Programmation Objet. Cours 1

Marie-Pierre Béal
UPEM DUT 1

Programmation objet. Classes et objets.

Principaux styles de programmation

Style impératif

- Fondé sur l'exécution d'instructions modifiant l'état de la mémoire.
- Utilise une structure de contrôle et des structures de données.
- Usage intensif de l'itération.
- Langages typiques : Fortran, C, Pascal.

Principaux styles de programmation

Style applicatif

- Fondé sur l'évaluation d'expressions, où le résultat ne dépend que de la valeurs des arguments (et non de l'état de la mémoire).
- Donne programmes courts, faciles à comprendre.
- Usage intensif de la récursivité.
- Langage typique : Lisp, Caml.

Principaux styles de programmation

Style objet

- Un programme est vu comme une communauté de composants autonomes (objets) disposant de ses ressources et de ses moyens d'interaction.
- Utilise des classes pour décrire les structures et leur comportement.
- Usage intensif de l'échange de messages.
- Langages typiques : C++, Java, Ruby, Python.

Principaux styles de programmation

Souvent un même langage permet différents styles de programmation.

Avantages du style objet

Facilite la programmation modulaire

- La conception par classes conduit à des composants réutilisables.
- Un composant offre des services et en utilise d'autres.
- Les services sont exposés à travers une *interface*.
- Les détails d'implémentations sont cachés (encapsulation).
- Les composants sont réutilisables.

Avantages du style objet

Abstraction et spécialisation

- L'abstraction sépare la définition de son implémentation.
- L'abstraction extrait un modèle commun à plusieurs composants.
- Le modèle commun est partagé par le mécanisme d'*héritage*.
- La spécialisation traite de cas particuliers.
- Le mécanisme de dérivation rend les cas particuliers transparents.

Java

Java (nom dérivé de Kawa) a vu le jour en 1995.
Actuellement version JDK 1.8 (aussi appelée Java 8).
Distributions et spécifications Oracle : Java SE, Java ME, Java EE.

- est fortement typé,
- est orienté objet,
- est compilé–interprété,
- est sans héritage multiple,
- à partir de la version 1.5, Java offre de la généricité (les `generics` sont différents des *template* du C++)

Premier exemple

- On écrit le code source java suivant dans le fichier `HelloCharlie.java` :

```
class HelloCharlie {
  public static void main (String[] args) {
    System.out.println("Bonjour Charlie");
  }
}
```

- On compile avec `javac HelloCharlie.java`
- On lance l'exécution (l'interprétation du bytecode) par la machine virtuelle Java avec `java HelloCharlie`

```
$ javac HelloCharlie.java
$ ls
HelloCharlie.class HelloCharlie.java
$ java HelloCharlie
Bonjour Charlie
```

Autre exemple

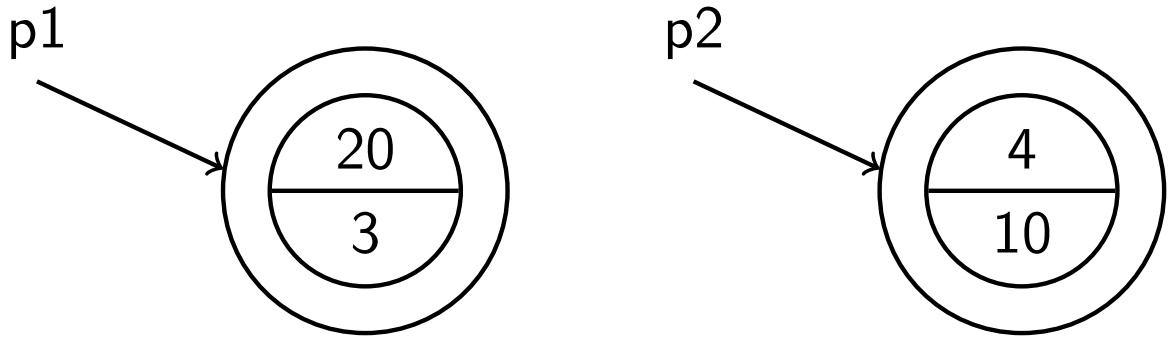
- On écrit ce code source java suivant dans le fichier `AdditionTest.java` :

```
class AdditionTest {
  public static void main (String[] args) {
    int n = 100;
    int m = 200;
    System.out.println(n + m);
  }
}
```

- On compile avec `javac AdditionTest.java`
- On lance l'exécution avec `java AdditionTest`, on obtient 300.

Les objets

Un objet est un composant situé dans une zone mémoire.
On le manipule par son adresse appelée référence.




```
class Test {
  public static void main (String[] args) {
    Pixel p1 = new Pixel(20,3);
    Pixel p2 = new Pixel(4,10);
  }
}
```

Les classes

Une classe

- définit la structure de l'objet,
- définit le comportement de l'objet (par les méthodes),
- définit le moule pour la création des objets,
- permet de déclarer un nouveau type.

Les classes

Une classe est composée de

- déclarations de variables (*attributs ou champs*) ;
- définitions de fonctions (*méthodes*) ;
- (déclarations d'autres classes (*nested classes*)).

On écrit chaque classe dans un fichier séparé qui a comme nom `Pixel.java` si la classe se nomme `Pixel`. Le nom est en majuscule.

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

Les méthodes

Les noms des méthodes commencent par une minuscule.

```
class Pixel {
  int x;
  int y;
  void printX(){
    System.out.println(x);
  }
  void move(int dx, int dy){
     x = x + dx;
     y = y + dy;
  }
}
```

Les constructeurs

Les constructeurs permettent la création des objets par *new*.
Son type de retour est `void` sans l'écrire.

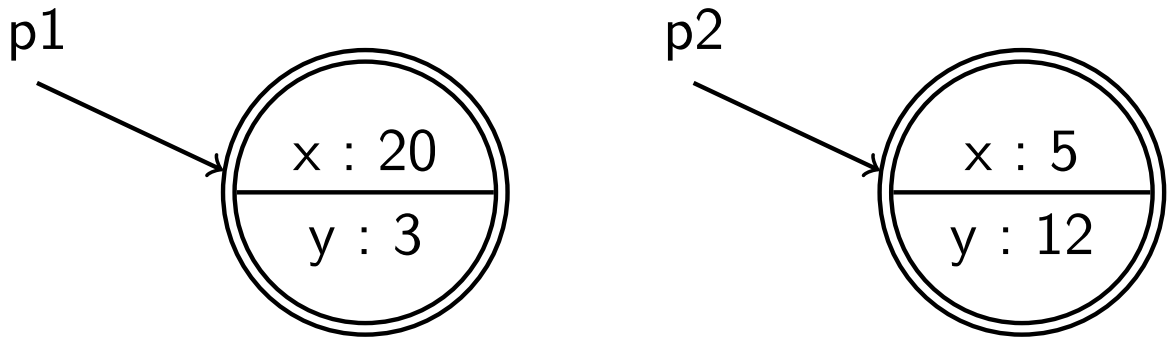
```
class Pixel {
  int x;
  int y;
  void printX(){
    System.out.println(x);
  }
  void move(int dx, int dy){
     x = x + dx;
     y = y + dy;
  }
  Pixel(int x0, int y0) {
     x = x0;
     y = y0;
  }
}
```

Utilisation des classes et objets

Dans une autre classe `PixelTest`, on fabrique et utilise des points.
On *appelle une méthode sur un objet* avec le symbole $\cdot$.

```
class PixelTest {
  public static void main (String[] args) {
    Pixel p1 = new Pixel(20,3);
    Pixel p2 = new Pixel(5,12);
    p1.printX();
    p2.move(1,1);
    p2.printX();
  }
}
```

Utilisation des classes et objets




```
$ java PixelTest
20
6
```

Utilisation de *this*

Le mot clé `this` est utilisé pour désigner une référence sur l'objet courant dans une classe.

```
class Pixel {
  int x;
  int y;
  Pixel(int x, int y) {
     this.x = x;
     this.y = y;
  }
}
```