Université Paris 7 — Java
Licence Informatique — Année 2005-2006

TD n°7 - Correction

TD sur machine, héritage

Exercice 1 La commande **javap** permet de désassembler le code produit par le compilateur **javac**. Cela peut vous permettre d'observer concrêtement ce que produit implicitement la compilation.

Par exemple, si vous définissez une classe qui n'a aucun constructeur, nous avons vu en cours que la compilation avec **javac** générait un constructeur par défaut.

De même, si dans le constructeur d'une sous-classe vous n'effectuez aucun appel explicite au constructeur de la super-classe, le code produit par la compilation de la sous-classe pour le constructeur comprendra comme première instruction l'appel du constructeur sans paramètre de la super-classe.

1. Ecrivez le code suivant, issu du TD 6. Sauvegardez dans un fichier, et compilez-le avec **javac**.

```
class Ex1b{
    int a ;
}

class Ex1b2 extends Ex1b{
    int b ;
    Ex1b2(int a , int b){
        this.a = a ;
        this.b = b ;
    }
}
```

2. Désassemblez le fichier Ex1b.class produit par la compilation en tapant **javap Ex1b**. Vous obtenez la description de la classe Ex1b telle qu'elle a été compilée par **javac**. De quelle classe Ex1b est-elle implicitement une sous-classe ? Repérez le prototype du constructeur par défaut.
Correction :

```
Compiled from "Ex1b2.java"
class Ex1b extends java.lang.Object{
    int a;
    Ex1b();
}
```

3. De même, désassemblez le fichier Ex1b2.class. Utilisez ensuite la commande **javap -c Ex1b2**, ce qui vous permettra d'obtenir le détail du code exécuté par le constructeur, sous forme d'instructions élémentaires de la machine virtuelle. Il n'est pas nécessaire que vous compreniez le détail de toutes ces instructions. Quelle ligne vous semble être l'invocation du constructeur par défaut de Ex1b dans le constructeur de Ex1b2 ?
Correction :

```
> javap Ex1b2
Compiled from "Ex1b2.java"
class Ex1b2 extends Ex1b{
    int b;
    Ex1b2(int,int);
}
> javap -c Ex1b2
Compiled from "Ex1b2.java"
class Ex1b2 extends Ex1b{
int b;

Ex1b2(int,int);
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method Ex1b."<init>":()V
   4:   aload_0
   5:   iload_1
   6:   putfield        #2; //Field a:I
   9:   aload_0
   10:  iload_2
   11:  putfield        #3; //Field b:I
   14:  return

}
```

4. Que pouvez vous en déduire sur une des toutes premières instructions du constructeur de Ex1b ? Vérifiez votre hypothèse à l'aide de la commande **javap -c Ex1b**.
   Correction : Le constructeur généré de Ex1b appellera le constructeur de la classe Object.

```
> javap -c Ex1b
Compiled from "Ex1b2.java"
class Ex1b extends java.lang.Object{
int a;

Ex1b();
  Code:
   0:   aload_0
   1:   invokespecial   #1; //Method java/lang/Object."<init>":()V
   4:   return

}
```

Exercice 2 Cet exercice est une reformulation du deuxième exercice du TD 6.

1. Ecrivez une classe `Figure`. Cette classe a des attributs privés abscisse et ordonnee, ainsi qu'une couleur. Les couleurs seront représentées par des objets de la classe `Color` que vous pouvez utiliser à condition de mettre en tête de votre fichier : `import java.awt.Color;`. Le constructeur de la classe `Figure` instanciera chaque attribut privé. La méthode `toString()` permettra de donner une représentation de l'objet sous forme de chaîne de caractères. Vous regarderez la documentation de la classe `java.awt.Color` pour choisir la constante correspondant à votre couleur préférée pour faire un test de votre classe.
   Correction :

```
import java.awt.Color ;
```

```
public class Figure{
    private int abscisse ;
    private int ordonnee ;
    private Color couleur ;

    public Figure(int abscisse , int ordonnee , Color couleur){
        this.abscisse = abscisse ;
        this.ordonnee = ordonnee ;
        this.couleur = couleur ;
    }
    public String toString(){
        return ("abscisse: " + abscisse + " ordonnee: " + ordonnee + " couleur: " + couleur) ;
    }
    public static void main(String[] argv){
        Figure f = new Figure(12 , 13 , Color.cyan) ;
        System.out.println(f) ;
    }
}
```

2. Ecrivez deux classes `Carre` et `Rectangle` qui héritent de `Figure`. `Carre` a un attribut côté et `Rectangle` a des attributs hauteur et longueur. Comme d'habitude, testez vos classes.
   Correction :

```
//fichier Carre.java
import java.awt.Color ;

public class Carre extends Figure{
    private int cote ;

    public Carre( int abscisse , int ordonnee , Color couleur , int cote ){
        super( abscisse , ordonnee , couleur ) ;
        this. cote = cote ;
    }
    public String toString(){
        return ( super.toString() + " cote: " + cote ) ;
    }
    public static void main(String[] argv){
        Cercle c = new Cercle(12,10,Color.cyan, 50) ;
        System.out.println(c) ;
    }
}
//fichier Rectangle.java
public class Rectangle extends Figure{
    private int hauteur ;
    private int longueur ;

    public Rectangle( int abscisse , int ordonnee , Color couleur , int hauteur , int longueur
        super( abscisse , ordonnee , couleur ) ;
        this.hauteur = hauteur ;
        this.longueur = longueur ;
    }
    public String toString(){
        return ( super.toString() + " hauteur: " + hauteur + " longueur: " + longueur ) ;
    }
```

```
    public static void main(String[] argv){
        Rectangle r = new Rectangle( 12 , 10 , Color.cyan , 50 , 100 ) ;
        System.out.println(r) ;
    }
}
```

3. Ajoutez à `Figure` une variable de classe privée `Vector` qui devra référencer **toutes les instances** de sa classe et de ses sous classes. Comment allez vous remplir ce vecteur ? Vous définirez également une méthode publique `getInstances()` qui permettra d'accéder à ce vecteur.
   Correction :

```
//Fichier Figure.java
import java.awt.Color ;
import java.util.Vector ;
public class Figure{
    private static Vector instances = new Vector() ;
    private int abscisse ;
    private int ordonnee ;
    private int couleur ;

    public Figure( int abscisse , int ordonnee , Color couleur ){
        this.abscisse = abscisse ;
        this.ordonnee = ordonnee ;
        this.couleur = couleur ;
        instances.add(this) ;
    }
    public static Vector getInstances(){
        return instances ;
    }
    public String toString(){
        return (abscisse + " " + ordonnee + " " + couleur) ;
    }
    public static void main(String[] argv){
        Figure f = new Figure(12 , 13 , Color.cyan) ;
        System.out.println((f.getInstances()).size()) ;
    }
}
```

4. Dans `Carre` et `Rectangle`, redéfinissez cette méthode `getInstances()` de manière à ne récupérer que les instances qui correspondent à leur type. Il n'est pas nécessaire d'ajouter un attribut à ces classes ou de modifier leur constructeur.

Correction : On déduira aisément une solution pour la classe Rectangle d'une solution pour la classe Carre :

```
public static Vector getInstances(){
      int nCarre = 0 ;
      Vector instancesCarre = new Vector() ;
      Vector instances = Figure.getInstances() ; // static pas de super
      Enumeration e = instances.elements() ;
      Figure uneFigure ;
      while( e.hasMoreElements()){
          uneFigure = (Figure) e.nextElement() ;
          if (uneFigure instanceof Carre)
```

```
                instancesCarre.add(uneFigure) ;
        }
        return instancesCarre ;
    }

//Fichier Test.java
import java.util.Vector ;
import java.util.Enumeration ;
public class Test{
    public static void main(String[] argv){
        Figure f1 = new Figure(1,1,Color.cyan) ;
        Figure f2 = new Figure(2,2,Color.red) ;
        Carre c3 = new Carre(3,3,Color.green,3) ;
        Figure f4 = new Figure(4,4,Color.pink) ;
        Carre c5 = new Carre(5,5,5,Color.black) ;


        System.out.println("Liste des figures") ;
        Enumeration e = Figure.getInstances().elements() ;
        while(e.hasMoreElements()){
            System.out.println(e.nextElement()) ;
        }
        System.out.println("Liste des carres") ;
        e = Carre.getInstances().elements() ;
        while(e.hasMoreElements()){
            System.out.println(e.nextElement()) ;
        }
    }
}
```

Exercice 3 [Evaluateur d'expressions booléennes] **Pour ceux qui ont encore du temps** L'objectif de cet exercice est d'écrire un programme prenant une expression booléennes en notation postfixe en argument, affichant l'expression en ordre infixe (avec parenthèses) puis affichant le résultat de l'évaluation de cette expression.
Par exemple :

`java Eval 1 0 + 1 .`

affichera :

`(true.(false+true)) = true`

On utilise des booléens comme valeurs et les opérateurs sont : +(ou), .(et) et -(non). On définira une interface ExprBool ainsi que les classes suivantes : `Eval`, `OpBin`, `OpUn`, `Ou`, `Et`, `Non` et `MyBoolean`. Pour la pile, on pourra utiliser la classe prédéfinie `Stack`.

1. Ecrire le graphe d'héritage des classes demandées.
2. Ecrire l'interface `Expr` qui spécifie deux méthodes : `public boolean eval()` et `public String toString()`.
   Correction :

```
interface Expr {
    public boolean eval();
    public String toString();
}
```

3. Ecrire la classe `MyBoolean` qui encapsule un booléen et implémente l'interface `Expr`. Pour cette classe comme pour les suivantes, il faudra écrire le(s) constructeur(s) dont vous avez besoin.
   Correction :

```
class MyBoolean implements Expr {
    private boolean b;

    public MyBoolean (boolean b) { this.b = b; }
    public boolean eval() { return b; }
    public String toString() { return Boolean.toString(b); }
}
```

4. Ecrire les classes abstraites `OpBin`, `OpUn` qui contiennent les champs, constructeurs et méthodes communes respectivement aux opérateurs binaires et unaires.
   Correction :

```
abstract class OpUn implements Expr {
    private Expr child;

    public OpUn (Expr child) { this.child = child; }
    public boolean eval(){ return op(child.eval()); }
    public String toString() { return symbol()+"("+child.toString()+")"; }
    abstract public boolean op(boolean b);
    abstract public String symbol();
}

abstract class OpBin implements Expr {
    private Expr lc;
    private Expr rc;

    public OpBin(Expr lc, Expr rc) { this.lc = lc; this.rc = rc; }
    public boolean eval(){ return op(lc.eval(), rc.eval()); }
    public String toString() { return "("+lc.toString()+symbol()+rc.toString()+")"; }
    abstract public boolean op(boolean a, boolean b);
    abstract public String symbol ();
}
```

5. Ecrire les classes `Ou`, `Et` et `Non`.
   Correction :

```
class Non extends OpUn {
    public Non(Expr e) { super(e); }
    public boolean op(boolean b) { return !b; }
    public String symbol() { return "-"; }
}

class Ou extends OpBin {
    public Ou(Expr a, Expr b) { super(a, b); }
    public boolean op(boolean a, boolean b) { return a || b; }
    public String symbol() { return "+"; }
}

class Et extends OpBin {
    public Et(Expr a, Expr b) { super(a, b); }
```

```
    public boolean op(boolean a, boolean b) { return a && b; }
    public String symbol() { return "."; }
}
```

6. Ecrire la classe `Eval` qui contiendra la méthode `main` et tout ce qui est nécessaire pour construire un arbre à partir des arguments de la ligne de commande.
   Correction : Pour connaître votre version de JAVA, vous pouvez utiliser la commande : **java -version**.

```
class Eval {
    // Cette version de la methode createExpr fonctionne
    // avec les versions 1.4 et anterieures de JAVA
     private static Expr createExpr(String[] args) {
Stack pile = new Stack();
for (int i = 0; i < args.length; i++){
    switch (args[i].charAt(0)) {
    case '0': pile.push(new MyBoolean(false)); break;
    case '1': pile.push(new MyBoolean(true)); break;
    case '+': pile.push (new Ou((Expr)pile.pop(), (Expr)pile.pop())); break;
    case '.': pile.push (new Et((Expr)pile.pop(), (Expr)pile.pop())); break;
    case '-': pile.push(new Non((Expr)pile.pop())); break;
    default: System.out.println ("Erreur: expression non reconnue");
System.exit(0);
    }
}
return (Expr)pile.peek();
    }

    public static void main (String[] args) {
Expr expr = createExpr(args);
System.out.println (expr.toString()+" = "+expr.eval());
    }
}
```

Correction :

```
    // Cette version de la methode createExpr fonctionne
    // avec la version 1.5 de JAVA
    private static Expr createExpr(String[] args) {
Stack<Expr> pile = new Stack<Expr>();
for (int i = 0; i < args.length; i++){
    switch (args[i].charAt(0)) {
    case '0': pile.push(new MyBoolean(false)); break;
    case '1': pile.push(new MyBoolean(true)); break;
    case '+': pile.push (new Ou(pile.pop(), pile.pop())); break;
    case '.': pile.push (new Et(pile.pop(), pile.pop())); break;
    case '-': pile.push(new Non(pile.pop())); break;
    default: System.out.println ("Erreur: expression non reconnue");
System.exit(0);
    }
}
return pile.peek();
    }
```