Language basics

Videogames Technology
Asignatura transversal

Departamento de Automática

Universidad de Alcalá
ISG

Objective

Develop simple algorithms in Java

Bibliography

1. The Java™ Tutorials. Oracle. (Link)

Table of Contents

Introduction

Java is based on the C syntax

- i.e., Java is similar to C

... however, Java is not C (neither C++)

- Java is Object-Oriented (OO) (classes, interfaces, inheritance, etc)
- OOP: Object-Oriented Programming

Java was designed to be secure and portable

- Intermediate code (bytecode)
- Safe execution environment (Java Virtual Machine)
- Automatic memory management (no more mallocs)
- No pointers in Java (oh yeah!) → Garbage collector

If you know C, then you almost know Java

- Variables, operators, expressions and control flow

Variables

Primitive types (I)

Warning!: Several names in OOP meaning the same

- Attribute, fields, member, ...

Types of variables

- Instance variable (non-static variables): Belongs to the object
- Class variable (static variables): Belongs to the class
- Local variable: Like in C
- Parameters: Do you remember `public static void main(String[] args)`? That's it

Keywords

public, protected, private, static, final, new

Variables

Primitive types (II)

Variable size is architecture independent

Definition	Type	Default	Size
`byte`	Integer	0	8-bit
`short`	Integer	0	16-bit
`int`	Integer	0	32-bit
`long`	Integer	0L	64-bit
`float`	Float	0.0f	32-bit
`double`	Float	0.0d	64-bit
`char`	Character	'\u0000'	16-bit (Unicode)
`string`	String	null	X
`boolean`	Logic	false	

Same initialization than C: `int i = 0;`

Variables

Primitive types (III)

TypesExample.java

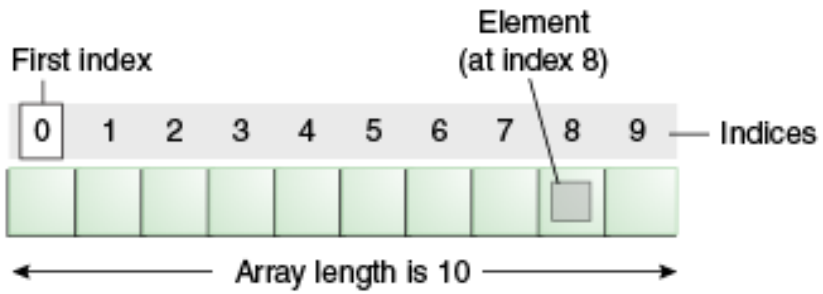
```
public class TypesExample {
    public static void main(String[] args) {
      int a = 0;
      double b = 1.3;
      String text = "The sum is: ";
      boolean condition = true;

      if (condition) {
        System.out.println(text + (a + b));
        System.out.println(text + a + b);
      }
    }
}
```

Variables

Arrays (I)

An array is a sequence of variables of the same type

- The variable is identified by an index
- The array length remains constant



(Source)

Example

```
int[] array1 = new int[10];
array1[3] = 5;
float[] array2 = new float[10];
int size=5;
boolean array3 = new boolean[size];
```

Variables

Arrays (II)

Declaration, creation and initialization are different concepts

```
public class ArrayDemo {
  public static void main(String[] args) {
    int[] anArray; //Declaration

    anArray = new int[3]; // Creation

    anArray[0] = 100; // Initialization
    anArray[1] = 200;
    anArray[2] = 300;

    System.out.println("Element 1: " + anArray[0]);
    System.out.println("Element 2: " + anArray[1]);
    System.out.println("Element 3: " + anArray[2]);
  }
}
```

Variables

Arrays (III)

The `main()` function provides the parameters as an array

ArrayDemo2.java

```
class ArrayDemo2 {
    public static void main(String[] args) {
      System.out.println("Argumentos: " + args.length);
      for (int i=0; i<args.length; i++) {
        System.out.println(args[i]);
      }
  }
}
```

Variables

Casting

We can assign a value to a variable of different type

- The compiler might guess some converssions
- We can force a conversion: Casting

CastDemo.java

```
public class CastDemo {
  public static void main(String[] args) {
    float a = 1.5; // Error
    float b = (float) 1.5; // Good

    int res;
    res = java.lang.Math.pow(2,2); // Error
    res = (int) java.lang.Math.pow(2,2); // Good
  }
}
```

Operators

Assignment and arithmetic operators

=	Assignment
+	Add
-	Substration
*	Multiplication
/	Division
%	Modulus
+=	Assign +
-=	Assign -
*=	Assign *
/=	Assign /

ArithmeticDemo.java

```
class ArithmeticDemo {
  public static void main(String[] args){
    int number = 1 + 2;
    number *= 5;
    number = number + 3;
    System.out.println("Number = " + number);
  }
}
```

Operators

Unary operators

var++	Increment
var−−	Decrement
++var	Increment
−−var	Decrement

UnaryDemo.java

```
public class UnaryDemo {
  public static void main(String[] args){
    int a=1, b=2;
    int res;
    res = a+b;
    System.out.println("Res=" + res);
    res = a*b;
    res *= 2;
    System.out.println("Res=" + res);
    a++;
    System.out.println("a=" + a);
    System.out.println("a=" + a++);
    System.out.println("a=" + ++a);
  }
}
```

Operators

Relational and conditional operators (I)

Relational operators

==	Equal to
!=	Not equal to
>	Greater than
>=	Great. or eq. to
<	Less than
<=	Less than or eq. to

Conditional operators

&&	AND
||	OR
!	Negation

- Result: `true` or `false`
  - Java supports native boolean variables
  - The result is a boolean
- Widely used in loops and conditions
- Truth tables represent the conditional operators

Truth tables

A	TTFF
B	TFTF
A&&B	TFFF

A	TTFF
B	TFTF
A||B	TTTF

Operators

Relational and conditional operators (II)

ComparisonDemo.java

```
public class ComparisonDemo {
  public static void main(String[] args){
    int value1 = 1;
    int value2 = 2;

    if(value1 == value2)
      System.out.println("value1==value2");
    if(value1 != value2)
      System.out.println("value1!=value2");
    if(value1 > value2) System.out.println("value1>value2");
    if(value1 < value2) System.out.println("value1<value2");
    if(value1 <= value2)
      System.out.println("value1<=value2");
  }
}
```

Operators

Bitwise and shift operators (I)

Bitwise operators

&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	Bit inversion

Examples

AND

A	1100
B	1010
A&B	1000

SHIFT

A	1100
A»1	0110

OR

A	1100
B	1010
A|B	1110

XOR

A	1100
B	1010
A^B	0110

Bit-level operators

- Conditional operators are variable-level

Classic logic operations

- AND, OR, XOR, shift and inversion
- AND is used to put bits to 0
- OR is used to put bits to 1

Operators

Bitwise and shift operators (II)

BitDemo.java

```
public class BitDemo {
  public static void main(String[] args) {
    int bitmask = 0x000F;
    int val = 0x2222;
    // prints "2"
    System.out.println(val & bitmask);
  }
}
```

Operations over certain bits: Bitmasks

- Example: Get the value of bit 5

```
xxxixxxx    &
oooioooo    =
--------
oooioooo
```

```
xxxoxxxx    &
oooioooo    =
--------
oooooooo
```

Control flow

Control flow introduction

Control flow breaks up the flow of execution

- Conditions (if-then, switch)
- Loops (for, while, do-while)
- Branching (break, continue, return)

Usually used with code blocks

Control flow

Blocks

A block is a group of statements between braces (“{” y “}”)

- A block is a unity of code
- Braces never end with semicolon (“;”)
- Used with loops, methods, conditions, etc

```
if (a != b)
  a = c;
  c = c+2;
```

≠

```
if (a != b) {
  a = c;
  c = c+2;
}
```

A variable can be defined at block-level

- The variable only exists within the block
- Good practice: Limiting the scope of the variables

```
if (a != b) {
  int c;
  c = a;
  a = b;
  b = c;
}
```

Control flow

Conditional statements: if-else statement (I)

Conditional statements implement decision making

- They are based on a conditional statement
- The result is a boolean
- Remember: Java supports a booleans!

```
int cond = true;
if (cond)
  // Some code
else
  // More code
```

≈

```
int a = 0;
if (a == 0)
  // Some code
else
  // More code
```

Good practice: The usage of `else` is optional, try to avoid it!

Control flow

Conditional statements: if-else statement (II)

- Many times decisions are not binary (true/false)
- Grouped conditions
  - Conditions are evaluated until first true
  - If all conditions are false, then it executes `else`
  - `else` is optional (try not to use it!)

```
if (condicion1)
  orden1;
else if (condicion2)
  orden2;
else if (condicion3)
  orden3;
else
  orden4;
```

Control flow

Conditional statements: if-else statement (III)

Example

```
int mes=2

if (mes == 1)
  System.out.println("Enero");
else if (mes == 2)
  System.out.println("Febrero");
else if (mes == 3)
  System.out.println("Marzo");

......

else if (mes == 12)
  System.out.println("Diciembre");
else {
  System.out.println("Error, mes no reconocido");
}
```

Control flow

Conditional statements: Ternary operator (IV)

The statement `if ... else` is pretty common

- Sometimes to simply assign a variable value
- It makes code hard to read

Ternary operator: `condition? if-true : if-false;`

```
if (a>b) {
   greater = true;
} else {
   greater = false;
}
```

=

```
greater = (a>b)? true :
     false;
```

Control flow

Conditional statements: switch (I)

- Closely related to `if else`
  - Sometimes `switch` is more convenient
- It evaluates an expression and, depending on its result, takes a branch
- There is a `default` action
- The keyword `break` exits the statement

```
switch (expression) {
  case constant1:
    statement1;
    break;
  case constant2:
    statement2;
    break;
  case constant3:
    statement3;
    break;
  default:
    statement4;
}
```

Control flow

Conditional statements: switch (II)

Example

```
switch (ke.getKeyCode()) {
  case KeyEvent.VK_UP:
    location = 0;
    break;
  case KeyEvent.VK_DOWN:
    location = 1;
    break;
  case KeyEvent.VK_LEFT:
    location = 3;
    break;
  case KeyEvent.VK_RIGHT:
    location = 4;
    break;
}
```

Control flow

Loop statements: for and while (I)

`For` and `while` are the two most common loop statements

for syntax

```
for (expr1; expr2; expr3) {
  // Some code here
}
```

=

while syntax

```
expr1;
while (expr2) {
  // Some code
  expr3;
}
```

- The statements `for` and `while` are equivalents
- The `for` statement can be read as
  - For `expr1`, while `expr2`, do `expr3`

Control flow

Loops: iterating collections

Old syntax

```
public static void main(String[] args) {
  int[] vector = {1,2,3,4,5};

  for (int i=0; i<vector.length; i++)
    System.out.println(vector[i]);
}
```

Modern syntax (> Java 5)

```
public static void main(String[] args) {
  int[] vector = {1,2,3,4,5};

  for (int number: vector)
    System.out.println(number);
}
```

Control flow

Loops: do-while (I)

Statements `for` y `while` assess first

- Execution is not assured

Statement do-while first executes, then assesses

- The body is executed at least one time
- Some times, using a do-while saves code

It can be read as “do ... while ...”

do-while syntax

```
do
  // Some code
while (expression);
```

Control flow

Loops: do-while (II)

Example

```
class DoWhileDemo {
    public static void main(String[] args) {
      int count = 1;
      do {
        System.out.println("Count is: " + count);
        count++;
      } while (count < 11);
    }
}
```

Control flow

Branching statements: Break and continue (I)

- `break`: Exit the loop
- `continue`: Jump to next iteration
- `break` and `continue` are valids in loops
  - `break` can be used, in addition, in a `switch` statement

Example: break

```
for (i=0; i<MAX, i++) {
  // Some code
  if (a == b) break;
  // More code
}
```

Example: Continue

```
for (i=0; i<MAX, i++) {
  // Some code
  if (a == b) continue;
  // More code
}
```

Control flow

Branching statements: Break and continue (II)

Break example

```
int[] array = {32, 87, 3, 589, 12, 1076, 2000, 8};
int searchfor = 12;
int i;
boolean foundIt = false;

for (i = 0; i < array.length; i++) {
  if (array[i] == searchfor) {
    foundIt = true;
    break;
  }
}
if (foundIt)
  System.out.println("Found " + searchfor + " at " + i);
else
  System.out.println(searchfor + " not in the array");
```

Exercise: Use loops modern syntax

Control flow

Branching statements: Break and continue (III)

Continue example

```
String searchMe = "peter piper picked a "
  + "peck of pickled peppers";
int max = searchMe.length();
int numPs = 0;

for (int i = 0; i < max; i++) {
  // interested only in p's
  if (searchMe.charAt(i) != 'p') continue;
  // process p's
  numPs++;
}

System.out.println("Found " + numPs + " p's");
```