INF112 - Programmation orientée objet en Java

Types de données, classes, objets et constructeurs

Dominique Blouin
Télécom Paris, Institut Polytechnique de Paris
dominique.blouin@telecom-paris.fr

Objectifs d'apprentissage

- Types scalaires et leurs opérateurs.
- Classes, objets et constructeurs.
- Types références.
- Variables de classes et d'instance
- A propos de Java…

Les types de base (scalaires) en Java

- **Variable**: zone de mémoire nommée.

- Toute variable ou donnée manipulée par le langage Java possède un **type**.
  - Comme en langage C.

- Lorsqu'une variable est déclarée, le programmeur déclare son type. La variable ne pourra contenir que des valeurs de ce type.

- Java est un langage **fortement typé**.
  - A l'opposé de Python ou de Javascript par exemple.

Le type `int`

- Un attribut ou une variable nommée **`var`** de type **`int`** se déclare ainsi :

```
int var;
```

- Un attribut de type **`int`** est initialisé par défaut avec la valeur **`0`**.

- Une variable de type **`int`** peut être explicitement initialisée avec **l'opérateur d'affectation** :

```
int var = 0;
```

- C'est une très bonne pratique d'écrire **explicitement** l'initialisation.

- Les valeurs entières sont représentées en entiers signés de 32 bits et donc comprises entre -2147483648 et 2147483647.

Autres types de base

Type	Signification	Valeurs possibles
`char`	Caractère	Jeu de caractères Unicode notés **`'a'`**, **`'b'`**,...
`byte`	Entier très court	
`short`	Entier court	**`-32768`** à **`32767`**
`int`	Entier	**`-2 147 483 648`** à **`2 147 483 647`**
`long`	Entier long	**`-9223372036854775808`** à **`9223372036854775807`**
`float`	Flottant (réel)	$1.4*10^{-45}$ à $3.4*10^{38}$
`double`	Flottant double	$4.9*10^{-324}$ à $1.7*10^{308}$
`boolean`	Booléen	**`true`** ou **`false`**

- **Initialisation par défaut des variables:**
  - Les variables dont le type est numérique sont initialisés avec la valeur zéro.
  - Les variables de type **`boolean`** sont initialisés avec **`false`**.

Opérateurs arithmétiques

- Java connaît les principales opérations arithmétiques qui permettent de construire des expressions :

```
int xVar = 20;
int yVar = xVar * xVar; // multiplication
int zVar = xVar + yVar; // addition
xVar = zVar / xVar; // division

yVar = zVar % xVar; // modulo (reste de la division)

int uVar = (zVar * (xVar + yVar)) - xVar; // parenthèses
```

- Les opérateurs arithmétiques fonctionnent également avec les **float** et les **double** (à l'exception de **%**).

Opérateurs de comparaison

- Les opérateurs de comparaisons permettent de comparer les nombres. Ils renvoient une valeur de type **`boolean`**.

- Exemples:

```
boolean myBool0 = (xVar <= yVar);
boolean myBool1 = (xVar < yVar);
boolean myBool2 = (xVar == yVar); // test d'égalité
boolean myBoolB3 = (xVar != yVar);
boolean myBoolB4 = (xVar > yVar);
boolean myBool5 = (xVar >= yVar);
```

Opérateurs logiques

- Les opérateurs logiques agissent sur les booléens.

- Exemples:

```
boolean myBool0 = (xVar < yVar) && (xVar > zVar) ; // ET logique
boolean myBool1 = (xVar < yVar) || (xVar > zVar) ; // OU logique
boolean myBool2 = ! myBool1 // NEGATION logique
```

Le type Classe

- Supposons que l'on écrive un programme traitant de concepts géométriques dans un plan. On créera alors une classe **`Point`** pour décrire un point d'un plan :

```
class Point {
    int xCoord;
    int yCoord;
}
```

- Un point est caractérisé par ses deux coordonnées dans le plan.
  - On déclare donc deux attributs dans la classe nommés **`xCoord`** et **`yCoord`**.
  - Ils sont de type **`int`**, ce qui signifie qu'ils peuvent prendre des valeurs entières (0, positives ou négatives).

Le type référence

- Si une classe est définie (par exemple la classe **`Point`** de notre exemple), il est possible de créer un **objet** de cette classe.

- Pour cela, il faut déclarer une **variable** pouvant contenir une **référence** sur un objet de la classe **`Point`**.

- Celle-ci se déclare ainsi :

```
Point myPoint;
```

- Une variable est une zone de mémoire nommée et typée:
  - Ici, le nom est **`myPoint`**.
  - Elle ne peut contenir que des valeurs d'un certain **type**.
  - Ce type est une **référence** sur un **objet** de la classe **`Point`**.
  - C'est le même principe que pour une variable de type **`struct`** en langage C

Initialisation de variable de type référence

- Une variable de type référence doit être initialisée, possiblement avec une valeur très particulière qui s'écrit **`null`**.

- Cette variable ne référence alors aucun objet.

- Toute tentative d'envoyer un message en utilisant une variable de référence non valorisée (de valeur **`null`**) se traduira par une erreur à l'exécution et par l'arrêt du programme si cette erreur n'est pas gérée.
  - A voir plus tard lors de la gestion des exceptions…

Création d'un objet

- La variable nommée **`myPoint`** est une variable de type **référence** sur un **objet** de la classe **`Point`**.

- On peut alors créer un objet de la classe **`Point`** avec l'instruction **`new`**:

```
myPoint = new Point();
```

- Le symbole **`=`** est appelé **symbole d'affectation**:
  - La variable à gauche du symbole reçoit la valeur à droite du symbole.
  - Ici, la valeur est une **référence** sur un **objet** de la **classe `Point`** qui est créé par l'instruction **`new`**.

- Une erreur commune chez les débutants est de confondre le symbole d'affectation (**`=`**) des langages de programmation avec le symbole d'égalité (=) des mathématiques.
  - Dans les langages de programmation, le symbole d'égalité s'écrit `==`.

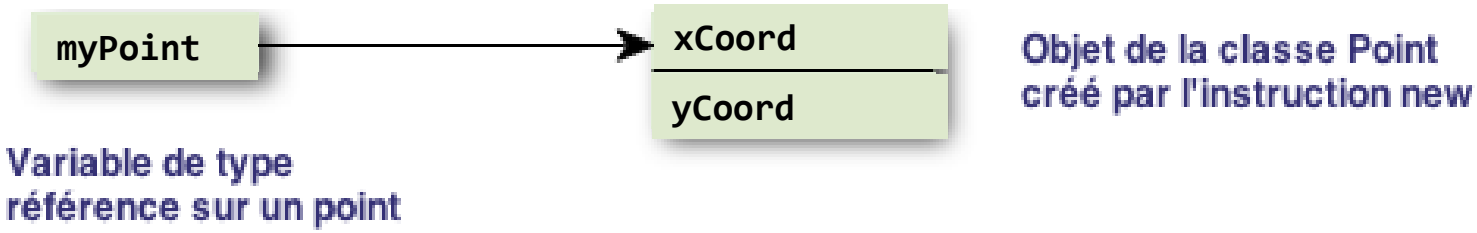
Création d'un objet

- On peut regrouper la déclaration de la variable et la création de l'objet :

```
Point myPoint = new Point();
```

- Dans les deux cas, le résultat est le même.

- Illustration de ce qui se passe en mémoire :

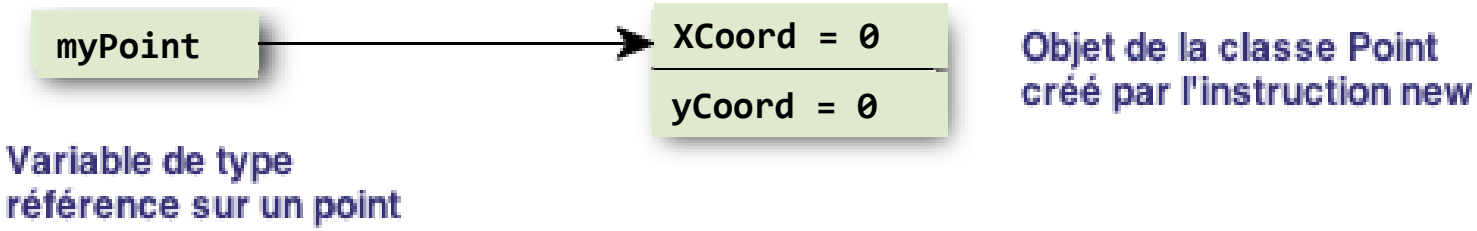


La référence est représentée par une flèche vers l'objet référencé.

On notera que **`myPoint`** ne contient pas le point mais une **référence** vers la **mémoire** qui stocke les données du point (ses coordonnées).

Initialisation des attributs d'un objet

- Quelles sont les valeurs des attributs **`xCoord`** et **`yCoord`** après la création de l'objet ?



- La spécification de Java nous apprend que les données (variables et attributs) de type **`int`** sont initialisés avec la valeur **`0`**.

- Attention, cela est vrai en Java mais pas pour tous les langages de programmation!
  - D'où l'importance d'initialiser les variables **explicitement**.

Les constructeurs

- Nous avons vu que Java initialise les attributs avec des valeurs par défaut.
  - Par exemple, un attribut de type **`int`** sera initialisé à **`0`**.

- Cependant, on peut vouloir initialiser les attributs avec des valeurs autres que les valeurs par défaut.

- Pour cela, il faut définir un **constructeur spécifique**.

Les constructeurs

- Un **constructeur** se présente comme une **méthode** (fonction dans une classe) qui a le même nom que la classe.

- Elle aura éventuellement des **paramètres** mais aucun type pour une valeur de retour.

- Exemple:

```
Point(int xCoordInit,
      int yCoordInit) {
    xCoord = xCoordInit;
    yCoord = yCoordInit;
}
```

- On peut alors créer un **`Point`** en utilisant le constructeur:

```
Point myPoint = new Point(23, 67);
```

Les constructeurs

- Il est possible de définir plusieurs constructeurs dans une classe.
- Cela permet de définir des initialisations différentes des attributs.
- Exemple de constructeur pour coordonnées polaires :

```
Point(double rho,
      double theta) {
    xCoord = (int)(rho * Math.cos(theta));
    yCoord = (int)(rho * Math.sin(theta));
}

Point myPoint = new Point(10.12, 1.34); // angle in radians
```

Les constructeurs

- Une classe peut avoir autant de constructeurs que nécessaire.
  - Chaque constructeur propose une initialisation particulière.

- Chaque constructeur devra avoir des **paramètres différents** des autres constructeurs, soit en nombre, soit en types.

- Le constructeur utilisé lors de la création d'un objet est déterminé par les types des paramètres donnés à l'instruction **`new`**.

  `new Point(23, 67)` utilisera un constructeur ayant deux paramètres de type **`int`**

  `new Point(10.62, 1.67)` utilisera un constructeur ayant deux paramètres de type **`double`**.

- C'est pour cela que différents constructeurs doivent être déclarés avec des **paramètres différents**.

- Si un ou plusieurs constructeurs existent dans une classe, alors la création d'un objet devra obligatoirement utiliser l'un de ces constructeurs.

- Sinon un **constructeur par défaut** (n'ayant aucun paramètre) sera fourni automatiquement, bien que pas nécessairement visible dans le code.

Appel de constructeur et mot clé `this`

- Il est possible qu'un constructeur fasse appel à un autre constructeur.
- Cela se fait avec le mot clé **`this`** suivi des paramètres entre parenthèses.
- Ce mot clé **`this`** sert à spécifier que l'on parle d'un élément de la classe dans laquelle le code est écrit (i.e., le constructeur à appeler).

```
Point(double rho, double theta) {
    this((int)(rho * Math.cos(theta)),
         (int)(rho * Math.sin(theta)));
}
```

- Cet appel de constructeur doit être la première instruction du constructeur appelant.
- Le mot clé **`this`** peut également être utilisé pour spécifier que l'on parle d'un **attribut** de la classe et ainsi éviter une **ambiguïté de nommage** avec les paramètres.

```
 9    Point(int xCoord,
10          int yCoord) {
11        xCoord = xCoord;
12        yCoord = yCoord;
13    }
```

```
Point(int xCoord,
      int yCoord) {
    this.xCoord = xCoord;
    this.yCoord = yCoord;
}
```

L'envoi de message entre les objets : les méthodes

- Pour qu'un objet puisse recevoir un message, il doit déclarer une **méthode** associée au type de message souhaité.

- L'en-tête d'une méthode est composée de:
  - Le mot clé de visibilité (à voir plus tard)
  - Le type de la réponse, ou **`void`** s'il n'y a pas de réponse.
  - Le nom de la méthode.
  - La parenthèse ouvrante **`(`**.
  - La liste des paramètres avec leurs noms et leurs types.
  - La parenthèse fermante **`)`**.

- L'en-tête est aussi appelé **signature** de la méthode.

L'envoi de message entre les objets : les méthodes

- Exemple: afficher les coordonnées d'un point:

```
class Point {
    int xCoord;
    int yCoord;

    void writeCoordinates() {
        System.out.print("x = ");
        System.out.println(xCoord);
        System.out.print("y = ");
        System.out.println(yCoord);
    }
}
```

- Pour accéder à un attribut ou à une méthode d'un objet, on utilise le caractère « `.` ».

```
Point myPoint = new Point(10, 10);
myPoint.writeCoordinates();
```

Autre solution

- Accéder directement aux attributs de la classe :

```
Point myPoint = new Point(10, 10);
System.out.print("x = ");
System.out.println(myPoint.xCoord);
System.out.print("y = ");
System.out.println(myPoint.yCoord);
```

- Quelle est la meilleure solution entre l'accès direct aux attributs et l'utilisation de la méthode **`writeCoordinates()`**?

Délégation et non-intrusion

- Une bonne pratique de l'OO est d'éviter d'agir depuis l'extérieur sur l'état d'un objet (**non-intrusion**).
- Ainsi, on évitera autant que possible les opérations de lecture et d'écriture des attributs.
- On va donc demander à l'objet de type **`Point`** d'afficher ses coordonnées (principe de **délégation**).
- La meilleure manière:

```
Point myPoint = new Point(10, 10);
myPoint.writeCoordinates();
```

Variables de classes et d'instance

- Chaque objet d'une classe possède ses propres valeurs d'attributs.
- Donc, les objets d'une même classe n'ont aucun lien entre eux, aucune valeur partagée, si ce n'est leur classe.
- Tout ce qui est déclaré dans la classe, les méthodes par exemple, est **partagé** par tous les objets de la classe.
- Il peut cependant arriver qu'il soit préférable que les instances d'une même classe aient besoin d'une **valeur partagée**.

Exemple : compter le nombre total de points

```
class Point {
    static int numberOfPoints = 0;
    int xCoord;
    int yCoord;

    Point(  int xCoordInit,
            int yCoordInit) {
        xCoord = xCoordInit;
        yCoord = yCoordInit;

        numberOfPoints = numberOfPoints + 1;
    }

    void writeNumberOfPoints() {
        System.out.print("number of points = " );
        System.out.print(numberOfPoints);
    }
}
```

Exemple : qu'est-ce qui sera affiché à la console?

```
Point point1 = new Point(0, 4);
Point point2 = new Point(5, 9);
Point point3 = new Point(9, 0);

point1.writeNumberOfPoints();
point2.writeNumberOfPoints();
```

- L'attribut **`numberOfPoints`** est déclaré avec le mot clé **`static`**.
- C'est une variable de **classe** : sa valeur est **détenue par la classe**.
- Elle est partagée par **tous les objets (instances)** de la classe.
- Ainsi, la même valeur **3** du nombre de points sera affichée pour chaque instance **`point1`** et **`point2`**.
- Comme nous le verrons plus tard, dans la pratique, les variables de classe sont **peu utilisées**.

Méthode de classe

- Une méthode peut aussi être qualifiée avec le mot clé **`static`**.

```
class Point {
    static int numberOfPoints = 0;
    int xCoord;
    int yCoord;

    Point(  int xCoordInit,
            int yCoordInit) {
        xCoord = xCoordInit;
        yCoord = yCoordInit;

        numberOfPoints = numberOfPoints + 1;
    }

    static void writeNumberOfPoints() {
        System.out.print("number of points = " );
        System.out.print(numberOfPoints);
    }
}
```

Appel d'une méthode de classe

```
Point point1 = new Point(0, 4);
Point point2 = new Point(5, 9);
Point point3 = new Point(9, 0);

point1.writeNumberOfPoints();
Point.writeNumberOfPoints();
```

- Une méthode de classe est exécutée par la classe, et non pas par l'objet.
- En conséquence il n'est pas nécessaire de connaître une **instance particulière** pour appeler la méthode.
- Par ailleurs, une méthode de classe n'a accès qu'aux **variables de classe**.
- Si on accède à une variable d'instance dans une méthode de classe, il y aura une **erreur de compilation**.

Exécuter un programme Java

- Un programme Java doit toujours contenir au moins une classe qui aura une **méthode de classe** qui sera appelée au tout début du programme :

```
public static void main(String[] args) {
    Point myPoint = new Point(10, 10);
    myPoint.writeCoordinates();
}
```

- **`void main(String[] args)`** : Signature requise pour indiquer le point d'entrée du programme.

- **`String[] args`** : La classe **`String`** sera introduite au prochain cours…

Bonnes pratiques de nommage dans le code

- L'anglais est la langue de l'informatique. Utilisez donc l'anglais pour nommer vos programmes, écrire vos commentaires, etc.

- Utiliser des lettres autres que celles de l'alphabet anglais risque de créer des problèmes.
  - Eviter les lettres accentuées par exemple.

- Choisir des noms **représentatifs** (éviter les noms à quelques lettres) et les **capitaliser.**

```
class Point {
        int xCoord;
        int yCoord;

        void writeCoordinates() {
            …
        }
}
```

Autres règles de nommage

- Le nom d'une classe commence par une **majuscule**.
- Le nom des variables, attributs, et méthodes par une **minuscule**.

```
class Point {
    int xCoord;
    int yCoord;

    void writeCoordinates() {
        …
    }
}
```

- Permet d'identifier ces éléments plus facilement

A propos de Java

- Java a été créé en 1991 par Sun Microsystems. L'objectif était la programmation de petits appareils comme des télécommandes.
- Java fut ensuite appliqué à la programmation d'applications dans les navigateurs Web: les applets.
- Java porte son nom à cause de la boisson préférée de ses concepteurs.
- Sun Microsystems a été racheté par Oracle.

A propos de Java

- Java est un langage de programmation généraliste utilisé par l'industrie dans une multitude de domaines.

- Le JDK (Java Development Kit) propose un très grand nombre de bibliothèques qui forment une boîte à outils pour le développement de logiciels.
  - Dans cet enseignement, nous apprendrons à utiliser cette boîte à outils.

- Des communautés de développeurs proposent sur le Web des solutions (un ensemble de classes) plus ou moins élaborées, génériques, éprouvées et documentées, permettant le développement d'applications en général à grande échelle.
  - Exemple: les fondations Eclipse et Apache.

- C'est une des grandes forces de Java.

Portabilité de Java

- Une caractéristique importante de Java est sa **portabilité** :
  - Une même application peut s'exécuter sur différents ordinateurs, en faisant abstraction des spécificités du matériel et du système d'exploitation.

- Pour une architecture classique de compilateur, le compilateur produit, à partir du programme, un fichier d'octets, le fichier **exécutable**, qui est exécutable par l'UAL.
  - Ce fichier exécutable est **dépendant** du processeur et du système d'exploitation.

- Java est un langage portable car il est mis en œuvre par une **machine virtuelle**.

Portabilité de Java

- Cette machine virtuelle cache les détails du processeur et du système d'exploitation en leur substituant (ou en émulant) un environnement d'exécution qui est toujours le même sur tous les ordinateurs.

- Une machine virtuelle est un ordinateur qui n'existe pas. Ses caractéristiques (processeur, mémoire, système d'exploitation, entrées-sorties, etc.) sont définies par un document papier.

- Le compilateur de Java produit un fichier exécutable mais pour une **machine virtuelle**.

- Une machine virtuelle Java s'appelle **JVM** (Java Virtual Machine).

Portabilité de Java

- Un fichier exécutable Java n'est pas exécuté directement par l'ordinateur.
- Un programme émulant la JVM exécute le fichier exécutable.
- Des programmes émulant la JVM existent pour tous les ordinateurs et tous les systèmes d'exploitation.
- Devise de SUN: **Compile once, run everywhere**.