Java 101 - Magistère BFA

Lesson 1

Stéphane Airiau

Université Paris-Dauphine

Today

⇨ The basics

All about writing basic code without using the concept of object.

- Variables, operators, types
- Tests, loops
- Arrays
- methods

Evaluation

- small project mainly to make you implement a small application

Slides (in English) will be posted at this webpage.

There are also lecture notes (but in French).

`http://www.lamsade.dauphine.fr/~airiau/Teaching/BFA-Java/`

Instructions and comments

```
// This is a comment
```

```
/*this is a
 comment
 using different lines */
```

An instruction must satisfy the grammar of the langage `Java`.

Most of the instructions finish with a `;`

Elementary Types

Elementary Types	number of bits	value interval
`boolean`	1	`true` and `false`
`byte`	8	an integer between -128 and 127
`short`	16	an integer between -2^{15}=-32768 et $2^{15}-1$=32767
`int`	32	an integer between -$2^{31} \approx -2.1 \cdot 10^9$ and $2^{31}-1 \approx 2.1 \cdot 10^9$
`long`	64	an integer between -$2^{63} \approx -9.2 \cdot 10^{18}$ and $2^{63}-1 \approx 9.2 \cdot 10^{18}$
`char`	16	unicode characters, there are 65536 codes
`float`	32	a floating point number following theIEEE norm
`double`	64	a floating point number following the IEEE norm

Variables : declaration and initialisation

- *simple declaration* :
  <type> <nom>;
- *declaration with affectation* :
  <type> <name> **=** <value in the type> | <variable> | <expression>;
- *multiple declarations* :
  <type> <name$_1$>, <name$_2$>, ..., <name$_k$>;
- *multiple declaration with partial affectation* :
  <type> <name$_1$>, <name$_2$>= <value in the type>, ..., <name$_k$>;

Examples

```java
short population;
population = 30000;
```

Examples

```java
short population;
population = 30000;
```

```java
short population = 1.000.000;
```

Examples

```
short population;
population = 30000;
```

```
short population = 1.000.000;
```

```
long nbParticules = 10.000.000.000;
```

Examples

```java
short population;
population = 30000;
```

```java
short population = 1.000.000;
```

```java
long nbParticules = 10.000.000.000;
```

```java
long nbParticules = 10.000.000.000L;
```

Examples

```
short population;
population = 30000;
```

```
short population = 1.000.000;
```

```
long nbParticules = 10.000.000.000;
```

```
long nbParticules = 10.000.000.000L;
```

```
char letter = 'c';
```

Examples

```java
short population;
population = 30000;
```

```java
short population = 1.000.000;
```

```java
long nbParticules = 10.000.000.000;
```

```java
long nbParticules = 10.000.000.000L;
```

```java
char letter = 'c';
```

```java
boolean test = true;
test = false;
```

Cast

Here is the situation :

```
<type1> <nom1> = <valeur1>;
<type2> <nom2> = <nom1>;
<type2> <nom2> = <valeur1>;
```

- Implicit cast : when type_2 is « stronger » than type_1

  ```java
  int i = 10;
  double x = 10;
  double y = i;
  ```

  an `int` « fits » inside a `double`.

- Explicit cast when <type$_1$> is « strictly stronger » than <type$_2$> : we need to tell the compiler to do something

  ```java
  double x= 3.1416;
  int i = (int)x;
  ```

  We need to tell `Java`to « cut » the `double` to make it fit inside and `int`.

Unary operator

Operator	degree of priority	action	examples
+	1	positive sign	+a; +7
-	1	negative sign	-a; -(a-b); -7
!	1	logical negation	!(a<b);
++		affectation and increment by one	n++;++n;
--		increment by one then affectation	n++; --i;

Binary operator

<table>
<tr><th>Operator</th><th>degree of priority</th><th>action</th><th>examples</th></tr>
<tr><td>*</td><td>2</td><td>multiplication</td><td>a * i</td></tr>
<tr><td>/</td><td>2</td><td>division</td><td>n/10</td></tr>
<tr><td>%</td><td>3</td><td>remainder of integer division</td><td>k%n</td></tr>
<tr><td>+</td><td>3</td><td>addition</td><td>1+2</td></tr>
<tr><td>-</td><td>3</td><td>substraction</td><td>x-5</td></tr>
<tr><td><</td><td>5</td><td>strictly smaller than</td><td>i<n</td></tr>
<tr><td><=</td><td>5</td><td>smaller or equal to</td><td>i <= n</td></tr>
<tr><td>></td><td>5</td><td>strictly greater</td><td>i < n</td></tr>
<tr><td>>=</td><td>5</td><td>greater or equal</td><td>i >= n</td></tr>
<tr><td>==</td><td>6</td><td>equality</td><td>i==j</td></tr>
<tr><td>!=</td><td>6</td><td>different</td><td>i!=j</td></tr>
<tr><td>&</td><td>7</td><td>conjunction (logical and)</td><td>(i<j) & (i<n)</td></tr>
<tr><td>|</td><td>9</td><td>disjunction (logical or)</td><td>(i<j) | (i<n)</td></tr>
<tr><td>&&</td><td>10</td><td>optimised conjunction</td><td>(i<j) && (i<n)</td></tr>
<tr><td>||</td><td>11</td><td>optimised disjunction</td><td>(i<j) || (i<n)</td></tr>
<tr><td>=</td><td></td><td>affectation</td><td>x = 10; x=n;</td></tr>
<tr><td>+=, -=</td><td></td><td>affectation and increment</td><td>i += 2; j-=4</td></tr>
</table>

Example

```java
int i=2, j = i++;
i=2;
j= ++i;
```

Warning : do **not** use = for equality test!

One ternary operator!

```
result = test ? value1 : value2;
```

If if a test (a boolean expression) is satisfied
then the variable `result` is assigned to `value1`,
otherwise it is assigned to `value2`.

```
double x,y, r=1.0;
  ...
boolean inside = x*x + y*y < r ? true : false
```

Expression type

Is the following code correct ?

```
int i = 5,j;
double x = 5.0;
j=i/2;
j=x/2;
```

Expression type

Is the following code correct ?

```
int i = 5,j;
double x = 5.0;
j=i/2;
j=x/2;
```

```
double x=2.75;
int y = (int) x * 2;
int z = (int) (x *2);
```

What are the values of y and z ?

Arrays

How to declare an array :

```
<type> [] line;
<type> [][] rectangle;
<type> [][][] cube;
```

How to create an array : you **must** tell the array **size** !

```
<type> [] line = new <type>[<taille1>];
<type> [][] rectangle = new <type>[<taille2>][<taille3>];
```

How to get the size of the array : `cube.length`

Warning : in computer Science,

- the first entry of an array is indexed by `0`.
- ⇨ the last entry of an array is then `length-1`.

How to use the array : use brackets `[]` :

```
rectangle[3][4] + cube[1][2][5];
```

Examples

It is possible to initialise an array using a « list » notation :

primes :

2	3	5	7	11	13	17	19

triangle :

1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

```java
int[] premiers = {2, 3, 5, 7, 11, 13, 17, 19 };
int[][] triangle = {{1,1,1,1}, { 0,1,1,1},
      { 0, 0, 1, 1}, {0, 0, 0, 1} };
```

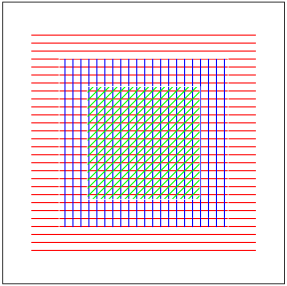
A 2-dimensional array is a 1-dimensional array of 1-dimensional array... so

```java
int[][][] cube = new int[3][4][5];
int[][] rectangle = cube[2];
int n1 = cube.length;
int n2 = cube[0].length;
int n3 = cube[0][0].length;
```

Instruction Blocks

A block gathers together instructions.
The variables that are declared **inside** a block are **only known** inside a block.
i.e. outside the block, the variable does not exist !

```
int a,b=10;
 {
   int d=2*b;
   {
     int e=b+d;
     a=e*d;
     {
       int g= b+ d*e;
     }
   }
 }
```



a and b are known everywhere.

d only exists in the red part.

e only exists in the blue part.

g only exists in the green part

Conditionnals : `if` … `then` … `else`

```
if ( <boolean expression> )
   <block to execute
     when the condition is satisfied>
else
   <block to execute
     if the condition is not satisfied>
```

The **`else`** block is **optional**.

```
int gains,payment,withdraw,invest;
// some code that modify the gains
   ...
if (gains<0)
  payment = gains;
else if (gains > 10) {
  withdraw = 10;
  invest = gains-10;
}
else
  withdraw = gains;
```

Multiple choices

```
int choice;
...
// something is done with choice
...
switch(choice) {
  case 1:
    //instruction block for case 1
    ...
    break;
  case 2:
    //instruction block for case 2
    ...
    break;
  default
    // default instruction block
    ...
}
```

Inside a **switch** we can use a variable with types **int**, **char**, and **String**

Loop : for loop

```
for (<initialisation>
    <stopping condition> ;
    <update>)
  <instruction block>
```

What happens in that case ? Is this valid ?

```
for ( ; ; ) {
    // instructions
}
```

a classical example :

```
int n=10;
for (int i=0; i< n; i++ ) {
    // instructions
}
```

Another example

```java
int n=10;
for (int i=0, j=n; j< i; i++; j-- ){
    // instructions
}
```

Loop : `while` loop

```
while(<condition>)
    <instruction block>
```

The instruction block is execute as long as the condition is met.

Example for checking convergence of a series $u : n \to r^n$:

```
double epsilon = 0.0000001;
double r = 0.75, u=1;
while( u-u*r <= -epsilon && u - u* r >= epsilon)
    u = u * r;
```

Loop do ... while

```
do
    <Instruction block>
while(<condition>);
```

Warning ! Do **not** forget the semi column ; right after the condition !

```
double epsilon = 0.0000001;
double r = 0.75, u=1;
do
    u = u * r;
while ( -epsilon >= u-u*r || u - u* r >= epsilon);
```

choosing a while loop or a do while loop is a matter of elegance, one usually comes easier than the other.

Choosing between a while loop or a for loop

- if we know exactly how many times we iterate : use a `for` loop
- otherwise, use a while loop.
- what is more expected ?

ex :

- search an element in an array ?
- search for the largest element in an array ?

Methods

It is the term used for *function* in a Object Oriented Programming Language.

Goal : Factorise/gather together a sequence of instruction that could be used multiple times.
The code becomes

- more readable (if a pertinent name is used !)
- shorter
- **important** If one wants/needs to modify the code, one just need to update the code in one location !

Method

```
public static <type of what is returned> <name>
      ( <parameter list>) {
   body of the method
}
```

`public` and `static` will become clear in the coming lectures

- Choose an illustrative name !
- the **order** of the parameters is significant :
  `Java`does not match the parameters using names, it uses the order !
- If the method does not return something (it is a procedure), we use the return type **`void`**.
- when the method returns something, the instruction **`return`** `<result>` is used to terminate the method and to output `result`.

Example

```java
public static int max( int[] tab) {
  int m= tab[0];
  for (int i=1;i<tab.length; i++){
    if (tab[i] > m)
      m = tab[i];
  }
  return m;
}
```

Appel de la méthode

```java
int tab = {7, 12, 15, 9, 11, 17, 13};
int m = max(tab);
```

Overloading

We call **signature** the name of the method with its list of argument.

The signature of a method must be unique to avoid ambiguities.

➯We can have two methods with the same name but different lists of parameters !

```
public static double max( double[] tab) {
  double m= tab[0];
  for (int i=1;i<tab.length; i++){
    if (tab[i] > m)
      m = tab[i];
  }
  return m;
}
```

Passing arguments of primitive types

```
public int f(int n){
   n = 3 * n * n -2 *n + 1
   if (n > 0)
     return n;
   else
     return 0;
}
```

```
int i=13;
   int j= f(i);
```

What is the value of `i` ?

We pass the argument of primitive type **by value**.