

## Programmation orientée objet

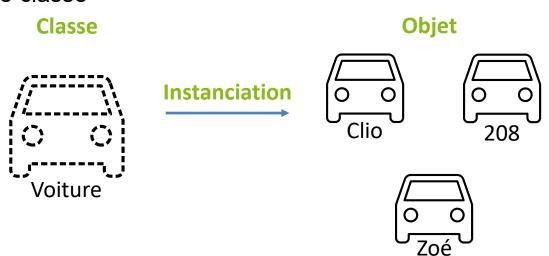
**Classes et encapsulation** 



- 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 Afpa

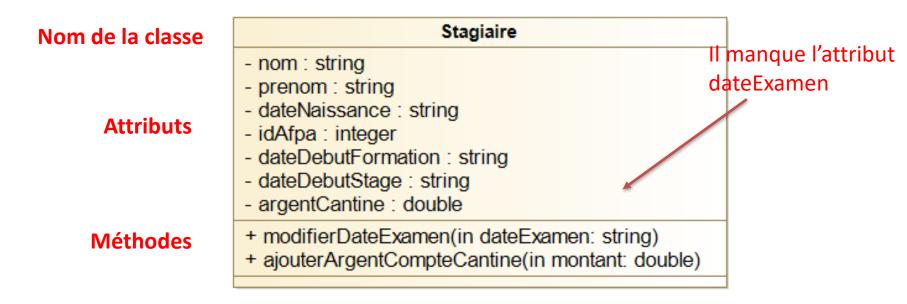
- Diverses fonctionnalités administratives :
- Inscription en formation
- Rémunération
- Compte cantine
- Inscription en session d'examen



- 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 :



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



```
public class Stagiaire {
     private String nom;
     private String prenom;
     private String dateNaissance;
     private int idAfpa;
                                                    Nouvelle date d'examen
     private String dateDebutFormation;
                                                    Va venir écraser l'ancienne date
     private String dateExamen;
                                                    Dans l'objet
     private String dateDebutStage;
     private double argentCantine;
     public void modifierDateExamen(String dateExamen) {
          this.dateExamen = dateExamen;
                                    Objet manipulé
     public void ajouterArgentCompteCantine(double montant) {
           this.argentCantine = this.argentCantine + montant;
```



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

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

## Instance 2 : Stagiaire 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

# - nom: string - prenom: string - dateNaissance: string - idAfpa: integer - dateDebutFormation: string - dateDebutStage: string - argentCantine: double + modifierDateExamen(in dateExamen: string) + ajouterArgentCempteCantine(in mentant: double) + Stagiaire(in idAfpa: integer, in nom: string, in prenom: string, in dateNaissance: string, in dateDebutFormation: string) c0

Stagiaire



```
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 :

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

11



- 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 luimême
- 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
      - > I StagiaireData.java
    - v 🖶 fr.afpa.gui
      - MainWindow.java
      - › I StagiaireTableview.java
    - → □ fr.afpa.gui.resources
      - logo-afpa.png

### Visibilité



```
public class Stagiaire {
Fichier Main.java
                                                                                             Fichier Stagiaire.java
Package fr.afpa.cantine
                                                                                          Package fr.afpa.cantine
                                                               private String nom;
Autorisé:
                                                               private String prenom;
                                                               private String dateNaissance;
Stagiaire stagiaire1 = new Stagiaire("Jebot",
"Thomas", "26/06/1990", 150230, "21/03/2022");
                                                               private int idAfpa;
                                                               private String dateDebutFormation;
                                                               private 9tring dateExamen;
Interdit:
                                                               private String dateDebutStage;
                                                               private int argentCantine;
stagiaire1.dateExamen = "26/02/2023";
                                                               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;
 Autorisé :
                                                               public void modifierDateExamen(String dateExamen) {
 stagiaire1.modifierDateExamen("26/02/2023");
                                                                     this.dateExamen = dateExamen;
                                                                                                              14
```



### 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 Stagia
<ul> <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>	
+ ajouterArgentCompteCantine(in monta	ant: double)  tring, in prenom: string, in dateNaissance: string, in dateDebutFormation: string)  co
+ getNom(): string + getPrenom(): string + getDateNaissance(): string + getIdAfpa(): integer + getDateDebutFormation(): string + getDateDebutStage(): string + getArgentCantine(): string	



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

- 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) + setNom(in nom: string) + setPrenom(in prenom: string) + setDateNaissance(in dateNaissance: string)	Stagiaire Stagiaire	
+ ajouterArgentCompteCantine(in montant: double) + Stagiaire(in idAfpa: integer_in nom: string_in prenom: string, in dateNaissance: string, in dateDebutFormation: string) + setNom(in nom: string) + setPrenom(in prenom: string) + setDateNaissance(in dateNaissance: string)	- prenom: string - dateNaissance: string - idAfpa: integer - dateDebutFormation: string - dateDebutStage: string	
+ setNom(in nom: string) + setPrenom(in prenom: string) + setDateNaissance(in dateNaissance: string)	+ ajouterArgentCompteCantine(in montant: double)	0
+ setDateDebutStage(in dateDebutStage: string)  + setDateDebutStage(in dateDebutStage: string)	+ setNom(in nom: string) + setPrenom(in prenom: string) + setDateNaissance(in dateNaissance: string) + setDateDebutFormation(in dateDebutFormation: 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;
}

• Appel de la fonction
int result = 0;
try {
    result = addPositiveNumbers(-10, 10);
} catch (IllegalArgumentException e) {
    System.out.print(e.getMessage());
}</pre>
```

