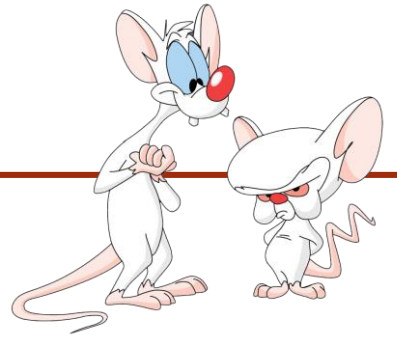
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 1 - Java syntax Part 1

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- Java programs
- General Java syntax
- Variable declaration
- Operators

JAVA RESOURCES

- Check out the free online Java book "How to think like a computer scientist:
  http://greenteapress.com/wp/think-java

- If you are a Python programmer, you might want to try:
  http://interactivepython.org/runestone/static/java4python/index.html

EXAMPLES


```
1. System.out.println("Hello World!");
```



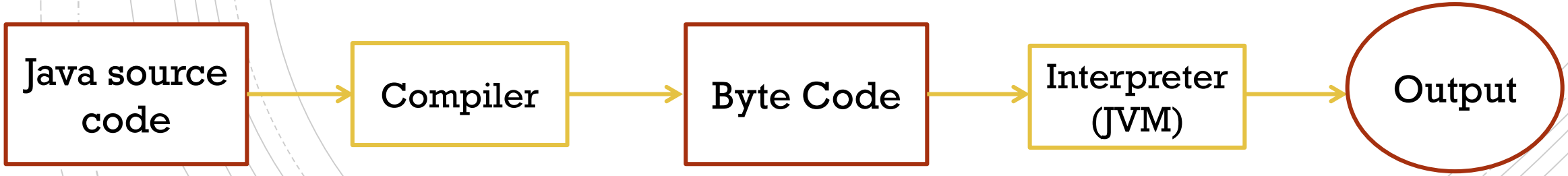
```
1. public class Hello {
2.     System.out.println("Hello World!");
3. }
```

EXAMPLES


```
1. public class Hello {
2.     public static void f(){
3.         System.out.println("Hello World!");
4.     }
5. }
```

JAVA

- High-level programming language
- Both compiled and interpreted:
  - The Java compiler translated the source code into **bytecode**.
    - As machine language, it is easy and fast to interpret.
    - As high-level languages, it is **portable**!
  - Then the Java Virtual Machine (JVM), an interpreter, runs the bytecode.



STEPS TO PROGRAMMING IN JAVA

1. Write a program and save it.

2. Compile the program (**javac**) – it is enough to save your file in Eclipse

3. Run the program (**java**) – the run button in Eclipse

HELLO WORLD!

Let's look at the code of Hello World! written in Java:


```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```


As expected, the program simply displays `Hello, World!` on your screen.

CURLY BRACES

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- Java uses curly braces to group things together.
- They denote a *block* of code.
- They help us keep track of what parts of the code are related.
- If one of them is missing or there's an extra one → **syntax error**

STATEMENTS

```
public class HelloWorld {
   public static void main (String[] args) {
      System.out.println("Hello, World!");
   }
}
```


- A statement is a line of code that performs a basic operation.
- All statements in Java end in a semi-colon.
- The statement in this program is a print statement: it displays a message on your screen.

PRINTING TO THE CONSOLE

```
public class HelloWorld {
   public static void main (String[] args) {
      System.out.println("Hello, World!");
   }
}
```


- **To print in Java you can use one of the following methods:**
  - `System.out.println()` **– which displays a new line character at the end**
  - `System.out.print()` **– which only display what it receives as input.**

- **NOTE, Java is case-sensitive:** `System` ≠ `system` ≠ `SYSTEM`

STRINGS

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- Phrases that appear in quotation marks are called Strings.
- Strings literals must start and end with **double quotes**.

METHODS AND CLASSES

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- Almost every line of code you will write in Java will be inside a method.
- Every method you will ever write will be part of a class.
- In this program: **`HelloWorld`** is a class, **`main`** is a method.

METHODS

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- A method is named sequence of statements
- These open and close curly brackets tell the computer where the **`main`** method (named block of code) starts and ends.

METHODS

```
public class HelloWorld {
   public static void main (String[] args) {
      System.out.println("Hello, World!");
   }
}
```


- This program defines a method called **`main`**, which is **`public`**, **`static`**, and **`void`** (but don't worry about this for now)
- The `main` method is a special one:
  - The execution of a program always starts from the first statement in the main method and ends when it finishes the last statement.

CLASSES

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- This program **must** be saved as a file named *HelloWorld.java*
- Convention: names of classes starts with capital letter.

CLASSES

```
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello, World!");
    }
}
```


- A class is a collection of methods.
- This program defines a class called `HelloWorld` which is:
  - `public` (we'll see more about this later)
  - defined by what is in between the curly brackets.

COMMENTS

```
public class HelloWorld {
   // This line is ignored
   public static void main(String[] args) {
      /* As well as this one
      and this one
      and this last one */
      System.out.println("Hello, World!");
   }
}
```


- A single line comment in Java starts with // and ends when you press enter.
- A multi-line comment starts with /* and ends with */.
- All comments are ignored by the computer.

ECLIPSE DEMO

- Open up Eclipse
- Create a Java Projects
- Write the HelloWorld program and run it.

WHICH LINES ARE STATEMENTS

Broadly speaking, there are 3 different kinds of 'lines' of code you can write:

1. Code that defines where a block starts and ends. These lines either end with an open curly bracket, or the whole line is a single close curly bracket.
2. A line of code that does something. These are statements and end with a semi-colon.
3. A comment.

CODE STRUCTURE

- All of your methods will be *inside* a class.
- (Almost) all of your statements will be *inside* of a method.
- You can only run a .java file which contains the main method.

GOOD PRACTICE

- In Java most spaces are optional.
  - For instance, you **cannot** write


```
publicstaticvoidmain (String[] args) {
```

  - But it is **ok** to write our program as:

```
public class HelloWorld { public static void main
(String[] args) { System.out.println("Hello,
World!");}}
```

GOOD PRACTICE

- Tabs and newlines are optional, but without them the program becomes hard to read!
- Some editors automatically format the code, but in general it is good practice to make sure to keep you program organized and easy to read!

VARIABLES

THE LIFE OF A VARIABLE

- Declaration
- Initialization
- Manipulation

DECLARATIONS


```
int aNumber;
```


- When you declare a variable, you give it a name and a type

DECLARATIONS


```
int aNumber;
```


- The type of this variable is `int`
- `int` is a keyword (reserved word) in Java.
  It is short for integer.

DECLARATIONS


```
int aNumber;
```


- The name of this variable is `aNumber`
- This is **not** a keyword in Java.
- `aNumber` is the name of the place in memory with enough space to store an integer.

ASSIGNMENT – RULES

We can store values inside a variable with an *assignment statement*.

- When we make an assignment we update the variable's value.
- Assignment operator: =

  It assigns the value on the right to the variable on the left.

- **The variable need to have the same type as the value we assign to it.**
- Variables must be **initialized** (assigned for the first time) before they can be used.

ASSIGNMENT – EXAMPLES

Examples:


```
String today;              // the variable today is declared

today = "Monday";          // today gets initialized

/* the variable hour is declared and initialized on
the same line */

int hour = 10;

int date = "Wednesday";    // NOT LEGAL!
```

VARIABLES

- **Declaration:**

```
int a;
```


VARIABLES

- **Declaration:**

```
int a;
```

- **Assignment:**

```
a = 3;
```


VARIABLES

- **Declaration:**

```
int a;
```

- **Assignment:**

```
a = 3;
```

- **New assignment:**

```
a = 5;
```


NAMING CONVENTIONS

- We use lowerCamelCase for names of variables and methods.
  E.g.: `isSnowing`, `catName`.

- We use UpperCamelCase for names of classes.
  E.g.: `SomeMethods`, `ShapeClass`.

EXPRESSIONS and OPERATORS

EXPRESSIONS

Recall that an *expression* represents a single value that needs to be computed.

That value has a specific type!

STANDARD INTEGER OPERATIONS

- Addition '+', Subtraction '-'
- Multiplication '*'
- Division '/'
  - The output of the division between two integers is an integer! Java will always round toward zero. That is, it computes the quotient between two numbers.
- Modulo (remainder) "%"
  - It performs integer division and outputs the remainder.

THE '+' OPERATOR

- If used between numbers, it will add the numbers together
- If used between strings, it will concatenate those strings.
- What happens in the following example?


```
System.out.println( 2 + 3 + "5");
System.out.println("5" + 2 + 3);
```

Output:

```
55
523
```

The two expressions are evaluated from left to right!

RELATIONAL OPERATORS

- Relational: <, >, <=, >=
- Equality: ==, !=
- They operates on compatible values (not on `String`)
- Expression containing them evaluate to a `boolean` value.

LOGICAL OPERATORS

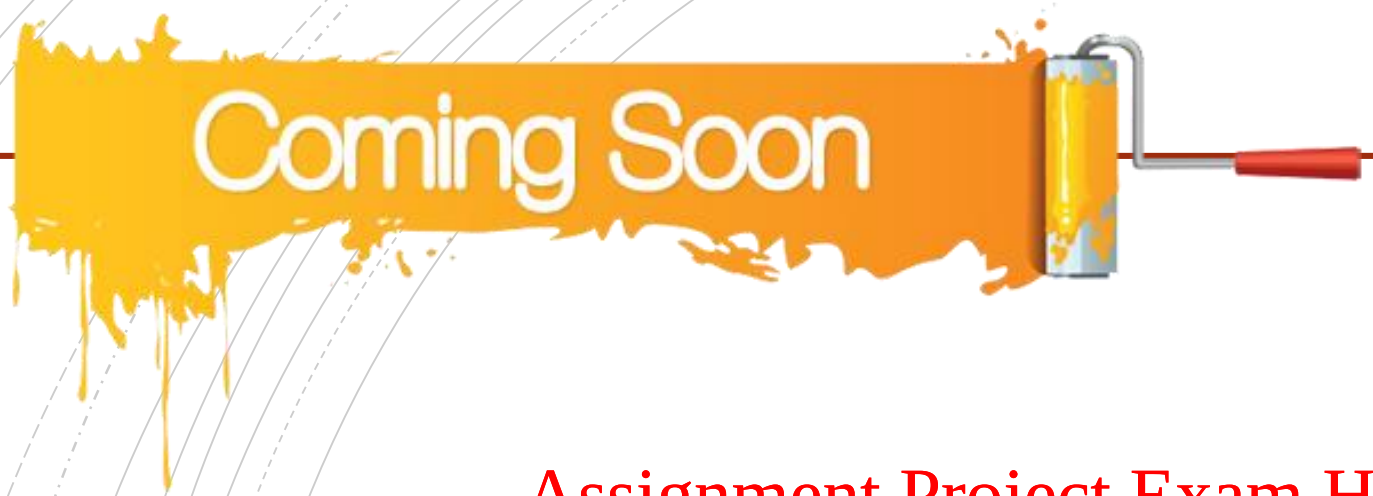
- Logical operators take boolean expressions (i.e. expressions that evaluate to a boolean value) as inputs and produce a result of type `boolean`

- Java has 3 logical operators:
  - NOT '`!`'
  - AND '`&&`'
  - OR '`||`"

ORDER OF OPERATIONS

From left to right:

1. Parenthesis
2. !
3. Typecasting
4. Arithmetic
   i. *, /, %
   ii. +, -
5. Comparison
   i. Relational: <, >, <=, >=
   ii. Equality: ==, !=
6. Boolean: &&, ||



In the next video we will be talking more about Java syntax and the scope of variables.