(3): Sous typage et polymorphisme

- Un objet peut **appartenir à plus d'un type** : c'est le ***polymorphisme*** ; permet d'utiliser des objets de types différents là où est attendu un objet d'un certain type.
  Définition de [Armstrong, 2006] : « Different classes may respond to the same message and each implements it appropriately ».

Un exemple bien connu ...

- Dans les langages objets deux mécanismes du typage :
  - le *typage dynamique* : le type des objets est déterminé à l'exécution lors de la création desdits objets (SMALLTALK, COMMON LISP, PYTHON, PHP...) ;
  - le *typage statique* : le type des objets est vérifié à la compilation et est soit explicitement indiqué par le développeur lors de leur déclaration (C++, JAVA, C#, PASCAL...), soit déterminé par le compilateur à partir du contexte (SCALA, OCAML...).
- De même, deux mécanismes de sous-typage existent : *l'héritage simple* (SMALLTALK, JAVA, C#) et *l'héritage multiple* (C++, PYTHON, COMMON LISP, EIFFEL).

(3): Le polymorphisme (suite)

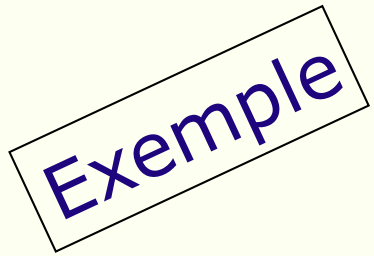
Exemple du périmètre d'un quadrilatère ou d'un rectangle :

```
class Test {
  public  static void main (String[] arguments) {
     Point a,b,c,d;
      . . .
     Quadrilatere q = new Quadrilatere(a,b,c,d);

     RectangleSpec r = new Rectangle(a,c);

     q = r;

 } }
```



(3): Polymorphisme (suite) - le typage récursif

- Possible dans d'autres langages mais adapté aux langages à objets

```
class Arbre {
  protected <T> valeur;
  protected Arbre filsGauche, filsDroit;
  public <T> valeur() { return valeur; }
  public boolean existeFilsGauche() {
  return filsGauche != null; }
  public boolean existeFilsDroit() { return filsDroit != null; }
  public Arbre filsGauche() { return filsGauche; }
  public Arbre filsDroit() { return filsDroit; }
  public void affecterValeur(<T> c) { valeur = c; }
  public void affecterFilsGauche(Arbre g) { filsGauche = g; }
  public void affecterFilsDroit(Arbre d) { filsDroit = d;}
  public boolean feuille() {return (filsDroit==null && filsGauche==null); }
}
```

pas expressément une question du polymorphisme mais du typage

Exemple

(4): Liaison dynamique

- Exemple du périmètre d'un quadrilatère ou d'un rectangle : (un exemple en Java)

Exemple

```
class Quadrilatere {
    protected Point p1, p2, p3, p4;

    public Quadrilatere(Point p, Point q, Point r, Point s) {
            p1= new Point(p); p2= new Point(q);
            p3= new Point(r); p4= new Point(s);
       }

    public double perimetre() {
          return p1.distance(p2)+p2.distance(p3)+p3.distance(p4)+p4.distance(p1);
       }
    public double aire() {
        return 0.;
       }
}
```

programmation incorrecte mais c'est juste pour comprendre

(4): Liaison dynamique (suite)

- Exemple du périmètre d'un quadrilatère ou d'un rectangle : (suite, en Java):

Exemple

```
class Rectangle extends Quadrilatere {
    protected double L,l;
    public Rectangle(Point bg, Point hd) {
        super(bg,new Point(hd.x(),bg.y()),hd,new Point(bg.x(),hd.y()));
        L = hd.x()-bg.x(); l = hd.y()-bg.y(); }

    public double perimetre() {
        return 2*(L+l)+.0001; } // pour différencier les deux méthodes
    public double aire() {
        return L*l; }
}
```

encore une programmation incorrecte
mais c'est juste pour comprendre

(4): Liaison dynamique (suite)

Exemple du périmètre d'un quadrilatère ou d'un rectangle :

```
class Test {
  public  static void main (String[] arguments) {
     Protected Point a,b,c,d;
a=new Point(100,200); b=new Point(200,200); c=new Point(200,250); d=new
Point(100,250);
System.out.print(a.toString()+b.toString()+c.toString()+d.toString());
     Quadrilatere q = new Quadrilatere(a,b,c,d);
     System.out.println("perimetre:"+q.perimetre()+"--- aire:"+q.aire());
     Rectangle r = new Rectangle(a,c);
     System.out.println("perimetre:"+r.perimetre()+"--- aire:"+r.aire());
     q = r;
     System.out.println("perimetre:"+q.perimetre()+"--- aire:"+q.aire());
 } }
```

Exemple (en Java)

(4): Liaison dynamique (suite)

Exemple du périmètre d'un quadrilatère ou d'un rectangle :

```
class Test {
  public  static void main (String[] arguments) {
     Point a,b,c,d;
a=new Point(100,200); b=new Point(200,200); c=new Point(200,250); d=new
Point(100,250);
System.out.print(a.toString()+b.toString()+c.toString()+d.toString());
     Quadrilatere q = new Quadrilatere(a,b,c,d);
     System.out.println("perimetre:"+q.perimetre()+"--- aire:"+q.aire());
     RectangleSpec r = new RectangleSpec(a,c);
     System.out.println("perimetre:"+r.perimetre()+"--- aire:"+r.aire());
     q = r; // q devient (dynamiquement) un rectangle
     System.out.println("perimetre:"+q.perimetre()+"--- aire:"+q.aire());
 } }
```

Exemple

```
>>> java Test
(100, 200) (200, 200) (200, 250) (100, 250)
perimetre : 300.0 --- aire : 0.0
perimetre : 300.0001 --- aire : 5000.0
perimetre : 300.0001 --- aire : 5000.0
```

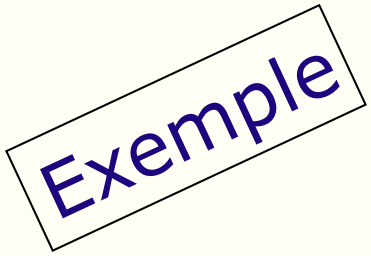
Quelques expériences sur le polymorphisme et la liaison dynamique

A.java

```
class A {
    public A() {
        System.out.println("je suis dans A, objet de la classe " + getClass().getName() );
    }
    public void aa() {
        System.out.println("execute la methode aa() de la classe "+getClass().getName() );
    }
}
```

B.java

```
class B extends A {
    public B() {
        super(); // est exécuté de toutes façons
        System.out.println("je suis dans B, objet de la classe " + getClass().getName() );
    }
    public void aa() {
        System.out.println("execute la methode aa() de la classe "+getClass().getName() );
    }
    public void bb() {
        System.out.println("execute la methode bb() de la classe "+getClass().getName() );
    }
}
```

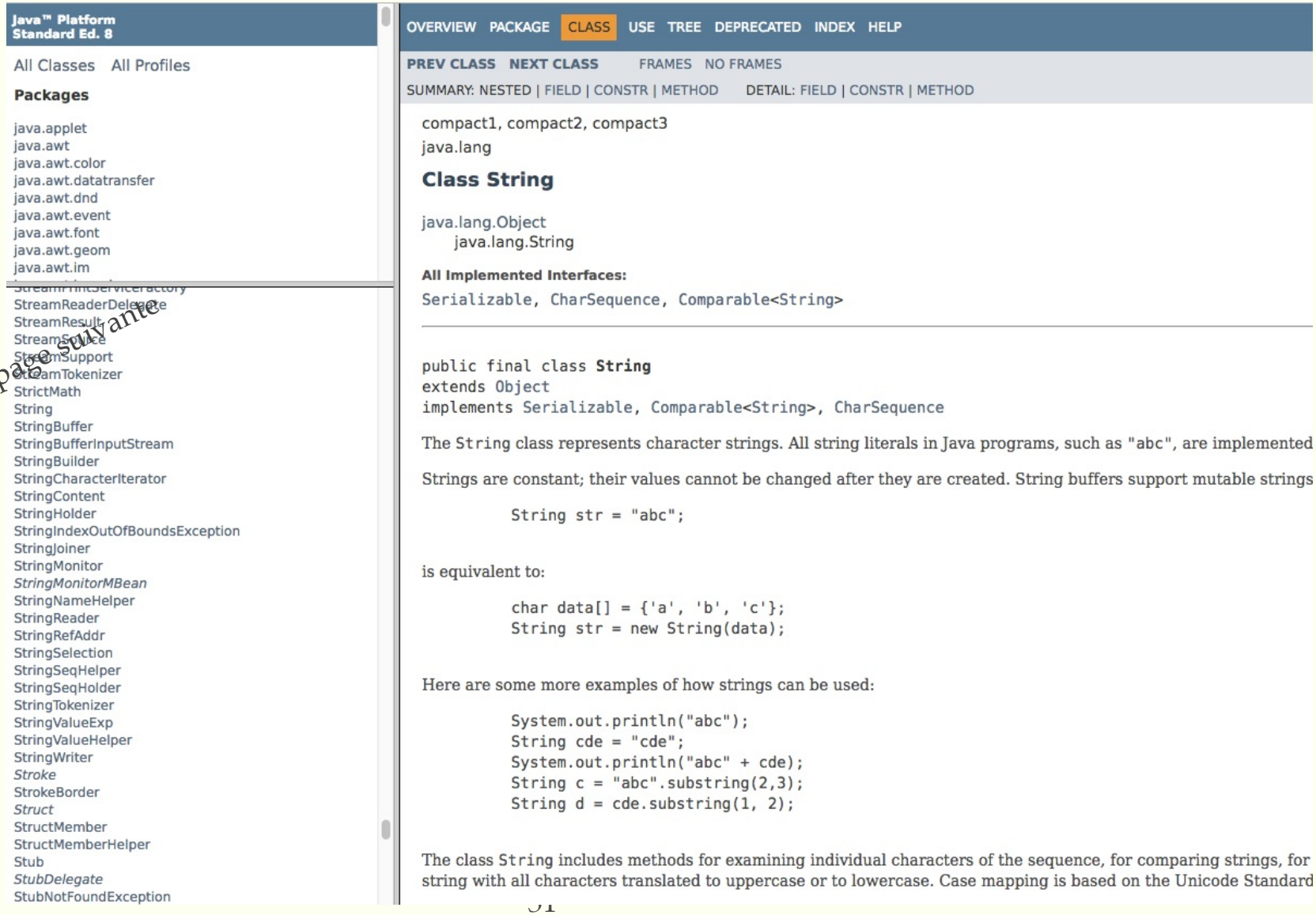


ScHeritage.java

Exemple d'une classe de la bibliothèque Java: la classe « `String` » héritant de « `Object` »

Exemple

plus lisible sur la page suivante



compact1, compact2, compact3

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>`

```
public final class String
extends Object
```

...able<String>, CharSequence

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

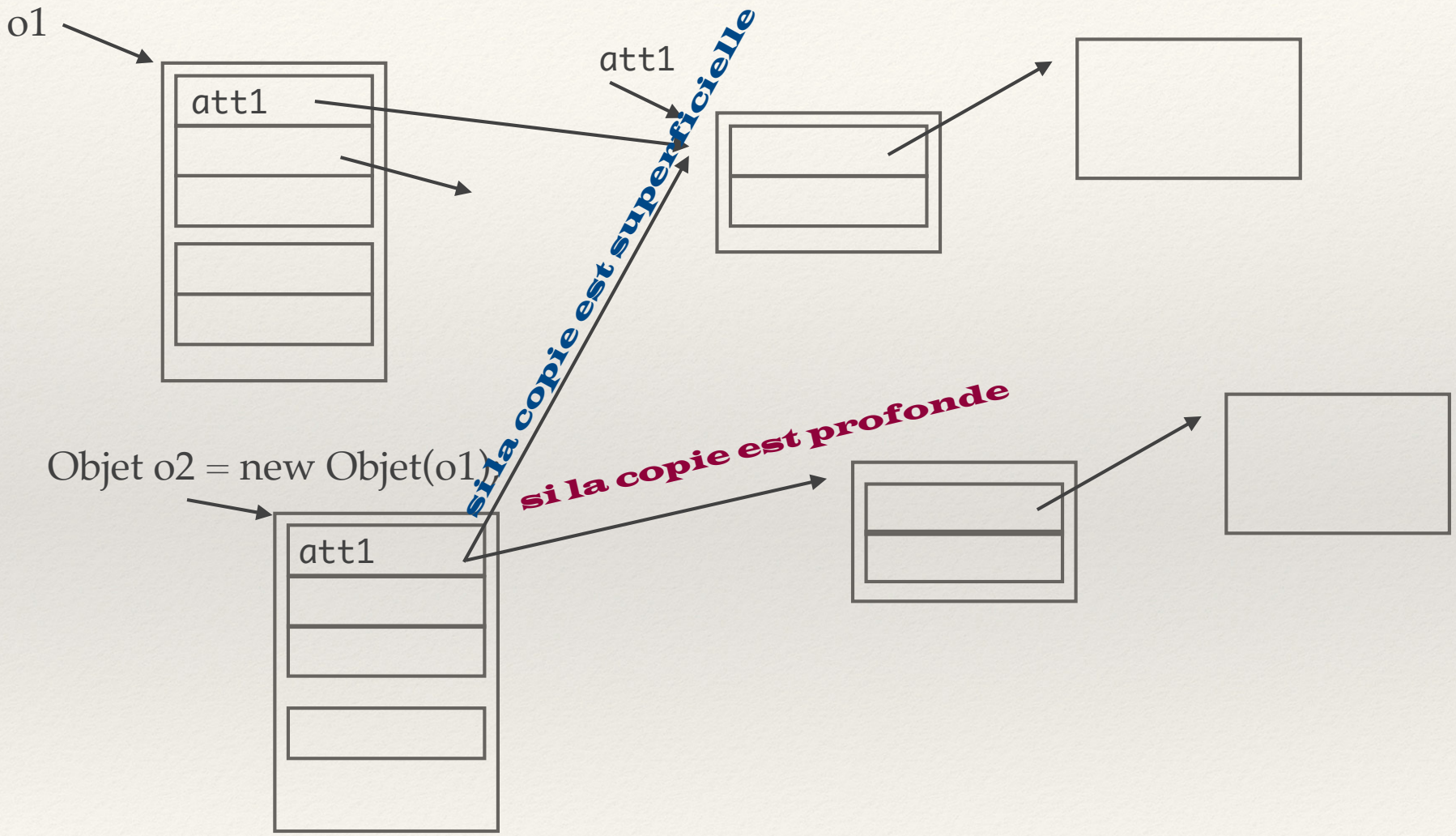
Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

The class String includes methods for examining individual characters of the sequence, for comparing strings, for string with all characters translated to uppercase or to lowercase. Case mapping is based on the Unicode Standard

Approfondissement : la copie d'objets

(copie **superficielle** / **profonde**)



Approfondissement : les objets ne sont pas des variables (distinction **type** / **classe**)

```
public class Point {
   protected int x;
   protected int y;
   public Point(int abs, int ord) {
      x = abs;
      y = ord;
   }
   public Point(Point p) {
    … // créer en copiant un autre objet Point
   }
   public int getX() {
    … // à compléter }
   public int getY() {
    … // à compléter }
   public void afficher()
{System.out.println("abs = "+x+", ord = "+y)}
   … // à compléter
}
```

```
public class Point {
   protected Integer x;
   protected Integer y;
   public Point(int abs, int ord) {
     x =  ??? ;
     y =  ??? ;
   }
   public Point(Point p) {
    … // créer en copiant un autre objet Point
   }
   public Integer getX() {
    … // à compléter }
   public Integer getY() {
    … // à compléter }
   public void afficher()
{System.out.println("abs = "+x+", ord = "+y)}
   … // à compléter
}
```

Approfondissement : les objets ne sont pas des variables (distinction **type** / **classe**)

```
public class Point {
   protected int x;
   protected int y;
   public Point(int abs, int ord) {
      x = abs;
      y = ord;
   }
   public Point(Point p) {
    … // créer en copiant un autre objet Point
   }
   public int getX() {
    … // à compléter }
   public int getY() {
    … // à compléter }
   public void afficher()
{System.out.println("abs = "+x+", ord = "+y)}
   … // à compléter
}
```

```
public class Point {
   protected Integer x;
   protected Integer y;
   public Point(int abs, int ord) {
     x =  new Integer(abs) ;
     y =  new Integer(ord) ;
   }
   public Point(Point p) {
    … // créer en copiant un autre objet Point
   }
   public Integer getX() {
    … // à compléter }
   public Integer getY() {
    … // à compléter }
   public void afficher()
{System.out.println("abs = "+x+", ord = "+y)}
   … // à compléter
}
```