Conditions de distribution et de copie

Exercices (V) : Programmation d'objets

Fabrice Rossi

16 mars 2002

1 Analyse de programmes

Exercice 1.1 :

On considère la définition d'objet suivante :

A

```
1  public class A {
2    public int x;
3    public A(int x) {
4      this.x=x;
5    }
6    public int b() {
7      return 2*x;
8    }
9    public A c(A y) {
10     return new A(x+2*y.x);
11   }
12 }
```

Questions :

1. Quel est l'affichage produit par le programme suivant :

TestA

```
1  public class TestA {
2    public static void main(String[] args) {
3      A u=new A(2);
4      System.out.println(u.b());
5      A v=new A(3);
6      System.out.println(v.b());
7      A w=u.c(v);
8      System.out.println(w.b());
9      w.x=4;
10     System.out.println(w.b());
11   }
12 }
```

2. Dessinez l'état de la mémoire juste après l'exécution de la dernière ligne du programme de la question précédente.
3. Quel est l'affichage produit par le programme suivant :

```
1 A u=new A(6);
2 System.out.println(u);
```

Proposez une méthode d'instance permettant de rendre plus utile cet affichage.

4. A la ligne 2 de la classe `A`, on remplace `public` par `private`. Après cette modification, à quelle(s) condition(s) le programme de la question 1 est-il compilable sans erreur ?

Exercice 1.2 :

On suppose donnée la classe suivante :

Comb

```
1  public class Comb {
2    public int x,y;
3    public double lambda;
4    public Comb(int x,double lambda,int y) {
5      this.x=x;
6      this.lambda=lambda;
7      this.y=y;
8    }
9    public Comb(int x,int y) {
10     this(x,0,y);
11   }
12   public double valeur() {
13     return x+lambda*y;
14   }
15   public Comb somme(Comb u){
16     return new Comb(x+u.x,lambda+u.lambda,y+u.y);
17   }
18   public String toString() {
19     if(lambda==0 || y==0) {
20       return String.valueOf(x);
21     }
22     if(lambda>0) {
23       return x+"+"+lambda+"*"+y;
24     }
25     return String.valueOf(x)+lambda+"*"+y;
26   }
27   public void décale(double t) {
28     lambda=t;
29   }
30 }
```

On propose le programme suivant qui utilise la classe `Comb` :

TestComb

```
1  public class TestComb {
2    public static void main(String[] args) {
3      Comb c=new Comb(2,3);
4      System.out.println(c);
5      System.out.println(c.valeur());
```

```
6       Comb d=new Comb(1,2,4);
7       System.out.println(d);
8       System.out.println(d.valeur());
9       Comb e=d.somme(c);
10      System.out.println(e);
11      Comb u=c;
12      System.out.println(u);
13      c.décale(-1);
14      System.out.println(c);
15      System.out.println(u);
16    }
17  }
```

Questions :

1. Dessinez l'état de la mémoire juste après l'exécution de la ligne 15 du programme. On ne tiendra pas compte dans le dessin des objets `String`s éventuellement créés par la méthode `toString`.
2. Donnez l'affichage produit par `TestComb`
3. On remplace la ligne 3 de `TestComb` par :

```
1  Code c=new Comb();
```

   Expliquez pourquoi le nouveau programme n'est pas accepté par le compilateur.

4. Comment modifier la méthode `décale` pour éviter les effets de bord ? Que faut-il aussi modifier dans la classe `Comb` pour interdire toute possibilité d'effet de bord ?
5. On remplace la ligne 18 de la classe `Comb` par :

```
1  public String transforme() {
```

   Quelles sont les conséquences cette modification sur les classes `Comb` et `TestComb` ?

6. Dans la version d'origine de `TestComb`, on remplace la méthode `toString` par la version suivante :

toString
```
1  public void toString() {
2    if(lambda==0 || y==0) {
3      System.out.println(x);
4    }
5    if(lambda>0) {
6      System.out.println(x+"+"+lambda+"*"+y);
7    }
8    System.out.println(String.valueOf(x)+lambda+"*"+y);
9  }
```

   Quelles sont les conséquences cette modification sur les classes `Comb` et `TestComb` ?

Exercice 1.3 :

On considère la classe suivante :

Compteur

```
1  public class Compteur {
2    private int max;
3    private int val;
4    public Compteur(int max) {
5      this.max=max;
6    }
7    public int val() {
8      return val;
9    }
10   public String toString() {
11     return String.valueOf(val);
12   }
13   public void inc() {
14     val++;
15     if(val>=max) {
16       val=0;
17     }
18   }
19   public void dec() {
20     val--;
21     if(val<0) {
22       val=max-1;
23     }
24   }
25 }
```

Questions :

1. On propose le programme suivant :

TestCompteur

```
1  public class TestCompteur {
2    public static void main(String[] args) {
3      Compteur c=new Compteur(7);
4      System.out.println(c);
5      for(int i=0;i<10;i++) {
6        c.inc();
7        System.out.println(c);
8      }
9    }
10 }
```

   Quel est l'affichage produit par ce programme ?

2. On crée un objet `Compteur` grâce à l'appel `new Compteur(5)`. Quel est le contenu de la variable d'instance `val` juste après la création de l'objet ? Quelle que soit l'utilisation de l'objet ainsi créé, la variable `val` ne peut prendre que certaines valeurs. Quelles sont ces valeurs ? Justifiez avec précision votre réponse.

3. On propose le programme suivant :

TestCompteur2

```
1  public class TestCompteur2 {
2    public static void main(String[] args) {
3      Compteur c=new Compteur(4),d=c;
4      System.out.println(c);
5      for(int i=0;i<5;i++) {
6        c.inc();
7        d.dec();
8        System.out.println(c);
9      }
10   }
11 }
```

Quel est l'affichage produit par ce programme ?

4. On remplace la ligne 15 de la classe `Compteur` par :

```
1  if(val==max) {
```

Quelles sont les conséquences de cette modification sur les programmes `TestCompteur` et `TestCompteur2` ?

Exercice 1.4 :

On considère la classe suivante :

Intervalle

```
1  public class Intervalle {
2    private int inf,sup;
3    public Intervalle(int i,int s) {
4      inf=i;
5      sup=s;
6    }
7    public Intervalle a(Intervalle u) {
8      return new Intervalle(Math.max(inf,u.inf),Math.min(sup,u.sup));
9    }
10   public Intervalle b(Intervalle u) {
11     return new Intervalle(Math.min(inf,u.inf),Math.max(sup,u.sup));
12   }
13   public boolean contient(double x) {
14     return x>=inf && x<=sup;
15   }
16 }
```

1. Soit `i` un objet de type `Intervalle`. On suppose que `i.contient(x)` renvoie `true` si et seulement si le réel `x` est contenu dans l'intervalle représenté par `i`. En étudiant le code de la méthode `contient` déterminer la nature des intervalles représentés (ouvert, fermé, ouvert à gauche ou encore ouvert à droite).
2. Que font les méthodes `a` et `b` ? Réponse à justifier en quelques lignes.
3. La classe `Intervalle` n'est pas complète. Il lui manque en particulier des méthodes élémentaires qui permettraient de simplifier son utilisation. Proposer des méthodes utiles en donnant leur programmation.

2 Programmation

2.1 Représentation

Exercice 2.1 :

Programmez une classe `Date` permettant de représenter une date. Vous vous baserez sur le squelette suivant :

Date

```
1  public class Date {
2    public Date(int année,int mois,int jour) { }
3    public int dayOfYear();
4    public int weekOfYear();
5    public int dayOfWeek();
6    public int days();
7    public int distance(Date d);
8    public int compareTo(Object o);
9    public String toString();
10 }
```

Les méthodes doivent se comporter selon la documentation suivante :

constructeur : `Date(int année,int mois,int jour)`

fabrique un objet `Date` représentant la date indiquée par les paramètres (les mois sont numérotés de 1 à 12 et les jours de 1 à 31 (en fonction du mois, bien entendu))

`int dayOfYear()`

renvoie le numéro du jour représenté par l'objet appelant (la numérotation va de 1 à 365 ou 366)

`int weekOfYear()`

renvoie le numéro de la semaine du jour représenté par l'objet appelant dans l'année (la numérotation va de 1 à 52)

`int dayOfWeek()`

renvoie le numéro du jour représenté par l'objet appelant dans la semaine (la numérotation va de 0 pour dimanche à 6 pour 6 samedi)

`int days()`

renvoie le nombre de jours dans l'année du jour représenté par l'objet appelant

`int distance(Date d)`

renvoie la distance en nombre de jours entre la date appelante et la date paramètre

`int compareTo(Date d)`

permet la comparaison entre deux dates selon le modèle classique

`String toString()`

permet la transformation d'une date en une chaîne de caractères, selon la notation numérique classique en France, c'est-à-dire par exemple 15/06/02 pour le 15 Juin 2002

Exercice 2.2 :

Programmez une classe `Genre` permettant de représenter les genres féminin et masculin. On se basera sur le squelette suivant :

Genre

```
1  public class Genre {
2    public static final Genre FEMININ=...;
3    public static final Genre MASCULIN=...;
4    public boolean estFeminin() {}
5    public boolean estMasculin() {}
6    public String toString() {}
7  }
```

On pourra bien entendu ajouter toute méthode jugée utile. L'utilisation de la classe `Genre` se fera cependant essentiellement par l'intermédiaire des deux constantes de classe.

Exercice 2.3 :

Programmez une classe `EtatCivil` permettant de représenter l'état civil sommaire d'une personne, constitué des éléments suivants :
- nom
- prénom
- date de naissance (en utilisant un objet `Date` proposé dans l'exercice 2.1)
- sexe (en utilisant un objet `Genre` proposé dans l'exercice 2.2)
- statut marital (célibat, comcubinage, PACS ou mariage), avec le cas échéant l'`EtatCivil` du partenaire

La classe proposée devra permettre des manipulations importantes, comme par exemple le changement de statut marital.

Exercice 2.4 :

Programmez une classe `Carte` permettant de représenter une carte à jouer d'un jeu de 52 cartes. Programmez ensuite une classe `Poker` et une classe `Belote` proposant chacune une méthode de classe de comparaison entre deux `Carte`s selon les règles du jeu considéré.

2.2 Objects mathématiques

Exercice 2.5 :

Programmez une classe `Rationnel` permettant de représenter un rationnel au sens mathématique du terme (c'est-à-dire une fraction $\frac{p}{q} \in \mathbb{Q}$. On se basera sur le squelette suivant :

Rationnel

```
1  public class Rationnel {
2    public Rationnel(int p,int q) {}
3    public int num() {}
4    public int dén() {}
5    public double toDouble() {}
6    public Rationnel somme(Rationnel r) {}
7    public Rationnel produit(Rationnel r) {}
8    public Rationnel différence(Rationnel r) {}
9    public Rationnel quotient(Rationnel r) {}
10   public int compareTo(Rationnel r) {}
11   public Rationnel abs() {}
12   public static Rationnel valueOf(int i) {}
13 }
```

On programmera les méthodes en accord avec la documentation suivante pour la classe :

constructeur : `Rationnel(int p,int q)`
Fabrique un objet `Rationnel` représentant $\frac{p}{q}$.

`int num()`
Renvoie la valeur du numérateur du rationnel appelant.

`int dén()`
Renvoie la valeur du dénominateur du rationnel appelant.

`double toDouble()`
Renvoie le `double` le plus proche du rationnel représenté par l'objet appelant.

`Rationnel somme(Rationnel r)`
Renvoie un nouvel objet `Rationnel` somme du rationnel appelant et du rationnel `r`.

`Rationnel produit(Rationnel r)`
Renvoie un nouvel objet `Rationnel` produit du rationnel appelant et du rationnel `r`.

`Rationnel différence(Rationnel r)`
Renvoie un nouvel objet `Rationnel` différence du rationnel appelant et du rationnel `r`.

`Rationnel quotient(Rationnel r)`
Renvoie un nouvel objet `Rationnel` quotient du rationnel appelant et du rationnel `r`.

`int compareTo(Rationnel r)`
Renvoie un entier strictement négatif si le rationnel appelant est strictement plus petit que le rationnel `r`. Renvoie zéro si les deux rationnels sont égaux, et un entier strictement positif si le rationnel appelant est strictement plus grand que le rationnel `r`.

`Rationnel abs()`
Renvoie le rationnel valeur absolue du rationnel appelant.

`Rationnel valueOf(int i)`
Méthode **de classe** qui construit et renvoie la représentation sous forme de rationnel de l'entier `i` paramètre.

On ajoutera à la classe une méthode `toString` et une méthode `equals`.

Exercice 2.6 :

Programmez une classe `Intervalle` permettant de représenter un intervalle fermé de $\mathbb{R}$. On se basera sur le squelette suivant :

Intervalle

```
1  public class Intervalle {
2    public Intervalle(double inf,double sup) {}
3    public boolean contient(double x) {}
4    public boolean inclut(Intervalle i) {}
5    public Intervalle intersection(Intervalle i) {}
6    public Intervalle union(Intervalle i) {}
7    public Intervalle somme(double x) {}
8    public Intervalle somme(Intervalle i) {}
9    public Intervalle produit(double x) {}
10   public Intervalle produit(Intervalle i) {}
11   public Intervalle exp() {}
12   public Intervalle sin() {}
```

```
13    public String toString() {}
14    public boolean equals(Object o) {}
15  }
```

Pour programmer les méthodes, on tiendra compte de la documentation suivante :

constructeur : `Intervalle(double inf,double sup)`
fabrique l'objet correspondant à l'intervalle [`inf`, `sup`]

`boolean contient(double x)`
renvoie `true` si et seulement si le réel paramètre appartient à l'intervalle appelant

`boolean inclut(Intervalle i)`
renvoie `true` si et seulement si l'intervalle paramètre est inclus dans l'intervalle appelant

`Intervalle intersection(Intervalle i)`
renvoie l'intervalle intersection de l'intervalle appelant et de l'intervalle paramètre

`Intervalle union(Intervalle i)`
renvoie l'intervalle défini comme le plus petit intervalle contenant à la fois l'intervalle appelant et l'intervalle paramètre

`Intervalle somme(double x)`
renvoie le plus petit intervalle contenant $y + x$ pour tout y de l'intervalle appelant

`Intervalle somme(Intervalle i)`
renvoie le plus petit intervalle contenant $x + y$ pour tout x dans l'intervalle appelant et pour tout y dans l'intervalle paramètre

`Intervalle produit(double x)`
renvoie le plus petit intervalle contenant yx pour tout y de l'intervalle appelant

`Intervalle produit(Intervalle i)`
renvoie le plus petit intervalle contenant xy pour tout x dans l'intervalle appelant et pour tout y dans l'intervalle paramètre

`Intervalle exp()`
renvoie le plus petit intervalle contenant e^x pour tout x dans l'intervalle appelant

`Intervalle sin()`
renvoie le plus petit intervalle contenant $\sin x$ pour tout x dans l'intervalle appelant

`String toString()`
renvoie la représentation de l'intervalle appelant sous forme d'une chaîne de caractères

`boolean equals(Object o)`
compare l'intervalle appelant avec l'intervalle paramètre

Proposez une classe permettant de représenter les intervalles quelconque de $\mathbb{R}$ (fermé, ouvert et semi-ouvert).

2.3 Géométrie

Problème 2.7 :

On souhaite construire un ensemble d'objets représentants des formes géométriques simples. Pour simplifier le problème, les objets ne seront pas représentés de façon graphique.

Questions :

1. On commence par la classe `PointR2`[1] dont le squelette est le suivant :

PointR2

```
1  public class PointR2 {
2    public double x;
3    public double y;
4    public PointR2(double a,double b) {}
5    public double distance(PointR2 p) {}
6    public boolean equals(Object o) {}
7    public String toString() {}
8  }
```

Un objet `PointR2` représente un point du plan, repéré par ses coordonnées `x` et `y`. Programmez le constructeur et les méthodes en accord avec la documentation suivante :

constructeur : `PointR2(double a,double b)`

Fabrique un `PointR2` de coordonnées `a` et `b`.

`double distance(PointR2 p)`

Renvoie la distance (euclidienne) entre le point représenté par l'objet appelant et le point `p`.

`boolean equals(Object o)`

Renvoie `true` si et seulement si le point appelant et le point paramètre sont les mêmes au sens mathématique (mêmes coordonnées).

`String toString()`

Renvoie une représentation sous forme de chaîne de caractères du point appelant. Par exemple la chaîne `"(2.0,3.0)"` pour le point de coordonnées $(2, 3)$.

2. On programme ensuite la classe `Vecteur` dont le squelette est le suivant :

Vecteur

```
1  public class Vecteur {
2    public double x;
3    public double y;
4    public Vecteur(double a,double b) {}
5    public Vecteur(PointR2 A,PointR2 B) {}
6    public double norme() {}
7    public double scalaire(Vecteur v) {}
8    public double angle(Vecteur v) {}
9  }
```

Programmez les méthodes de cette classe en accord avec la documentation suivante :

constructeur : `Vecteur(double a,double b)`

Fabrique un vecteur de coordonnées `a` et `b`.

constructeur : `Vecteur(PointR2 A,PointR2 B)`

Fabrique le vecteur `AB`.

`double norme()`

Renvoie la norme du vecteur appelant.

[1] Il existe déjà en `Java` une classe `Point`, c'est pourquoi je propose d'utiliser un nom un peu plus lourd dans ce problème.

`double scalaire(Vecteur v)`

Renvoie le produit scalaire du vecteur appelant et du vecteur paramètre.

`double angle(Vecteur v)`

Renvoie la valeur (en radian) de l'angle orienté entre le vecteur appelant et le vecteur paramètre.

3. On propose ensuite la classe `Droite` dont le squelette est le suivant :

Droite

```
1  public class Droite {
2    public Droite(PointR2 a,PointR2 b) {}
3    public static Droite directeur(PointR2 a,Vecteur v) {}
4    public static Droite normal(PointR2 a,Vecteur n) {}
5    public boolean contient(PointR2 a) {}
6    public double distance(PointR2 a) {}
7    public Vecteur normal() {}
8    public Vecteur directeur() {}
9    public Intersection intersection(Droite d) {}
10 }
```

Proposer des variables d'instance pour cette classe (on les choisira `public`) et programmez le constructeur et les méthodes en accord avec la documentation suivante :

constructeur : `Droite(PointR2 a,PointR2 b)`

Fabrique la `Droite` passant par les deux points paramètres.

`Droite directeur(PointR2 a,Vecteur v)`

Méthode **de classe** qui fabrique la `Droite` passant par `a` et de vecteur directeur `v`.

`Droite normal(PointR2 a,Vecteur n)`

Méthode **de classe** qui fabrique la `Droite` passant par `a` et de vecteur normal `n`.

`boolean contient(PointR2 a)`

Renvoie `true` si et seulement si le point paramètre appartient à la droite appelante.

`double distance(PointR2 a)`

Renvoie la distance entre le point paramètre et la droite appelante.

`Vecteur normal()`

Renvoie un vecteur normal à la droite appelante.

`Vecteur directeur()`

Renvoie un vecteur directeur pour la droite appelante.

`Intersection intersection(Droite d)`

Renvoie un objet `Intersection` représentant l'intersection de la droite appelante et de la droite paramètre. Vous devez définir la classe `Intersection` avec des méthodes adaptées.

4. On programme ensuite la classe `Disque` dont le squelette est le suivant :

Disque

```
1  public class Disque {
2    public Disque(PointR2 c,double r) {}
3    public boolean contient(PointR2 p) {}
4    public double surface() {}
5    public double circonférence() {}
6  }
```

Proposer des variables d'instance pour cette classe (on les choisira `public`) et programmez le constructeur et les méthodes en accord avec la documentation suivante :

constructeur : `Disque(PointR2 c,double r)`

Fabrique un `Disque` de centre `c` et de rayon `r`.

`boolean contient(PointR2 p)`

Renvoie `true` si et seulement si le point `p` est contenu dans le disque appelant.

`double surface()`

Renvoie la surface du disque appelant.

`double circonférence()`

Renvoie la circonférence du disque appelant.

5. On continue avec la classe `Rectangle` dont le squelette est le suivant :

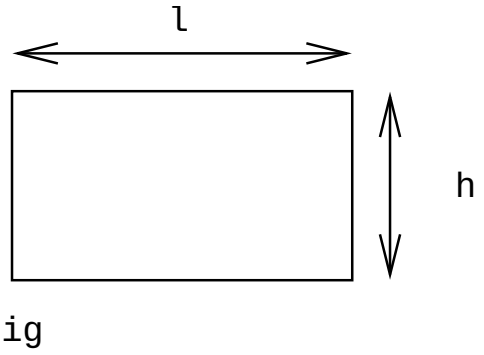
Rectangle

```
1  public class Rectangle {
2    public Rectangle(PointR2 cid,double l,double h) {}
3    public boolean contient(PointR2 p) {}
4    public double surface() {}
5    public double périmètre() {}
6    public Rectangle intersection(Rectangle r){}
7  }
```

Proposer des variables d'instance pour cette classe (on les choisira `public`) et programmez le constructeur et les méthodes en accord avec la documentation suivante :

constructeur : `Rectangle(PointR2 cig,double l,double h)`

Fabrique un `Rectangle` dont le coin inférieur gauche est le point `cig`, la largeur est `l` et la hauteur `h` :



`boolean contient(PointR2 p)`

Renvoie `true` si et seulement si le point `p` est contenu dans le rectangle appelant.

`double surface()`

Renvoie la surface du rectangle appelant.

`double périmètre()`

Renvoie le périmètre du rectangle appelant.

`Rectangle intersection(Rectangle r)`

Renvoie le rectangle intersection du rectangle appelant et du rectangle `r`. Si l'intersection est vide, la méthode renvoie la référence `null`.