# 1: Types, Variables, Operators

ISe
Ingeniería de Servicios
www.ssme.es

Universidad
Rey Juan Carlos

- Clic para editar la descripción

# Goal
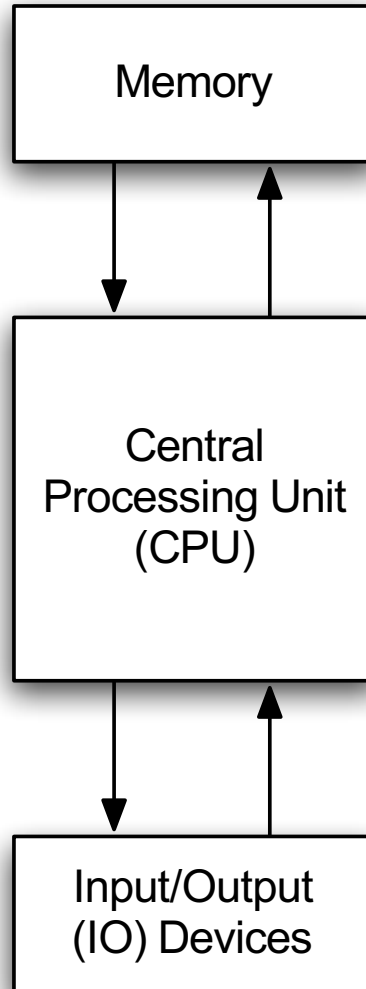
Learn enough Java to do something useful

Examples:

- Simulate a natural/engineering process
- Manipulate PDFs
- Draw pretty graphics

# Assignments

- View and submit via Codeboard.io

- Collaborate with others

- Write your **own** code

- Must submit every assignment

$$z = x + y$$

Read location x

Read location y

Add

Write to location z

# Programming Languages

- Easier to understand than CPU instructions
- Needs to be translated for the CPU to understand it

# Java

- "Most popular" programming language
- Runs on a "virtual machine" (JVM)
- More complex than some others (eg. Python)
- Simpler than others (eg. C++)

## What is Java

- Object Oriented programming language from the 90s

- A programming tool developd by 13 people manged by James Gosling for the *7 project

- Syntactically similar to C/C++ but much more simple

- Platform independent: "write once, run anywhere"

- Oak → Green → Java
  - Just Another Vague Acronym
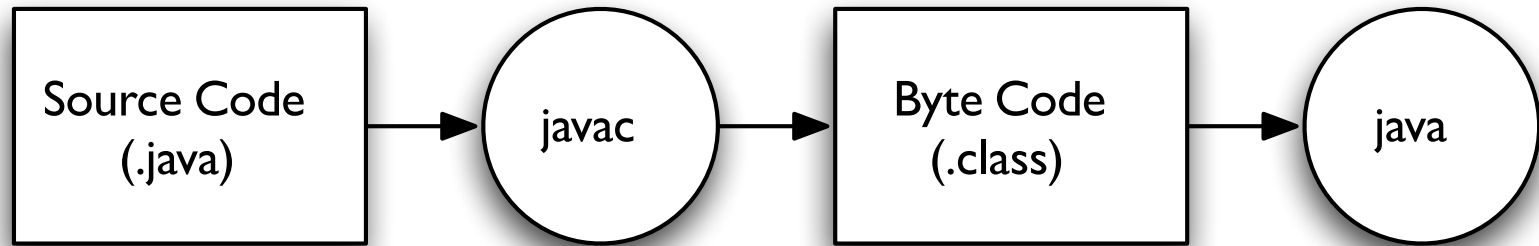  - James Gosling, Arthur Van Hoff, Andy Bechtolsteim

# Java

## Java history

- Java 1.0 -1995
- Java 1.1 - 1997
- Java 1.2 - 1998 (Playground) → Java 2
- Java 1.3 - 2000 (Kestrel)
- Java 1.4 - 2002 (Merlín)
- Java 1.5 - 2004, (Tiger) → Java 5
- Java 1.6.0 - 2006, (Mustang) → Java 6
- Java SE7 - 2011, (Dolphin) → Java 7
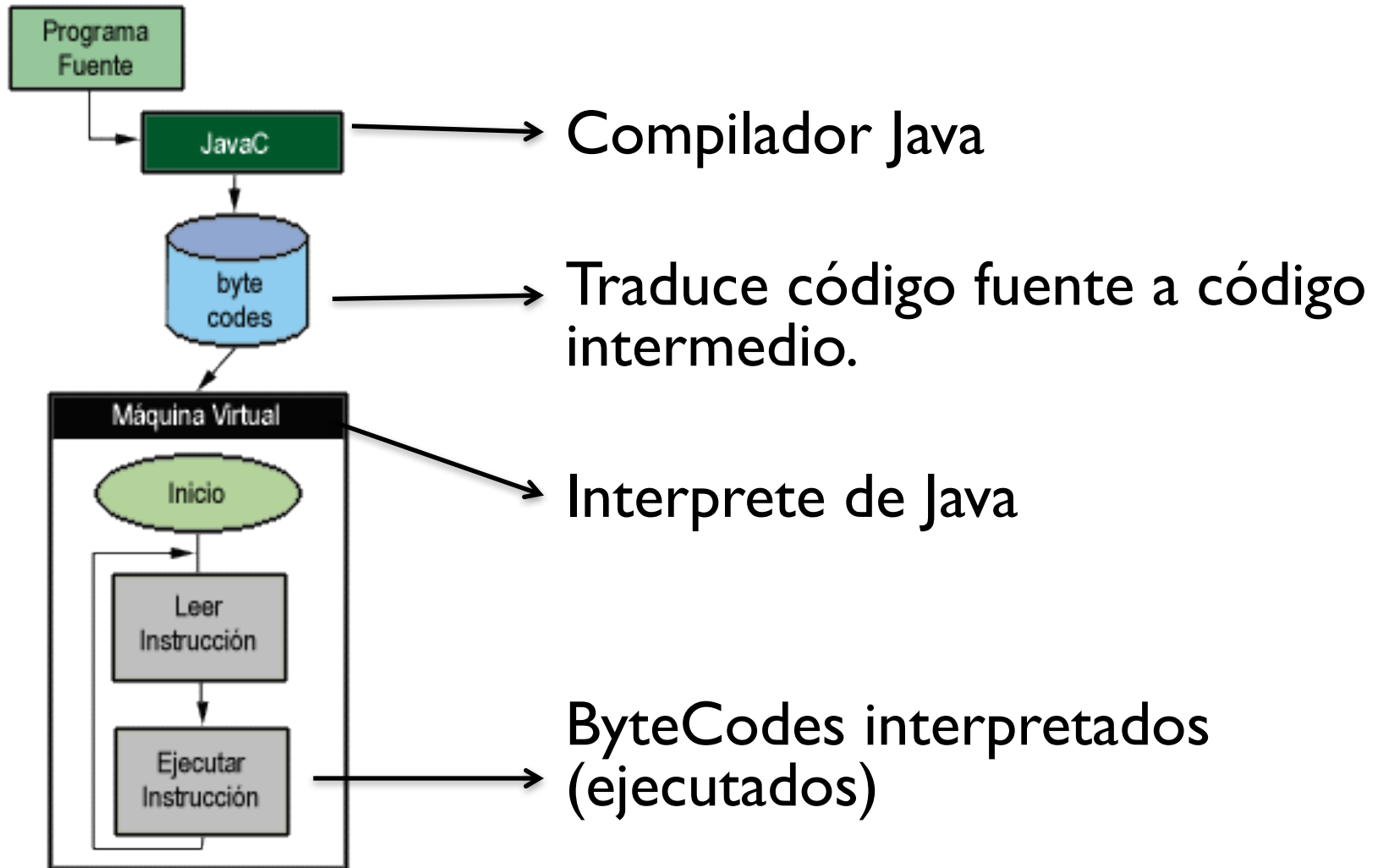- Java SE8 - 2014, (Spider) → Java 8

## Java Virtual Machine



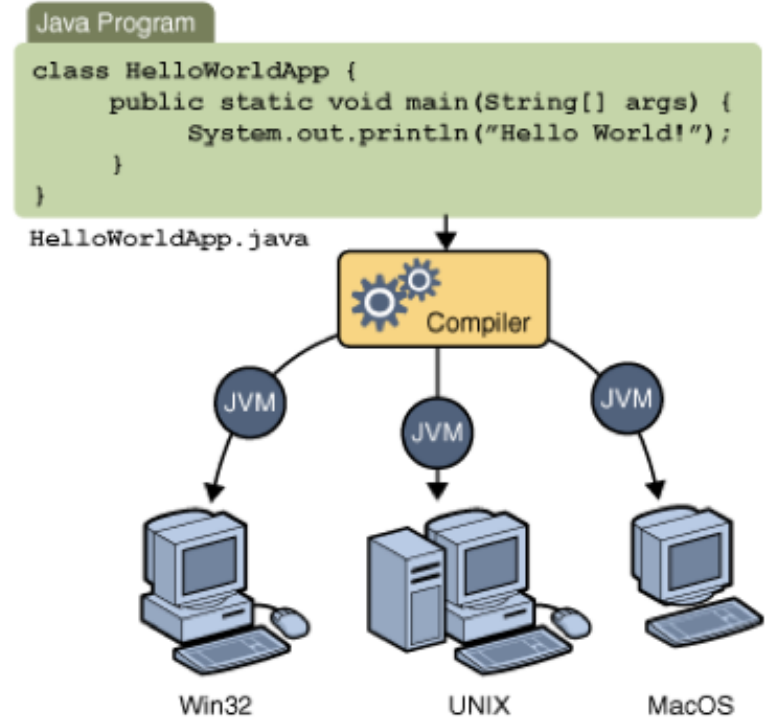– A JVM is needed to run a Java program

Compilador Java

Traduce código fuente a código intermedio.

Interprete de Java

ByteCodes interpretados (ejecutados)

# Bytecode

- – Native language for any JVM.
- – A Java program runs in any platform



```
Java Program
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```
HelloWorldApp.java

Compiler

JVM    JVM    JVM

Win32    UNIX    MacOS

**Write once, Run anywhere**

```java
class Hello {
  public static void main(String[] arguments)
  {
      // Program execution begins here
      System.out.println("Hello
      world.");
  }
}
```

```
class CLASSNAME {
  public static void main(String[]
arguments) {
    STATEMENTS
  }
}
```

System.out.println(*some String*) outputs to the console

Example:

  – System.out.println("output");

```java
class Hello2 {
  public static void main(String[] arguments) {
    System.out.println("Hello world."); // Print once
    System.out.println("Line number 2");  // Again!
  }
}
```

# Types

Kinds of values that can be stored and manipulated.

**boolean**: Truth value (**true** or **false**).

**int**: Integer (0, 1, -47).

**double**: Real number (3.14, 1.0, -2.1).

**String**: Text ("hello", "example").

# Variable declaration

Named location that stores a value of one particular type.

- Form:
  - **_TYPE_ NAME**;
- Example

```
String foo;
```

Start with lower case and use an upper case for every new "word"

- Use intuitive and significative names for variables

# Assignment

Use = to give variables a value.

- Computes right part of the assignment and assigns the result to the variable on the left

Example:



```
String foo;
foo = "IAP 6.092";
```

foo

foo

"IAP 6.092"

Can be combined with a variable  declaration.

Example

```
double badPi = 3.14;
boolean isJanuary = true;
```

```java
class Hello3 {

  public static void main(String[] arguments)

  {

      String foo = "IAP 6.092";
      System.out.println(foo);
      foo = "Something else";
      System.out.println(foo);

  }

}
```

# Operators

# Symbols that perform simple computations

– Simple arithmetic

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| – | operador unario de cambio de signo | -4 | -4 |
| + | Suma | 2.5 + 7.1 | 9.6 |
| – | Resta | 235.6 - 103.5 | 132.1 |
| * | Producto | 1.2 * 1.1 | 1.32 |
| / | División (tanto entera como real) | 0.050 / 0.2<br>7 / 2 | 0.25<br>3 |
| % | Resto de la división entera | 20 % 7 | 6 |

– Combined arithmetic

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| += | Suma combinada | a+=b | a=a+b |
| -= | Resta combinada | a-=b | a=a-b |
| *= | Producto combinado | a*=b | a=a*b |
| /= | División combinada | a/=b | a=a/b |
| %= | Resto combinado | a%=b | a=a%b |

# String Concatenation (+)

## The + operator is overriden

- Conventional add operator when used with numbers
- String concatenation when used with Strings

```
int a = 3;
int b = 2;
int c = a + b;
// c = 5
String text = a + "000" + b;
// text = "30002"
System.out.println(c);
System.out.println(text);
```

# Operators

**Relational**

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| == | igual que | `7 == 38` | `false` |
| != | distinto que | `'a' != 'k'` | `true` |
| < | menor que | `'G' < 'B'` | `false` |
| > | mayor que | `'b' > 'a'` | `true` |
| <= | menor o igual que | `7.5 <= 7.38` | `false` |
| >= | mayor o igual que | `38 >= 7` | `true` |

**Logical**

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ! | Negación - NOT (unario) | `!false`<br>`!(5==5)` | `true`<br>`false` |
| \| | Suma lógica – OR (binario) | `true | false`<br>`(5==5)|(5<4)` | `true`<br>`true` |
| ^ | Suma lógica exclusiva – XOR (binario) | `true ^ false`<br>`(5==5)|(5<4)` | `true`<br>`true` |
| & | Producto lógico – AND (binario) | `true & false`<br>`(5==5)&(5<4)` | `false`<br>`false` |
| \|\| | Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true` | `true || false`<br>`(5==5)||(5<4)` | `true`<br>`true` |
| && | Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false` | `false && true`<br>`(5==5)&&(5<4)` | `false`<br>`false` |

# Truth tables

| A | B | A OR B |
|---|---|--------|
| F | F | F |
| F | V | V |
| V | F | V |
| V | V | V |

| A | B | A AND B |
|---|---|---------|
| F | F | F |
| F | V | F |
| V | F | F |
| V | V | V |

| A | NOT A |
|---|-------|
| F | V |
| V | F |

# Order of Operations

Follows standard (math) rules:

1. Parentheses

2. Arithmethic operators
   1. Multiplication and division
   2. Addition and subtraction
3. Relational operators
4. Logic operators

Which is the resut of the following expression?

> 3 + 5 < 5 * 2 || 3 > 8 && 7 > 6 - 2

Which is the resut of the following expression?

| 3 + 5 < 5 * 2 \|\| 3 > 8 && 7 > 6 - 2 |

```
3 + 5 < 5 * 2 || 3 > 8 && 7 > 6 - 2
       8 < 10 || 3 > 8 && 7 > 4
        true || false && true
           true || false
               true
```

```
int i = 7;
float f = 5.5F;
char c = 'w';
```

| Expresión | Resultado |
|---|---|
| (i >= 6) && (c == 'w') | |
| (i >= 6) \|\| (c == 119) | |
| (f < 11) && (i > 100) | |
| (c != 'p') \|\| ((i + f) <= 10) | |
| i + f <= 10 | |
| i >= 6 && c == 'w' | |
| c != 'p' \|\| i + f <= 10 | |

```
int i = 7;
float f = 5.5F;
char c = 'w';
```

| Expresión | Resultado |
|---|---|
| `(i >= 6) && (c == 'w')` | true |
| `(i >= 6) || (c == 119)` | true |
| `(f < 11) && (i > 100)` | false |
| `(c != 'p') || ((i + f) <= 10)` | true |
| `i + f <= 10` | false |
| `i >= 6 && c == 'w'` | true |
| `c != 'p' || i + f <= 10` | true |

```java
class DoMath {
  public static void main(String[] arguments) {
      double score = 1.0 + 2.0 * 3.0;
      System.out.println(score);
      score = score / 2.0;
      System.out.println(score);
  }
}
```

```java
class DoMath2 {

    public static void main(String[] arguments) {
        double score = 1.0 + 2.0 * 3.0;
        System.out.println(score);
        double copy = score;
        copy = copy / 2.0;
        System.out.println(copy);
        System.out.println(score);

    }

}
```

Every Java program must contains at least one Class

– Java methods container

A Java file could contain several classes, but just one of them being **Public**

– File name must be that of the Public class (ClassName.java)

```java
public class PayrollApp {

    public static void main(String[] args) {
        int hours = 40;
        double payRate = 25.0, grossPay;

        grossPay = hours * payRate;
        System.out.print("Gross Pay: $");
        System.out.println(grossPay);
    }

}
```

# Defining classes

- – Class definition starts with the **Class** keyword
- – Every declaration and instruction is located between the start and end brackets of the class
- – They are referred to as the **body** of the class

```
public class PayrollApp {

    public static void main(String[] args) {
        int hours = 40;
        double payRate = 25.0, grossPay;

        grossPay = hours * payRate;
        System.out.print("Gross Pay: $");
        System.out.println(grossPay);
    }
}
```

Body
Class

# The main method

- – Every program needs a **main** method
- – It is the starting point of the program
- – Always the same heading
- – Always invoked when the program is run

```java
public class PayrollApp {

    public static void main(String[] args) {
        int hours = 40;
        double payRate = 25.0, grossPay;

        grossPay = hours * payRate;
        System.out.print("Gross Pay: $");
        System.out.println(grossPay);
    }
}
```

Punctuation symbols have well-defined purposes

- ; → end of instruction
- "" → Strings are located between
- {} → blocks delimitation
- () → methods arguments
- [ ] → arrays indexing

They also guide indentation and ease the identification of the different blocks

## Each pair of brackets identifies a code block

```java
class DoMath2 {

    public static void main(String[] arguments) {
        double score = 1.0 + 2.0 * 3.0;
        System.out.println(score);
        double copy = score;
        copy = copy / 2.0;
        System.out.println(copy);
        System.out.println(score);

    }

}
```

## Code inside each block is indented to ease identificaction

# Human readable information in the code

- Comments are intended to introduce information that will be ignored by the compiler inside the code

- End line comment (//)

```
int usu = 0; // el número de usuarios
```

- Block comment (/* … */)

```
/*
 * A program to list the good things in life
 * Author: Barry Burd, BeginProg2@BurdBrain.com
 * February 13, 2005
 */

class ThingsILike {

    public static void main(String args[]) {
        System.out.println("Chocolate, royalties, sleep");
    }
}
```

- Javadoc comment (/** ...*/)

```
/**
 * Print a String and then terminate the line.
 */
```

# 1: Types, Variables, Operators

- Clic para editar la descripción