Introduction à la Programmation Orientée Objet

Dr Khadim DRAME
kdrame@univ-zig.sn

Département Informatique
UFR Sciences et Technologies
Université Assane Seck de Ziguinchor

Mai 2022

Objectifs du cours

- distinguer des concepts fondamentaux de la programmation orientée objet : classes, objets, encapsulation, attributs, méthodes, constructeurs
- utiliser ces concepts de manière adéquate
- concevoir et implémenter les classes en Java

Plan

1. Introduction
2. Concepts de la POO
3. Objets et classes
   - Constructeurs
   - Accesseurs et modificateurs
4. Manipulation des objets
5. Membres statiques

Principe de la POO

- La Programmation Orientée Objet (POO) est un paradigme de programmation basé sur les objets
  - un programme est composé d'un ensemble d'objets et de leurs interactions
- Exemple : l'interface suivante est composée de 7 objets

Motivation

- La POO
  - pour concevoir et maintenir facilement des applications
  - incontournable dans le développement logiciel
- La POO offre plusieurs avantages
  - modularité
    - facilite la compréhension et la maintenance de codes
    - facilite la réutilisation de codes
  - sûreté
    - robustesse des programmes grâce à l'**encapsulation**

Langages de POO

- 1970 : **Simula** (1967), **Smalltalk** (1972)
- 1980 : **C++**, **Objective C**
- 1990 : **Java** par Sun Microsystems
- 2000 : **C#** par Microsoft
- Aujourd'hui, de nombreux langages ont adopté le principe de la POO : **PHP**, **Python**, **JavaScript**

Plan

1. Introduction
2. Concepts de la POO
3. Objets et classes
   - Constructeurs
   - Accesseurs et modificateurs
4. Manipulation des objets
5. Membres statiques

C'est quoi un objet ?

- Un objet est une entité, concrète ou abstraite, manipulée dans un programme
- Un objet est caractérisé par
  - une identité (référence)
  - un état interne : valeurs de ses attributs (variables)
  - un comportement : ensemble de méthodes (fonctions)
- Exemples

  une personne, une voiture, un point

Notion de classe

- Les objets manipulées ont des structures et comportements proches voire identiques
- Ces objets peuvent être regroupés par types d'objets
- Une classe représente un type de données
  - famille d'objets ayant une même structure et un même comportement
  - chaque objet est une instance d'une classe
- Exemples

  Nombre complexe, Point, Personne

Utilisation des classes

- Définir de nouveaux types de données : classes
- Créer des objets de ces types : instances
- Décomposer une application en de petites entités
- Encapsuler des données

Membres d'une classe

- Attributs
  - Un attribut ou variable d'instance est une donnée de la classe, une caractéristique
  - Exemples

    prénom, nom, âge
- Méthodes
  - Une méthode est une fonction définie dans une classe
  - Elle permet de faire des manipulations spécifiques sur les objets
  - Exemples

    modifier le prénom, augmenter l'âge
- Les attributs et les méthodes constituent les membres de la classe

Membres d'une classe

- Un contructeur est une méthode particulière pour créer des instances de la classe
- Deux types de méthodes permettent d'accéder aux attributs depuis l'extérieur de la classe de manière sécurisée
  - les accesseurs (getters) pour récupérer les valeurs des attributs
  - les modificateurs (setters) pour modifier les valeurs des attributs

Encapsulation

- L'encapsulation consiste à masquer l'accès aux attributs
  - protéger l'intégrité des objets en cachant leur état
  - imposer de passer par des méthodes pour les manipuler
- Trois niveaux de visibilité
  - les membres privés (private) ne sont accessibles que dans la classe
  - les membres protégés (protected) seulement accessibles dans la classe et ses classes dérivées (voir héritage)
  - les membres publics (public) sont accessibles partout
- L'encapsulation est mise en œuvre dans l'ensemble des bibliothèques de Java SE

Encapsulation

- Mise en œuvre de l'encapsulation
  - déclarer les attributs privés (**private**)
  - fournir des méthodes (**public**) d'accès sécurisés à ces attributs (accesseurs et modificateurs)
  - faire des contrôles dans les modificateurs

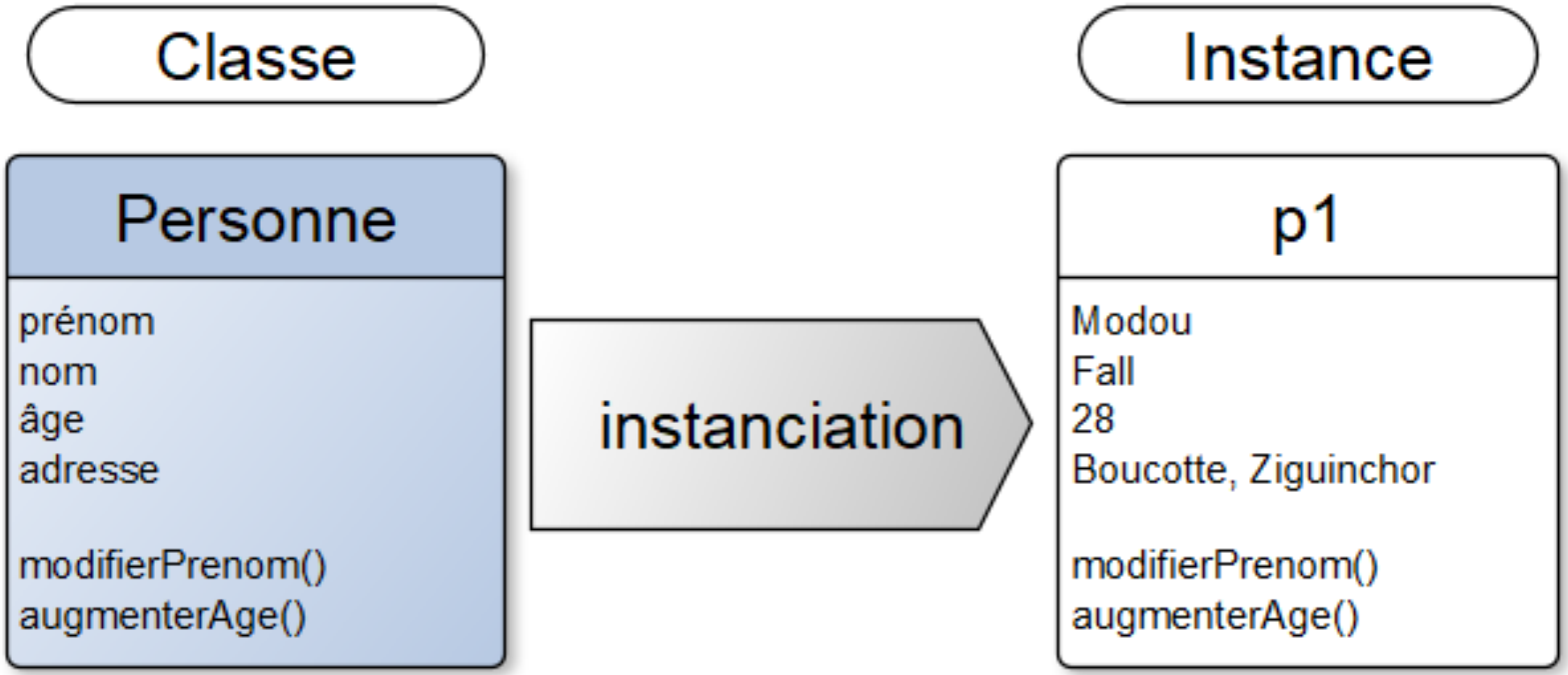
```
1  public class Rationnel{
2    private int numerateur, denominateur;
3    public int getNumerateur(){
4      return numerateur;
5    }
6    public void setDenominateur(int den){
7      if (den !=0){
8        denominateur = den;
9      }else {...}
10   }
11   ...
12 }
```

Plan

1. Introduction
2. Concepts de la POO
3. Objets et classes
   - Constructeurs
   - Accesseurs et modificateurs
4. Manipulation des objets
5. Membres statiques

Création de classes et d'objets

- Pour définir une classe, il faut spécifier
  - les données ou attributs associés aux objets de la classe
  - les opérations ou méthodes pour manipuler ces objets
- Objets = instances de classes
- Exemple



Conventions de nommage

- Les noms de classes commencent par une majuscule

  `Etudiant`, `Personne`, `Object`
- Les identificateurs des membres commencent par une minuscule

  `nom`, `age`, `concat`, `contains`
- Les mots contenus dans un identificateur commencent par une majuscule

  `isEmpty`, `parseInt`
- Les constantes sont en majuscules et les mots séparés par le «_»

  `PI`, `MAX_VALUE`
- Si possible, des noms pour les classes et des verbes pour les méthodes

Définition de classes en Java

- Syntaxe

  <qualificateur> class <nom_classe>{

  spécification des attributs de la classe

  définition des méthodes de la classe

  }

- Exemple

```
1  public class Employe{
2    //déclaration des attributs
3    private String prenom, nom;
4    private double salaire;
5    //définition des méthodes
6    public double salaireActuel(){
7      return salaire;
8    }
9    ...
10 }
```

Définition de méthodes

- Une méthode est une séquence d'instructions permettant de manipuler des objets
- La classe définit l'ensemble des méthodes qu'on peut appeler sur les objets de ce type
- Exemple

```
1 String  prenom = "Amadou";
2 String  nom = "Diop";
3 int n = prenom.length(); //affecte 6 à n
4 nom = nom.toUpperCase(); //affecte "DIOP" à nom
5 int m = prenom.longueur(); /*génère une erreur car
6 cette méthode n'existe pas pour les objets de type
7 String */
```

Définition de méthodes

- Syntaxe

  <qualificateur> <type> <nom_methode>(<paramètres>){
  <bloc_instructions>
  }

  - <qualificateur> : public, protected, privated, ou rien
  - <nom_methode> : nom de la méthode
  - <paramètres> : paramètres explicites avec leurs types
  - <bloc_instructions> : corps de la méthode
  - <type> : le type de retour (void s'il y en a pas)

- Exemple

```
1 public class Employe{
2   ...
3   public double salaireActuel(){
4     return salaire;
5   }
6   public void modifierSalaire(double sal){
7     salaire = sal;
8   }
9 }
```

Paramètres d'une méthode

- Paramètres explicites
  - données en entrée de la méthode
  - ils ne sont pas obligatoires
  - exemples

```
System.out.println("Bonjour le monde");
System.exit()
```

- Paramètre implicite ou receveur
  - objet sur lequel on appelle la méthode
  - exemple

```
nom.length()
```

Surcharge

- On peut définir des méthodes de même nom dans une même classe mais avec des signatures différentes : **surcharge**
- Exemples
  - double calculerImpots(double revenus) ; //célibataire
  - double calculerImpots(double revenus, int nbEnf) ;//marié avec enfants
- En Java, il est interdit de surcharger une méthode en changeant le type de retour

Constructeur

- Un constructeur est une méthode spéciale qui permet de
  - créer des instances (objets) d'une classe
  - initialiser les attributs de ces objets
- Un constructeur
  - a le **même nom que la classe**
  - n'a pas de type de retour (mais pas de void !)
- Chaque classe possède un ou plusieurs constructeurs
  - dans le cas de plusieurs constructeurs (avec des signatures différentes), on parle de surcharge

Types de constructeurs

- Trois types de constructeurs
  - Constructeur par défaut produit par le compilateur, utilisé si aucun constructeur n'est défini
    - il ne prend pas d'argument
  - Constructeur paramétrique, appelé si la signature correspond à celle du constructeur
  - Constructeur de copie qui a comme unique argument un objet de même type
- Si un constructeur est défini, le constructeur par défaut n'est pas produit

Surcharge de constructeurs

- Exemple de surcharge de constructeurs

```
1  public class Employe{
2    private String prenom, nom;
3    private double salaire;
4    // Constructeur 1
5    public Employe(String p, String  n){
6      prenom = p;
7      nom = n;
8      salaire = 50000;
9    }
10   // Constructeur 2
11   public Employe(String p, String  n, double sal){
12     prenom = p;
13     nom = n;
14     salaire = sal;
15   }
16   ...
17 }
```

Référence **this**

- Chaque objet a accès à une référence à lui même : this
- Utilisé si une méthode (constructeur) a un paramètre ayant le même nom qu'un attribut de la classe

```
1  public class Employe{
2    private String prenom, nom;
3    private double salaire;
4    // Constructeur 1
5    public Employe(String prenom, String  nom){
6      this.prenom = prenom;
7      this.nom = nom;
8      this.salaire = 50000;
9    }
10   // Constructeur 2
11   public Employe(String prenom, String  nom, double salaire){
12     this.prenom = prenom;
13     this.nom = nom;
14     this.salaire = salaire;
15   }
16   ...
17 }
```

Délégation de constructeurs

- Un constructeur peut faire appel à d'autres constructeurs

```
1  public class Employe{
2    private String prenom, nom;
3    private double salaire;
4    // Constructeur 1
5    public Employe(String prenom, String  nom){
6      this.prenom = prenom;
7      this.nom = nom;
8      this.salaire = 50000;
9    }
10   // Constructeur 2
11   public Employe(String prenom, String  nom, double salaire){
12     this(prenom, nom);
13     this.salaire = salaire;
14   }
15   ...
16 }
```

Accesseur/Modificateur

- Deux types de méthodes permettent d'accéder aux attributs depuis l'extérieur de la classe
  - les accesseurs pour récupérer les valeurs des attributs d'un objet
    - ils ne modifient pas l'état interne d'un objet
  - les modificateurs pour modifier les valeurs des attributs
    - ils modifient l'état interne d'un objet

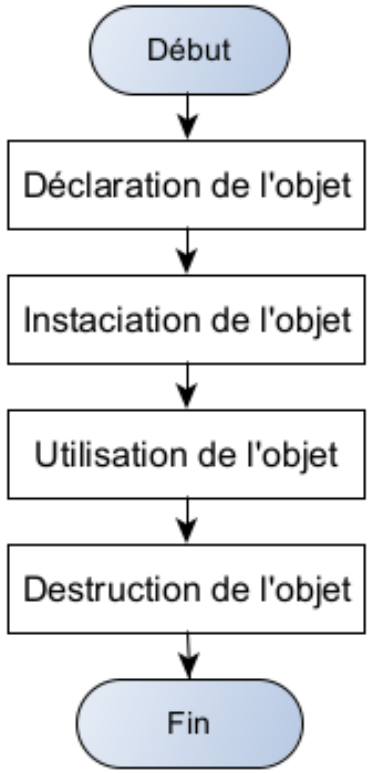
```
1  public class Employe{
2    // accesseur sur l'attribut prénom
3    public String getPrenom(){
4      return prenom;
5    }
6    // accesseur sur l'attribut salaire
7    public double getSalaire(){
8      return salaire;
9    }
10   // modificateur de l'attribut prénom
11   public void setPrenom(String prenom){
12     this.prenom = prenom;
13   }
14   // modificateur de l'attribut salaire
15   public void setSalaire(double salaire){
16     this.salaire = salaire;
17   }
18   ...
19 }
```

Plan

1. Introduction
2. Concepts de la POO
3. Objets et classes
   - Constructeurs
   - Accesseurs et modificateurs
4. Manipulation des objets
5. Membres statiques

Cycle de vie d'un objet

- Déclaration de l'objet
- Instanciation de l'objet
- Utilisation de l'objet en appelant ses méthodes
- Suppression de l'objet
  - utilisation d'un destructeur
  - arrêt de la JVM
  - arrêt du programme



Instanciation d'un objet : opérateur **new**

- En Java, l'opérateur new
  - crée une instance d'une classe
  - initialise les attributs de l'objet avec ses paramètres
  - retourne l'objet créé
- Syntaxe

  new <constructeur>(<paramètres>)
- L'objet créé est généralement conservé dans une variable

  <nom_classe> <variable> = new

  <constructeur>(<paramètres>)
- Exemples

```
1 Employe emp1 = new Employe("Fallou", "Ndiaye");
2 Employe emp2 = new Employe("Aida", "Dione", 90000.5);
```

Encapsulation

- Exemple

```
1  public class Employe{
2    private String prenom, nom;
3    private double salaire;
4    // Constructeur 1
5    public Employe(String prenom, String  nom){
6      this.prenom = prenom;
7      this.nom = nom;
8      this.salaire = 50000;
9    }
10   // Constructeur 2
11   public Employe(String prenom, String  nom, double salaire){
12     this(prenom, nom);
13     this.salaire = salaire; // ça passe
14   }
15   ...
16 }
17 public class Test{ // une autre classe
18   public static void main(String args[]){
19     Employe emp = new Employe("Cheikh","Diop");
20     emp.salaire = 105000; // Erreur de compilation
21    }
22 }
```

Utilisation de méthodes

- Syntaxe

  <objet>.<nom_methode>(<arguments>);

  - <objet> : objet qu'on veut manipuler
  - <nom_methode> : nom de la méthode
  - <arguments> : arguments de la méthode

- Exemple 1

```
1 Employe emp = new Employe("Aida", "Dione", 90000.5);
2 double sal = emp.salaireActuel();
```

- Exemple 2

```
1 String  s1 = "Bonjour ", s2 = "Amadou";
2 String message = s1.concat(s2); // "Bonjour Amadou"
3 boolean b1 = s1.contains("jour"); // true
4 boolean b2 = s1.contains("soir"); // false
```

Utilisation des accesseurs et modificateurs

- Exemple

```
1  public class TestMethodes{
2    public static void main (String args[]){
3      Employe e1 = new Employe("Fallou", "Ndiaye");
4      Employe e2 = new Employe("Aida", "Dione", 90000.5);
5      e1.setSalaire(120000.40):
6      System.out.print("Le 1er employé se nomme "+e1.getNom());
7      System.out.println(" et il gagne "+e1.getSalaire());
8      System.out.print("Le 2e employé se nomme "+e2.getNom());
9      System.out.println(" et il gagne "+e2.getSalaire());
10   }
11 }
```

Référence d'un objet

- Une variable instanciée contient une référence vers un objet
- Exemple

```
1 Employe e1 = new Employe("Fallou", "Ndiaye");
2 Employe e2 = new Employe("Fallou", "Ndiaye");
3 Employe e3 = e1;
4 boolean b1 = e1 == e2; // false
5 boolean b2 = e1 == e3; // true
6 e3.modifierSalaire(90000); // modifie aussi e1
```

  - ***e1*** et ***e2*** font références à deux objets
  - ***e1*** et ***e3*** font références au même objet
- Pour comparer des objets, définir des méthodes spécifiques

Référence d'un objet : **null**

- Une variable de type non primitif qui ne référence rien a pour valeur null
- **null** ne peut pas être utilisé comme un objet normal
  - pas d'appel de méthode
- Exemple

```
1 Employe e1 = null;
2 Employe e2 = null;
3 Employe e3 = e1;
4 boolean b1 = e1 == e2; // true
5 boolean b2 = e1 == e3; // true
6 e1.modifierSalaire(90000); // erreur: référence à null
```

- null est la valeur par défaut des attributs de type non primitif

Plan

1. Introduction
2. Concepts de la POO
3. Objets et classes
   - Constructeurs
   - Accesseurs et modificateurs
4. Manipulation des objets
5. Membres statiques

Variables statiques

- Aussi appelées **variables de classe**
- Déclarées avec le mot clé static dans une classe

  static <type> <identificateur_variable> ;
- Les variables statiques sont
  - utilisées pour faire référence aux propriétés communes aux objets d'une classe (nom de formation pour des étudiants)
  - définies pour l'ensemble du programme
  - visibles depuis toutes les méthodes de la classe

Variables statiques

- Exemple

```
1  public class Compteur{
2      int compt1 = 0;
3      static int compt2 = 0;
4      public Compteur(){
5        compt1++;
6        compt2++;
7        System.out.print("compteur 1 = "+compt1);
8        System.out.println(" et compteur 2 = "+compt2);
9      }
10     public static void main(String args[]){
11       Compteur c1 = new Compteur();
12       Compteur c2 = new Compteur();
13       Compteur c3 = new Compteur();
14     }
15 }
```

Variables statiques

- Exemple

```java
public class Compteur{
    int compt1 = 0;
    static int compt2 = 0;
    public Compteur(){
      compt1++;
      compt2++;
      System.out.print("compteur 1 = "+compt1);
      System.out.println(" et compteur 2 = "+compt2);
    }
    public static void main(String args[]){
      Compteur c1 = new Compteur();
      Compteur c2 = new Compteur();
      Compteur c3 = new Compteur();
    }
}
```

- Sortie du programme

  Compteur 1 = 1 et compteur 2 = 1
  Compteur 1 = 1 et compteur 2 = 2
  Compteur 1 = 1 et compteur 2 = 3

Méthodes statiques

- Déclarées avec le mot clé static dans une classe
- Exemple : main()
- Elles sont aussi dites **méthodes de classe**
- Une méthode statique
  - se rapporte à la classe plutôt qu'aux objets
  - peut être invoquée sans créer une instance d'une classe
  - ne peut utiliser de référence à une instance courante
    - elle ne peut utiliser une variable d'instance (non statique)
    - elle ne peut pas faire appel à une méthode non statique

Méthodes statiques

- Exemple

```
1  import java.lang.Math;
2  public class TestMethode{
3    static int carre(int x){
4      return x * x;
5    }
6    public static void main(String args[]){
7      int result = TesteMethode.carre(6); //préfixé par le nom
       de la classe
8      System.out.println(result); //36
9      System.out.println(Math.PI); //3.141592653589793
10     System.out.println(Math.max(14, 17));//17
11     System.out.println(Math.abs(-24));//24
12     System.out.println(Math.sqrt(81));  //9.0
13   }
14 }
```