



# Programmation orientée objet

**Classes et encapsulation**

1. Classes et objets
2. Encapsulation
3. Exemple de classe
4. Instanciation et utilisation d'objets
5. Généricité
6. Exceptions

- Principe de programmation orientée objet :
- Paradigme de représentation de **concept**, **idée** ou **entité** manipulable par un langage de programmation.
- Représentation par une **classe**
- Patron de conception : description de ce que doit être un objet
- Manipulation d'une entité par un **objet**
- Instance d'une classe



- Logiciel à développer :
- Application de gestion des informations stagiaires d'un centre Afpas
  
- Diverses fonctionnalités administratives :
  - Inscription en formation
  - Rémunération
  - Compte cantine
  - Inscription en session d'examen

- Classe « Stagiaire » :
- **Attributs** : Quels sont les caractéristiques qui représentent un.e stagiaire ?
- **Méthodes** : Quelles sont les opérations que l'on peut faire sur un.e stagiaire ?
- Exemple d'attributs :
  - Nom
  - Prénom
  - Date de naissance
  - Identifiant Afpa
  - Argent compte cantine
  - Date d'entrée en formation
  - Date examen
- Exemple de méthodes :
  - Ajouter module au parcours
  - Inscrire à une session d'examen
  - Ajouter de l'argent sur son compte cantine

- Représentation UML :

**Nom de la classe**

**Stagiaire**

**Attributs**

- nom : string
- prenom : string
- dateNaissance : string
- idAfpa : integer
- dateDebutFormation : string
- dateDebutStage : string
- argentCantine : double

**Méthodes**

- + modifierDateExamen(in dateExamen: string)
- + ajouterArgentCompteCantine(in montant: double)

Il manque l'attribut  
dateExamen

- Lors du développement d'une classe il faudra écrire le code des méthodes → **implémentation**


## Exemple de classe - Java


```
public class Stagiaire {  
  
    private String nom;  
    private String prenom;  
    private String dateNaissance;  
    private int idAfpa;  
    private String dateDebutFormation;  
    private String dateExamen;  
    private String dateDebutStage;  
    private double argentCantine;  
  
    public void modifierDateExamen(String dateExamen) {  
        this.dateExamen = dateExamen;  
    }  
  
    public void ajouterArgentCompteCantine(double montant) {  
        this.argentCantine = this.argentCantine + montant;  
    }  
}
```

Nouvelle date d'examen  
Va venir écraser l'ancienne date  
Dans l'objet

**Objet manipulé**

- Instanciación :
- Création d'un objet de la classe qu'il représente
- **Allocation** en mémoire sur le tas (ou la « heap »)
- Objets de la classe « Stagiaire » :

 <u>Instance 1 : Stagiaire</u>
nom : string = Jebot prenom : string = Thomas dateNaissance : string = 26/06/1990 idAfpa : integer = 150230 dateDebutFormation : string = 21/03/2022 dateDebutStage : string = null

 <u>Instance 2 : Stagiaire</u>
nom : string = Bojet prenom : string = Nicolas dateNaissance : string = 25/07/1995 idAfpa : integer = 150231 dateDebutFormation : string = 21/03/2022 dateDebutStage : string = null



- Le constructeur :
- Méthode particulière appelée à l'instanciation d'un objet
- Obligatoire
- Ses paramètres doivent permettre d'initialiser des attributs de l'objet
- Possibilité de déclarer plusieurs constructeurs avec des paramètres différents

Stagiaire	
<ul style="list-style-type: none"> <li>- nom : string</li> <li>- prenom : string</li> <li>- dateNaissance : string</li> <li>- idAfpa : integer</li> <li>- dateDebutFormation : string</li> <li>- dateDebutStage : string</li> <li>- argentCantine : double</li> </ul>	
<ul style="list-style-type: none"> <li>+ modifierDateExamen(in dateExamen: string)</li> <li>+ <del>ajouterArgentCompteCantine(in montant: double)</del></li> <li>+ Stagiaire(in idAfpa: integer, in nom: string, in prenom: string, in dateNaissance: string, in dateDebutFormation: string)</li> </ul>	
	C0

Nom du paramètre

Type du paramètre

in = paramètre d'entrée

in\out : paramètre d'entrée-sortie

```
public class Stagiaire {  
  
    private String nom;  
    private String prenom;  
    private String dateNaissance;  
    private int idAfpa;  
    private String dateDebutFormation;  
    private String dateExamen;  
    private String dateDebutStage;  
    private int argentCantine;  
  
    public Stagiaire(String nomStagiaire, String prenom, String dateNaissance,  
                     int idAfpa, String dateDebutFormation) {  
        this.nom = nomStagiaire;  
        this.prenom = prenom;  
        this.dateNaissance = dateNaissance;  
        this.idAfpa = idAfpa;  
        this.dateDebutFormation = dateDebutFormation;  
    }  
}
```

# 1. Développement de la classe

## 2. Utilisation :

- Instanciation d'objets avec l'opérateur **new**
- Manipulation en appelant des **méthodes**
- Manipulation **directe des attributs** (possible si la visibilité est correcte)
- Code Java :

```
public Stagiaire(String nomStagiaire, String prenom, String dateNaissance,  
                 int idAfpa, String dateDebutFormation)
```

```
Stagiaire stagiaire1 = new Stagiaire("Jebot", "Thomas", "26/06/1990", 150230,  
    "21/03/2022");
```

```
// appel d'une méthode de la classe
```







```
stagiaire1.ajouterArgentCompteCantine(50.0);
```

```
// accès à un attribut direct
```

```
String nomStagiaire = stagiaire1.nom;
```

Il peut s'agir de développeurs/développeuses différent.e.s !

- Principe d'encapsulation :
  - les attributs et les méthodes d'un objet lui sont associés
  - le champ d'action des attributs et des méthodes est l'objet lui-même
- Intérêts :
  - protéger **l'intégrité de l'objet**
  - masquer toute la complexité des calculs effectués en interne (**abstraction du fonctionnement**)
  - Échanges codifiés avec l'extérieur (interface de programmation)
- Exemple :
  - Ajouter de l'argent sur le compte cantine d'un stagiaire  
→ **opération pouvant être complexe**

- Visibilité :
  - protéger l'accès aux attributs et aux méthodes
- Structure d'un projet Java :
  - concept de **packages**
  - **organisation des fichiers** de projets
    - ▼  src
      - ▼  fr.afpa.database
        - >  FormationData.java
        - >  StagiaireData.java
      - ▼  fr.afpa.gui
        - >  MainWindow.java
        - >  StagiaireTableview.java
      - ▼  fr.afpa.gui.resources
        -  logo-afpa.png

Fichier Main.java  
Package fr.afpa.cantine

## Autorisé :

```
Stagiaire stagiaire1 = new Stagiaire("Jebot",  
"Thomas", "26/06/1990", 150230, "21/03/2022");
```

## Interdit :

```
stagiaire1.dateExamen = "26/02/2023";
```

## Autorisé :

```
stagiaire1.modifierDateExamen("26/02/2023« ");
```

Fichier Stagiaire.java  
Package fr.afpa.cantine

```
public class Stagiaire {  
  
    private String nom;  
    private String prenom;  
    private String dateNaissance;  
    private int idAfpa;  
    private String dateDebutFormation;  
    private String dateExamen;  
    private String dateDebutStage;  
    private int argentCantine;  
  
    public Stagiaire(String nom, String prenom,  
                     String dateNaissance,  
                     int idAfpa,  
                     String dateDebutFormation) {  
  
        this.nom = nom;  
        this.prenom = prenom;  
        this.dateNaissance = dateNaissance;  
        this.idAfpa = idAfpa;  
        this.dateDebutFormation = dateDebutFormation;  
  
    }  
  
    public void modifierDateExamen(String dateExamen) {  
        this.dateExamen = dateExamen;  
    }  
}
```

- Mots clefs :
  - **public**
  - **private**
  - **protected**
- Visibilité d'une classe :
  - **public** : visible de partout
  - Par défaut : visible que dans son package
- Visibilité d'attributs et de méthodes :

	Classe	Package	Partout ailleurs
Défaut (sans mot clef)	OUI	OUI	NON
public	OUI	OUI	OUI
protected	OUI	OUI (si classe publique)	NON
private	OUI	NON	NON

- Accesseurs (« getters ») :
- Méthode permettant de récupérer le contenu d'un attribut

Stagiaire	
<ul style="list-style-type: none"><li>- nom : string</li><li>- prenom : string</li><li>- dateNaissance : string</li><li>- idAfpa : integer</li><li>- dateDebutFormation : string</li><li>- dateDebutStage : string</li><li>- argentCantine : double</li></ul>	
<ul style="list-style-type: none"><li>+ ajouterArgentCompteCantine(in montant: double)</li><li>+ Stagiaire(in idAfpa: integer, in nom: string, in prenom: string, in dateNaissance: string, in dateDebutFormation: string)</li></ul>	
<ul style="list-style-type: none"><li>+ getNom(): string</li><li>+ getPrenom(): string</li><li>+ getDateNaissance(): string</li><li>+ getIdAfpa(): integer</li><li>+ getDateDebutFormation(): string</li><li>+ getDateDebutStage(): string</li><li>+ getArgentCantine(): string</li></ul>	C0



- Mutateurs (« setters ») :
- Méthode permettant de modifier le contenu d'une donnée membre

Stagiaire	
- nom : string - prenom : string - dateNaissance : string - idAfpa : integer - dateDebutFormation : string - dateDebutStage : string - argentCantine : double	
+ modifierDateExamen(in dateExamen: string) + ajouterArgentCompteCantine(in montant: double) + Stagiaire(in idAfpa: integer, in nom: string, in prenom: string, in dateNaissance: string, in dateDebutFormation: string) <span style="color: green;">C0</span> + setNom(in nom: string) + setPrenom(in prenom: string) + setDateNaissance(in dateNaissance: string) + setDateDebutFormation(in dateDebutFormation: string) + setDateDebutStage(in dateDebutStage: string)	

- Classe paramétrable avec une autre → **classe générique**
- Exemple → **ArrayList<E>** (package java.util) :
  - Liste d'objets d'une **classe qui doit être spécifiées**
  - Avec la classe Stagiaire :

```
ArrayList<Stagiaire> list = new ArrayList<Stagiaire>();
```

```
Stagiaire stagiaire1 = new Stagiaire("Jebot", "Thomas", "26/06/1990",  
150230, "21/03/2022");
```

```
list.add(stagiaire1);
```

- Mécanisme de gestion des erreurs → **les exceptions**
- Permet d'arrêter une fonction **si un problème est rencontré**
- Utile pour la vérification de paramètres de fonctions
- Fonctionnement :
- Doit être « **jetée** » lorsqu'une erreur est rencontrée

```
public static int addPositiveNumbers(int number1, int number2) throws IllegalArgumentException {
    if (number1 < 0 || number2 < 0) {
        throw new IllegalArgumentException();
    }
    return number1 + number2;
}
```

Nouvel objet

- Appel de la fonction

```
int result = 0;
try {
    result = addPositiveNumbers(-10, 10);
} catch (IllegalArgumentException e) {
    System.out.print(e.getMessage());
}
```



Agence nationale pour la formation professionnelle des adultes  
Établissement public à caractère industriel et commercial  
Tour Cityscope  
3 rue Franklin, 93100 Montreuil  
824 228 142 RCS BOBIGNY