Java 101 - Magistère BFA

Lesson 2

Stéphane Airiau

Université Paris-Dauphine

Object Oriented Programming in Java

Objects and Classes

An **object** can be defined by its **states** and its **behaviours**

A car seen as an *object*

States	Behaviours
brand	accelerate
model	gear up
power	gear down
fuel level	turn wheel
oil level	opening door
tires pressure	closing door
rpm	break

A **class** can be seen as a *blue print* used for creating objects

- *states* are represented by variables
- *behaviours* are represented by *methods*.

An **object** is a class **instance**.

The state of an object can only be changed by the behaviour of the object ⇨ using the methods of the object.

Object

An object is an instance of a class.

The running example for the course will be a class of characters of comic books such as Astérix.

We will create a class `Character`. When we will create a particular character, say Astérix, we will instantiate the class `Character` to create the instance / the object Astérix.

By **convention**, the name of a class **always** start by an **upper case letter**. The instances/objects and everything else will start by a **lower case letter**.

We will write a class `MyClass` in a file `MyClass.java`. We will write the code of the class starting with the keyword **`class`**

```
class Character {
   ...
}
```

to be saved in a file `Character.java`

Java comes with *many* classes!

`Java` comes with a large class library. The library is organised in different `packages`.

`http://docs.oracle.com/javase/8/docs/api/overview-summary.html`

For instance, the `package java.lang` contains all the basic classes of `Java`. The class to manipulate strings of characters is located in that package and is called `String`.

Instance variables

- **instance variables** :
  these variables define the characteristics of the objects.
  - initialisation is optional.

  access : `<name object>.<instance variable name>`
- **class variables** : these variables are *common to all* the instances of the class,
  - declaration with the keyword `static`
  - initialisation is compulsory

  access : `<class name>.<class variable name>`

example : the class `Float` encapsulates a floating point number `float`.

- class variables : `MAX_VALUE`, `MAX_EXPONENT`, `NaN`, etc.

Class method and Instance methods

- **Instance methods** : these methods allows to *access* or *modify* the *state* of an instance/object
- **Class methods** : these methods do *not* modify the state of an object. Usually, there are utility methods to work with object of the class.

Example : `Float` class

- instance method `String toString()`
  - ⇨ returns a representation of the **current object** as a character string
- class method `static String toString(Float f)`
  - ⇨ returns a representation of an object passed in parameter

```
1 Float f;
2 ...
3 System.out.println(f.toString());
4 System.out.println(Float.toString(3.1419));
```

Encapsulation

The behaviour or the state of an object can be
known by every other object ⇨ `public`
any class can

- execute a public method
- access or modify a public variable

hidden to other classes ⇨ `private`
one can call a private method, access or modify a private variable only if it is inside the class

- ⇨ goal is to hide what is "under the hood"
  (one will be able to change code without affecting any other class).
- ⇨ protection

Creating an object : call a constructor

- A class is just a *blue print* to create instances.
- To create an object, we use a special method called a **constructor**.
- In the class, we need to implement a constructor
- *signature of the constructor*
  - the name of the method is the name of the class
  - there is not return type nor `void`

The **default** constructor is the constructor with **no** argument :

```
1  public class <class name> {
2     // declare variables
3     // (class or instance)
⋮
      ...
5     // default constructor
6     public <class name>(){
7       // body
8     }
9  }
```

Example

With overloading, we can then have several constructors

```
1  public class Character {
2     public String name;
3
4     // default constructor
5     public Character(){
6       nom = "unknown";
7     }
8
9     public Character(String name){
10      this.name = name;
11    }
12 }
```

In the example, we have two constructors.

Creating an object

- Declaration : exactly as primitive types :
  `<class name> <object name>;`
- Creation with the keyword **new** and we call the constructor :
  new `<class name>(<arguments list>);.`
- as for primitive types, we can declare and create the object in the same instruction

```
1 Character asterix = new Character("Astérix");
2 Character obelix = new Character("Obelix"),
3    idefix = new Character(”Idéfix”),
4    romain = new Character();
```

Equality between object

```
1  Character asterix = new Character("Astérix");
2  Character asterixBis = asterix;
3  Character asterixTer = new Character("Astérix");
4  if (asterix == asterixBis)
5    System.out.println("Red");
6  else
7    System.out.println("Green");
8  if (asterix == asterixTer)
9    System.out.println("Blue");
10 else
11   System.out.println("Yellow");
```

What is written in the output ?

Equality between object

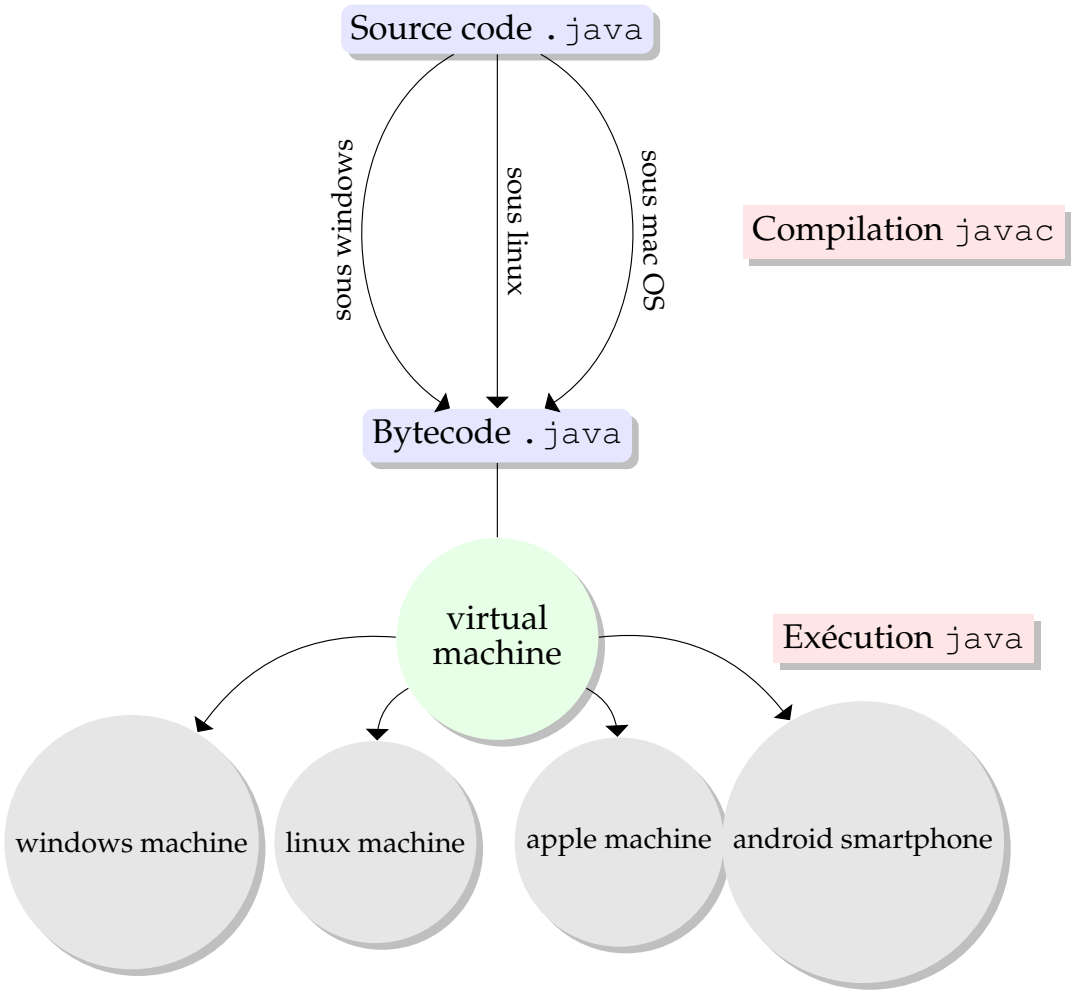
```
1  Character asterix = new Character("Astérix");
2  Character asterixBis = asterix;
3  Character asterixTer = new Character("Astérix");
4  if (asterix == asterixBis)
5    System.out.println("Red");
6  else
7    System.out.println("Green");
8  if (asterix == asterixTer)
9    System.out.println("Blue");
10 else
11   System.out.println("Yellow");
```

What is written in the output ?

- a variable is a *reference* to an object in memory
  and **not** the object !
- == is the equality between references :
  two references are equal if they refer to the same object in memory
- For testing the equality between **properties** of an object
  we use a special method `boolean equals(Object o)`.

Compilation, execution, virtual machine

`Java` is not only a langage and a library of classes
`Java` has tools for *generating* and *executing* code.



Compilation

A class `<MyClass>` is saved in a file `<MyClass>.java` : the name of the class matches the name of the file with the extension `.java`.

To *compile*, we use a program called `javac` that translate your code into machine readable code.

Le compiler *translates* your code into a langage that the virtual machine understands.

For `Java` it produces `bytecode`.

The result of the compilation is a file name `<MyClass>.class`

Compilation

Roughly, there are two stages in the compilation process :

- *syntaxic analysis* : we check the grammar of the code
- *semantic analysis* : translation of the code in bytecode
  and we check if everything is well known (other classes)

Execution

What is executed is a special method called **main**.
Each class can have one **main**.
If a method `main` is implemented in a class `MyClass`, we lauch the virtual `Java`machine, which runs the main :
`java MaClass`
(on linux or mac os, you can run this command)

The `main` method has a well specified signature

```
1 | public static void main(String[] args)
```

- `public` : it must be called from outside the class
- `static` : we have not yet been able to create an object !
- `void` : lthe method does not return anything (to whom should it return something ?)
- `String[] args` : when we start the execution, we can add some text, which will be accessible in this array of string. This is useful when we want to launch an application with some options.

Write your first class

Code a class that represent students. Each students has a name and 4 grades.

- `String toString()` that returns a representation of the student
- a method to add each grade
- a method that compute the average of the grades. If one note is missing, write a message. As you must return a value, choose an appropriate one.
- a method that tells whether the student passes.

Use a `main` method to test your code.

PS : to write a message on the console, use the following instruction : `System.out.println(<a string>)`

PPS : for Strings, the binary operator + appends the two strings