Exercices sur la surcharge de méthodes et l'héritage

Fabrice Rossi

29 novembre 1998

Exercice 1 :

On considère la classe `Calcul` :

```
 1 :  public class Calcul {
 2 :    /* 1 */
 3 :    public static byte sqr(byte b) {
 4 :      return (byte)(b*b) ;
 5 :    }
 6 :    /* 2 */
 7 :    public static int sqr(int b) {
 8 :      return b*b ;
 9 :    }
10 :    /* 3 */
11 :    public static double sqr(double b) {
12 :      return b*b ;
13 :    }
14 :    public static void main(String[] args) {
15 :      byte a=sqr(2) ;
16 :      byte b=2,c=4 ;
17 :      byte d=sqr(b) ;
18 :      byte e=sqr(b+c) ;
19 :      int i=sqr(4) ;
20 :      double x=sqr(2) ;
21 :      double y=sqr(3.5) ;
22 :      double z=sqr(2.7f) ;
23 :    }
24 :  }
```

Pour chaque appel d'une méthode `sqr`, indiquez si l'appel est possible et quelle est dans ce cas la méthode utilisée.

Exercice 2 :

On considère le programme suivant :

```
1 :  class Base {
2 :    double divise(double a,double b) {
3 :      return a/(a+b) ;
```

```
 4 :     }
 5 :     double calcul(double a,double b) {
 6 :       return (a+b)/(a-b) ;
 7 :     }
 8 :     double combine(double a) {
 9 :       return divise(a+1,a-1) ;
10 :     }
11 :   }
12 :
13 :   class WithParam extends Base {
14 :     private double scal ;
15 :     WithParam(double val) {
16 :       scal=val ;
17 :     }
18 :     double divise(double a,double b) {
19 :       return scal*a/(a+b) ;
20 :     }
21 :   }
22 :
23 :   class MoreParam extends WithParam {
24 :     private double translate ;
25 :     MoreParam(double val,double trans) {
26 :       super(val) ;
27 :       translate=trans ;
28 :     }
29 :     double calcul(double a,double b) {
30 :       return super.calcul(a,b)+translate ;
31 :     }
32 :   }
33 :
34 :   class Compose extends Base {
35 :     private Base base ;
36 :     Compose(Base theBase) {
37 :       base=theBase ;
38 :     }
39 :     double divise(double a,double b) {
40 :       return base.divise(a,base.divise(a,b)) ;
41 :     }
42 :   }
43 :
44 :   public class Redefinition {
45 :     public static void main(String[] args) {
46 :       Base b=new Base() ;
```

```
47 :      System.out.println(b) ;
48 :      System.out.println(b.divise(3,5)) ;
49 :      System.out.println(b.calcul(3,5)) ;
50 :      System.out.println(b.combine(2)) ;
51 :      WithParam c=new WithParam(2) ;
52 :      System.out.println(c) ;
53 :      System.out.println(c.divise(3,5)) ;
54 :      System.out.println(c.calcul(3,5)) ;
55 :      System.out.println(c.combine(2)) ;
56 :      MoreParam d=new MoreParam(3,4) ;
57 :      System.out.println(d) ;
58 :      System.out.println(d.divise(3,5)) ;
59 :      System.out.println(d.calcul(3,5)) ;
60 :      System.out.println(d.combine(2)) ;
61 :      Compose e=new Compose(b) ;
62 :      System.out.println(e) ;
63 :      System.out.println(e.divise(3,5)) ;
64 :      System.out.println(e.calcul(3,5)) ;
65 :      System.out.println(e.combine(2)) ;
66 :      Compose f=new Compose(d) ;
67 :      System.out.println(f) ;
68 :      System.out.println(f.divise(3,5)) ;
69 :      System.out.println(f.calcul(3,5)) ;
70 :      System.out.println(f.combine(2)) ;
71 :    }
72 :  }
```

Indiquez l'affichage produit par le programme en justifiant vos réponses (c'est-à-dire en indiquant avec précision quelles sont les méthodes appelées lors de chaque affichage).

On remplace la ligne 67 par :

```
Base f=new Compose(d) ;
```

Donnez, en justifiant votre réponse, l'affichage produit par les dernières lignes du programme.

Exercice 3 :

On considère le programme suivant :

```
class A {}
class B extends A {}
class C extends B {}
class D extends C {}

class W {
  void foo(D d) {
    System.out.println("D") ;
  }
```

```
}

class X extends W {
  void foo(A a) {
    System.out.println("A");
  }
  void foo(B b) {
    System.out.println("X.B");
  }
}

class Y extends X {
  void foo(B b) {
    System.out.println("Y.B");
  }
}

class Z extends Y {
  void foo(C c) {
    System.out.println("C");
  }
}

public class CallSelection {
  public static void main(String[] args) {
    Z z=new Z();
    C c=new C();
    ((X) z).foo(c);
    z.foo(c);
    // ((W) z).foo(c);
    z.foo((B) c);
    // z.foo((D) c);
    D d=new D();
    // z.foo(d);
    // ((X) z).foo(d);
    ((W) z).foo(d);
    z.foo((C) d);
    ((Y) z).foo((C) d);
  }
}
```

Indiquez l'affichage produit par le programme. Expliquez pourquoi les lignes mises en commentaires correspondent à des appels incorrects.

Exercice 4 :

On souhaite stocker les informations suivantes pour un ensemble de personnes : nom, prénom, date de naissance et sexe. Proposer une représentation par une classe usuelle, donc chaque instance permet de stocker les informations relatives à une personne. Proposer ensuite une solution utilisant l'héritage pour représenter le sexe. Quels avantages possède cette solution ?

Exercice 5 :

On souhaite stocker des informations sur l'ensemble des employés d'une entreprise. Chaque employé est caractérisé par son nom, son numéro de bureau et son salaire annuel. Certains employés sont des cadres et dirigent une équipe constituée d'autres employés.

1. Proposer une classe `Employe` représentant un employé non cadre.
2. Proposer ensuite une classe `Cadre` héritant de `Employe` et permettant de représenter un employé cadre.
3. Proposer une classe `Entreprise` dont chaque instance contient une liste d'employés. On programmera les méthodes suivantes :
   (a) ajout d'un employé ;
   (b) suppression d'un employé désigné par son nom ;
   (c) calcul du nombre de cadres et du nombre de non cadres ;
   (d) calcul de la liste des employés occupant un bureau donné ;
   (e) calcul de la liste des cadres dont dépend un employé (désigné par son nom) ;
   (f) calcul du salaire moyen des deux catégories d'employés.

Exercice 6 :

On souhaite réaliser les éléments de base d'un éditeur de texte.

1. Proposer une classe `Position` qui représente une position dans un texte. Cette position est constituée du numéro de la ligne considérée (toujours strictement positif) et du numéro du caractère dans cette ligne (numéroté à partir de 0).
2. Proposer une classe `Curseur` qui possède une `Position` modifiable.
3. On suppose donnée une classe `Texte` de squelette suivant :

```
public class Texte {
  /**
   * Renvoie le caractère situé en position p.
   * @param p position étudiée
   * @return le caractère
   */
  public char getChar(Position p) { }
  /**
   * Renvoie la ligne numéro n sous forme d'une chaîne de caractères
   * @param n numéro de la ligne
   * @return la ligne
   */
  public String getLigne(int n) { }
  /**
```

```
   * Renvoie sous forme d'une chaîne de caractères le texte compris entre la
   * position begin (incluse) et la position end (non incluse)
   * @param begin début du texte
   * @param end fin du texte
   * @return le texte
   */
  public String getSubText(Position begin,Position end) {}
  /**
   * Renvoie sous forme d'une chaîne de caractères le texte compris entre la
   * position begin (incluse) et la position end (non incluse), et supprime ce
   * texte de celui stocké dans l'objet appelant
   * @param begin début du texte
   * @param end fin du texte
   * @return le texte
   */
  public String cutSubText(Position begin,Position end) {}
  /**
   * Insére à partir de la position where le texte contenu dans le paramètre
   * text. Le premier caractère de ce texte devient celui de position where.
   * @param where emplacement de l'insertion
   * @param text texte à insérer
   */
  public void insertSubText(Position where,String text) {}
}
```

On souhaite réaliser sur une instance de cette classe des modifications produites par un éditeur de texte. Chaque modification implantera l'interface suivante :

```
public interface Modification {
  public void do(Position p,Curseur c,Texte t);
}
```

Le but de la méthode `do` est de modifier le texte courant et éventuellement la position du curseur. On programmera les modifications suivantes :

(a) `Avance` : cette modification ne transforme pas le texte, elle se contente de faire passer le curseur sur le caractère suivant. Pour ce faire, elle augmente de 1 le numéro du caractère. Si ce numéro devient strictement supérieur au nombre de caractères présents dans la ligne courante, on passe à la ligne suivante.

(b) `Recule` : même opération, mais en faisant passer le curseur sur le caractère précédent.

(c) `Insère` : cette modification hérite de `Avance` et prend comme paramètre de création un caractère à insérer dans le texte. Elle insère à la position du curseur le caractère en question, puis avance le curseur d'une position.

(d) `Backspace` : cette modification hérite de recule. Elle faite reculer le curseur d'un cran, puis supprime le caractère situé à la nouvelle position.

4. Proposer une autre organisation qui sépare les mouvements du curseur des modifications du texte. Comparer les deux techniques.