Université Bordeaux 1
Département Licence

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

Sujet proposé par Robert Cori

Epreuve d'examen du 22 mai 2006, Durée 1h 30

Documents autorisés : sujets, corrigés de travaux dirigés et copies des transparents. Les livres et photocopies de livres sont interdits. Le barême est indicatif.

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

On se propose de construire une 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.

Pour simplifier, ces horaires de passage seront des entiers considérés comme des minutes ; ainsi le TGV qui part de Bordeaux à 10h 25 pour atteindre Irun à 12h 54 sera affecté des tableaux suivants :

```
[Bordeaux Dax Bayonne Biarritz Saint-Jean-de-Luz Hendaye Irun]
[625 697 730 742 755 767 774]
```

Les valeurs données s'expliquent par : $625 = 10 \times 60 + 25$, $774 = 12 \times 60 + 54$.

Cette classe devra contenir les composants suivants :

- Constructeur d'un objet `TGV` à partir d'un entier, un tableau de chaînes de caractères (`String[]`) et un tableau d'entiers (`int[]`).
- Deux méthodes : l'une `int heurePassage(String u)` indiquera l'horaire d'arrêt du TGV auquel s'applique la méthode dans la ville `u` ; l'autre `int duree(String u, String v)` donnera le temps mis entre deux villes de noms `u, v` par le `TGV` à qui s'applique la méthode. Le résultat sera -1 dans les deux cas si le TGV ne passe pas par les villes considérées.
- Une méthode `String toString()` qui renvoie une chaîne de caractères contenant le numéro du TGV suivi de la gare de départ de l'heure de départ, du terminus, de l'heure d'arrivée. Les heures de départ et d'arrivée seront exprimées sous forme d'entiers. Pour le TGV précédent cette chaîne sera "`8505 : Bordeaux 625 Irun 774`"
- Une méthode `int nbIntersecte(TGV t)` qui calcule le nombre de gares communes au TGV `t` et à celui à qui est appliquée la méthode.
- Une méthode `String intersection(TGV t)` qui dans le cas où le TGV `t` et celui à qui est appliquée la méthode ont une seule gare commune donne comme résultat le nom de cette gare. S'il n'y a pas de gare commune ou s'il y en a plus d'une le résultat sera `null`.

1.1. *(5 points)* Ecrire la définition de cette classe, du constructeur et des cinq méthodes.

1.2. *(1,5 points)* On considère une autre classe `Test` qui utilise la classe précédente. On vous demande d'écrire une méthode dans la classe `Test`

```
public static void afficheTGV (TGV[] tab, String u, String v)
```

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. Ces TGV seront affichés à l'aide de la méthode `toString()` demandée plus haut, ainsi il n'y aura pas d'affichage des heures de passage dans ces gares s'il ne s'agit pas de la gare de départ ou du terminus.

1.3. *(1,5 points)* On souhaite trouver des TGV qui relient deux villes en effectuant un changement de train, pour simplifier on considère que les heures de passage des trains ne sont pas prises en compte (ainsi on peut envisager qu'un voyageur attende toute une nuit sa correspondance). Ecrire une fonction

```
public static void afficheTGVCorres (TGV[] tab, String u, String v)
```

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.

Exercice 2 : Exceptions *(6 points)*

On considère la classe suivante :

```
public class RecursifExcp{
    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 Quelles sont les valeurs affichées par les commandes suivantes, (pour les deux premiers cas vous expliquerez les actions qui se déroulent à l'exécution, dans le troisième cas vous donnerez simplement le résultat).

- `java RecursifEx 0`
- `java RecursifEx 2`
- `java RecursifEx 5`

2.2 Que se passe-t-il si on exécute la commande :

```
java RecursifEx
```

Ecrire dans `main` une récupération d'une exception de type `IndexOutOfBoundsException` qui affiche dans le cas précédent le message :

```
Recommencez en mettant une valeur entiere après votre commande
```

2.3 Que se passe-t-il si on exécute la commande :

```
java RecursifEx -1
```

2.4 Ecrire une classe `RecurException` dont les objets seront lancés si un argument négatif est donné à la méthode statique `test`. Il y aura alors un affichage du message `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.

Exercice 3 Héritage *(6 points)*

On considère les classes suivantes :

```
class Client{
    private static int nombre = 0;
    protected int numero;
    private String nom;
    private String adresse;
    protected double totalAchat;
    Client  (String u, String adr){
        nom = u;
        numero = nombre++;
        adresse = adr;
        totalAchat = 0;
    }
    public  String toString(){
        return nom + "  " + adresse + " " + totalAchat;
    }
    public void achete (double montant){
        totalAchat += montant;
    }

}

class Particulier extends Client{
    private String prenom;
    Particulier (String u, String v, String w){
        super(u, w);
        prenom = v;
    }
   public  String toString(){
        return prenom +"  " +  super.toString();
    }

}

class Test {
    public static void main(String[] args) {
        Client[] carnet = new Client[3];
```

```
        carnet[0] = new Client("Morues" , "Begles");
        carnet[1] =  new Client("EADS", "Toulouse");
        carnet[2] = new Particulier("Zidane", "Zinedine", "Madrid");
        for (int i  = 0; i < carnet.length;i++)
            System.out.println(carnet[i]);
        carnet[0].achete(1000);
        for (int i  = 0; i < carnet.length;i++)
            System.out.println(carnet[i]);
    }
}
```

3.1 Quelles sont les valeurs affichées lorsqu'on effectue la commande ci-dessous ?

```
java Test
```

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

3.3 Ecrire une nouvelle classe `Entreprise` qui hérite de la classe `Client` qui a une nouvelle méthode `achete(double mont, double tauxReduc)` ou `tauxReduc` est un réel compris entre 0 et 1. Cette méthode tient compte d'une réduction du montant de la facture, ce montant étant alors égal à `mont * tauxReduc`.

Une fois définie cette nouvelle classe que se passe-t-il 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);
```

Quelle(s) classe(s) faut-il modifier pour éviter cette erreur à la compilation ? Précisez les modifications à effectuer.