Programmation orientée objet

Chapitre3: Programmation orientée objet en java

Principe de base de la POO

La Programmation Orientée Objets (**POO**) consiste en une structuration de plus haut niveau. Il s'agit de regrouper l'ensemble des données et toutes les procédures qui permettent la gestion de ces données. On obtient alors des entités comportant à la fois un ensemble de données et une liste de procédures et de fonctions pour manipuler ces données. La structure ainsi obtenue est appelée : **Objet**.

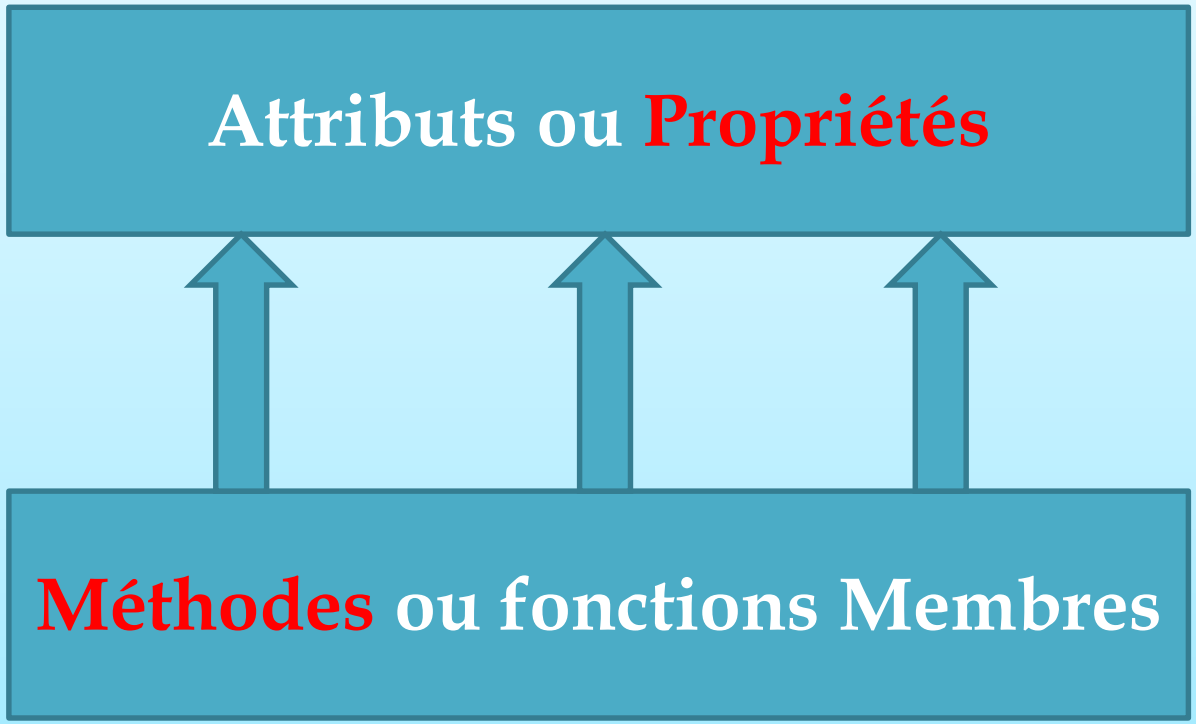
Un objet est alors une généralisation de la notion d'enregistrement. Il est composé de deux parties :

- Une partie statique (fixe) composée de la liste des données de l'objet. On les appelle :

Attributs ou **Propriétés, ou encore : Données Membres**.

- Une partie dynamique qui décrit le comportement ou les fonctionnalités de l'objet. Celles-ci sont appelées: Méthodes ou **Fonctions Membres**.

Principe de base de la POO



Encapsulation

- La définition des propriétés et des méthodes de l'objet devra être réalisée dans une structure de données appelée **classe.**
- Une **classe** est donc le support de l'encapsulation : c'est un ensemble de données et de fonctions regroupées dans une même entité.
- Créer un objet depuis une classe est une opération qui s'appelle **l'instanciation**. L'objet ainsi créé pourra être appelé aussi : **instance**. Entre classe et objet il y a, en quelque sorte, le même rapport qu'entre type de données et variable.
- Pour utiliser une classe il faut en déclarer une instance de cette classe (appelée aussi objet).

Encapsulation

L'encapsulation consiste à réduire la visibilité des attributs et méthodes d'une classe, afin de n'exposer que les fonctionnalités réellement nécessaires pour les classes utilisatrices.

- Cette réduction de visibilité est rendue possible avec
  - private : visibilité interne à la classe
  - public : visibilité pour toutes les classes
  - protected : visibilité pour les classes enfants
  - default (aucun des modes ci-dessus n'est spécifié) : visibilité pour les classes du même package

Syntaxe de déclaration d'une classe

```
class NomDeClasse {
      Propriétés de la classe
      Méthodes de la classe
}
```

L'ordre des méthodes dans une classe n'a pas d'importance. Si dans une classe, on rencontre d'abord la méthode m1() puis la méthode m2(), m2() peut être appelée sans problème dans m1().

Instanciation

Il est nécessaire de déclarer un objet avant son instanciation. La déclaration est réalisée de la manière suivante :

```
NomDeClasse nomDeVariable ;
```

L'instanciation et réalisée ensuite à l'aide de l'opérateur **new** qui se charge de créer une instance (ou un objet) de la classe et de l'associer à la variable

```
nomDeVariable = new NomDeClasse();
```

Il est possible de réunir les 2 instructions en une seule :

```
NomDeClasse nomDeVariable = new NomDeClasse();
```

Instanciation

- Si **objet1** désigne un objet de type NomDeClasse, et si **objet2** est un nouvel objet créé à l'aide de l'instruction **objet2 = objet1**, il est alors à noter que cette affectation ne définit pas un nouvel objet mais objet1 et objet2 désignent tous les deux le même objet.

- L'opérateur **new** est un opérateur de haute priorité qui permet d'instancier des objets et d'appeler une méthode particulière de cet objet : le **contructeur**.

Constructeur

Le constructeur est une méthode de classe (sans valeur de retour) qui port le même nom que celui de la classe et dans laquelle sont réalisées toutes les initialisations de l'objet en cours.

Le constructeur peut avoir des paramètres et dans tel cas l'instanciation est réalisée avec des arguments d'appel :

```
NomDeClasse nomDeVariable = new NomDeClasse(a1, a2, ...);
```

Exemple :

```
String S = new String("abc");
```

Pour le cas des chaînes de caractères, l'écriture précédente et équivalente à l'instruction suivante: **`String S = "abc";`**

Les constantes chaînes sont des objets **String**

Durée de vie d'un objet

Les objets en java sont tous dynamiques. La durée de vie d'un objet passe par trois étapes :

- La déclaration de l'objet et son instanciation à l'aide de l'opérateur **new**
- L'utilisation de l'objet en appelant ces méthodes
- La suppression de l'objet : elle est réalisée automatiquement à l'aide du **garbage collector** qui libère l'espace mémoire associé à l'objet ne référence plus cet espace mémoire (par exemple affectation de **null** à l'objet) et lorsque cet espace mémoire n'est plus référencée par aucune autre variable).

  Il n'existe pas d'instruction **delete** comme en C++.

Affectation d'objets

Exemple:

```
NomDeClasse objet1 = new NomDeClasse ();
NomDeClasse objet2 = objet1;
```

objet1 et objet2 contiennent la même référence et pointent donc tous les deux sur le même espace mémoire : les modifications faites à partir de l'une des deux variables modifient le contenu aussi bien pour l'une que pour l'autre des 2 variables.

Références et comparaison d'objets

Les variables de type objet que l'on déclare ne contiennent pas un objet mais une référence vers cet objet. Lorsque l'on écrit c1 = c2 (c1 et c2 sont des objets), on copie la référence de l'objet c2 dans c1 : c1 et c2 réfèrent donc le même objet (ils pointent sur le même objet).

L'opérateur **==** compare ces références.

Deux objets avec des propriétés identiques sont deux objets distincts :

Exemple:

```
Rectangle r1 = new Rectangle(100,50);
Rectangle r2 = new Rectangle(100,50);
if (r1 == r2) { ... } // Faux
if (r1.equals(r2)) { ... } // Vrai
```

Références et comparaison d'objets

Pour tester l'égalité des contenus de deux instances, il faut munir la classe d'une méthode permettant de réaliser cette opération : la méthode **equals** héritée de la classe **Object**.

Remarque : Pour s'assurer que deux objets sont de la même classe, il faut utiliser la méthode **getClass()** de la classe Object dont toutes les classes héritent.

Exemple:

```
(obj1.getClass().equals(obj2.getClass())
```

Le mot clé this

- Ce mot clé sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. **`this`** est un objet qui est égale à l'instance de l'objet dans lequel il est utilisé.
- Ainsi dans une classe, pour accéder à une propriété «p », on peut écrire **`this.p`**. Cette notation devient nécessaire dans le cas ou une autre variable du contexte (un paramètre ou une variable globale) porte le même nom de la propriété.

Le mot clé this

Exemple:

```
class A {
 private int nombre;
 public maclasse(int nombre) {
      nombre = nombre;
 }
}
```

Cette référence est habituellement implicite

L'opérateur instanceof

L'opérateur **instanceof** permet de déterminer la classe de l'objet qui lui est passé en paramètre.

La syntaxe est :

```
… objet instanceof classe …
```

Exemple:

```
void testClasse(Object objet) {
      if (objet instanceof A ) ...
}
```

Les constantes

Les constantes sont définies avec le mot clé **`final`** : leur valeur ne peut pas être modifiée. Elles sont généralement aussi : publiques et statiques.

Exemple:

```
public class MaClasse {
      public static final double PI = 3.14 ;
}
```

La surcharge des méthodes

- La surcharge d'une méthode permet de définir plusieurs fois une même méthode avec des arguments différents. Lors de l'appel à la méthode, le compilateur choisi la méthode qui doit être appelée en fonction du nombre et du type des arguments.
- Une méthode est surchargée lorsqu'elle exécute des actions différentes selon le type et le nombre de paramètres transmis.

La surcharge des méthodes

Exemple:

```
public class Number {
 private int value;
 public Number(int value) {
  this.value = value;
 }
 public void inc() {
  value ++;
 }
 public void inc(int step) {
  value = value + step;
 }
}
```

La surcharge des méthodes

Exemple: (l'appel)

```
Number n = new Number(20) ;
n.inc(); //appellera la première méthode inc() et incrémentera la valeur par 1
n.inc(5); //appellera la 2ème méthode inc(int) et incrémentera la valeur par 5
```

Remarque **:**

Les constructeurs peuvent aussi être surchargés

Les accesseurs

Un accesseur est une méthode publique qui donne l'accès à une propriété privée d'une classe. L'accès peut être réalisé en lecture ou en écriture. Par convention, les accesseurs en lecture commencent par le mot « **get** » (ou « **is** » si le type de la propriété est **boolean**) et sont appelés des **getters** et les accesseurs en écriture commencent par le mot « **set** » et sont appelés des **setters.**

Les accesseurs

Exemple:

```
public class Number {
	private int value;
	public Number(int value) {
		this.value = value;
	}
	public void setValue(int value) {
		this.value = value;
	}
	public int getValue() {
		return value;
	}
}
```

Utilisation d'un package

Pour utiliser ensuite le package ainsi créé, on l'importe dans le fichier :

```
import nom.de.package.*;
```

Pour utiliser une classe A particulière du package, on peut utiliser l'instruction suivante :

```
import nom.de.package.A;
```

Remarque :

L'utilisation du symbole * donne l'accès à toutes les classes du package mais pas aux sous packages de celui-ci.

TP3

Programmation orientée objet en java