PROGRAMMATION ORIENTÉE OBJETS

PROGRAMMATION ORIENTÉE OBJETS

PRINCIPES DE BASE

APPLICATION AU LANGAGE JAVA

La **Programmation Orientée Objets (POO)** est un modèle de programmation basé sur les objets et les données

POO - AVANTAGES

- Flexibilité
- Réutilisabilité
- Maintenabilité

4 PILIERS DE LA POO

- Abstraction
- Encapsulation
- Héritage
- Polymorphisme

MODÉLISATION

- Deux catégories de caractéristiques dans le modèle objet :
  - **L'état** : les propriétés qui caractérisent l'objet à un instant donné
  - **Le comportement** : les choses que l'objet est capable de faire

EXEMPLE - MODÉLISATION D'UN ARBRE

- **État** : hauteur, diamètre du tronc, type d'arbre
- **Comportement** : capacité à grandir

CLASSES

- L’idée est de modéliser des concepts existants (physique ou pas) avec des **classes**
- Une **classe** est donc une représentation abstraite d’un concept

CLASSE - CONSTITUTION

- On peut voir une classe comme un patron de plan, un moule, un « modèle vide »
- Une classe est constituée de :
  - **attributs** : variables
  - **méthodes** : blocs de codes nommés qui s'exécutent quand on les appellent

CLASSE - ÉTAT ET COMPORTEMENT

- Quand on modélise un concept avec une classe :
  - les **attributs** (variables) caractérisent l'**état**
  - les **méthodes** caractérisent le **comportement**

RETOUR SUR L'ARBRE

- Le concept « arbre » devient une classe `Arbre` :
  - **attributs** : `hauteur`, `diametreTronc`, `type`
  - **méthodes** : `grandir()`

CLASSE ARBRE

- **État**
  - hauteur
  - diamètre du tronc
  - type d’arbre
- **Comportement**
  - capacité à grandir

- **Attributs**
  - `hauteur`
  - `diametreTronc`
  - `type`
- **Méthodes**
  - `grandir()`

DIFFÉRENCE CLASSE/OBJET

- Tous les arbres ont les caractéristiques citées
  - ils sont tous de la **classe** `Arbre`
- Chaque arbre est différent en taille, diamètre et type
  - chaque **objet** arbre est une **instance** de la classe `Arbre`
- On a donc une représentation d'un concept (la classe) et autant d'instances de ce concept que l'on veut (les objets)

```java
public class Arbre {

  double taille;
  double diametreTronc;
  TypeArbre type;

  void grandir() {
    taille = taille + 10;
    diametreTronc = diametreTronc + 0.5;
  }
}

public enum TypeArbre {
  HETRE, PEUPLIER, CHENE, BOULEAU
}
```

RETOUR SUR LA CLASSE

- Se souvenir que ce n'est qu'un modèle, un « formulaire vide »
- Pour l'instant, **on n'a créé aucun arbre**
- Notre classe dit juste : « voici ce qui caractérise un arbre »
- Lorsqu'on va effectivement créer un arbre, on va vouloir préciser la valeur que prend chacun des attributs
  - ⇒ c'est le rôle du **constructeur**

CONSTRUCTEUR DE CLASSE

- Le constructeur est une méthode spéciale de la classe
- Il est appelé automatiquement lorsque qu'un objet de cette classe est créé
- Syntaxiquement, un constructeur a deux caractéristiques qui le distingue des autres méthodes :
  - il porte le **même nom que la classe**
  - il n'a **pas de type de retour** (même pas `void`)

```
public class Arbre {

  double taille;
  double diametreTronc;
  TypeArbre type;

  Arbre() {
    taille = 500;
    diametreTronc = 25;
    type = TypeArbre.PEUPLIER;
  }

  // Méthode grandir() omise...
}

public enum TypeArbre {
  HETRE, PEUPLIER, CHENE, BOULEAU
}
```

CRÉATION D'UNE INSTANCE

- Pour créer un objet de type `Arbre`, on va faire une **instanciation** avec le mot-clé `new`
- On instancie hors de la classe `Arbre`, là où l'objet est requis, par exemple ici dans la méthode principale du programme : `main()`

```
public class AppArbre {
  public static void main(String[] args) {
    Arbre peuplier = new Arbre(); // Instanciation
    // Affichage de la taille
    System.out.println("Taille : " + peuplier.taille);
    // Appel de la méthode grandir sur cette instance
    peuplier.grandir();
    // Quelle taille fait ici le peuplier ?
  }
}
```

INITIALISATION DE L'INSTANCE

- Problème : actuellement, on ne peut instancier que des peupliers, et d'une taille spécifique
- Comment choisir le type de l'arbre, ainsi que sa taille initiale
- ⇒ Il faut écrire un **constructeur paramétré**
- Les paramètres (entre parenthèses) vont permettre au client de préciser les caractéristiques initiales
- NB : on appelle **client** d'une classe tout code qui **utilise la classe**

CONSTRUCTEUR PARAMÉTRÉ

```
public class Arbre {
  double taille;
  double diametreTronc;
  TypeArbre type;

  Arbre(double taille, double diametreTronc, TypeArbre type) {
    this.taille = taille;
    this.diametreTronc = diametreTronc;
    this.type = type;
  }
}
```

LE MOT-CLÉ "THIS"

- Le paramètre est plus local à la méthode que l'attribut, et donc « cache » celui-ci
- Le mot-clé `this` permet de désigner explicitement l'instance courante
- En indiquant `this.taille`, on spécifie donc que l'on accède à l'attribut `taille` de l'instance courante, et non à la variable `taille` passée en paramètre

UTILISATION DU CONSTRUCTEUR PARAMÉTRÉ

```
public class AppArbre {

  public static void main(String[] args) {
    // Instanciation en donnant des valeurs initiales
    Arbre peuplier = new Arbre(90, 5, TypeArbre.PEUPLIER);
    Arbre bouleau = new Arbre(300, 15, TypeArbre.BOULEAU);
    // Appels de méthodes sur les instances
    peuplier.grandir();
    peuplier.grandir();
    bouleau.grandir();
  }
}
```

EXERCICE - CLASSE "EMPLOYE"

SOLUTION - CLASSE EMPLOYE

```java
public class Employe {
  String nom;
  String prenom;
  int age;
  double salaire;

  Employe(String nom, String prenom, int age, double salaire) {
    this.nom = nom;
    this.prenom = prenom;
    this.age = age;
    this.salaire = salaire;
  }

  void augmenterSalaire(int pourcentage) {
    salaire = salaire + salaire * pourcentage / 100;
  }
}
```

SOLUTION - CLIENT

```
public final class AppEmploye {
  public static void main(String[] args) {
    Employe gerard = new Employe("Lambert", "Gérard", 20, 1000);
    Employe jean = new Employe("Jaurès", "Jean", 50, 3000);

    jean.augmenterSalaire(20);

    System.out.println(gerard.prenom + " "
        + gerard.nom + " ; "
        + gerard.age + " ans ; "
        + gerard.salaire + " euros");
    System.out.println(jean.prenom + " "
        + jean.nom + " ; "
        + jean.age + " ans ; "
        + jean.salaire + " euros");
  }
}
```