Programmation Orientée Objet

Mohamed Tounsi

Institut Supérieur d'Informatique et de Multimédia Sfax

Novembre 2014

Programmation orientée objet
Introduction

- Un programme informatique est construit à partir de deux choses: **variables** et **instructions**.
- En programmation orientée objet, c'est la même chose ! La différence réside dans le fait que l'on doit bâtir les applications à partir de briques que l'on appelle les **objets**.
- Ainsi, au lieu de développer un seul gros programme, nous en développons de nombreux petits, qui communiquent ensemble.

Définition

Les objets sont des petits programmes qui possèdent leurs propres variables et leurs propres instructions.

Programmation orientée objet
Caractérisation simple d'une personne

Caractérisation d'une personne

- Une personne est caractérisée par son prénom, son nom de famille, son sexe, sa date de naissance, son lieu de naissance, sa profession...: ce sont ses **propriétés** (ces propres attributs),
- Une personne peut exécuter les **opérations** (ces propres méthodes) suivantes: se réveiller, se lever, marcher, courir, dormir, manger, lire,....

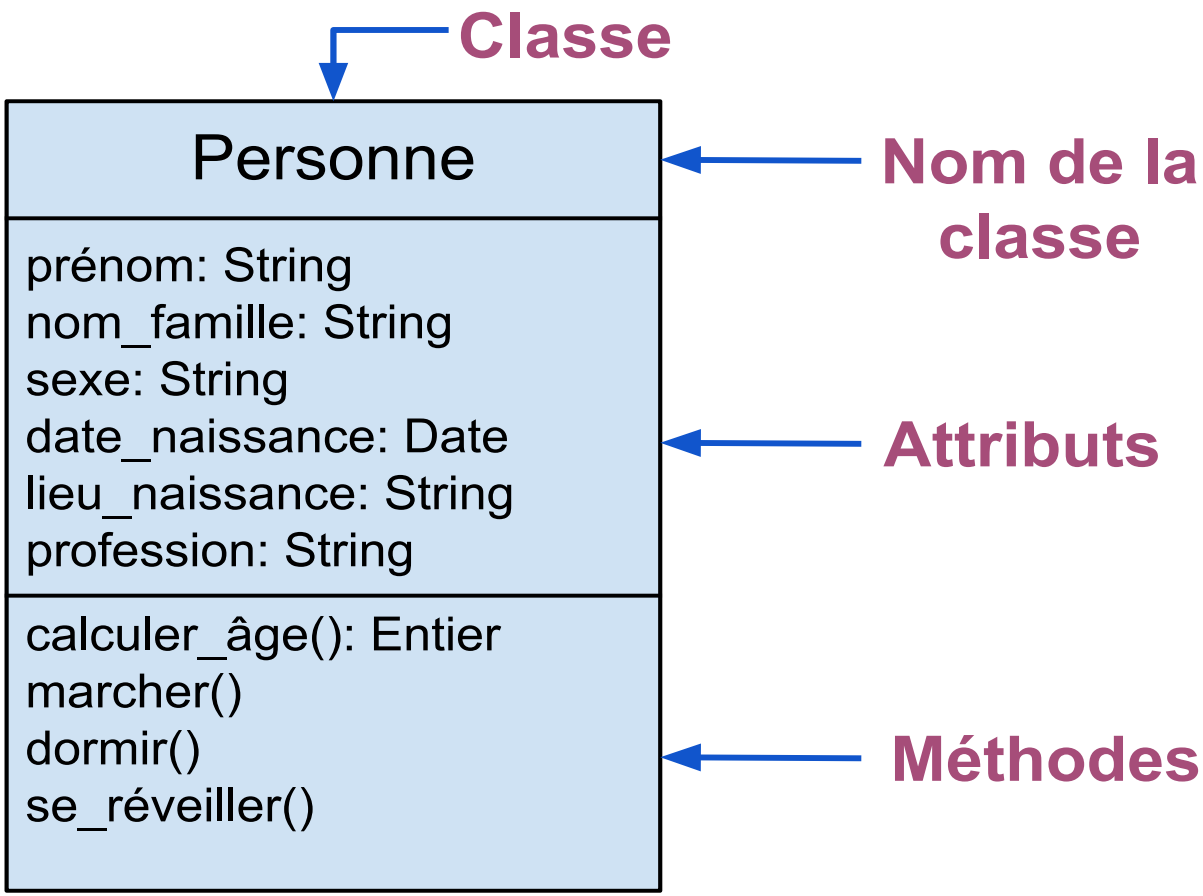
Programmation orientée objet

Modèle textuel simple des objets de type “Personne”

```
Classe: //on déclare un type personne
          Personne
Attributs: //on déclare les propriétés
          prénom: Chaîne de caractère
          nom_famille: Chaîne de caractère
          sexe: Chaîne de caractère
          date_naissance: Date
          lieu_naissance: Chaîne de caractère
          profession: Chaîne de caractère
Méthodes: // on déclare le comportement de l'objet
          calculer_âge(): type de retour Entier
          marcher(): type de retour Rien
          dormir(): type de retour Rien
          se_réveiller(): type de retour Rien
```

Programmation orientée objet

Modèle graphique simple des objets de type "Personne"



Programmation orientée objet

La classe Personne

Soit le fichier “Personne.java”

```
public class Personne
  { String prenom;
    String nom_famille;
    String sexe;
    Date date_naissance;
    String lieu_naissance;
    String profession;

    public int calculer_age ()
    { return (...); }

    public void marcher ()
    { .... }
    ......
  }
```

Programmation orientée objet
L'objet "Fairuz"

Exemple: l'objet **Fairuz**

Fairuz=(prenom= "Nouhad", nom_famille="Haddad", sexe="femme", date_naissance=21-11-1935, lieu_naissance="Beyrouth", profession="chanteuse")

La classe

Définition

- Une classe permet de décrire de façon commune des éléments ayant les mêmes propriétés.
- Une classe est une sorte de “modèle” pour fabriquer des objets partageant la même structure.
- Une classe X permet de définir un type d'objet X.
- Une classe regroupe un ensemble de données (qui peuvent être des variables primitives ou des objets) et un ensemble de méthodes (opérations) de traitement de ces données et/ou de données extérieures à la classe.

La classe

Syntaxe de la déclaration d'une classe

- En Java, les classes sont définies dans des groupes d'instructions (entre accolades) précédés de la déclaration **class** suivie du nom de la classe:

```
[private/public][abstract/final] class Nom_de_la_classe
[extends Nom_super_classe]
[implements Nom_interface1, Nom_interface2,...]
{
        // définition des attributs (variables)
        // définition des méthodes (opérations)
  }
```

- Le nom de la classe doit être unique;
- Le nom de la classe sera utilisé pour créer des objets à partir de la classe.

La classe

Modificateurs d'accès & qualificateurs

- **[private/public]**: deux modificateurs d'accès précisant la possibilité d'utilisation de la classe:
  - **private**: indique que l'utilisation de la classe est réservée aux classes du même package (**private** est la valeur par défaut);
  - **public**: donne la possibilité d'utiliser la classe à toutes les autres classes.
    Il ne peut y avoir qu'une seule classe publique par fichier.
- **[abstract/final]**: deux qualificateurs en relation avec le concept d'héritage;
- **extends**: annonce l'héritage de la classe Nom_super_classe
- **implements**: annonce l'implémentation d'une ou plusieurs interfaces par la classe.

La classe
Déclaration des attributs

- En Java, la syntaxe de déclaration d'un attribut est la suivante:

```
[private/public/protected][static][final]
        type_attribut nom_attribut;
```

- **[private/public/protected]**: des modificateurs d'accès à l'attribut à partir d'autres classes:
  - **private**: indique que l'accès est réservé aux méthodes de la même classe;
  - **public**: donne la possibilité d'accès à toutes les méthodes des autres classes;
  - **protected**: limite l'accès aux méthodes de la même classe, des classes dérivées et des classes du même package.
- **static**: précise le fait qu'il y aura une seule copie de cet attribut qui est commune à tout objet crée à partir de la classe en question;
- **final**: indique que l'attribut est une constante.

La classe
Déclaration des méthodes

```
[private/public/protected][static][final/abstract]
    type_retour nom_méthode
            (type_paramètre1 paramètre1, ...)
    {   //déclaration des variabls locales;
        [type_retour variable;]
        //instructions de la méthode
        [return variable;]      }
```

- **[private/public/protected]**: des modificateurs d'accès autorisant l'appel aux méthodes de la classe;
- **static**: précise que l'exécution de la méthode n'est pas liée aux instances de la classe;
- **abstract**: la méthode ne comporte pas de code. Par conséquent toute la classe devient abstraite;
- **final**: la méthode ne pourra pas être redéfinie dans une classe dérivée.

La classe
Surcharge des méthodes

Définition

- Écrire plusieurs méthodes de même nom dans une classe;
- Les méthodes doivent différer par le nombre des arguments et/ou par le type des arguments;
- Le type de retour peut également être différent, mais pas uniquement (c'est à dire ne peut être la seule différence)

Remarques

- Avoir plusieurs constructeurs est un cas particulier de surcharge;
- Surcharger des méthodes c'est le polymorphisme statique.

La classe
L'encapsulation

- L'encapsulation est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure (la classe).
- Les détails d'implémentation sont cachés, le monde extérieur n'ayant accès aux données que par l'intermédiaire d'un ensemble d'opérations constituant l'Interface de l'objet.
- L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

La classe

Référence sur l'objet courant "this"

- Le mot-clé "this" représente une référence sur l'objet courant, c'est à dire, celui qui possède la méthode qui est en train de s'exécuter.
- La référence "this" peut être utile:
  - Lorsqu'une variable locale (ou un paramètre "caché") porte le même nom d'un attribut de la classe.
  - Lorsqu'on ne connaît pas encore une référence pour l'objet en cours.

```
class Personne {
  public String nom;
  public Personne (String nom)
  { this.nom=nom;} }
```

```
public MaClasse (int a, int b) {...}

public MaClasse (int c) { this(c,0); }

public MaClasse () { this(10); }
```

La classe

Les accesseurs

- Un accesseur est une méthode publique qui donne l'accès à une variable privée d'un objet.
- Un accesseur peut modifier ou retourner les valeurs des attributs même qui sont déclarés "private".
- Par convention:
  - les accesseurs en lectures commencent par "get";
  - les accesseurs en écriture commencent par "set".
- Ces méthodes sont utiles surtout quand l'attribut en question n'est pas public, c'est pour cela que le modificateur d'accès à ces méthodes est public.

```
public class MonClass {
private int valeur = 13;
public int getValeur () { return(valeur); }
public void setValeur (int val) { valeur = val; }
}
```

La classe

Les constructeurs

- Le constructeur d'une classe est une méthode particulière appelée une seule fois par objet et ce au moment de sa création;
- Le constructeur porte le même nom que celui de la classe;
- Le rôle d'un constructeur est d'initialiser les attributs d'un objet.
- Le constructeur est généralement publique et ne possède pas de type de retour (même pas de **void**)
- Une classe peut avoir plusieurs constructeurs qui diffèrent par le nombre et la nature de leurs paramètres.
- En l'absence d'un constructeur défini par le programmeur, Java offre un constructeur non paramétré qui initialise les attributs de l'objet crée (0 pour les valeurs numériques, "null" pour les références, "false" pour les boolean).

La classe

Les constructeurs: Exemple

```
public class Personne
  { String prenom;
    String nom;
    int age;

    public Personne (String p, String n, int a)
    { prenom = p;
      nom = n;
      age = a;
     }
     //....
   }
```

```
...
public static void main (String args[])
{Personne Fairuz = new Personne (``Nouhad'',``Haddad'',78);}
...
```

La classe
Destructeur d'objets

- Le destructeur est une méthode qui exécute certaines tâches avant la destruction d'un objet,
- L'exécution du destructeur se fait automatiquement par la ramasse miettes,
- Le destructeur est optionnel, s'il est décrit il est unique,
- En Java, le destructeur s'appelle "finalize". Il est de type "void" et ne reçoit pas de paramètres.

```
protected void finalize()
```

Les objets
Définitions

- Un objet est une instance de classe. Il est crée (on dit aussi construit) à partir d'une classe.
- Un objet est un module logiciel encapsulé. Il possède:
  - Un **état**: ce sont les valeurs de ses attributs qui sont définies dans la classe.
  - Un **comportement**: c'est l'ensemble de ses méthodes, définies et implémentées dans la classe.
  - Une **identité**: c'est ce qui permet de l'identifier pour le manipuler.
- Tous les objets issus d'une même classe ont strictement la même structure et les même comportements.

Les objets
Construction des objets (1)

- En Java, c'est l'opérateur "**new**" qui permet de créer des objets.

```
Nom_Classe Nom_Objet = new Nom_Classe();
```

- L'opérateur **new** va réserver l'espace mémoire nécessaire pour créer l'objet "Nom_Objet" de la classe "Nom_Classe".

- si on définit un constructeur, est ce qu'on peut utiliser le constructeur par défaut ?
- est ce qu'on peut redéfinir le constructeur sans paramètres ?

Les objets
Construction des objets (2)

L'appel de "**new**" pour créer un nouvel objet déclenche, dans l'ordre:

- L'allocation mémoire nécessaire au stockage de ce nouvel objet et l'initialisation par défaut de ces attributs,
- L'initialisation explicite des attributs, s'il y a lieu,
- L'exécution d'un constructeur,
- Retourner une référence sur l'objet.

Les objets
Détermination de la nature d'un objet

L'opérateur **instanceof** permet de tester si l'objet référencé est une instance d'une classe donnée (ou d'une de ses sous-classes).

Exemple

```
Circle c = new Circle(2.0);
```

"c instanceof Circle" retourne **true**
"c instanceof Personne" retourne **false**