Programmation Objet. Cours 2

Marie-Pierre Béal
UPEM DUT 1

Encapsulation et visibilité. Délégation. Immutabilité.
Expressions, types, variables. Attributs et méthodes statiques.

Encapsulation et visibilité

- En programmation objet, l'encapsulation consiste à protéger l'information contenue dans un objet et à ne proposer que des méthodes pour manipuler cet objet.
- En java on utilise des modificateurs de visibilité.
  - `public` : les attributs publics sont accessibles partout,
  - `protected` : les attributs protected sont accessibles seulement dans la classe elle-même et dans les classes dérivées ;
  - `private` : les attributs privés sont accessibles seulement dans la classe elle-même.

Par défaut les attributs sont visibles dans les classes du répertoire (ou du paquetage) mais on ne doit normalement pas laisser de visibilité par défaut.

```
public class Pixel {
  private int x;
  private int y;
  public int getX(){
    return x;
  }
  public int getY(){
    return y;
  }
  public void setX(int x){
       this.x = x;
  }
  public Pixel(int x, int y){
      this.x = x;
      this.y = y;
  }
}
```

Exemple : la classe PixelTest

```
public class PixelTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(2,3);
       System.out.println(p1.getX());
       System.out.println(p1.x); //erreur a la compilation
   }
}
```

Pas de constructeurs ?

```
public class Pixel {
  private int x;
  private int y;
}
```

Il existe un constructeur par défaut

```
public class PixelTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(); // le point (0,0) est cree
   }
}
```

Trop de constructeurs ?

On peut créer plusieurs constructeurs ou méthodes de même nom à condition que la suite des types des arguments soient différentes.

```
public class Pixel {
  private int x;
  private int y;
  public Pixel(int x, int y){ // (int, int)
      this.x = x;
      this.y = y;
  }
  public Pixel(int x){ // (int)
      this.x = x;
      y = 0;
  }
  public Pixel(){ // ()
  }
}
```

Plusieurs constructeurs

```
public class PixelTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(); // le point (0,0) est cree
       Pixel p2 = new Pixel(2,3); // le point (2,3) est cree
       Pixel p3 = new Pixel(2); // le point (2,0) est cree
   }
}
```

Des segments

On veut créer une classe `Segment` pour manipuler des segments formés de deux points.

```
public class Segment {
  private Pixel start;
  private Pixel end;
  public Segment(Pixel start, Pixel end){
      this.start = start;
      this.end = end;
  }
  public Pixel getStart() {
      return start;
  }
  public Pixel getEnd() {
      return end;
  }
}
```

Des segments

```
public class SegmentTest {
   public static void main(String[] args) {
       Pixel p1 = new Pixel(2,3);
       Pixel p2 = new Pixel(4,5);
       Segment s1 = new Segment(p1,p2);
       Segment s2 = new Segment(new Pixel(), new Pixel(2,2));
       System.out.println(s1.getStart().getX());
   }
}
```

Des segments

Plusieurs constructeurs dans la classe `Segment`.

```
public class Segment {
  private Pixel start;
  private Pixel end;
  public Segment(Pixel start, Pixel end){
      this.start = start;
      this.end = end;
  }
  public Segment(int x1, int y1, int x2, int y2){
      start = new Pixel(x1,y1);
      end = new Pixel(x2,y2);
  }
  ....
}
```

Des segments

```
public class SegmentTest {
   public static void main(String[] args) {
       Segment s1 = new Segment(new Pixel(), new Pixel(2,2));
       Segment s2 = new Segment(0,0,2,2);
       System.out.println(s1 == s2); // affiche false
   }
}
```

Expressions, types, variables

Toute expression a une valeur et un type. Les valeurs sont

- les valeurs primitives ;
- les références, qui sont des références à des tableaux ou à des objets.

Il existe une référence spéciale `null`. Elle peut être la valeur de n'importe quel type non primitif.

```
public class SegmentTest {
   public static void main(String[] args) {
       Segment s = null; //pas d'objet
       s.getStart(); //NullPointerException
   }
}
```

Un objet ne peut être manipulé, en Java, que par une référence.

Expressions, types, variables

Une *variable* est le nom d'un emplacement mémoire qui peut contenir une valeur. Le *type* de la variable décrit la nature des valeurs de la variable.

- Si le type est un type primitif, la valeur est de ce type.
- Si le type est une classe, la valeur est une référence à un objet de cette classe (ou d'une classe dérivée).

Les types primitifs sont `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`.

Immutabilité

Pour rendre une classe non mutable, on indique que les champs ne peuvent être modifiés avec le mot clé `final`.

```
public class Person {
   private final String firstName;
   private final String lastName;
   private int age;

   public Person(String firstName, String lastName, int age)
      this.firstName = firstName;
      this.lastName = lastName;
      this.age = age;
   }
}
```

Immutabilité

```
public class Person {
   private final String firstName;
   private final String lastName;
   private int age;
   public String getFirstName() {
      return firstName;
   }
   public void badMethod() {
      firstName = "toto"; //erreur
   }
}
```

```
public class PersonTest {
   public static void main(String[] args) {
      Person p = new Person("Philip","Mortimer",42);
}}
```

Immutabilité

Dans ce cas, il faut mettre des "getter" mais pas de "setter".

Les chaînes de caractères ne peuvent pas être modifiées en Java (comme en Python).

En Python les listes sont mutables, les tuples sont non mutables. En Java les tableaux, listes, ensembles, dictionnaires sont mutables.

Attributs statiques

Les attributs peuvent être

- des attributs de classe (`static`) ;
- des attributs d'objet (ou d'instance).

Les attributs de classe `static` sont partagés par tous les objets de la classe. Il n'en existe qu'un par classe au lieu de un pour chaque instance ou objet d'une classe lorsqu'il s'agit de membre d'objets.
Exemple d'attributs `static`

- un compteur du nombre d'objets d'une classe ;
- un élément particulier de la classe, par exemple une origine.

```
public class Pixel {
   private int x, y;
   public static Pixel origin = new Pixel(0,0);
}
```

Méthodes statiques

Les méthodes peuvent aussi être ou non `static`.
Les méthodes statiques sont appelées en donnant le nom de la classe ou le nom d'une instance de la classe. Une méthode statique ne peut pas faire référence à `this`.
Elles sont utiles pour fournir des services (helper).
On peut utiliser la méthode *static* `random` de la classe `Math`.

```
public class AleaTest {
   double d = Math.random(); // double dans [0.0,1.0[
}
```

On peut aussi utiliser une méthode *non static* sur un objet `Random`.

```
import java.util.Random;
public class AleaTest {
   int i = new Random().nextInt(10); // entier dans [0,10[
}
```