ITI 1521. Introduction à l'informatique II

Programmation orientée objet : attributs, variables et méthodes d'instance

by

Marcel Turcotte

Version du 15 janvier 2020

Préambule

Préambule

Aperçu

Aperçu

Programmation orientée objet : attributs, variables et méthodes d'instance

Nous analysons ensemble un système informatique complexe, tel qu'une application Web pour le commerce électronique, afin d'identifier les principaux objets, leurs attributs et comportements, ainsi que les associations entre ces objets. Nous découvrons ensemble que la programmation orientée objet rend concrète une activité abstraite.

Objectif général :

- Cette semaine, vous serez en mesure d'identifier les principaux objets d'un système informatique.

Une videeo d'introduction :

- `https://www.youtube.com/watch?v=IE-rP-xPoaY`

Préambule

Objectifs d'apprentissage

Objectifs d'apprentissage

- **Identifier** les attributs et les comportements des objets d'un système informatique.
- **Reconnaitre** les principales associations entre les objets d'un système informatique.
- **Expliquer** dans vos propres mots les concepts suivants : variable d'instance et de méthode d'instance
- **Concevoir** un programme Java simple illustrant les concepts de base de la programmation orientée objet.

Lectures :

- Pages 573-579 de E. Koffman et P. Wolfgang.

Lectures (suite) :

Entrée en matière

- docs.oracle.com/javase/tutorial/java/concepts/object.html
- docs.oracle.com/javase/tutorial/java/concepts/class.html

Informations détaillées

- docs.oracle.com/javase/tutorial/java/javaOO/classes.html
- docs.oracle.com/javase/tutorial/java/javaOO/objects.html
- docs.oracle.com/javase/tutorial/java/javaOO/more.html

Exercises

- docs.oracle.com/javase/tutorial/java/concepts/QandE/questions.html
- docs.oracle.com/javase/tutorial/java/javaOO/QandE/creating-questions.html

Préambule

Plan du module

Plan

Théorie

Discussion

- Java est un langage **orienté objet**
  - **Qu'est-ce que** l'orienté objet ?
  - **Pourquoi** l'orienté objet ?

Programmation orientée objet

- La conception de logiciels est une **activité abstraite**, en ce sens qu'on ne peut toucher ou voir ses éléments constitutifs.
- La programmation orientée objet fournit des outils, classes et objets, qui rendent ce processus **concret, visible, palpable** ; on peut dessiner des diagrammes illustrant les objets d'un système ainsi que leurs interactions.

Abstractions

- Les **abstractions** nous permettent d'**ignorer les détails** d'un concept et de nous concentrer sur certains aspects jugés importants.
- On compare souvent une abstraction à une « **boîte noire** ». On peut l'utiliser sans se préoccuper de son contenu.

Abstractions en informatique

- **Variable** : associe un **nom** à **un emplacement de la mémoire**
- **Fonction*** : découper un problème, cacher les détails, réutilisable

*Sous-routine, routine, procédure, méthode

Programmation orientée objet

Le concept **central** de la programmation orientée objet est l'**objet** !
Un **objet** possède

- des **propriétés** (attributs)
- des **comportements** (méthodes)

Un logiciel est vu comme **une collection d'objets** qui interagissent, les uns avec les autres, afin de résoudre un problème commun.

Définir de nouveaux types

- Java est un langage **orienté objet**.
- Le programmeur définit de nouveaux types à l'aide de **classes et d'objets**.
- Une **classe** définit les caractéristiques (propriétés et comportements) des **objets**.

Exemple : Entier

- Définissons donc un **nouveau type** !
- Pour rendre l'exemple simple,
  - Chaque « élément » du type **Entier** comprend simplement une **valeur** de type entier (un **int**).
  - Il y a deux **opérations** : **getValue** et **plus**

Exemple : Entier

```
class Entier {
  int value;

  Entier(int v) {
    value = v;
  }

  int getValue() {
    return value;
  }

  Entier plus(Entier other) {
      int sum;
      sum = value + other.value
      return new Entier(sum);
  }

}
```

Exemple : Entier (suite)

```
Entier a, b, c;

a = new Entier(10);

b = new Entier(5);

c = a.plus(b);
```

Programmation orientée objet

Identifier les objets :

- **Commerce-e :** clients, items, inventaire, transactions, . . . ;
- **Jeu d'échecs :** pièces, tableau de jeu, joueurs ;
- **Manufacture :** lignes d'assemblage, robots, items, pièces, . . . ;

Programmation orientée objet

- Pour certaines «classes» d'objets, il n'y a qu'une «instance» ; c'est le cas du tableau dans l'application du jeu d'échecs.
- Alors que d'autres classes décrivent les caractéristiques communes d'une collection d'objets.
- Chaque objet est **unique** bien que 2 objets peuvent avoir le même état (le contenu des variables d'instances est le «même»).

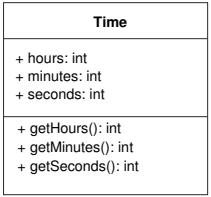
Objet

Un **objet** a

- des **propriétés** qui définissent son **état** ;
- des **comportements** : ce que l'objet peut faire, ses réponses aux requêtes.

Unified Modeling Language (UML)

- **UML** est un langage graphique standardisé afin de modéliser des systèmes logiciels orientés objet.
- Un diagramme de classe est constitué d'une boîte divisée en trois sections : le nom de la classe, les attributs et les méthodes.



Nous introduirons d'autres éléments de notation lorsque nécessaires.

Simon Benett, Steve McRobb and Ray Farmer (1999) *Object-Oriented Systems Analysis and Design using UML*. McGraw-Hill.

Exemple

Modéliser un compteur manuel

- Imaginer le dispositif utilisé dans les sports afin de **cumuler** les points.
- Une fenêtre permettant de **lire la valeur** courante.
- Un bouton permettant d'**incrémenter la valeur du compteur de 1**
- Un bouton afin de **remettre la valeur à zéro**.

Wikimedia Commons/usager **Wesha**

Pourquoi pas ?

- Une **seule valeur** est nécessaire pour modéliser le compteur.
- De plus, cette valeur peut facilement être représentée à l'aide d'**un type primitif** de Java, tel que **int** ou **long**.

Voici un compteur,

```
int c1;
```

Compteur : orienté objet

```
public class Counter {

    private int value = 0;

    public int getValue() {
      return value;
    }

    public void incr() {
        value = value + 1;
    }

    public void reset() {
        value = 0;
    }
}
```

Compteur : orienté objet

```
public class Test {
    public static void main(String[] args) {
        Counter c;
        c = new Counter();
        System.out.println(c.getValue());
        for (int i=0; i<5; i++) {
            c.incr();
            System.out.println(c.getValue());
        }
    }
}
```

Compteur : orienté objet

```java
public class Test {
    public static void main(String[] args) {
        Counter c;
        c = new Counter();
        System.out.println(c.getValue());
        for (int i=0; i<5; i++) {
            c.incr();
            System.out.println(c.getValue());
        }
        c.value = -9;
    }
}
```

```
Test.java:13: value has private access in Counter
            c.value = -9;
              ^
```

Concepts

Définition : variable d'instance

Une **variable d'instance** est une variable définie dans le corps de la classe et telle que **chaque instance** (objet) en possède **sa propre copie**.

```
class Point {
    int x;
    int y;
}
```

- **Par défaut**, les variables définies dans le corps de la classe sont des **variables d'instance**.

Définition : méthode d'instance

Une **méthode d'instance** est une méthode définie dans le corps de la classe, que l'**on ne peut qu'appeler que dans le contexte d'une instance** (objet), et qui **a accès aux variables d'instance de cet objet**.

```
class Point {
    int x, y;

    int getX() {
        return x;
    }

    int getY() {
        return y;
    }
}
```

- **Par défaut**, les méthodes définies dans le corps de la classe sont des **méthodes d'instance**.

Définition : contexte d'une instance

La **notation pointée** est utilisée afin de faire appel à une méthode d'instance de l'objet désigné par une variable référence.

```
Point p1;
p1 = new Point();

Point p2;
p2 = new Point();

p1.getX();
p2.getX();
```

- La variable référence fournit le **contexte de l'objet** : on appelle la méthode de l'objet désigné.

Définition : constructeur

Un **constructeur** est une méthode d'instance bien spéciale :

- Le constructeur a le **même nom** que **la classe**.
- On ne l'appelle que dans le contexte de la création d'un objet, à la suite du mot clé **new**.
  - Il est impossible d'y faire appel dans un autre contexte.
- Cette méthode **ne retourne aucune valeur**.
  - C'est logique ! Voyez-vous **pourquoi** ?
- Le constructeur sert à **initialiser les variables d'instances** !

Définition : constructeur (suite)

```
class Point {
    int x;
    int y;

    Point(int xInit, int yInit) {
        x = xInit;
        y = yInit;
    }

}
```

```
Point p;
p = new Point(1024, 20148);
```

La classe Point

```
class Point {
    int x;
    int y;

    Point(int xInit, int yInit) {
        x = xInit;
        y = yInit;
    }

    int getX() {
        return x;
    }

    int getY() {
        return y;
    }

}
```

Méthodes d'accès

```
class Point {
    int x;
    int y;

    int getX() {
        return x;
    }

    int getY() {
        return y;
    }

    void setX(int xVal) {
        x = xVal;
    }

    void setY(int yVal) {
        y = yVal;
    }
}
```

Passer la référence d'un objet comme paramètre

```
class Point {
    int x;
    int y;

    boolean equals(Point other) {
        boolean truth;
        if (x == other.x && y == other.y) {
            truth = true;
        } else {
            truth = false;
        }
        return truth;
    }
}
```

Passer la référence d'un objet comme paramètre

```
class Point {
    int x;
    int y;

    boolean equals(Point other) {
        return (x == other.x && y == other.y);
    }
}
```

```
Point p1, p2, p2;

p1 = new Point(10, 10);
p2 = new Point(1024, 1032);
p3 = new Point(10, p1.getY());
```

Quel sera le résultat de **p1.equals(p2) p1.equals(p3)** ?

- Les concepts de **variables d'instance** et de **méthodes d'instances** sont fondamentaux pour bien comprendre la programmation orientée objet.

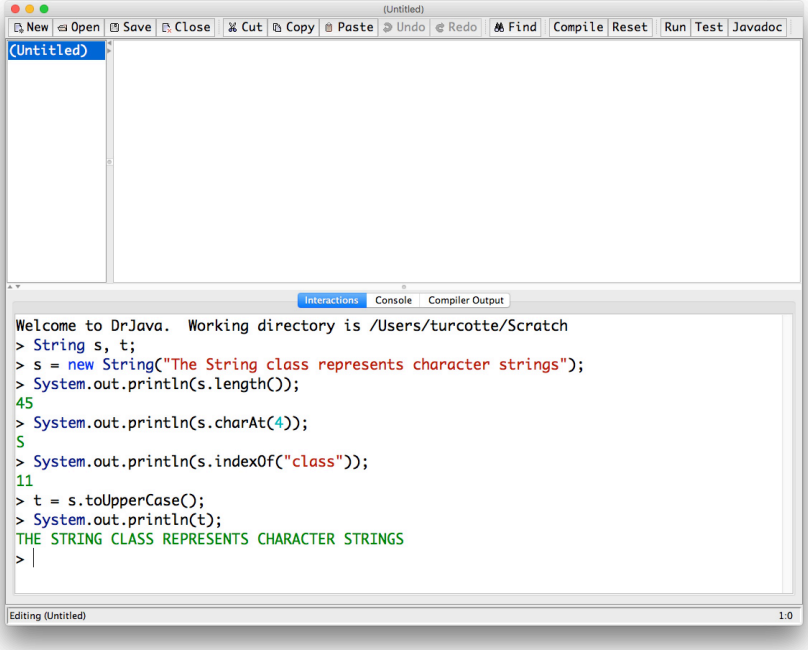
```
class Point {
    int x;
    int y;

    void translate(int deltaX, int deltaY) {
        x = x + deltaX;
        y = y + deltaY;
    }
}
```

Classes prédéfinies de Java

- Familiarisez-vous avec la **documentation de Java**
  - `http://docs.oracle.com/javase/8/docs/api/overview-summary.html`
- En particulier, les classes du «package» **java.lang**
  - `http://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html`
- Consultez la documentation de la classe **String**
  - `http://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

Classes prédéfinies de Java : String

```
String s, t;

s = new String("The String class represents ...");

s.length();

s.charAt(4);

s.indexOf("class");

t = s.toUpperCase();
```


(Untitled)
New
Open
Save
Close
Cut
Copy
Paste
Undo
Redo
Find
Compile
Reset
Run
Test
Javadoc
(Untitled)
Interactions
Console
Compiler Output
Welcome to DrJava. Working directory is /Users/turcotte/Scratch
> String s, t;
> s = new String("The String class represents character strings");
> System.out.println(s.length());
45
> System.out.println(s.charAt(4));
S
> System.out.println(s.indexOf("class"));
11
> t = s.toUpperCase();
> System.out.println(t);
THE STRING CLASS REPRESENTS CHARACTER STRINGS
>
Editing (Untitled)
1:0

Prologue

Résumé

- Les langages à haut niveau sont **expressifs**
- La programmation orientée objet rend l'activité de programmation plus **concrète**
- Un **objet** a
  - un **état** (valeurs des variables d'instance), et
  - des **comportements** (ses méthodes d'instance)

Prochain module

- **Programmation orientée objet** : variables de classes, méthodes de classe, modificateurs de visibilité, la référence **this**.

References I


E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa