# Unit 1: Primitive Types
## Basic Java Syntax

Adapted from:

1) Building Java Programs: A Back to Basics Approach

by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum

https://longbaonguyen.github.io

# Java

What do Minecraft, Android phones, and Netflix have in common? They're all programmed in Java!

Many of the apps you use in an Android phone or tablet are also written in Java. Netflix uses Java for some of its software too. Java is used worldwide to create software that we all use.

# Java

Java is a **programming language**, which means that we can use Java to tell a computer what to do.

Computers don't actually speak Java so we have to **compile** (translate) Java source files (they end in .java) into class files (they end in .class).

compiling with javac

| Person.java | → | Person.class |
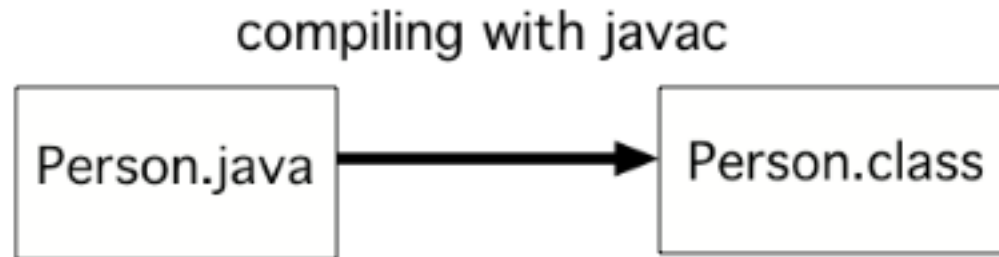
The source file is something humans can read and edit, and the class file is code that a computer can understand and can run.

# Java Terminology

All Java code are organized into units called **classes**.

**class**:
*(a)* A module or program that can contain executable code.
*(b)* A description of a type of objects. (Animal class, Human class, Employee class, Car class)

**statement**: An executable piece of code that represents a complete command to the computer.

– every basic Java statement ends with a semicolon ;

**method**: A named sequence of statements that can be executed together to perform a particular action or computation.

# Structure of a Java program

```
public class name {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

**class**: a program

**method**: a named group of statements

**statement**: a command to be executed

- Every executable Java program consists of a **class**, called the **driver class**,
  - that contains a **method** named `main`,
    - that contains the **statements** (commands) to be executed.

# Program Template

Here's a simple program that prints out a message on the screen. Code highlighted in red should be in every program!

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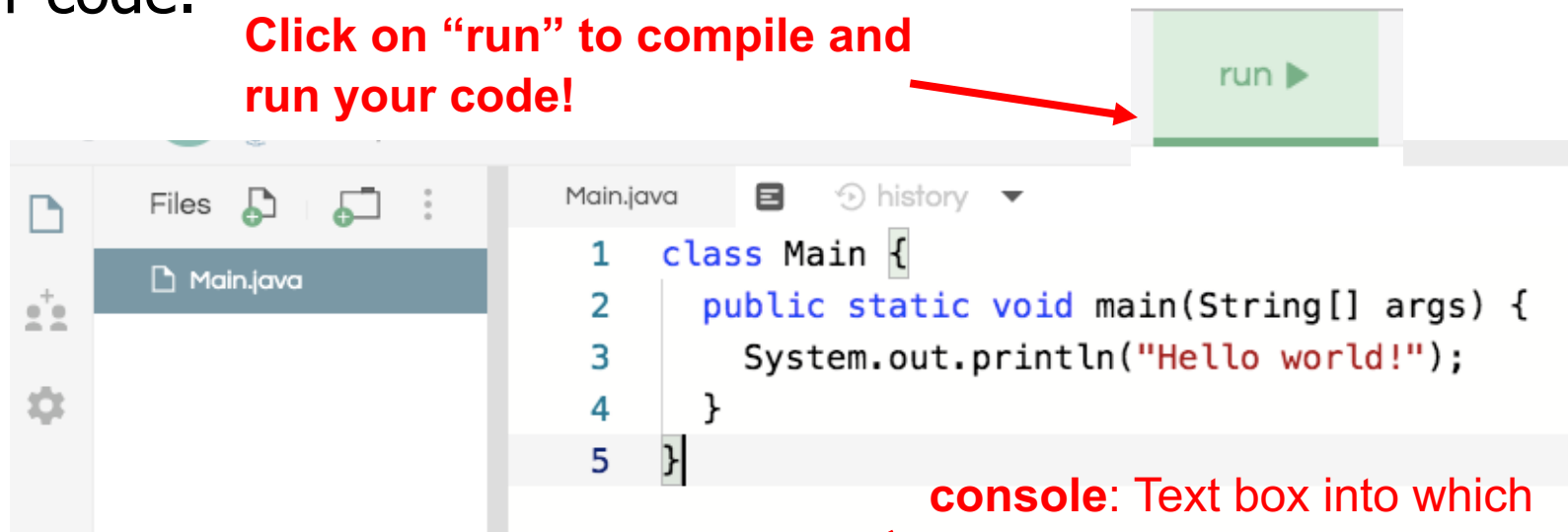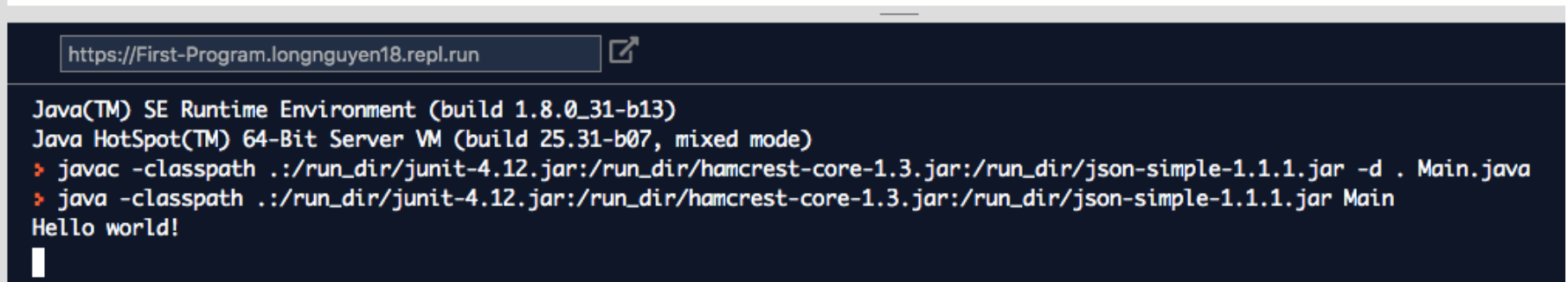
```
Output:
Hello, World!
```

# First Program: repl.it

We will use repl.it, an online integrated development environment(IDE), for the first part of this course to write all of our code.

**Click on "run" to compile and run your code!**

```
run ▶
```

Files
Main.java

```java
1  class Main {
2    public static void main(String[] args) {
3      System.out.println("Hello world!");
4    }
5  }
```

Main.java    history

**console**: Text box into which the program's output is printed.

```
https://First-Program.longnguyen18.repl.run

Java(TM) SE Runtime Environment (build 1.8.0_31-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.31-b07, mixed mode)
> javac -classpath .:/run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar -d . Main.java
> java -classpath .:/run_dir/junit-4.12.jar:/run_dir/hamcrest-core-1.3.jar:/run_dir/json-simple-1.1.1.jar Main
Hello world!
```

# File naming

The name of the class has to match up with the name of the file.

For example, the class below is called `Main` therefore the name of the file is Main.java. On replit, the main class must be called Main.java.(in other IDEs, you can pick any name.)
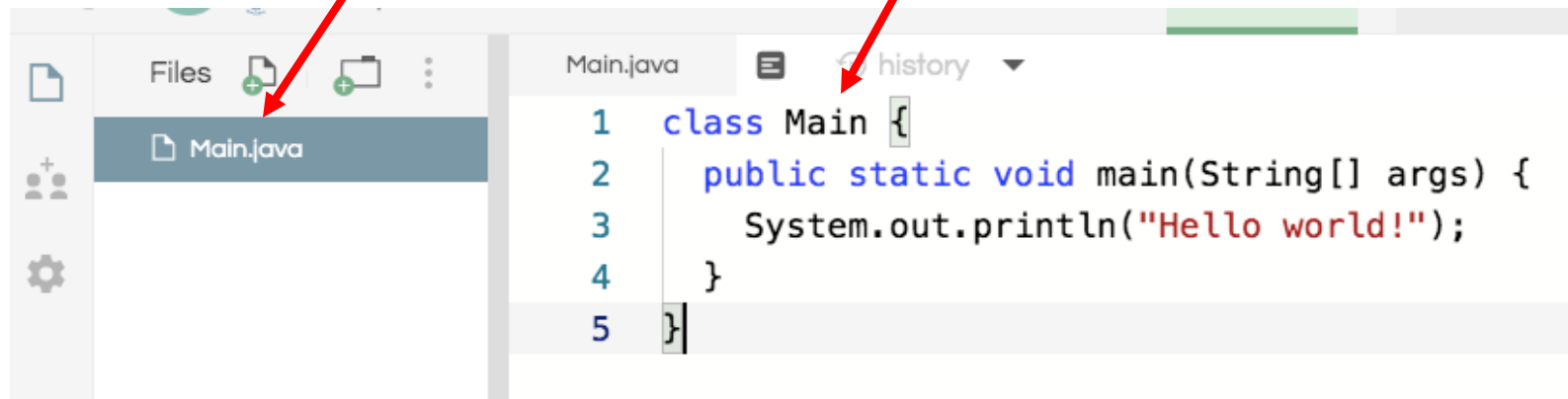
# **Printing**

Two ways to print a line of output on the console:
`System.out.println()` and `System.out.print()`.

`System.out.println()` is just the way that you ask Java to print out the value of something followed by a new line (ln).

`System.out.print()` without the ln will print out something without advancing to the next new line.

# `System.out.println`

```
public class Welcome{

    public static void main(String[] args){

        System.out.println("Hi there!");

        System.out.println("Welcome to APCS A!");

    }

}
Output:

Hi There!

Welcome to APCS A!
```

The "System" in System.out.println() must be capitalized. And the command line must end with a semicolon (;).

# System.out.print

```
public class SecondClass{

    public static void main(String[] args){

        System.out.print("Hi there!");

        System.out.println("Welcome to APCS A!");

        System.out.print("We will learn Java!");

    }

}
Output:
Hi There!Welcome to APCS A!
We will learn Java!
```

Do you see why there are two lines of output as above?

# Find the errors.

```
pooblic class Errors

    public static void main(String args){

        System.out.print("Good morning! ")

        system.out.print("Good afternoon!);

        System.Print "And good evening!";


}
```

See next slide for all of the corrections.

# Corrected!

```java
public class Errors {
    public static void main(String[] args){
        System.out.print("Good morning! ");
        System.out.print("Good afternoon!");
        System.out.print("And good evening!");
    }
}
```

# Strings

- **string**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
    - The quotes do not appear in the output.

  - Examples:

    A string enclosed in quotes is called a **string literal**.

    ```
    "hello"
    "This is a string.  It's very long!"
    ```

- Restrictions:
  - May not span multiple lines.
    ```
    "This is not
    a legal String."
    ```

  - May not contain a " character.
    ```
    "This is not a "legal" String either."
    ```

14

# Comments

- **comment**: A note written in source code by the programmer to describe or clarify the code.
  - Comments are not executed when your program runs.
- Syntax:
  **// comment text, on one line**
  or,
  **/* comment text; may span multiple lines */**

- Examples:
  ```
  // This is a one-line comment.

  /* This is a very long
     multi-line
     comment. */
  ```

# Using comments

- Where to place comments:
  - at the top of each file (a "comment header")
  - at the start of every method (seen later)
  - to explain complex pieces of code

- Comments are useful for:
  - Understanding larger, more complex programs.
  - Multiple programmers working together, who must understand each other's code.

# Comments example

```java
/* Suzy Student, CS 101, Fall 2019
   This program prints lyrics about ... something. */

public class BaWitDaBa {
    public static void main(String[] args) {
        // first verse
        System.out.println("Bawitdaba");
        System.out.println("da bang a dang diggy diggy");
        System.out.println();

        // second verse
        System.out.println("diggy said the boogy");
        System.out.println("said up jump the boogy");
    }
}
```

# Indent Nicely!

```
public class Welcome{ public static void main(String[]
args){ System.out.println("Hi there!"
);System.out.println("Welcome to APCS A!");}}
```

The code above will compile and run correctly. Java ignore whitespaces. But it is very hard to read, please make an effort to indent nicely!

```
public class Welcome{

    public static void main(String[] args){

        System.out.println("Hi there!");

        System.out.println("Welcome to APCS A!");

    }

}
```

# Lab 1

Create a new repl on your repl.it account and write a program that has the following outputs:

You must use exactly 5 different print statements.(println and/or print).

Output:

I am Sam. Sam I am. I do not like them, Sam-I-am.

I do not like green eggs and ham.

# Lab 2

You must reuse the same repl you created for Lab 1.

Create two integer variables: pens and pencils. Initialize it to some positive values. Use print statements and String concatenation to have the following output:

Output: (Assuming that the initialized values for pens=18, pencils = 10)

There are 18 pens.

There are 10 pencils.

Total is 28 pens and pencils.

# References

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum:
https://runestone.academy/runestone/books/published/csawesome/index.html

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

https://longbaonguyen.github.io