ITI 1121. Introduction to Computing II

Object-oriented programming: attributes, instance variables and methods

by

Marcel Turcotte

Version January 13, 2020

Preambule

Preambule

Overview

Overview

Object-oriented programming: attributes, instance variables and methods

We analyze together a complex computer system, such as a web-based e-commerce application to identify the main objects, their attributes and behaviours, as well as the associations between these objects. We discover together that the object-oriented programming makes concrete an abstract activity.

General objective:

- This week, you will be able to identify the main objects of a software system.

Preambule

Learning objectives

Learning objectives

- **Identify** the attributes and behaviours of objects in a computer system.
- **Recognize** the main associations between the objects of a computer system.
- **Explain** in your own words the following concepts: instance variables and instance methods
- **Design** a simple Java program illustrating the basic concepts of object-oriented programming.

Lectures:

- Pages 573-579 from E. Koffman and P. Wolfgang.

Lectures (continued):

Basics

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

Detailed

- 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

Preambule

Plan

Plan

Théorie

Discussion

- Java is an **object-oriented** programming language
  - **What is** object-oriented programming?
  - **Why** object-oriented programming?

Object-oriented programming

- Software design is an **abstract activity**, in the sense that one cannot touch or see one's various components.
- Object-oriented programming provides tools, classes, and objects, which make this process **concrete, visible, palpable**; we can draw diagrams illustrating the objects of a system as well as their interactions.

Abstractions

- **Abstractions** are allowing us to **ignore the details** of a concept and to focus on some aspects deemed important.
- We often compare an abstraction to a "**black box**" It can be used without worrying about its content.

Abstractions in computer science

- **Variable** : associates a **name** to **a memory location**
- **Fonction*** : associates a name with a set of instructions, thus hiding the details, and making the instructions reusable

*Sub-routine, routine, procedure, method

Object-oriented programming

The **central** concept of object-oriented programming is the **object**!
An **object** has

- **properties** (attributes)
- **behaviours** (methods)

A software is seen as **a collection of objects** interact with one another to solve a problem common problem.

Defining data types

- Java is **object-oriented**.
- The programmer defines new data types using **classes and objects**.
- A **class** defines the characteristics (properties and behaviours) that are common to a set of objects.

Example: Integer

- Let's define a new **type**!
- To make things simple,
  - Each "element" of this type has a single value, of type **int**.
  - There are two **operations**: **getValue** and **plus**.

Example: Integer

```
class Integer {
  int value;

  Integer(int v) {
    value = v;
  }

  int getValue() {
    return value;
  }

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

}
```

Example: Integer (continued)

```
Integer a, b, c;

a = new Integer(10);

b = new Integer(5);

c = a.plus(b);

System.out.println(c.getValue());
```

Object-oriented programming

Identify the objects:

- **E-commerce:** clients, items, inventory, transactions, . . . ;
- **Chesss game:** pieces, board, players, . . . ;
- **Manufacture:** assembly lines, robots, items, . . . ;

Object-oriented programming

- For some “classes” of objects, there is only one “instance”; this is the case of the board in the game of chess.
- While other classes describe features of a collection of objects.
- Each object is **unique** although 2 objects can have the same state (the content of the instance variables is the same.)

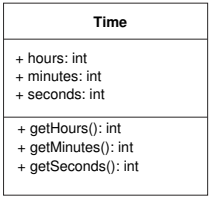
Object

An **object** has

- **properties** that define its **state**;
- **behaviours**: what the object can do in response to requests.

Unified Modelling Language (UML)

- **UML** is a graphical language to model sofware systems.
- A **class diagram** is a box with three sections: the name of the class, the attributes, and the methods.



We will introduce other elements as needed.

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

Exemple

Modelling a counter

- Imagine a device used to count **points** in sports.
- A window allows us to **read the current value**.
- A button allows us to **increment the current value by 1**
- Finally, another button allows us to **reset the value to zero**.

Wikimedia Commons/usager **Wesha**

Why not?

- A **single value** is necessary to model this counter.
- Furthermore, this value can easily be represented with a **primitive type** of Java, such as **int** or **long**.

Here is a counter:

```
int c1;
```

Counter: object-oriented

```
public class Counter {

    private int value = 0;

    public int getValue() {
      return value;
    }

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

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

Counter: object-oriented

```
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());
        }
    }
}
```

Counter: object-oriented

```
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

Definition: instance variable

An **instance variable** is a variable defined in the body of the class and such as **each instance** (object) has **its own copy**.

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

- **By default**, the variables defined in the body of the class are **instance variables**.

Definition: instance methods

An **instance method** is a method defined in the body of the class, that **we can only call in the context of an instance** (object), and that **has access to the instance variables of this object**.

```
class Point {
    int x, y;

    int getX() {
        return x;
    }

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

- **By default**, the methods defined in the body of a class are **instance methods**.

Definition: instance context

The **dot notation** is used in order to call a method of the object designated by the reference variable.

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

Point p2;
p2 = new Point();

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

- The reference variable provides **context of the object**: we call the method of the designated object.

Definition: constructor

A **constructor** is a very special instance method:

- The constructor has the **same name** as **the class**.
- It is only called in the context of creating an object, after the keyword **new**.
  - It cannot be used in another context.
- This **method does not return any value**.
  - It's logical! Do you see **why**?
- The constructor is used to **initialize instance variables**!

Definition: constructorr (continued)

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

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

}
```

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

The class Point

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

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

    int getX() {
        return x;
    }

    int getY() {
        return y;
    }

}
```

Acces methods

```
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;
    }
}
```

Passing the reference of an object as parameter

```
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;
    }
}
```

Passing the reference of an object as parameter

```
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());
```

What will be the result of **p1.equals(p2)**, **p1.equals(p3)**?

Fundamentals of object-oriented programming

- The concepts of **instance variables** and **instance methods** are fundamental to understand object oriented programming.

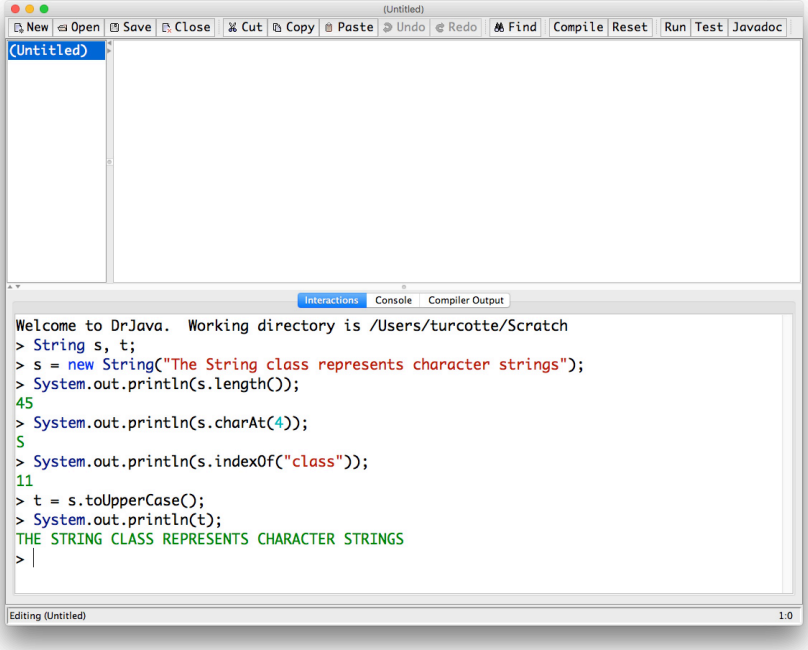
```
class Point {
    int x;
    int y;

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

Predefined classes of Java

- Familiarize yourself with the **Java documentation**
  - `http://docs.oracle.com/javase/8/docs/api/overview-summary.html`
- In particular, the classes of the package **java.lang**
  - `http://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html`
- Consult the documentation of the class **String**
  - `http://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

Predefined classes of 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

- High-level programming languages are **expressive**.
- Object-oriented programming makes programming **concrete**.
- An **object** has
  - a **state** (values of its instance variables), and
  - **behaviours** (its instance methods).

- **Object-oriented programming**: class variables, class methods, visibility modifiers, reference **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

School of Electrical Engineering and **Computer Science** (EECS)

University of Ottawa