R201 — Développement orienté objets

Programmation par objets en Java

Clément Pagès
clement.pages@u-pec.fr

Université Paris-Est Créteil
BUT Informatique 1re année

2025-2026

Sommaire

Idée générale

- Un objet est une entité caractérisée par ses états (appelés attributs) et les actions (appelées méthodes) qu'il peut réaliser.
- Les objets partageant les mêmes structures appartiennent à la même classe.

Idée générale

- Un objet est une entité caractérisée par ses états (appelés attributs) et les actions (appelées méthodes) qu'il peut réaliser.
- Les objets partageant les mêmes structures appartiennent à la même classe.

Exemple

- Les étudiant·es forment une classe.
  - Attributs : âge, année d'étude, formation suivie, notes, etc.
  - Méthodes : étudier, rendre un travail, passer un examen, faire la sieste, etc.

Idée générale

- Un objet est une entité caractérisée par ses états (appelés attributs) et les actions (appelées méthodes) qu'il peut réaliser.
- Les objets partageant les mêmes structures appartiennent à la même classe.

Exemple

- Les étudiant·es forment une classe.
  - Attributs : âge, année d'étude, formation suivie, notes, etc.
  - Méthodes : étudier, rendre un travail, passer un examen, faire la sieste, etc.
- Chaque étudiant·e représente une instance ou un objet de la classe.
  - Chaque étudiant·e a des valeurs spécifiques pour ses attributs, et réalise (ou pas !) certaines actions (méthodes).

Exemple concret

- En Java, chaque fichier `.java` doit contenir une et une seule classe.
- Même principe qu'en programmation modulaire.

Exemple concret

- En Java, chaque fichier `.java` doit contenir une et une seule classe.
- Même principe qu'en programmation modulaire.

Personnage.java

```java
public class Personnage { // Commence par une majuscule
    /* Attributs : variables de classe */
    private int age;    // private : attribut privé
    private final String nom; // private : attribut privé.
                              // final   : immuable

    /* Méthodes de la classe */
    public String getNom() { // public : méthode publique
        return this.nom;
    }
    public void setAge(int age_) {
        this.age = age_;
    }
    public void anniversaire() {
        this.age++;
    }
}
```

Constructeur

- Le code d'une classe ne réserve pas de mémoire.
- Chaque classe doit contenir les méthodes pour construire les objets : les constructeurs.

Constructeur

- Le code d'une classe ne réserve pas de mémoire.
- Chaque classe doit contenir les méthodes pour construire les objets : les constructeurs.

Personnage.java

```
public class Personnage {
    /* Attributs : variables de classe ... */

    /* Constructeurs */
    public Personnage() {
        this.nom = "";
        this.age = 0;
    }
    public Personnage(String nom_) {
        this.age_ = 0;
        this.nom = nom_;
    }

    /* Méthodes de la classe ... */
}
```

Intanciation

- Pour créer les objets, il faut appeler le constructeur ailleurs dans le code.
- On dit que l'on crée une instance de l'objet.

Intanciation

- Pour créer les objets, il faut appeler le constructeur ailleurs dans le code.
- On dit que l'on crée une instance de l'objet.

Main.java

```
public class Main {
    public static void main(String[] args){
        Personnage asterix   = new Personnage("Asterix");
        Personnage anonymous = new Personnage();

        asterix.setAge(32);
        System.out.println(anonymous.getNom());
    }
}
```

Sommaire

Attributs de classes

- Les attibuts définis jusqu'à maintenant se rapportent à des objets instanciés.
- On peut avoir besoin d'attributs relatifs à la classe elle-même.
- Les attributs de classe sont partagés entre toutes les instances.
- On utilise le mot-clé `static`.

Attributs de classes

- Les attibuts définis jusqu'à maintenant se rapportent à des objets instanciés.
- On peut avoir besoin d'attributs relatifs à la classe elle-même.
- Les attributs de classe sont partagés entre toutes les instances.
- On utilise le mot-clé `static`.

Personnage.java

```
public class Personnage {
    private static int nbPersonnages = 0;

    public Personnage() {
        this.nom = "";
        this.age = 0;
        nbPersonnages++;
    }
    public Personnage(String nom_) {
        this.age_ = 0;
        this.nom = nom_;
        nbPersonnages++;
    }
```

Méthodes de classes

Principe

- Les méthodes usuelles modifient l'état de l'objet sur lequel elles agissent.
- Besoin de méthodes qui ne modifient pas les objets, mais qui sont simplement des opérations, en lien avec la classe.
- Ressemblent aux fonctions de la programmation impérative.

Méthodes de classes

Principe

- Les méthodes usuelles modifient l'état de l'objet sur lequel elles agissent.
- Besoin de méthodes qui ne modifient pas les objets, mais qui sont simplement des opérations, en lien avec la classe.
- Ressemblent aux fonctions de la programmation impérative.

Exemple

Dans la classe `Math`, la méthode `Math.pow(x, a)` calcule x^a.

Méthodes de classes

Principe

- Les méthodes usuelles modifient l'état de l'objet sur lequel elles agissent.
- Besoin de méthodes qui ne modifient pas les objets, mais qui sont simplement des opérations, en lien avec la classe.
- Ressemblent aux fonctions de la programmation impérative.

Exemple

Dans la classe `Math`, la méthode `Math.pow(x, a)` calcule x^a.

- On utilise encore le mot-clé `static`.

Méthodes de classes

Exemple

Personnage.java

```
public class Personnage {
    /* Reste du code … */
    public static int getNbPersonnages() {
        return nbPersonnages;
    }
}
```

Méthodes de classes

Exemple

Personnage.java

```java
public class Personnage {
    /* Reste du code … */
    public static int getNbPersonnages() {
        return nbPersonnages;
    }
}
```

Main.java

```java
public class Main {
    public static void main(String[] args){
        Personnage asterix   = new Personnage("Asterix");
        Personnage anonymous = new Personnage();

        System.out.println(Personnage.getNbPersonnages());
    }
}
```

Sommaire

Portée des attributs et des méthodes

Notion de visibilité

- Les attributs et méthodes sont munis d'une visibilité : `public` ou `private`.

Portée des attributs et des méthodes

Notion de visibilité

- Les attributs et méthodes sont munis d'une visibilité : `public` ou `private`.
- Visibilité `private`
  - Un attribut ou une méthode `private` ne peut être appelé qu'à l'intérieur de la classe.
  - Toute tentative d'accès ou de modification depuis l'extérieur échoue.

Portée des attributs et des méthodes

Notion de visibilité

- Les attributs et méthodes sont munis d'une visibilité : `public` ou `private`.
- Visibilité `private`
  - Un attribut ou une méthode `private` ne peut être appelé qu'à l'intérieur de la classe.
  - Toute tentative d'accès ou de modification depuis l'extérieur échoue.
- La visibilité `public` autorise les accès et les modifications depuis l'extérieur.
- Teasing : il existe d'autres modes de visibilité. :-)

Portée des attributs et des méthodes

Exemple

Personnage.java

```java
public class Personnage {
    private final String nom;
}
```

Main.java

```java
public class Main {
    public static void main(String[] args){
        Personnage asterix = new Personnage("Asterix");

        System.out.println(asterix.nom);
    }
}
```

Terminal

```
> javac Main.java
Main.java:5: error: age has private access in Personnage
        System.out.println(asterix.age);
```

Portée des attributs et des méthodes

Exemple

Personnage.java

```java
public class Personnage {
    public final String nom;
}
```

Main.java

```java
public class Main {
    public static void main(String[] args) {
        Personnage asterix = new Personnage("Asterix");

        System.out.println(asterix.nom);
    }
}
```

Terminal

```
> javac Main.java
> java Main.class
Asterix
```

Portée des attributs et des méthodes

Les bonnes pratiques

- L'objet est une boite sur laquelle on agit en appelant les méthodes.
- En règle générale, l'état de l'objet ne peut être modifié qu'à l'intérieur d'une classe.

Portée des attributs et des méthodes

Les bonnes pratiques

- L'objet est une boite sur laquelle on agit en appelant les méthodes.
- En règle générale, l'état de l'objet ne peut être modifié qu'à l'intérieur d'une classe.

Les bonnes pratiques de POO : encapsulation

- Les attributs sont en visibilité `private`.
- Les méthodes sont en visibilité `public`.

Portée des attributs et des méthodes

Les bonnes pratiques

- L'objet est une boite sur laquelle on agit en appelant les méthodes.
- En règle générale, l'état de l'objet ne peut être modifié qu'à l'intérieur d'une classe.

Les bonnes pratiques de POO : encapsulation

- Les attributs sont en visibilité `private`.
- Les méthodes sont en visibilité `public`.

- D'autres situations existent, cf. partie sur l'héritage.
- Certaines méthodes peuvent parfois être `private`
  - Méthodes utilitaires qui effectuent des opérations internes à la classe.
  - Méthodes inutiles à l'extérieur de la classe.

L'encapsulation

- Le fait de placer en `private` les attributs d'un objet s'appelle l'encasulation.
- Les instances sont des « capsules » dont on ne voit pas l'intérieur.
- Accès et modification des attributs depuis l'extérieur de la classe : via des méthodes (getters et setters).

L'encapsulation

- Le fait de placer en **`private`** les attributs d'un objet s'appelle l'encasulation.
- Les instances sont des « capsules » dont on ne voit pas l'intérieur.
- Accès et modification des attributs depuis l'extérieur de la classe : via des méthodes (getters et setters).

Personnage.java

```
public class Personnage {
    private int age;
    private final String nom; // final : immuable

    public String getNom() { // public : méthode publique
        return this.nom;
    }
    public int getAge() { // public : méthode publique
        return this.age;
    }
    public void setAge(int age_) {
        this.age = age_;
    }
}
```