Université de Nice-Sophia Antipolis — POLYTECH
PeiP2 — 2018–2019

Examen de Algorithmique et Programmation JAVA

Durée : 2h
Aucun document autorisé
Mobiles interdits

Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.

---

1 Question 1

On se propose de programmer, en Java, un petit système de location de voitures (sans héritage).

▶ 1. Écrivez une classe `Voiture` avec :
- trois attributs `modèle`, `prix` et `dispo` qui représentent respectivement le modèle de la voiture, son prix de location par jour et sa disponibilité. Vous veillerez à choisir des types appropriés pour ces attributs.
- Un constructeur prenant en paramètres le modèle et le prix de la voiture. À sa construction, une voiture sera toujours disponible à la location.
- La méthode `toString` permettant d'afficher une voiture sous la forme : `"Le modèle de voiture ... est à ... euros par jour. Disponibilité: ... ."`.
- Une méthode `comparePrix` qui compare, c'est-à-dire qui renvoie la différence, entre le prix (par jour) de la voiture courante et celui d'une voiture passée en paramètre.
- Une méthode `calculePrix` qui prend en paramètre une durée de location (en jours) et renvoie le prix de la location de la voiture courante pour cette durée.

```
public class Voiture {
    //Attributs
    private String modèle;
    private double prix;
    private boolean dispo;
    //Constructeur
    public Voiture(String modèle, double prix) {
        this.modèle = modèle;
        this.prix = prix;
        this.dispo = true;
    }
    //toString
    public String toString() {
        return "Le␣modèle␣de␣voiture␣" + this.modèle +
            "est␣à␣" +  this.prix + "Par␣jour.␣Disponibilité␣:␣" +
            this.dispo;
    }
    //Méthodes
    public double comparePrix(Voiture v) {
        return this.prix - v.prix;
        //négatif si la voiture en paramètre est plus chère
    }
    public double calculerPrix(int nbJours) {
        assert nbJours>0;
        return nbJours*this.prix;
    }
    //Accesseur
    public boolean getDispo() {
        return this.dispo;
    }
    //Mutateur
    public void changeDispo() {
        this.dispo = !this.dispo;
    }
}
```


.....

▶ 2. Écrivez une classe `Client` avec :
- deux attributs `nom` et `num` qui représentent, respectivement, le nom du client et son numéro d'enregistrement dans la base de données du système.
- Un constructeur prenant en paramètres uniquement le nom du client. Le numéro attribué au premier client sera 1, celui attribué au second sera 2 et ainsi de suite. Vous pourrez vous aider d'une variable de classe permettant de comptabiliser le nombre de clients construits.
- Une méthode `réserveVoiture` permettant au client courant de réserver une voiture pour un nombre de jours donné. Cette méthode vérifiera la disponibilité de la voiture avant de la réserver, mettra à jour sa disponibilité si le client peut la réserver et affichera le montant total à payer.

```
public class Client {
    //Variable de classe
    static private long compteurClient = 1;
    //Attributs
    private String nom;
    private long num;
    //Constructeur
    public Client(String nom) {
        this.nom = nom;
        this.num = compteurClient++;
    }
    //Méthode
    public void réserveVoiture(int nbJours, Voiture v) {
        assert nbJours>0;
        if (!v.getDispo())
            System.err.println("Voiture␣indisponible␣!");
        else {
            v.changeDispo();
            System.out.println("Prix␣:␣" + v.calculerPrix(nbJours));
        }
    }
}
```

.....

▶ 3. Écrivez une classe `Réservations` contenant une méthode `main` permettant de :
- créer un client,
- créer un tableau de 3 voitures ayant des prix différents,
- comparer le prix des voitures du tableau (en utilisant la méthode `comparePrix`) pour permettre au client de réserver la moins chère (pour 4 jours).

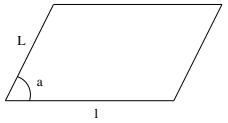
```
public class Réservations {
    public static void mai(String []args) {
```

```
        Client c = new Client("Brian");
        Voiture [] flotte = {
            new Voiture("Talisman",150),
            new Voiture("Micra", 50),
            new Voiture("Zoé", 100.0),
        };
        //recherche la voiture la moins chère
        Voiture moinsChère = flotte[0];
        for (int i=1; i<flotte.length; i++)
            if (moinsChère.comparePrix(flotte[i]) > 0)
                moinsChère = flotte[i];
        //réserver la moins chère
        c.réserveVoiture(4, moinsChère);
    }
}
```

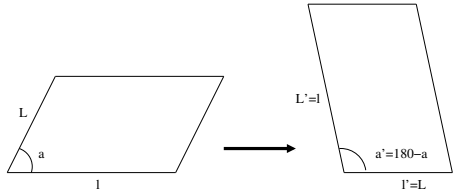
.....

2 Question 2

On définit les *parallélogrammes* dont un des cotés est posé sur un axe horizontal par leur longueur horizontale l, leur longueur inclinée L et l'angle inférieur gauche a. La figure ci-dessous montre un exemple.



- 4. Déclarez en Java la hiérarchie de classes qui représente les parallélogrammes *généraux*, les rectangles (des parallélogrammes dont l'angle est toujours égal à 90), et les losanges (un parallélogramme ayant deux côtés consécutifs de même longueur). Dans les constructeurs des classes filles, veillez bien à ne déclarer que les paramètres d'appel que vous jugez utiles.
- 5. On veut effectuer des déformations sur les différents parallélogrammes. Écrivez au niveau de la classe racine (et redéfinir si nécessaire dans ses classes filles) les méthodes `allongerX(double coeff)`, `allongerY(double coeff)` et `changerAngle(double coeff)` qui modifie respectivement l, L et a. `allongerX` sera par exemple `l=l*coeff`. **Important :** les déformations ne doivent pas changer la nature du parallélogramme sur lequel elle sont appliquées. Par exemple, un losange devra garder ses cotés égaux, ou rectangle son angle droit quelle que soit la déformation.
- 6. Programmez la méthode `pivoter` qui effectue la transformation donnée ci-dessous :



- 7. Écrivez la classe `Carré`. Veillez à ce qu'on puisse appeler des méthodes pour déformer ou faire pivoter un carré, néanmoins, tout en le laissant carré : est-il ou pas nécessaire de redéfinir certaines méthodes ?

```
/** Cette classe définit les parallélogrammes dont un des cotés est
 *  posé sur un axe horizontal par leur longueur horizontale l, leur
 *  longueur inclinée L et l'angle inférieur gauche a.
 */
public class Parallélogramme {
    protected double l, L, a;

    public Parallélogramme(double l, double L, double a) {
        assert l>0 && L>0 && a>=0 && a <=180;
        this.l = l;
        this.L = L;
        this.a = a;
    }

    /**
     *  Antécédent : c>0
     *  Rôle : allonge la longueur horizontale d'un coefficient
     *         multiplicateur c
     */
    public void allongerX(double c)    {
        assert c>0;
        this.l*=c;
    }
    /**
     * Antécédent : c>0
     * Rôle : allonge la longueur inclinée d'un coefficient
     *        multiplicateur c
     */
    public void allongerY(double c)    {
        assert c>0;
        this.L*=c;
    }

    /**
     * Antécédent : a>=0 & a<=180
     * Rôle : change l'angle du parallélogramme courant à la valeur a
     */
    public void changerAngle(double a) {
        assert a>=0 && a <=180;
        this.a=a;
    }

    /**
     *  Rôle : pivote le parallélogramme courant vers la droite
     */
    public void pivoter() {
        //échanger la longueur et la largeur
        double aux = this.l;
        this.l = this.L;
        this.L = aux;
        //prendre l'angle opposé
        this.a =  this.a-180;
```

```
    }

    public String toString() {
        return "(l=" + this.l + ", L=" + this.L + ", a=" + this.a + ")";
    }
}

/**
 *  Cette classe définit des Losange par héritage de Parallélogramme
 *  Elle garantit que les cotés restent égaux par redéfinition des méthodes
 *  allongerX et allongerY
 */
public class Losange extends Parallélogramme {

    public Losange(double c, double a) {
        super(c, c, a);
    }

    /**
     *  Antécédent : c>0
     *  Rôle : allonge la longueur horizontale ET
     *  inclinée d'un coefficient multiplicateur c, et donc assure que
     *  les longueurs des 2 cotés du Losange courant restent égales
     */

    @Override
    public void allongerX(double c) {
        super.allongerX(c);
        super.allongerY(c);
    }

    /**
     *  Antécédent : c>0
     *  Rôle : allonge la longueur horizontale ET
     *  inclinée d'un coefficient multiplicateur c, et donc assure que
     *  les longueurs des 2 cotés du Losange courant restent égales
     */
    @Override
    public void allongerY(double c) {
        this.allongerX(c);
    }
};

/**
 *  Cette classe définit des Rectangle par héritage de Parallélogramme
 *  Elle garantit que l'angle reste à 90 par redéfinition de la méthode
 *  changerAngle.
 */
public class Rectangle extends Parallélogramme {
    public Rectangle(double l, double L) {
        super(l, L, 90);
    }

    /**
     * Rôle : assure que l'angle du Rectangle courant est à 90 degrés
     */
    @Override
    public void changerAngle(double c) { assert c==90; }
};
```



```
/**
 *  Cette classe définit les Carré par héritage des Losanges.
 *  Elle garantit que l'angle reste à 90 par redéfinition de la méthode
 *  changerAngle.
 */
public class Carré extends Losange {
    public Carré(double c) {
        super(c,90);
    }

    /**
     * Rôle : assure que l'angle du Carré courant est à 90 degrés
     */
    @Override
    public void changerAngle(double c) { assert c==90; }
}
```

.....

3 Question 3

Dans cet exercice, vous manipulerez en écriture un fichier de nom f contenant des entiers naturels représentés par le type **int**. En Java, la classe `FileOutputStream` définit des flots d'octets séquentiels en écriture. La classe `DataOutputStream` est également utilisée, en combinaison de la première, pour l'écriture d'un fichier. Comme dans le TD7, nous supposerons donc qu'une classe `X` définit un attribut `String f` représentant le nom du fichier que l'on veut ainsi créer, et ce avec

```
DataOutputStream dis = new  DataOutputStream(new FileOutputStream(f));
```

Les principales méthodes fournies par DataOutputStream sont :

`Classe DataOutputStream`
`DataOutputStream(OutputStream out)` Associe un objet de type `DataOutputStream` avec le fichier ouvert en écriture transmis en paramètre
`void writeBoolean(boolean b) throws IOException` Écrit un booléen
`void writeChar(char c) throws IOException` Écrit un caractère
`void writeDouble(double d) throws IOException` Écrit un réel double précision
`void writeFloat(float f) throws IOException` Écrit un réel simple précision
`void writeInt(int i) throws IOException` Écrit un entier
`void writeLong(long l) throws IOException` Écrit un entier long
`void writeShort(short s) throws IOException` Écrit un entier court
`void writeUTF(String S) throws IOException` Écrit une chaîne de caractères au format UTF
`void close() throws IOException` Ferme le fichier

▶ 8. Écrivez dans la classe `X` une méthode `générerTrié` qui tire au hasard et écrit n entiers naturels, pris sur l'intervalle $[0; 100[$, ordonnés de manière croissante, dans le fichier `f`. L'objectif est de remplir ce fichier des n entiers ordonnés de manière croissante sans exécuter d'algorithme de tri. Ainsi, si le $i$ème entier tiré est 99, avec $i < n$, il vaut mieux « jeter » ce nombre, pour en tirer un à

nouveau qui soit à la fois supérieure ou égal au dernier nombre écrit mais pas égal à 99. Si par contre le $n$ème nombre tiré est 99, et est plus grand que le précédent, on pourra le conserver et l'écrire dans le fichier. Vous veillerez à correctement gérer les exceptions de type `FileNotFoundException` et `IOException`.

```
/**
  * Antécédent : n nombre positif d'entiers de [0..100[ triés à écrire
  *
  * Rôle : crée le fichier de nom f, contenant (au mieux) n nombres entiers triés
  *
  */
void générerTrié(int n) {
  int i = 1;
  int nb;  //nombre courant tiré
  int precNb=0; //précédent du nombre tiré

  java.util.Random gr= new java.util.Random();
  try (DataOutputStream dis = new DataOutputStream(new FileOutputStream(this.f)))
  {
       while (i<=n) {
          nb=gr.nextInt(100);
          if (i!=n || nb!=99)
              if (nb >= precNb) {
                 i++;
                 dis.writeInt(nb);
                 precNb=nb;
              }
      }
  }
  catch(FileNotFoundException e) { /* traiter l'erreur */ }
  catch(IOException e) { /*  traiter l'erreur */ }
}
```

.....