

# COMPTE RENDU LAB3

## Date

```
1+ /**  
4 package gse.labs.java;  
5  
6- /**  
7  * @author romain  
8  *  
9  */  
10 public class Date {  
11     private int day;  
12     private int month;  
13     private int year;  
14  
15-     public Date(int d, int m, int y) {  
16         day = d;  
17         month = m;  
18         year = y;  
19     }  
20  
21 }  
22
```

Dans le main :

```
Date date1 = new Date(5, 2, 2016);  
System.out.println("getClass date1 -> "+date1.getClass());  
  
Constructor<?> c[] = date1.getClass().getConstructors();  
for(int i = 0; i < c.length; i++) {  
    System.out.println(c[i]);  
}
```

Output :

```
getClass date1 -> class gse.labs.java.Date  
public gse.labs.java.Date(int,int,int)
```

La classe Date hérite automatiquement de la classe par défaut *java.lang.Object*. Cette classe contient la méthode *getClass()* donc notre classe Date est capable d'appeler cette méthode.

## toString

Dans le main :

```
Date date2 = new Date (5, 2, 2016);
System.out.println("date2 -> "+date2);
System.out.println(date2.getClass());
```

Output :

```
date2 -> gse.labs.java.Date@d716361
class gse.labs.java.Date
```

Il serait plus pratique de pouvoir afficher la date lorsqu'on fait un *print* d'un objet *Date*. Ici on voit que cette opération nous renvoie l'adresse de l'objet *Date*. C'est parce que cette opération appelle la fonction *toString()* définie dans la classe mère. Si on veut changer l'affichage, il faut réécrire cette fonction.

## SuperDate

```
1  /**
4  package gse.labs.java;
5
6  /**
7   * @author romain
8   *
9   */
10 public class SuperDate {
11     private int day;
12     private int month;
13     private int year;
14
15     public SuperDate(int d, int m, int y) {
16         day = d;
17         month = m;
18         year = y;
19     }
20
21     @Override
22     public String toString() {
23         return month+"/"+day+"/"+year;
24     }
25
26
27 }
28
```

Dans le main :

```
SuperDate date3 = new SuperDate (5, 2, 2016);  
System.out.println("date3 -> "+date3);  
System.out.println(date3.getClass());
```

Output :

```
date3 -> 2/5/2016  
class gse.labs.java.SuperDate
```

On peut voir maintenant que l'affichage est beaucoup plus *user friendly*.

## equals

Dans le main :

```
SuperDate date4 = new SuperDate (5, 2, 2016);  
SuperDate date5 = new SuperDate (5, 2, 2016);  
if(date4.equals(date5))  
    System.out.println(date4+" is equal to "+date5);  
else  
    System.out.println(date4+" is not equal to "+date5);
```

Output :

```
2/5/2016 is not equal to 2/5/2016
```

On peut voir que l'appel de la fonction *equals* fait référence à l'implémentation de la classe par défaut. Cette implémentation ne correspond pas au concept d'égalité dans le cas de notre classe. Il faut surcharger cette méthode pour obtenir un résultat correct.

## GreatDate

```
1  /**
4  package gse.labs.java;
5
6  /**
7   * @author romain
8   *
9   */
10 public class GreatDate {
11     private int day;
12     private int month;
13     private int year;
14
15     public GreatDate(int d, int m, int y) {
16         day = d;
17         month = m;
18         year = y;
19     }
20
21     @Override
22     public String toString() {
23         return month+"/"+day+"/"+year;
24     }
25
26     @Override
27     public boolean equals(Object obj) {
28         if (getClass() == obj.getClass()) {
29             GreatDate d = (GreatDate)obj;
30             if(day != d.day || month != d.month || year != d.year)
31                 return false;
32             return true;
33         }
34         return false;
35     }
36 }
37
```

Dans le main :

```
GreatDate date6 = new GreatDate (5, 2, 2013);
GreatDate date7 = new GreatDate (5, 2, 2013);
if(date6.equals(date7))
    System.out.println(date7+" is equal to "+date6);
else
    System.out.println(date7+" is not equal to "+date6);

System.out.println(date6.equals(date7));
System.out.println(date6.equals(0));
```

Output :

```
2/5/2013 is equal to 2/5/2013
true
false
```

On peut voir qu'avec la nouvelle implémentation de *equals*, on peut tester correctement l'égalité de deux dates.

On remarque que l'argument de la fonction *equals* est de type *Object*, il faut donc tester le type de cet argument avant de tester l'égalité.

## Rectangle

```
1  /**
4  package gse4.labs.java;
5
6  /**
7   * @author romain
8   *
9   */
10 public class Rectangle {
11     protected double h_;
12     protected double w_;
13
14     protected static int cnt = 0;
15     protected int uid_;
16
17     public Rectangle (double h, double w) {
18         h_ = h;
19         w_ = w;
20         ++cnt;
21         uid_ = cnt;
22     }
23
24     public double area () { return h_*w_; }
25     public double perimeter () { return 2*h_+2*w_; }
26
27     public double getHeight() { return h_; }
28     public double getWidth() { return w_; }
29     public static int getCnt() { return cnt; }
30     public int getUid() { return uid_; }
31
32     public void setHeight(double h_) { this.h_ = h_; }
33     public void setWidth(double w_) { this.w_ = w_; }
34     public void setUid(int uid_) { this.uid_ = uid_; }
35 }
```

```

37-   protected String computeValues() {
38       String s = "";
39       s+="\t-id " + uid_+"\n";
40       s+="\t-height " + w_+"\n";
41       s+="\t-width " + w_+"\n";
42       s+="\t-perimeter " + perimeter()+"\n";
43       s+="\t-area " + area()+"\n";
44       return s;
45   }
46-   @Override
47   public String toString() {
48       String s = "Rectangle:\n";
49       s+=computeValues();
50       return s;
51   }
52
53
54

```

## Square

```

1- /**
4   package gse4.labs.java;
5
6- /**
7   * @author romain
8   *
9   */
10  public class Square extends Rectangle {
11
12
13-   public Square(double s) {
14       super(s, s);
15   }
16
17   public void setHeight(double h_) {throw new UnsupportedOperationException();}
18   public void setWidth(double w_) {throw new UnsupportedOperationException();}
19   public void setUid(int uid_) {throw new UnsupportedOperationException();}
20
21-   public void setSide(double s) {
22       super.w_=s;
23       super.h_=s;
24   }
25   public double getSide() {return super.w_;}
26
27-   @Override
28   public String toString() {
29       String s = "Square:\n";
30       s+=computeValues();
31       return s;
32   }
33
34 }

```

## Main

```
10 public class TestShape {
11
12     /**
13      * @param args
14      */
15     public static void main(String[] args) {
16         Square sq1 = new Square(7.0);
17         double height = sq1.getHeight();
18         System.out.println("height: "+height);
19         double width = sq1.getWidth();
20         System.out.println("width: "+width);
21         double area = sq1.area();
22         System.out.println("area: "+area);
23
24         Rectangle rect1 = new Rectangle(4.0, 6.0);
25         System.out.println("rect1 = "+ rect1);
26         Rectangle rect2 = new Square(4);
27         System.out.println(rect2);
28
29         // Square sq2 = new Rectangle(4,4);
30         // System.out.println(sq2);
31
32         Square sq = new Square(2.0);
33         sq.setWidth(3.0);
34         System.out.println(sq);
35
36
37     }
38
39 }
```

## Output :

```
height: 7.0
width: 7.0
area: 49.0
rect1 = Rectangle:
  -id 2
  -height 6.0
  -width 6.0
  -perimeter 20.0
  -area 24.0
```

```
Square:
  -id 3
  -height 4.0
  -width 4.0
  -perimeter 16.0
  -area 16.0
```

```
Exception in thread "main" java.lang.UnsupportedOperationException
    at gse4.labs.java.Square.setWidth(Square.java:18)
    at gse4.labs.java.TestShape.main(TestShape.java:33)
```

La classe *Square* hérite de la méthode *toString* implémentée dans la classe *Rectangle*. Il faut surcharger cette méthode pour que l’affichage soit cohérent.

Pour que la classe *Square* puisse accéder aux attributs de *Rectangle*, il faut les déclarer *protected*.

On remarque que lorsqu’on affecte un *Square* à un *Rectangle*, ce dernier va appeler les fonctions implémentées dans *Square*. Il s’agit du polymorphisme.

Si on essaye de faire l’inverse (affecter un *Rectangle* à un *Square*), nous obtenons une erreur. En effet, un carré ne peut pas avoir les propriétés d’un rectangle. Le polymorphisme ne fonctionne pas dans ce sens là.

## Shape

```
1 ⊕ /**
4 package gse4.labs.java;
5
6 import java.awt.Color;
7
8 ⊖ /**
9  * @author romain
10  *
11  */
12 public abstract class Shape {
13     protected Color color;
14
15 ⊖     public Shape(Color c) {
16         color = c;
17     }
18
19     public Color getColor() { return color; }
20
21     // calculate and compute the area of the shape
22     public abstract double area();
23     // calculate and compute the perimeter of the shape
24     public abstract double perimeter();
25
26 }
```



## Rectangle

```

12 public class Rectangle extends Shape{
13     private double h_;
14     private double w_;
15
16
17     public Rectangle (double h, double w, Color c) {
18         super(c);
19         h_ = h;
20         w_ = w;
21     }
22
23     public double getHeight() { return h_; }
24     public double getWidth() { return w_; }
25
26     public void setHeight(double h_) { this.h_ = h_; }
27     public void setWidth(double w_) { this.w_ = w_; }
28
29     @Override
30     public double area () { return h_*w_; }
31     @Override
32     public double perimeter () { return 2*h_+2*w_; }
33
34     @Override
35     public String toString() {
36         String s = "Rectangle:\n";
37         s+="\t-color " + color+"\n";
38         s+="\t-height " + w_+"\n";
39         s+="\t-width " + w_+"\n";
40         s+="\t-perimeter " + perimeter()+"\n";
41         s+="\t-area " + area()+"\n";
42         return s;
43     }
44 }

```

## Circle

```

12 public class Circle extends Shape{
13     private double radius_;
14
15
16     public Circle(double r, Color c) {
17         super(c);
18         radius_ = r;
19     }
20
21     public double getRadius() { return radius_; }
22
23     @Override
24     public double area() {
25         return Math.PI * Math.pow(radius_, 2);
26     }
27     @Override
28     public double perimeter() {
29         return 2*Math.PI * radius_;
30     }
31
32     @Override
33     public String toString() {
34         String s = "Circle:\n";
35         s+="\t-color " + color+"\n";
36         s+="\t-radius " + radius_+"\n";
37         s+="\t-perimeter " + perimeter()+"\n";
38         s+="\t-area " + area()+"\n";
39         return s;
40     }
41
42 }

```

## Square

```
12 public class Square extends Shape {
13
14     private double side_;
15
16     public Square(double s, Color c) {
17         super(c);
18         side_ = s;
19     }
20
21     public void setSide(double s) { side_ = s; }
22     public double getSide() {return side_;}
23
24     @Override
25     public double area () { return side_*side_; }
26     @Override
27     public double perimeter () { return 4*side_; }
28
29     @Override
30     public String toString() {
31         String s = "Square:\n";
32         s+="\t-color "+ color+"\n";
33         s+="\t-side "+ side_+"\n";
34         s+="\t-perimeter "+ perimeter()+"\n";
35         s+="\t-area "+ area()+"\n";
36         return s;
37     }
38
39 }
```

## Main

```
11 public class TestAbsractShape {
12
13     /**
14      * @param args
15      */
16     public static void main(String[] args) {
17         // An array of 9 shapes
18         Shape[] shapes = new Shape[9];
19         for(int i = 0; i < 7; i=i+3){
20             shapes[i] = new Circle(4+i, Color.black);
21             shapes[i+1] = new Square(2+i, Color.blue);
22             shapes[i+2] = new Rectangle(1+i,5+i, Color.green);
23         }
24         for(int i = 0; i < 9; i++) {
25             System.out.println(shapes[i]);
26         }
27         // Shape shapeObj = new Shape(Color.yellow);
28
29     }
30
31 }
```

## Output :

```
Circle:
  -color java.awt.Color[r=0,g=0,b=0]
  -radius 4.0
  -perimeter 25.132741228718345
  -area 50.26548245743669

Square:
  -color java.awt.Color[r=0,g=0,b=255]
  -side 2.0
  -perimeter 8.0
  -area 4.0

Rectangle:
  -color java.awt.Color[r=0,g=255,b=0]
  -height 5.0
  -width 5.0
  -perimeter 12.0
  -area 5.0

Circle:
  -color java.awt.Color[r=0,g=0,b=0]
  -radius 7.0
  -perimeter 43.982297150257104
  -area 153.93804002589985
```

On peut voir ici que par polymorphisme, chaque instance de type *Shape* se retrouve attribuée à un *Circle*, *Square* ou *Rectangle*. Ces instances sont toujours de type *Shape*, mais font maintenant référence à un autre type.

La méthode *toString* appelée est donc celle définie dans la classe référencée et non la classe *Shape*.

On remarque que comme la classe *Shape* est abstraite, toute tentative de création d'un objet purement *Shape* se verra retourner une erreur.

## Drawable

```

1 1+ /**
4  package gse4.labs.java;
5
6 6- /**
7   * @author romain
8   *
9   */
10 public interface Drawable {
11 11- /**
12   * Draws an object using ASCII characters
13   */
14   public void drawWithASCII();
15 15- /**
16   * Draws an object using dots
17   */
18   public void drawWithDots();
19 }

```

## Rectangle

```

10 public class Rectangle implements Drawable{
11     private int h_;
12     private int w_;
13
14
15 15- public Rectangle (int h, int w) {
16     h_ = h;
17     w_ = w;
18 }
19
20 public int getHeight() { return h_; }
21 public int getWidth() { return w_; }
22
23 public void setHeight(int h_) { this.h_ = h_; }
24 public void setWidth(int w_) { this.w_ = w_; }
25
26 public int area () { return h_*w_; }
27 public int perimeter () { return 2*h_+2*w_; }
28
29 29- @Override
30 public String toString() {
31     String s = "Rectangle:\n";
32     s+="\t-height "+ w_+"\n";
33     s+="\t-width "+ w_+"\n";
34     s+="\t-perimeter "+ perimeter()+"\n";
35     s+="\t-area "+ area()+"\n";
36     return s;
37 }
38
39 39- @Override
40 public void drawWithASCII() {
41     System.out.print(" ");
42     for(int first_line = 0; first_line < w_; ++first_line)
43         System.out.print(" ");
44     System.out.println("");
45     for(int i = 0; i < h_-1; ++i) {
46         System.out.print("|");
47         for(int j = 0; j < w_; ++j)
48             System.out.print(" ");
49         System.out.println("|");
50     }
51     System.out.print("|");
52     for(int last_line = 0; last_line < w_; ++last_line)
53         System.out.print(" ");
54     System.out.println("|");
55 }
56
57 57- @Override
58 public void drawWithDots() {
59     for(int i = 0; i < h_; ++i) {
60         for(int j = 0; j < w_; ++j)
61             System.out.print(".");
62         System.out.println("");
63     }
64 }
65 }

```

## Cow

```

10 public class Cow implements Drawable {
11
12     private String nalme_;
13     private int weight_;
14
15     public Cow(String str, int weight) {
16         nalme_ = str;
17         weight_ = weight;
18     }
19
20
21     @Override
22     public void drawWithASCII() {
23         String fat = "";
24         for(int w = 0; w < weight_/2; ++w) {
25             fat+="  \\";
26             for (int f = 0; f < weight_; ++f)
27                 fat+=" ";
28             fat+=" )\n";
29         }
30
31         String line2 = " (oo)\\";
32         String line3=" ( _)\\";
33         String line4="  ||";
34         String line5 = "  ||";
35         for(int w = 0; w < weight_; ++w) {
36             line2+="- ";
37             line3+="- ";
38             line4+="- ";
39             line5+="- ";
40         }
41         line2+="__ \n";
42         line3+=" )\\\\\\\\\\\\\\\\\\n";
43         line4+="w | \n";
44         line5+=" || \n";
45         System.out.println(" ^ _ ^\n" +
46                             line2+
47                             line3 +
48                             fat+
49                             line4 +
50                             line5);
51     }
52
53     @Override
54     public void drawWithDots() {
55         // TODO Auto-generated method stub
56     }
57
58     /**
59      * @return the str_
60      */
61     public String getName() {
62         return nalme_;
63     }
64
65     /**
66      * @return the weight_
67      */
68     public int getWeight() {
69         return weight_;
70     }
71
72     /**
73      * @param str_ the str_ to set
74      */
75     public void setName(String str_) {
76         this.nalme_ = str_;
77     }
78
79     /**
80      * @param weight_ the weight_ to set
81      */
82     public void setWeight(int weight_) {
83         this.weight_ = weight_;
84     }
85 }

```

## Main


```

10 public class TestDrawable {
11
12     /**
13      * @param args
14      */
15     public static void main(String[] args) {
16         Drawable drawable1 = new Rectangle(6, 5);
17         System.out.println(drawable1.getClass());
18         drawable1.drawWithASCII();
19         drawable1.drawWithDots();
20         Drawable drawable2 = new Cow("Cow", 20);
21         System.out.println(drawable2.getClass());
22         drawable2.drawWithASCII();
23
24         System.out.println(drawable1.area());
25         System.out.println(((Rectangle)drawable1).area());
26         System.out.println(((Rectangle)drawable2).area());
27
28         if (drawable1 instanceof Rectangle)
29             System.out.println(((Rectangle)drawable1).area());
30         if (drawable2 instanceof Cow)
31             System.out.println(((Cow)drawable2).getName());
32     }
33 }

```

Output :

```
class gse4.labs.java.Rectangle
```



• • • • •  
• • • • •  
• • • • •  
• • • • •  
• • • • •  
• • • • •

```
.....
class gse4.labs.java.Cow
```

30  
Cow

On remarque que les objets de type *Drawable* ne sont pas capable d'accéder aux méthodes des Classes référencées. Par exemple, si on essaye d'accéder à la méthode *area()* depuis l'objet *drawable1*, on aura une erreur.

Pour pouvoir utiliser ces méthodes, il faut *caster* l'objet dans la classe qui nous intéresse, après avoir vérifier que l'instance que l'on veut *caster* référence bien le bon type. On utilise pour cela le mot clé *instanceof*.

## Phone

```
10 public interface Camera {
11     public void captureImage();
12     public void captureVideo();
13 }
14
```

```
10 public interface MediaPlayer {
11     public void playMovie(String name);
12     public void playSong(String name);
13     public void displayPicture(String name);
14 }
```

```
6 /**
7  * @author romain
8  *
9  */
10 public class Phone {
11
12     public Phone() {
13         System.out.println("Manufacturing your phone...");
14     }
15     public void voiceCall(String contact) {
16         System.out.println("Calling "+contact+"...");
17     }
18     public void sendSMS(String contact, String message) {
19         System.out.println("Sending \""+message+"\" to "+contact+"...");
20     }
21 }
```

```
10 public class Smartphone extends Phone implements Camera, MediaPlayer {
11
12     public Smartphone() { super(); }
13
14     @Override
15     public void playMovie(String name) {
16         System.out.println("playing movie \""+name+"\""...");
17     }
18
19     @Override
20     public void playSong(String name) {
21         System.out.println("playing music \""+name+"\""...");
22     }
23
24     @Override
25     public void displayPicture(String name) {
26         System.out.println("displaying picture \""+name+"\""...");
27     }
28
29     @Override
30     public void captureImage() {
31         System.out.println("taking a picture...");
32     }
33
34     @Override
35     public void captureVideo() {
36         System.out.println("taking a video...");
37     }
38 }
39 }
```

## Main

```
10 public class TestSmartPhone {
11
12     /**
13      * @param args
14      */
15     public static void main(String[] args) {
16         Smartphone mytel = new Smartphone();
17         mytel.sendSMS("0687374202", "I have a new phone !");
18         mytel.voiceCall("BFF");
19         mytel.captureVideo();
20         mytel.displayPicture("landscape");
21         mytel.playSong("wrecking ball");
22     }
23 }
24 }
```

Output :

```
Manufacturing your phone...
Sending "I have a new phone !" to 0687374202...
Calling BFF...
taking a video...
displaying picture "landscape"...
playing music "wrecking ball"...
```