Object-oriented programming with Java – Part 1

Samuel Toubon

Ensai

Acknowledgment

This course is strongly inspired by Olivier Levitt's one, available at **formations.levitt.fr**

ENSAI École nationale de la statistique et de l'analyse de l'information

Agenda

- 4 parts
  - Part 1 & 2 : OOP with Java
  - Part 3 : How to use Java ?
  - Part 4 : How to deal with a real project ?
- 4 lessons (1.5h), each with a practical session (3h)
- A final exam (multiple choice, alone, on paper)

Table of contents

Reminder about OOP

How would you model this situation ? How would you implement it ?

A car has four wheels, each characterized with a unique id. Each car has a unique registration number, which can change, and a brand which cannot. At every time, a wheel belongs to only one car, but you could change the wheel of a car. You could destroy the car and still get back the wheels.

What if you should store thousands of such cars in a database ?

Reminder about OOP

What is a class ? An instance ?

Reminder about OOP

What is an attribute ? A method ?

What can be found inside a class ?

Table of contents

What is Java ?

- A language
- A **programming** language
- An **object-oriented** programming language
- A **compiled** object-oriented programming language (kind of, more on that later)

Why so many languages ?

https ://www.college-de-france.fr/site/gerard-berry/course-2015-11-04-16h00.htm

▸ Link

18’58

Why Java ?

- popular
- portable (desktop, servers, smartphones, more on that later)
- robust and secure
- simple
- open source
- fast (kind of, more on that later)
- INSEE-friendly : more than 9 out of 10 home-made apps running Java there

ENSAI École nationale de la statistique et de l'analyse de l'information

Java popularity (2019)

- ieee.org : Python, Java, C, C++, R
- tiobe.com : Java, C, Python, C++, C#
- tiobe.com : Javascript, Java, Python, PHP, C++

Java versions (source : wikipedia)

Version	Release date	End of Free Public Updates[7][8]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2020 for Oracle (personal use) At least September 2023 for AdoptOpenJDK	March 2025
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	At least September 2022 for AdoptOpenJDK	September 2026
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	TBA	TBA

Legend: Old version | Older version, still supported | **Latest version** | Latest preview version | Future release

ENSAI École nationale de la statistique et de l'analyse de l'information

Table of contents

A simple class

```
public class Student {
    public String name = "Toubon";
    public String firstName = "Samuel";
}
```

A simple instance

```
Student alice = new Student();
alice.firstName = "Alice";
```

Naming conventions

- starts with a letter
- only includes letters, numbers, and underscores
- case sensitive !
- cannot be a language keyword (such as **while**)
- camelCase is used, i.e. variables start with a lowercase and words are separated with an uppercase
- there is a special rule for constants (more on that later)

A simple method

```
public class Student {
        public String name = "Toubon";
        public String firstName= "Samuel";

        public void sayHello() {
                System.out.println("Hello, my name is
                    "+firstName);
        }
}
```

the constructor, a special method

- it has the name of the class and no type of return
- it's used to initialize an instance
- Java provides by default an hidden void constructor to each class... which is disabled if you implement you own
- you can have several constructors for each class

```
public class Student {
    public String name = "Toubon";
    public String firstName = "Samuel";

    public Student(String name, String firstName) {
        this.name = name;
        this.firstName = firstName;
    }
}
```

```
Student s = new Student("Toto","titi");
```

main, another special method

```
public class Main {

        public static void main(String[] args) {
                Student alice = new Student();
                alice.firstName = "Alice";
                alice.sayHello();
        }

}
```

Notice the signature of the method, it has to be exactly this one !

Conditional blocks

```
if (booleanExpression1) {
    ...
} else if (booleanExpression2) {
    ...
} else {
    ...
}
```

```
switch (value) {
    case value1:
        ...
    break;
    case value2:
        ...
    break;
    default:
        ...
}
```

Loop blocks

```
while (booleanExpression1) {
    ...
}
```

Do... while exists, too.

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```

Operators

- Comparative operators : <, >, <=, >=, ==, !=
- Boolean operators : !, &&, ||

Table of contents

Primitive types

Primitive types are the most basic data types available within the Java language.

- Integer :

type	size (bytes)	minimum value	maximum value
byte	1	$-2^7 = -128$	$2^7 - 1 = 127$
short	2	$-2^{15} = -32\,768$	$2^{15} - 1 = 32\,767$
int	4	$-2^{31} = -2\,147\,483\,648$	$2^{31} - 1 = 2\,147\,483\,647$
long	8	$-2^{63} \approx -9 \cdot 10^{18}$	$2^{63} - 1 \approx 9 \cdot 10^{18}$

- Floating-point :

type	size (bytes)	amplitude	precision
int	4	limited	limited
long	8	less limited	less limited

- Boolean : **boolean** true or false
- Characters : **char** on 2 bytes, delimited with single quotes ' '.

ENSAI École nationale de la statistique et de l'analyse de l'information

String

- Not a primitive type but very mainstream.
- Delimited by double quotes : " ".

```
String hey = "Hello world :)";
```

- As a non-primitive type, the name **String** begins with a capital.
- All **String** variables are instances of the class **String** ! So we could write it this way :

```
String hey = new String("Hello world :)");
```

Types : example

```
int myInteger = 5;
float myFloat = 5.0f/8; //0.625 will be stored !
char myChar = 'a';
String a = 15; //will fail !
int b = 3.5; //will fail !
```

Strong typing everywhere

```
public class Student {
    public String name;

    public void changeName(String newName) {
        name = newName;
    }
}
```

Table of contents

Let's speak about attributes

- **static** means it's attached to the class, not the instance
- **final** means it cannot change over time, i.e. once it has a value it keeps it forever, i.e. it is a constant

Game : I want to write a FrenchCitizen class. Can you find one example attribute for each of these empty cells ? What would be their types ?

FrenchCitizen	final	not final
static		
not static		

- NB : **final** can also be used for a simple "variable" inside a method, it's not only for attributes !

Let's speak about attributes : syntax

```
public class Car {
    public final static int NUMBER_OF_WHEELS = 4;

    public String name = "Model S";
}
```

```
Car.NUMBER_OF_WHEELS;  //we do not use camelCase on
    constants !

Car myCar = new Car();
myCar.name;
```

ENSAI École nationale de la statistique et de l'analyse de l'information

What about methods ?

- **static** means it's attached to the class, not the instance (easy, right ?)
- **final** is trickier but not so useful, more on that later

```
public class Maths {

    public static int add(int a, int b) {
        return a + b;
    }
}
```

```
int total = Maths.add(2, 3);
```

Table of contents

this keyword

this refers to things related to the current instance, precisely :

- Used as a function, it refers to the constructor of the class of the instance.
- Used as a variable, it refers to the current instance.

We have already seen this example :

```
public class Student {
    public String name = "Toubon";
    public String firstName = "Samuel";

    public Student(String name, String firstName) {
        this.name = name;
        this.firstName = firstName;
    }
}
```

ENSAI École nationale de la statistique et de l'analyse de l'information

Useful this vs unuseful this

Idea : what if we change the names of the function parameters ?

```
public class Student {
    public String name = "Toubon";
    public String firstName = "Samuel";

    public Student(String lastName, String givenName) {
        this.name = lastName;
        this.firstName = givenName;
    }
}
```

this used as a function : example

```
class Counter {
    int position, step;

    Counter(int position; int step) {
        this.position = position;
        this.step = step;
    }

    Counter(int position) {
        this(position, 1);
    }
}
```