Lecture 1: Diving into java

Pablo Oliveira <pablo@sifllez.org>

ENST

Outline

1. Introduction
2. Primitive types
3. Operators
4. Basic OOP
5. Control Flow
Introduction : Execution model

- Java programs run inside a Virtual Machine (JVM), "a software implementation" of a computer (making Java programs portable among different architectures).
- For the sake of performance and space, the JVM reads bytecode (a low level language).
- It is easier for humans to write programs in a more expressive, high level language, like Java.
- The Java compiler is an automatic translator from Java to Bytecode.

Introduction : Concepts

- A computer program can be seen as a transformation from a state of a computer to a new one.
- So very informally, programming is a matter of transforming data:
  - Find a computer representation of data
  - Write methods to transform this data
- In this lecture we will learn about:
  - primitive types: Basic representations of data.
  - operators: Simple operations on data.
  - objects: Combine data representation and methods to transform data.
  - control flow: Control which operations are done and how many times.
Outline

1 Introduction

2 Primitive types

3 Operators

4 Basic OOP

5 Control Flow

Representing data

Data is encoded in memory as a binary string.

<table>
<thead>
<tr>
<th>bin</th>
<th>00000000 01001000 00000000 01101001</th>
</tr>
</thead>
<tbody>
<tr>
<td>dec:</td>
<td>0 72 0 105</td>
</tr>
</tbody>
</table>

What does this value represent?

Example

```
2d line : (0,72) to (0,105)
integer : 4718697
float   : 6.612303E-39 (IEEE 754)
string  : 'H' 'i'
...
```
Variables and Types

- To lift this ambiguity we introduce types, which specify how a value should be used.
- We also give each value a name (should start with {letter, $, _} and contain only {alphanumeric, $, _}).
- We call the association (name, type, value) a variable.

**Example**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td><code>is_it_raining = true;</code></td>
</tr>
<tr>
<td>byte</td>
<td><code>life_expectancy = 70;</code></td>
</tr>
<tr>
<td>short</td>
<td><code>year;</code></td>
</tr>
<tr>
<td>char</td>
<td><code>unicode_character = 'A';</code></td>
</tr>
<tr>
<td>int</td>
<td><code>city_population = 2167994;</code></td>
</tr>
<tr>
<td>long</td>
<td><code>molecules;</code></td>
</tr>
<tr>
<td>float</td>
<td><code>mean_grade = 13.54;</code></td>
</tr>
<tr>
<td>double</td>
<td><code>angular_speed;</code></td>
</tr>
</tbody>
</table>

- 1 bit logic value
- 8 bit range $[-128..127]$  
- 16 bit defined on $[-2^{15}..2^{15} − 1]$  
- UTF-16 caracter
- 32 bit range $[-2^{31}..2^{31} − 1]$  
- 64 bit range $[-2^{63}..2^{63} − 1]$  
- single precision(ex. 8 mt. 23)  
- double precision(ex. 11 mt. 52)

Mutable variables, assignment, final variables

The value of a variable can be modified during the execution of a program, except for final variables (which can only be assigned once).

```
int a;  // variable declaration
float b = 3;  /* variable declaration with initial assignment */
final float pi;
pi = 3.14159;

a = 8;  // assignment
b = 1;  // assignment
pi = 0;  // illegal (won’t compile)
```
Arrays

Arrays model a contiguous, random access, fixed-length, collection of values. The values of an array are of the same type.

```java
// The type of the array price is float []
flo at [] price;

// reserve memory for 3 elements
price = new float [3];

// initialize the values of the elements
price[0] = 1.00;
price[1] = 5.99;
price[2] = 3.25;

// Syntax sugar for this is:
flo at [] price = {1.00, 5.99, 3.25};
```

Strings

Strings represent a collection of characters.

```java
String message = "Hello\World";
```

Caution

- Strings and arrays are a mix between objects and primitive types and thus some care must be taken when manipulating them.
- In the next lecture we will explain why when we'll talk about references and immutability.
Operators are special symbols which perform an operation on some operands.

The semantic of the operator depends on the type of the operands.

**Example**

```plaintext
a = 5; b[4] = 30  
5 + 4 → 9  
5 − 4 → 1  
"hello " + "world" → "hello world"  
4 * 4 → 16  
15 / 2 → 7, 15.0 / 2 → 7.5  
15 % 2 → 1  
15 > 12 → true  
< > = <=  
```

- assignment
- sum
- substraction
- concatenation
- multiplication
- integer or float division
- modulo
- bigger than
- other comparison operators
Increment operators

- Increment (++), decrement (--) operators:

```java
int a, b, c;

a = 42;
b = a++;\n\here b evaluates to 42 and a to 43

a = 42;
b = ++a;\n\here b evaluates to 43 and a to 43
```

- With ++a the value of a is incremented, then the right side of the assignment is evaluated.
- With a++ the right side of the assignment is evaluated, then a is incremented.

More Operators

- Operation + assignment (+= -= *= /= etc ...)

```java
a += b; // equivalent to:
a = a + b;
```

- Equality == and Inequality !=

```java
int a, b;
a = 5;
c = 2;
b = c + 3;
a == b; // true
a != c; // true
```

- Referential equality when used with objects, strings and arrays (we'll come back to this later)
Boolean operators

- **Boolean Operators**

  ```java
  boolean a = false;
  boolean b = true;

  /* not */
  !a // --> true
  
  /* and */
  a && b // --> false
  
  /* or */
  a || b // --> true
  
  /* Careful: Lazy */
  int a = 0;
  (false) && (a++ > 0) // a evaluates to 0
  (true) || (a++ > 0) // a evaluates to 0
  ```

Other operators

- **Ternary Operator** (condition?true_stm:false_stm) ex: (a>b) ? a : b
- **Bitwise Operators** (>> << >>> & |)
- **instanceof**
Outline

1. Introduction
2. Primitive types
3. Operators
4. Basic OOP
5. Control Flow

Objects

Definition

An object is the association of:
- a State (i.e. data)
- some Methods (i.e. operations that change/read state)

Example

A turtle has a state composed of its color, its position and its orientation, and some methods: turn, advance and readPosition.

<table>
<thead>
<tr>
<th>Turtle</th>
</tr>
</thead>
<tbody>
<tr>
<td>color: green</td>
</tr>
<tr>
<td>position: (2, 3)</td>
</tr>
<tr>
<td>rotation: 30.0°</td>
</tr>
<tr>
<td>turn(double angle)</td>
</tr>
<tr>
<td>advance()</td>
</tr>
<tr>
<td>readPosition()</td>
</tr>
</tbody>
</table>
**Classes**

**Definition**

A class is a blueprint for making objects. A class defines the common attributes of a family of objects:

- the methods they share
- the types of variables they have

**Example**

<table>
<thead>
<tr>
<th>Turtle 1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>color: red</td>
<td></td>
</tr>
<tr>
<td>position: (0, 0)</td>
<td></td>
</tr>
<tr>
<td>rotation: 90.0°</td>
<td></td>
</tr>
<tr>
<td>turn(double angle)</td>
<td>advance()</td>
</tr>
<tr>
<td></td>
<td>readPosition()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Turtle 2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>color: blue</td>
<td></td>
</tr>
<tr>
<td>position: (10, -5)</td>
<td></td>
</tr>
<tr>
<td>rotation: 35.4°</td>
<td></td>
</tr>
<tr>
<td>turn(double angle)</td>
<td>advance()</td>
</tr>
<tr>
<td></td>
<td>readPosition()</td>
</tr>
</tbody>
</table>
### Classes

#### Example

**Turtle 1**
- **red**
- **(0, 0)**
- **90.0°**

**Turtle 2**
- **blue**
- **'10, -5)**
- **35.,4°**

#### Class Turtle
- **color**: a color
- **position**: coordinates
- **rotation**: an angle
- **turn(double angle)**
- **advance()**
- **readPosition()**

### Anatomy of a method

```java
static float norm (float x, float y) {
    return x*x + y*y ;
}
```
In Java

```java
class Turtle {
    Color color;
    Position position;
    double rotation;

    void turn(double angle) { rotation += angle; }
    void advance() {
        int step_size = 5;
        position.x += step_size * cos(rotation*Math.PI/180);
        position.y += step_size * sin(rotation*Math.PI/180);
    }

    Position readPosition() {
        return position;
    }
}

Turtle turtle1 = new Turtle();
Turtle turtle2 = new Turtle();

Position pos = turtle1.readPosition();
turtle2.turn(20); turtle2.advance();
```

**Instance Methods/Variables**

- Objects `turtle1` and `turtle2` are called *instances* of class `Turtle`,
- `turtle2.color` represents the color for the specific instance `turtle2`:
  → color is an instance variable
- `turtle1.readPosition()` returns the position of the specific instance `turtle1`:
  → `readPosition()` is an instance method

**Definition**

An instance method or instance variable is only relevant in the context of a particular object.
Static Methods/Variables

Q: how to share a common variable between all the objects in a class?

```java
class Turtle {
    static int step_size = 5;
}
```

Q: how to write a method independent from a particular instance?

```java
class Turtle {
    static double degreesToRadians(double deg) {
        return deg * Math.PI / 180;
    }
}
Turtle.degreesToRadians(30); ➔ 1.0471975512
```

Definition

A static variable is shared among all the objects of a class. A static method is called in the context of a class and not for a specific object. Static variables are also called class variables.

Local Variables / Scope

Definition

Variables defined inside a method are called local variables. The scope of a variable is the portion of code where it is visible (i.e., where you can read it, or modify it).

- Local variables are only visible inside their method.
- Instance and Static variables are visible in all the methods from their class.

In the next lecture we will talk about the visibility of methods, and about access modifiers that regulate the visibility of methods/variables outside their class.
Main: the program entry point

Q: (At last ... ) How do we start things up?

```java
class MyFirstJavaProgram {
    public static void main (String args[]) {
        System.out.println("Hello world!");
    }
}
```

(Note: public means this method can be called from anywhere, this is a prerequisite for the main method)

Outline

1. Introduction
2. Primitive types
3. Operators
4. Basic OOP
5. Control Flow
If - else

Depending on a predicate choose which branch to execute

```java
if (predicate) block1 else block2
```

```java
static int collatz (int n) {
    if (n % 2 == 0) {
        return n/2;
    } else {
        return 3*n + 1;
    }
}
```

code between `{}` is called a block, variables defined inside a block are not visible outside

while

Execute a block repeatedly while a predicate is true

```java
while(predicate) {
    block;
}
```

```java
static int gcd(int a, int b) {
    while (b != 0) {
        int t = b;
        b = a % b;
        a = t;
    }
    return a;
}
```

Above if the predicate is false the block is never executed. When appropriate, you can use instead `do { ... } while (predicate);` which always executes the block at least once.
More sophisticated loop

```java
for (statement; predicate; statement)
    block

for (int i = 0; i < 100; i++) {
    System.out.println(i);
}
```

Syntax sugar for :

```java
int i = 0;
while (i < 100) {
    System.out.println(i);
    i++;
}
```

break and continue

Escaping from loops

- break will exit the loop from which it is called
- continue will jump to the next iteration of the loop from which it is called

```java
int whereIs(int element, int[] set) {
    int i;
    for (i = 0; i < set.length; i++) {
        if (element == set[i]) {
            break;
        }
    }
    return (i < set.length) ? i : -1;
}
```
You can also escape using return, which here avoids a test and is more elegant.

```java
int whereIs(int element, int[] set) {
    for (int i = 0; i < set.length; i++) {
        if (element == set[i]) {
            return i;
        }
    }
    return -1;
}
```

We won’t discuss switch/case control structure. If interested you may look it up!