Université Bordeaux 1
Département Licence

Licence Mention Informatique, Semestre 4, Programmation avec des Objets INF 203

Corrigé proposé par Robert Cori

Examen du 22 mai 2006

Exercice 1 : Construction d'une classe *(8 points)*

La classe `TGV` qui permet de gérer des horaires de trains. Chaque objet de type `TGV` contient un entier (numéro du train), un tableau de chaînes de caratères constitué des gares où s'arrête le train (le premier élément de ce tableau est la gare de départ, le dernier le terminus et les autres arrêts sont dans l'ordre de passage) et un autre tableau de nombre entiers représentant les heures de passage dans les gares, ceci dans le même ordre.

1.1. *(5 points)*

```
public class TGV{
   private int numero;
   private String[] gares;
   private int[] horaires;

   TGV (int n, String[] g, int[] h){
       numero = n; gares = g; horaires = h;
   }
   public int heurePassage(String u){
       for (int i = 0; i < gares.length; ++i)
           if (u.compareTo(gares[i]) == 0) return horaires[i];
       return -1;
    }
   public int duree(String u, String v){
       int a = heurePassage(u);
       int b = heurePassage(v);
       if (a==-1 || b == -1 || a > b) return -1;
       else return b -a;
   }
   public String toString(){
       int a = gares.length;
       String u = numero +" " + gares[0] + " " + horaires[0] +
         "  " + gares[a-1]+ "  " + horaires[a-1];
       return u;
   }
   public int nbIntersecte(TGV t) {
       int res = 0;
       for (int i = 0; i < gares.length; ++i)
           if( t.heurePassage(gares[i])!= -1) res++;
       return res;
```

```
    }
    public String intersection(TGV t){
       String res = null;
       if (nbIntersecte(t) == 1)
           for (int i = 0; i < gares.length; ++i)
               if( t.heurePassage(gares[i])!= -1) res = gares[i];
       return res;
    }
```

1.2. *(1,5 points)* On considère une autre classe `Test` qui utilise la classe précédente.

méthode qui étant donné un tableau `tab` de TGV et deux noms de villes `u, v` affiche tous les TGV qui relient ces deux villes et contenus dans ce tableau.

```
 static void afficheTGV(TGV[]  tab, String u, String v){
        for (int i = 0; i < tab.length ; ++i)
            if (tab[i].duree(u, v) != -1) System.out.println(tab[i]);
    }
```

1.3. *(1,5 points)*

Méthode qui affiche tous les couples de TGV qui relient la ville `u` à une ville `w` et la ville `w` à la ville `v` ; ces deux TGV devront avoir `w` comme seule gare d'arrêt commune.

```
 static void afficheTGVCorres (TGV[] tab, String u, String v){
        for(int i = 0 ; i < tab.length; ++i)
            for (int j = 0; j < tab.length; ++j)
                if (tab[i].nbIntersecte(tab[j]) == 1){
                     String w = tab[j].intersection(tab[i]);
                     if (tab[i].duree(u,w)!= -1 && tab[j].duree(w, v) != -1){
                        System.out.println(tab[i]);
                        System.out.println(tab[j]);
                    }
                 }
  }
```

Exercice 2 : Exceptions *(6 points)*

On considère la classe suivante :

```
public class RecursifEx{
    static long test(int n){
        try{
            long v = n*n/n;
            return v + test(n-1);
        }
        catch (ArithmeticException e){
            return 0;
        }
    }
```

```
    public static void main (String[] args){
        int m = Integer.parseInt(args[0]);
        System.out.println(test(m));
   }
}
```

2.1 Valeurs affichées par les commandes suivantes,

- `java RecursifEx 0` La fonction `main` récupère la valeur `m = 0` et appelle `test(0)` Lors de cet appel l'interpréteur effectue une division par 0 qui lance une Exception de type `ArithmeticException` laquelle est traitée dans la partie `catch` et donne la valeur `0` c' est donc `0` qui est affiché.
- `java RecursifEx 2` La fonction `main` récupère la valeur `m = 2` et appelle `test(2)` qui calcule `2 + test(1)` ; donc appel de `test(1)` qui calcule `1 = test(0` Lors de cet appel on se retrouve dans la situation plus haut et finalement le résultat est 3.
- `java RecursifEx 5` Le calcul est `5 + 4 + 3 + 2 + 1 + 0` qui donne pour résultat `15`

2.2 Si on exécute la commande :

```
java RecursifEx
```

il y aura lancement d'une exception lors de l'acces à `args[0]`.

Récupération de cette exception de type `IndexOutOfBoundsException` :

```
  public static void main (String[] args){
        try{
          int m = Integer.parseInt(args[0]);
        }
        catch (IndexOutOfBoundsException e){
    System.out.println("Recommencez avec une valeur
                               entiere après votre commande ");
}
        System.out.println(test(m));
    }
```

2.3 Si on exécute la commande :

```
java RecursifEx -1
```

dans ce cas `m` prend la valeur `-1` et il y a une suite infinie d'appels. L'ineterpréteur s'arrête en signalant que la pile de récursivité est peleine.

2.4 Classe `RecurException` dont les objets seront lancés si un argument négatif est donné à la méthode statique `test`.

```
class RecurException extends Exception{
    RecurExcp (){
        System.out.println("Valeur negative, changement de signe necessaire");
    }
}
```

Réécrire la méthode `main` de façon qu'elle traite le cas où un utilisateur donne une valeur négative en lançant une exception de type `RecurException` qui sera capturée pour remplacer la valeur donnée par sa valeur absolue.

```
public static void main (String[] args){
    int m = 0;
    try{
         m = Integer.parseInt(args[0]);
         if (m <0 ) throw new RecurException();
         System.out.println(test(m));
      }
    catch (RecurException e1){
        System.out.println (test(-m));
    }
    catch  (IndexOutOfBoundsException e){
    System.out.println("Recommencez avec une valeur entiere");
    }
 }
```

Exercice 3 Héritage *(6 points)*

On considère les classes données dans l'énoncé :

3.1 Valeurs affichées lorsqu'on effectue la commande ci-dessous ?

`java Test`

```
Morues  Begles 0.0
EADS  Toulouse 0.0
Zinedine  Zidane  Madrid 0.0

Morues  Begles 1000.0
EADS  Toulouse 0.0
Zinedine  Zidane  Madrid 0.0
```

3.2 Précisez pour chacune des deux classes `Client` et `Particulier` quels sont les méthodes redéfinies.

Dans la classe `Client` la méthode `toString` est redéfinie ainsi que dans la classe `Particulier`. Dans cette deuxième classe le constructeur est redéfini.

3.3 la nouvelle classe `Entreprise` qui hérite de la classe `Client` qui a une nouvelle méthode.

```
class Entreprise extends Client{
    Entreprise(String u, String v){
super(u,v);
    }
    public void achete(double mont, double taux){
totalAchat += mont*taux;
    }
}
```

Une fois définie cette nouvelle classe à la compilation si on remplace les deux lignes

```
carnet[0] = new Client("Morues" , "Begles");
carnet[0].achete(1000);
```

par

```
carnet[0] = new Entreprise("Morues" , "Begles");
carnet[0].achete(1000, 0.8);
```

Il y a un problème car la méthode `achete()` est considérée comme n'ayant qu'un seul paramètre pour les `carnet[i]` que le compilateur considère comme des objets de type `Client` alors qu'à l'exécution ce peut être des objets de type `Particulier` ou `Entreprise`.

Un façon de régler ce problème est de définir dans la classe `Client` une méthode `achete (double mont, double taux)` qui surcharge la méthode `void achete (double montant)` de la façon suvante

```
public void achete(double montant, double taux){
       achete(montant);
}
```