Introduction aux Classes en Java

Maria-Virginia Aponte, François Barthélémy

NFA031

1 Introduction aux structures de données

En programmation, nous avons souvent besoin de stocker et manipuler des ensembles de données. Jusqu'à présent, nous avons principalement utilisé les tableaux comme structure de données.

1.1 Rappel sur les tableaux

Un tableau en Java est une structure qui permet de stocker plusieurs éléments de **même type** dans une séquence continue en mémoire. Par exemple :

```
// Tableau d'entiers
int[] nombres = {1, 2, 3, 4, 5};

// Tableau de chaînes de caractères
String[] noms = {"Alice", "Bob", "Charlie"};
```

L'accès aux éléments se fait via un indice numérique, en commençant par 0 :

```
System.out.println(nombres[0]); // Affiche 1
System.out.println(noms[1]); // Affiche Bob
```

1.2 Limites des tableaux

Les tableaux présentent plusieurs limitations importantes :
— Ils ne peuvent contenir que des éléments de **même type** (homogènes)
— La taille est fixe une fois le tableau créé
— L'accès aux données se fait uniquement via des indices numériques
— Il n'y a pas de contrôle sur les valeurs stockées

Par exemple, si nous voulons stocker les informations d'un étudiant (nom, âge, moyenne), nous devrions créer plusieurs tableaux :

```
String[] noms = {"Dupont", "Martin", "Durand"};
int[] ages = {19, 20, 18};
double[] moyennes = {12.5, 14.0, 11.5};
```

Cette approche est problématique car :
— Les données liées à un même étudiant sont dispersées
— Il faut maintenir la cohérence entre les différents tableaux
— L'ajout d'une nouvelle information nécessite un nouveau tableau

1.3 Besoin d'un autre type de structure de données

On voudrait avoir une structure de donnée permettant de regrouper les différentes données d'un même étudiant : un seul nom, un seul age, une seule moyenne.

C'est ce que permettent de faire les classes. Elles permettent de décrire un type d'objets qui contiennent des données élémentaires et le moyen de manipuler (consulter et modifier) ces données élémentaires.

Les données d'un objet ne sont pas identifiées par un numéro comme dans un tableau mais par un nom.

2 Concept de classe

2.1 Définition et rôle

Une classe est un "moule" qui permet de créer des objets regroupant des données de types différents et des comportements associés. C'est une structure qui permet de représenter un type de données composées utilisées dans notre programme.

Prenons l'exemple d'un étudiant :

— Pour les besoins d'un programme, un étudiant possède plusieurs caractéristiques qu'il faut connaître : nom, prénom, âge, numéro étudiant, moyenne...
— Ces caractéristiques sont de types différents (String, int, double...)
— Un étudiant peut effectuer des actions : s'inscrire, passer un examen...

En programmation, une classe `Etudiant` va permettre de :

— Regrouper toutes les données relatives à un étudiant
— Assurer que ces données restent cohérentes
— Définir les opérations possibles sur ces données

Dans cette première approche des classes, nous allons nous intéresser à des classes simples comportant peu de méthodes spécifiques. Ce seront des méthodes permettant d'accéder ou de modifier une donnée élémentaire contenue dans l'objet.

L'opération de création d'un objet s'appelle l'instanciation et l'objet créé s'appelle une instance de la classe qui a servi à le créer.

3 Composants et syntaxe d'une classe

Une classe est un modèle servant à créer des objets et elle contient différents éléments décrivant ce que contiendront les objets et comment les créer :

— Les *attributs* sont les données élémentaires contenues dans chaque objet
— Le *constructeur* est une sorte de méthode appelée au moment de la création d'un objet pour initialiser cet objet
— Les *méthodes d'instance* sont des traitements, des méthodes du même genre que les méthodes statiques déjà vues, mais qui peuvent utiliser et modifier les données de l'objet.

Une classe commence par les mots-clés `public class` suivis du nom de la classe choisi par le codeur. Ensuite vient une accolade ouvrante, la listes des éléments contenus dans la classe et une accolade fermante. Il est généralement conseillé de mettre les composants dans l'ordre attributs, suivis du constructeur et pour terminer les méthodes.

3.1 Attributs

Les attributs sont des variables déclarées dans la classe. Ils définissent l'état interne de chaque instance de la classe. La syntaxe est analogue à la déclaration de variables dans une méthode, avec généralement le mot-clé `private` devant.

```
public class Etudiant {
   private String nom; // Un attribut de type String
   private int age; // Un attribut de type int
   private double moyenne; // Un attribut de type double
```

```
    ... constructeur ...
    ... méthodes ...
}
```

3.2 Le mot-clé this

Avant d'aborder les constructeurs, introduisons un mot-clé important : `this`.

Dans une classe, `this` est une référence à l'objet courant. Il permet de distinguer clairement les attributs de la classe des paramètres ou variables locales qui pourraient avoir le même nom.

Listing 1 – Utilisation de this pour référencer les attributs

```
public class Etudiant {
    private String nom;
    private int age;

    // Méthode qui utilise this pour accéder aux attributs
    public void afficherInfos() {
        System.out.println("Nom : " + this.nom);
        System.out.println("Age : " + this.age);
    }
}
```

Règle importante : `this.nom` fait toujours référence à l'attribut de l'objet, même s'il existe une variable locale ou un paramètre nommé `nom`.

Cette distinction devient cruciale dans les constructeurs où les paramètres ont souvent les mêmes noms que les attributs.

3.3 Constructeur

Le constructeur est une méthode spéciale qui initialise une nouvelle instance de la classe.

Caractéristiques du constructeur :

— Même nom que la classe

— Pas de type de retour (même pas `void`)

— Peut avoir des paramètres. Souvent, les paramètres sont utilisés pour donner une valeur aux attributs des objets créés.

```
public class Etudiant {
  private String nom; // Un attribut de type String
  private int age; // Un attribut de type int
  private double moyenne; // Un attribut de type double

  public Etudiant(String nom, int age){
    this.nom = nom;
    this.age = age;
  }

    ... méthodes ...
}
```

Dans le constructeur :

— `this.nom` désigne l'attribut de la classe

— `nom` (sans `this`) désigne le paramètre du constructeur

— L'affectation `this.nom = nom;` transfère la valeur du paramètre à l'attribut de l'objet.

À l'issue de l'exécution de ce constructeur au moment de la création d'un objet, les deux affectations auront donné des valeurs à deux des trois attributs de l'objet. En ce qui concerne le troisième, `moyenne`, il va contenir la *valeur par défaut* du type `double`, c'est-à-dire 0.0.

3.4 Accesseurs (getters)

Les accesseurs sont des méthodes qui permettent de lire la valeur d'un attribut.

Convention de nommage : `get` suivi du nom de l'attribut avec une majuscule.

Si l'attribut est déclaré `private`, il n'est pas possible de voir l'attribut (la variable) hors de la classe, mais il est possible de connaître la valeur qu'elle contient en utilisant l'accesseur (getter).

```
public class Etudiant {
   private String nom; // Un attribut de type String
   private int age; // Un attribut de type int
   private double moyenne; // Un attribut de type double

   public Etudiant(String nom, int age){
     this.nom = nom;
     this.age = age;
   }

  public String getNom() {
      return this.nom;
   }

   public int getAge() {
      return this.age;
   }

   public double getMoyenne() {
      return this.moyenne;
   }

   ... autres méthodes ...
}
```

3.5 Mutateurs (setters)

Les mutateurs (setters en anglais) sont des méthodes qui permettent de modifier les valeurs des attributs privés.

Convention de nommage : `set` suivi du nom de l'attribut avec une majuscule.

```
public class Etudiant {
   private String nom; // Un attribut de type String
   private int age; // Un attribut de type int
   private double moyenne; // Un attribut de type double

   public Etudiant(String nom, int age){
     this.nom = nom;
     this.age = age;
   }

  public String getNom() {
      return this.nom;
   }

   public int getAge() {
      return this.age;
   }
```

```
    public double getMoyenne() {
        return this.moyenne;
    }

    // Mutateurs
    public void setNom(String nom) {
        this.nom = nom;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void setMoyenne(double moy) {
        this.moyenne = moy;
    }
}
```

Note : Dans ce constructeur, nous initialisons seulement le nom et l'âge. L'attribut `moyenne` conservera sa valeur par défaut (0.0) jusqu'à ce qu'elle soit modifiée avec le mutateur `setMoyenne()`.

4 Instanciation

Une classe (par exemple `Etudiant`) doit être contenue dans un fichier source portant le même nom (`Etudiant.java`). Elle peut être utilisée dans d'autres classes situées dans le même dossier pour créer des objets. De plus le nom de la classe peut être utilisé comme un type pour déclarer des variables ou des paramètres de méthode.

Voici un exemple d'utilisation de la classe `Etudiant` dans une autre classe `Main`. La création d'un objet se fait au moyen de l'instruction `new`, la même que celle utilisée pour créer un tableau. Le mot-clé `new` est suivi du nom de la classe à instancier et de la liste des paramètres à transmettre au constructeur.

```
public class Main{
  public static void main(String[] args){
    // Création d'une instance avec le constructeur
    Etudiant etudiant1 = new Etudiant("Merlin", 18);

    // Utilisation des accesseurs
    System.out.println(etudiant1.getNom()); // Affiche "Merlin"
    System.out.println(etudiant1.getAge()); // Affiche 18

    // Utilisation des mutateurs
    etudiant1.setAge(19);
    etudiant1.setMoyenne(13.5);

    // On vérifie que les valeurs ont changé
    System.out.println(etudiant1.getAge()); // Affiche 19
    System.out.println(etudiant1.getMoyenne()); // Affiche 13.5
  }
}
```

La classe complète forme ainsi une unité cohérente où :

— Les attributs stockent les données

— Le constructeur initialise ces données
— Les accesseurs permettent de lire les données
— Les mutateurs permettent de modifier les données

5 La méthode toString

La méthode `toString()` permet de définir une représentation textuelle d'un objet. Elle est particulièrement utile pour l'affichage et le débogage.

Par défaut, sans définir `toString()`, l'affichage d'un objet donne quelque chose comme :

```
Etudiant etudiant1 = new Etudiant("Merlin", 18);
System.out.println(etudiant1);
       // Affiche quelque chose comme: Etudiant@7852e922
```

Ce résultat n'est pas très lisible car il affiche le nom de la classe suivi d'un code hexadécimal représentant l'adresse mémoire de l'objet.

Pour obtenir un affichage plus utile, on peut définir la méthode `toString()` dans notre classe :

```
public class Etudiant {
   private String nom;
   private int age;
   private double moyenne;

   public Etudiant(String nom, int age){
      this.nom = nom;
      this.age = age;
   }

   // ... getters et setters précédents ...

   @Override
   public String toString() {
      return "Etudiant{" +
            "nom=" + nom +
            ", age=" + age +
            ", moyenne=" + moyenne +
            "}";
   }
}
```

Maintenant, l'affichage sera beaucoup plus informatif :

```
Etudiant etudiant1 = new Etudiant("Merlin", 18);
etudiant1.setMoyenne(13.5);
System.out.println(etudiant1);
     // Affiche: Etudiant{nom=Merlin, age=18, moyenne=13.5}
```

La méthode `toString()` est automatiquement appelée lorsqu'un objet est utilisé dans un contexte nécessitant une chaîne de caractères, comme `System.out.println()`.

L'annotation `@Override` indique que nous redéfinissons une méthode qui existe déjà par défaut (c'est elle qui fait l'affichage peu lisible). L'annotation n'est pas obligatoire mais elle est recommandée parce qu'elle permet d'éviter certaines erreurs de programmation. Nous n'en dirons pas plus dans ce cours.

6 Objets en mémoire

6.1 Ressemblance avec les tableaux

Les objets instances de classes ressemblent beaucoup aux tableaux par la façon dont ils sont gérés :

- Il faut les créer pour qu'ils existent
- L'instruction qui crée les tableau et les objets est la même : c'est **new**
- Les objets comme les tableaux sont dans le segment de la mémoire appelé tas
- Les variables dont le type est une classe contiennent l'adresse de l'objet dans le tas. C'est ce qu'on appelle une *référence*.
- Les tableaux et les classes sont ce que l'on appelle les *types références* qui s'opposent aux types primitifs (int, double, char boolean)
- Plusieurs variables peuvent contenir l'adresse du même objet.
- Les variables dont le type est une classe peuvent contenir la valeur `null`
- Lorsqu'on réalise une affectation entre deux variables, seule la référence (adresse du tas) est recopiée, par l'objet lui-même
- Même chose lorsqu'on passe un objet en paramètre à une méthode : c'est l'adresse de l'objet qui est copiée dans le paramètre mais l'objet lui-même n'est pas recopié.

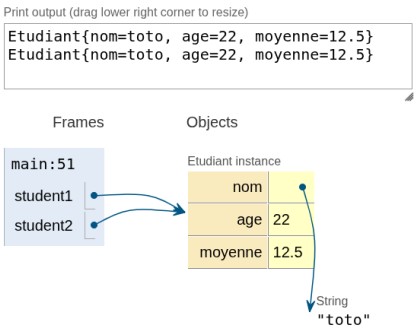
7 Deux variables pour un objet

L'exemple suivant montre le cas d'un objet référencé par deux variables de type étudiant.

```
1  public static void main(String[] args){
2    Etudiant student1, student2;
3    student1 = new Etudiant("toto",22);
4    student2 = student1;
5    student1.setMoyenne(12.5);
6    System.out.println(student1);
7           // affiche Etudiant{nom=toto, age=22, moyenne=12.5}
8    System.out.println(student2);
9           // affiche Etudiant{nom=toto, age=22, moyenne=12.5}
10 }
```

Dans ce programme, un seul objet est créé dans le tas parce qu'il y a un seul `new` et aucun appel à une méthode qui crée un objet. Il y a deux variables créées dans la pile qui contiennent la même référence. Lorsqu'une moyenne est ajoutée à `student1`, cela attribue aussi une moyenne à `student2` qui est le même objet, le même étudiant.

Voici l'image de la mémoire en fin d'exécution selon Pythontutor.

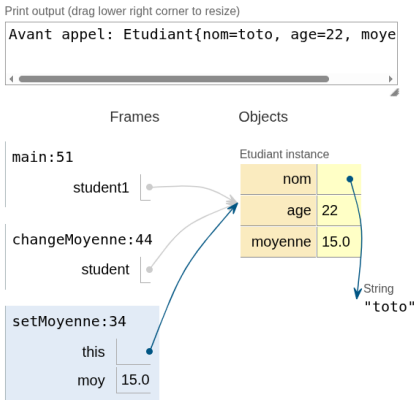


8 Passage d'un objet en paramètre

Voyons maintenant un exemple de passage d'un objet comme paramètre à une méthode.

```
package nfa031;
public class Parametre{
   public static void changeMoyenne(Etudiant student){
      student.setMoyenne(15);
   }
   public static void main(String[] args){
      Etudiant student1;
      student1 = new Etudiant("toto",22);
      System.out.println("Avant appel: " + student1);
      changeMoyenne(student1);
      System.out.println("Après appel: " + student1);
   }
}
```

Dans ce programme encore, il y a un seul objet de type `Etudiant` créé avec un seul `new`. Voyons le moment où l'appel à setMoyenne est exécuté (juste après l'exécution). La mémoire selon Pythontutor est la suivante :



On voit sur ce dessin que la variable `student` de la méthode `main`, le paramètre `student` de la méthode `changeMoyenne` et le `this` de la méthode `setMoyenne` sont la même adresse mémoire.

De ce fait, l'objet qui est modifié par le mutateur `setMoyenne` est le même objet que celui créé dans le `main` et la modification opérée par la méthode est visible lorsque l'état de cet objet est affiché dans le `main` à la fin du programme.

Le programme affiche :

```
Avant appel: Etudiant{nom=toto, age=22, moyenne=0.0}
Après appel: Etudiant{nom=toto, age=22, moyenne=15.0}
```

9 Conclusion

Nous avons mentionné au début de ce document qu'il est possible d'avoir des tableaux ou des objets à l'intérieur d'un objet. Cela se fait de façon simple en déclarant comme type d'un attribut un type de tableau ou un nom de classe. Dans ce cours, nous allons rester sur des cas simples où les attributs seront de type primitif ou du type String.

Nous avons présenté les méthodes de type accesseur et mutateur. Il peut y avoir d'autres méthodes (comme par exemple `toString`). Ce n'est pas une méthode de type getXXX, car elle utilise la valeur de plusieurs attributs et elle renvoie un résultat de type `String` quel que soit le

type des attributs. Elle partage avec les accesseurs le fait d'utiliser les valeurs des attributs sans les modifier, mais elle est plus complexe.

Par ailleurs, il n'est pas nécessaire de mettre systématiquement des accesseurs et des mutateurs pour tous les attributs d'un objet. Par exemple, si un attribut contient le code secret d'une carte de crédit, il est judicieux de garder l'information secrète et de ne fournir ni accesseur ni mutateur pour cet attribut. D'autres informations ne peuvent pas changer et dans ce cas, il ne faut pas fournir de mutateur. Par exemple, une personne ne change jamais de date de naissance. Dans un cas de ce genre on fournira un accesseur (car il est utile de connaître la date de naissance de la personne), mais pas de mutateur.

Ce chapitre a été rédigé avec l'assistance d'une intelligence artificielle, Claude 4 Sonnet de la société Anthropic