetage `banque.util`.

Le résultat de `toString()` est pour `PersonneId` la concaténation du nom et du prénom séparés d'un espace, et la valeur de `raisonSociale` pour `EntrepriseId`. Pour les deux cas la méthode `equals` teste l'égalité de tous les attributs.

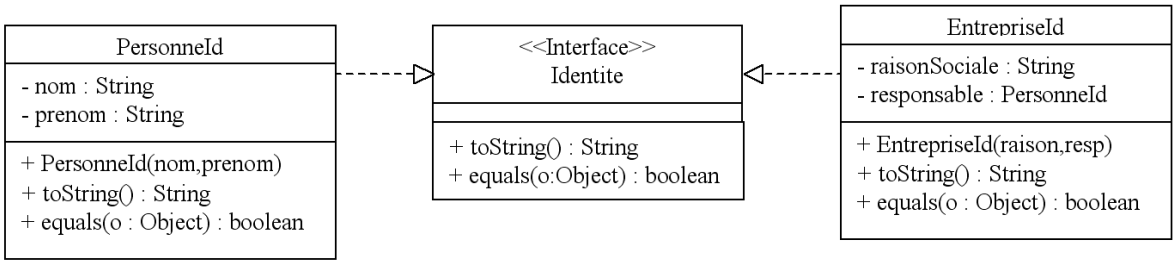

Figure 1: Graphe UML pour les identités

Exercice 3 : Ecritures

Une écriture correspond à une opération bancaire sur un compte. On les représente par une classe `Ecriture`. Une instance de cette classe est définie par une date (de type `Date`), un intitulé (une chaîne de caractères) et un montant (un flottant).

Q 1 . Donnez le diagramme d'une classe `Ecriture` fournissant comme méthodes uniquement les accesseurs pour les attributs.

On supposera cette classe définie dans le paquetage `banque.compte`.

Exercice 4 : Compte

Une interface `Compte` est donnée en annexe.

Il existe différents type de comptes : des comptes chèques, des comptes épargne et des plans d'épargne. A chacun de ces types correspond une classe implémentant l'interface `Compte`. Ces classes sont respectivement les classes `CompteCheque`, `CompteEpargne` et `PlanEpargne` du paquetage `banque.compte`.

Les instances des classes `CompteEpargne` et `PlanEpargne` disposent d'informations supplémentaires telles qu'un taux d'intérêt, une date d'échéance, etc. On suppose ces classes définies.

Q 1 . Les instances de la classe `CompteCheque` sont quant à elles caractérisées par un attribut `autorisationDecouvert` dont la valeur peut être changée par le banquier si besoin. Cette valeur (négative) correspond au découvert autorisé sur ce compte.

La méthode `addEcriture` de cette classe lève l'exception mentionnée dans la signature lorsque l'on essaie d'effectuer une écriture sur le compte qui amènerait à un solde inférieur à la valeur de `autorisationDecouvert`. Cette écriture n'est dans ce cas pas ajoutée.

Donnez le code **complet** de la classe `CompteCheque` (un numéro est fourni en paramètre du constructeur).

Exercice 5 : Clients

Un client est caractérisé par un nom (de type `banque.util.Identite`) une adresse (chaîne de caractères) et la liste de ses comptes. Il y a deux types de clients :

▷ les entreprises, dans ce cas le nom est une instance de la classe `banque.util.EntrepriseId` (d'autres informations particulières à ce type de client, comme son numéro SIRET, peuvent également être nécessaires).

▷ les personnes physiques (un particulier), le nom est alors une instance de la classe `banque.util.PersonneId`.

On veut pouvoir :

▷ avoir l'identité du client,

▷ avoir l'adresse du client,

▷ avoir la liste des comptes du client,

▷ deux clients sont dits égaux si ils ont la même identité et même adresse,

▷ ajouter un compte donné (en paramètre) pour le client,

▷ avoir le compte du client correspondant à un numéro donné (`null` si aucun compte ne correspond),

▷ disposer d'une méthode :

```
public void addEcriture(String numCpte,
                        String intitule,
                        Date date,
                        float montant) throws UnsupportedOperationException
```

qui ajoute au compte correspondant au numéro `numCpte`, une écriture avec la date, l'intitulé et le montant indiqués, quand c'est possible.

Q 1 . Que proposez-vous pour la définition des données de type client ? Donnez **tous** le code nécessaire à la gestion des clients correspondant à une personne physique, dans un paquetage `banque.client`.

Exercice 6 : La banque

Une banque est représentée par un objet de la classe `Banque`. Les instances de cette classe sont définies par un ensemble de clients. Pour faciliter la recherche, on gère également une table associant à un numéro de compte le client correspondant.

Q 1 . Faites les déclarations nécessaires (attributs et constructeur(s)) pour la classe `Banque`, à définir dans le paquetage `banque`.

Q 2 . Définissez une méthode `creeCompteCheque` qui prend en paramètre un numéro de compte et un client (supposé déjà créé et connu) et ajoute un nouveau compte chèque au client précisé.

Q 3 . Définissez une méthode `clientARisque` qui retourne la liste des clients dont au moins un compte est débiteur.

Q 4 . Définissez une méthode `getClient` qui retourne le client titulaire du numéro de compte passé en paramètre.

Q 5 . Définissez une méthode `dixMeilleursClients` qui retourne les dix clients dont les soldes cumulés de leurs différents comptes sont les dix plus élevés (on supposera pour simplifier qu'il n'y a pas d'ex-aequo possible).

Annexe

La classe `Mois`

```
package util.date;
public class Mois implements Comparable {
  private int numero;
  private static int cpt = 1;
  private Mois() {
    this.numero = cpt++;
  }
  public static final Mois JANVIER = new Mois();
  public static final Mois FEVRIER = new Mois();
  ...
  public static final Mois DECEMBRE = new Mois();

  private static final int[] NB_JOURS = new int[] {31, 28, 31, ..., 31, 30, 31};
  private static final String[] NOMS_MOIS = new String[] {"janvier", "fevrier",..., "decembre"};
  public static final Mois[] LES_MOIS = new Mois[] {JANVIER, FEVRIER,..., DECEMBRE};

  public String toString() { return lesMois[numero-1]; }
  public int getNumero() { return numero; }

  public int compareTo(Object o) {
    // ... à compléter
  }

  /** retourne le nombre de jours dans un mois
  /* @param bissextile  true ssi on est dans une année bissextile
  /* @return le nombre de jours dans un mois
  */
  public int getNbJours(boolean bissextile) {
    if (this == Mois.FEVRIER && bissextile) {
      return 29;
    }
    else {
      return nbJours[this.numero-1];
    }
  }
}
```

L'interface `Compte`

```
package banque.compte;
import java.util.*;
public interface Compte {
  // retourne le numéro du compte
  public String getNumeroCompte();
  // retourne le solde de ce compte
  public float getSolde();
  // retourne la liste des écritures pour le compte
  public List getEcritures();
  // ajoute une ecriture pour ce compte
  public void addEcriture(Ecriture e) throws UnsupportedOperationException;
  // retourne la liste des écritures du compte des <nbJours> jours
  // précédant <d> (inclus)
  public List ecrituresDepuis(util.date.Date d, int nbJours);
}
```