Classes et objets (en Java)

Gilles Trombettoni

IUT MPL-Sète, département info

Développement initiatique

Novembre 2021

Qu'est-ce qu'une classe ?

- Intuitivement, une classe est un moule dans le lequel on peut créer plusieurs objets/instances.
- Mathématiquement, une classe rassemble les caractéristiques communes (attributs, méthodes) des différents objets de la classe.
- Une classe décrit, pour ses instances :
  - leurs données : **attributs**
  - leur comportement : **méthodes**
- D'un point de vue programmation, une classe est un type et ses objets sont des variables.
- Une classe généralise les **enregistrements** en rassemblant au sein d'une même unité de programmation variables et procédures.

Attributs et méthodes

Les **attributs** d'un objet sont :

- d'autres objets (d'une autre classe) ;
- des variables de types de base (sauf dans les langages à objets purs).

Les **méthodes** sont des fonctions et des procédures qu'on peut appliquer à n'importe quelle instance de cette classe.

Toute instance doit être créée au moyen d'une méthode particulière appelée **constructeur**.

Les objets d'une même classe ont en commun le type des attributs et les méthodes, mais les valeurs des attributs d'un objet lui sont propres.

Encapsulation, portée

Les méthodes d'un objet accèdent aux attributs internes (à l'objet).

$\Rightarrow$ Inutile de les passer en paramètre des méthodes !

Dans les langages à objets purs, les attributs ne sont visibles que des instances de la classe où ils sont déclarés.

En Java, on déclare un attribut (de portée) *privé* (local à la classe) pour obtenir cette restriction.

Exemple d'une classe Etudiant

Cahier des charges :

Des étudiants passent un examen comportant deux épreuves : mathématiques et philosophie.

On doit observer les règles suivantes :

- les deux épreuves comptent à poids égal et sont notées sur 20 ;
- il faut obtenir une moyenne de 10 pour être reçu ;
- moins de 8 à une épreuve est éliminatoire.

Exemple d'une classe Etudiant

```
public class Etudiant  {

    private String nom;
    private int noteMaths;
    private int notePhilo;

    public Etudiant (String name, int noteM,
                     int noteP) {
        this.nom = name;
        this.noteMaths = noteM;
        this.notePhilo = noteP;
    }
    public String getNom() {
       return this.nom;
    }
    ...
```

Exemple d'une classe Etudiant

```
...
public boolean barre8 () {
   return (this.noteMaths >= 8 &&
           this.notePhilo >= 8);
}

public double moyenne () {
   return (this.noteMaths + this.notePhilo) / 2.0;
}

public String resultat () {
    if (this.moyenne() >= 10 && this.barre8())
        return "Examen reussi !";
    else
        return "Examen rate !";
}
...
```

Exemple d'une classe Etudiant

```
 ...
 public String toString () {
   String ch;
   ch = "nom : " + this.nom +
        "\n" + "notes obtenues : " +
        this.noteMaths + " "  + this.notePhilo +
        "\n" + "moyenne : " + this.moyenne() +
        "\n" + "résultat : " + this.resultat();

   return ch;
 }
} // end class
```

Exemple d'une classe Etudiant

```
public class TestEtudiant {

    public static void main (String args[]) {

        Etudiant etud1, etud2, etud3, etud4;
        etud1 = new Etudiant("Karl", 5, 17);
        System.out.println(etud1);

        etud2 = new Etudiant("Bob", 8, 17);
        System.out.println(etud2.toString());

        etud3 = new Etudiant("Ghislaine", 9, 9);
        System.out.println(etud3);

        etud4 = new Etudiant("Nathalie", 11, 9);
        System.out.println(etud4);
    }
}
```

Trace

```
trombe> java TestEtudiant

Nom : Karl
Notes obtenues : 5 17
Moyenne : 11.0
Resultat : Examen rate !
Nom : Bob
Notes obtenues : 8 17
Moyenne : 12.5
Resultat : Examen reussi !
Nom : Ghislaine
Notes obtenues : 9 9
Moyenne : 9.0
Resultat : Examen rate !
Nom : Nathalie
Notes obtenues : 11 9
Moyenne : 10.0
Resultat : Examen reussi !
```

Exemples de constructeurs

```
public class Fraction {

    private int numerateur;
    private int denominateur;

    // Constructeur vide: alloue la mémoire pour les attributs:
    public Fraction(){};

    public Fraction (int num, int denom) {
        this.numerateur = num;
        this.denominateur = denom;
    }

    public Fraction (String frac) {
        String str[] = frac.split("/");
        this.numerateur = Integer.parseInt(str[0]);
        this.denominateur = Integer.parseInt(str[1]);
    }

    public Fraction (Fraction frac) { // constructeur par recopie
        this(frac.numerateur, frac.denominateur);
    } ...
```

Portée publique et privée

Portée des attributs et des méthodes

- Une variable/attribut privé(e) ou une méthode privée n'est visible que de l'intérieur de sa classe.
- Une variable ou une méthode publique est visible par les objets des autres classes en utilisant la notation pointée ('.')
- Notion qui remplace avantageusement la notion de variable locale et globale des langages impératifs (comme C).

Exemple 1

Dans la classe TestFraction :

```
...
Fraction frac = new Fraction(34, 78);
System.out.println("Valeur de frac = "
  + ((float)frac.numerateur / (float)frac.denominateur));// NON
System.out.println("Valeur de frac = "
                   + frac.getValeurFlottante());          // OUI
...
```

Portée publique et privée

Exemple 2 : dans la classe `TestEtudiant`

```
public class TestEtudiant {

    public static void main (String args[]) {

        Etudiant etud1, etud2, etud3, etud4;
        etud1 = new Etudiant("Karl", 5, 17);
        System.out.println(etud1.nom); // pas autorisé
        System.out.println(etud1.toString()); // autorisé
        ...
    }
}
```

Portée *“rien”* ou *paquetage* en Java

Il est possible en Java de ne **rien** préciser devant un attribut : ni `public`, ni `private`, ni ...

Dans ce cas, la variable est publique... à l’intérieur du paquetage (*package*) de la classe. Autrement dit, seules les classes du paquetage peuvent la lire ou la modifier.

Accesseurs en lecture et écriture

Motivation

- En programmation à objets, on recommande de déclarer les attributs comme privés (non accessibles par une autre classe).
- Dans le cas où on veut offrir un accès de l'extérieur à un attribut, il faut alors munir la classe d'une méthode dédiée à cet usage appelée **accesseur**.

  Il est souvent inutile de fournir des accesseurs pour tous les attributs.

Accesseurs en lecture (*getters*)

```
public int getNumerateur() {// accesseur (inutile) de la classe
   return this.numerateur;
}
public float getValeurFlottante() {// méthode d'accès plus utile
   return (float)this.numerateur / (float)this.denominateur ;
}
```

Accesseurs en lecture et écriture

Accesseurs en écriture (mutateurs, *setters*)

```
public void setNumerateur(int num) {
    // mutateur (inutile) de la classe Fraction
    this.numerateur = num;
}
public void setFraction(int num, int denom) {
    // méthode plus utile
    this.numerateur = num;
    this.denominateur = denom;
}
```

Variables et méthodes de classe

Variables et méthodes de classe

- Les **variables de classe** sont propres à la classe et non pas à un objet particulier. Elles sont partagées par toutes les instances de la classe.
- Mot-clé : **static**
- Utile pour déclarer les constantes (ex : taux de TVA)
- Une **méthode de classe** est une méthode qui n'utilise que des variables de classe, ou bien une "méthode globale bien rangée", comme `Math.abs()`.

Exemple

```
class Bidon {

   private static int nbInstances = 0 ; // variable de classe

   public static int combien () {       // méthode de classe
      return nbInstances ;
   }
   public Bidon () {
      nbInstances = nbInstances + 1 ;
   }
}
```

La classe `Scanner` (du paquetage `java.util`)

Lecture des informations. Exemples de 3 usages courants.

Lecture sur le flux d'entrée standard

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Lecture dans une chaîne de caractères

```
Scanner s = new Scanner("40 30 deuce");
    System.out.println(s.nextInt());
    System.out.println(s.nextInt());
    System.out.println(s.next());
    s.close();
```

Lecture dans un fichier

```
Scanner sc = new Scanner(new File("myNumbers.txt"));
while (sc.hasNextInt()) {
    int i = sc.nextInt(); ...
}
```

La classe `String` (du paquetage `java.lang`)

Quelques généralités

- Les “variables” de type chaîne de caractères en Java sont constantes, ne peuvent pas être modifiées après la création.
- Les caractères sont codés avec Unicode.
- De nombreuses méthodes permettent de les manipuler.

Quelques méthodes

- `length()` : longueur de `this`
- `char charAt (int index)` : retourne le caractère de `this` à l'indice `index`
- `int compareTo (String s)` : retourne 0 si égales ;
  retourne nombre négatif si `this` $< s$ (lexicographiquement),...
  (voir aussi `equals` : ex : `return` s_1.equals(s_2))
- `String[] split (String regexp)` : voir plus haut (`regexp` est le séparateur),
- `String toString()` : à redéfinir pour afficher un objet
- `String toUpperCase()` : modification en lettres majuscules
- `static String valueOf(...)` : retourne la représentation par chaîne de ...

La classe `System` (du paquetage `java.lang`)

`public static final InputStream in`	entrée standard
`public static final PrintStream out`	sortie standard
`public static void exit (int etat)`	termine le processus
`public static void gc()`	lance le ramasse miettes
`static long currentTimeMillis()`	chronomètre

`System.out` : flux de sortie, de la classe `PrintStream`

`System.in` : flux d'entrée, de la classe `InputStream`

Les classes `Math` et `StrictMath` (de `java.lang`)

Quelques généralités

- Fournit les opérations mathématiques principales sur les nombres entiers et flottants. `StrictMath` donne une précision plus grande et offre des algorithmes documentés (publiés et agréés).
- Les opérations sont données sous la forme de méthodes de classe (`static`).

Quelques attributs et méthodes

- `static double E;static double PI`
- `Math.abs(-34.67),Math.arccos(Math.PI/2),Math.sin(3)`
- `Math.max(37.2,x),Math.max(-8,n),`
  `double random=Math.random()`
- `int arrondi=Math.round(65.98)`