
LETS LEARN JAVA
GET SET GO

Training Plan - Basics

Topics to be covered
First program, Understanding the program, How to compile, How to Execute Methods. Parameters and Arguments Creating Method in same Class Moving the method to another utility class and invoking the method String Formatting and String Printing Usage of JShell What are variables. Naming conventions. What are the data types

Our first program

```
public class MyFirstJavaProgram {

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

How to compile
How to run >> Console

What do the following terms mean in the example

```
public
class
static
void
main
String
[]
args
System.out.println
Hello World!
```

Why name of file is same as name of class

Our first program using a method/function

```
public class SumOfNumbers {

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int sum = findTotal(num1, num2);

        print(sum);
        print(345);
        print(findTotal(100, 200));
    }

    private static Integer findTotal(Integer num1, Integer num2) {
        Integer total = num1 + num2;
        return total;
    }

    private static void print(Integer value) {
        System.out.println(value);
    }
}
```

See the different ways in which the findTotal and print methods are invoked.

Reusability – At the heart of Java

```java
public class SumOfNumbers {

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int sum = findTotal(num1, num2);

        MyPrintUtils.print(sum);
        MyPrintUtils.print(345);
        MyPrintUtils.print(findTotal(100, 200));
    }

    private static Integer findTotal(Integer num1, Integer num2) {
        Integer total = num1 + num2;
        return total;
    }
}
```

```java
public class MyPrintUtils {

    public static void print(Integer value) {
        System.out.println(value);
    }
}
```

Notice how the method definition has changed from private to public

Using jshell (since Java 9)

```
jshell> String name = "Manish";
name ==> "Manish"

jshell> int age = 35;
age ==> 35

jshell> System.out.println("Hello World");
Hello World

jshell> System.out.println(name);
Manish

jshell> System.out.println("Name is " + name);
Name is Manish

jshell> System.out.println("Name is " + name + " and age is " + age);
Name is Manish and age is 35
```

Ways of printing information to console

```java
public class ConsoleDataPrinter {
    public static void main(String[] args) {

        // Printing text
        System.out.println("This line prints this text and includes a new line at the end");
        System.out.print("Only this text printed, but no new line at end.**");
        System.out.print("##This line continues from last line. No line break.");
        System.out.println("\n"); // This line just prints a new line nothing else

        // Printing variables - formatting string + printing in one shot
        System.out.printf("10 * 3 = %d", 10 * 3).println();
        System.out.printf("%d / %d = %d", 10, 2, 10 / 2).println();

        // Separating concerns - printing and formatting
        System.out.println(String.format("20 * 3 = %d", 20 * 3));
        System.out.println(String.format("%d / %d = %d", 20, 2, 20 / 2));
    }
}
```

Can you print 7 table using such print statements

`System.out` is common across all as it points to the console.

`System.out.print` and `System.out.println` are most common.

Formatting – `String.format()`

Formatting and Printing in one shot – `System.out.printf()`

Printing Variables – what are “variables”

```
public class VariableDemo {
    public static void main(String[] args) {

        int age = 30;
        char gender = 'M';
        boolean isMarried = false;

        System.out.println(String.format("Peter's is %d years old. \nHis gender: %c.", age, gender));
        System.out.println(String.format("'Peter is married' - This statement is %b", isMarried));
    }
}
```

Which are the variables here
What is happening behind the scenes

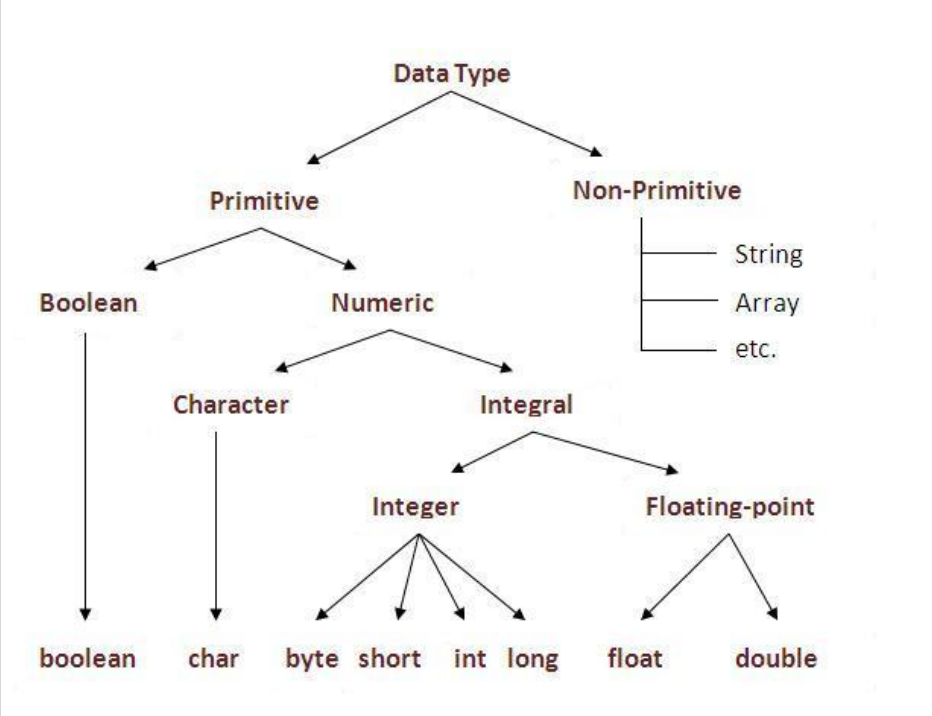
What are int, char, boolean

Variables – naming conventions

```
public class VariableDemo {
    public static void main(String[] args) {

        int age = 30;
        char gender = 'M';
        boolean ISMARRIED = false; // unnecessarily prominent.
        long nooffriends = 5; // less readable.

        boolean isMarried = false; // more readable. Follows camel casing
        long noOfFriends = 5; // more readable. Follows camel casing

        System.out.println(String.format("Peter's is %d years old. \nHis gender: %c.", age, gender));
        System.out.println(String.format("'Peter is married' - This statement is %b", isMarried));
        System.out.println(String.format("He has %d friends", noOfFriends));
    }
}
```

- Characters, Numbers, $ and _
- Cannot start with a number
- No length restriction
- Naming should follow camel casing
- Cannot be a **keyword**

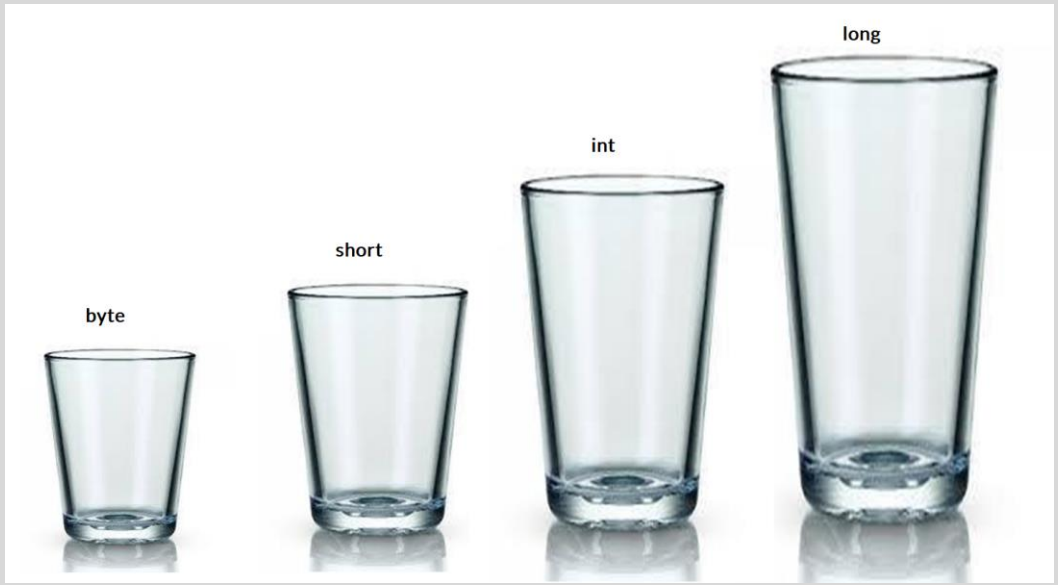
Data Types



Primitive Type	Wrapper Equivalent	Size	Range/Examples
boolean	Boolean	-	true/false
char	Character	2 bytes (16 bits)	'A', 'X', '0'
byte	Byte	1 byte (8 bits)	-128 to 127
short	Short	2 bytes (16 bits)	-32768 to 32767
int	Integer	4 bytes (32 bits)	- 2.14 Billion to 2.14 Billion
long	Long	8 bytes (64 bits)	Huge number
float	Float	4 bytes (32 bits)	Not precise
double	Double	8 bytes (64 bits)	Not precise

be **c**areful; **b**aboons **s**houldn't **i**nfuriate **l**arge **f**lying **d**ragons

Data Types



Larger glass/cup can easily hold what smaller glass/spoon can. Hence, implicit conversion is allowed.

```
int i = 1000;
long l = i; // allowed
float f = i; // allowed
double d = l; // allowed
```

Smaller glass may not be able to hold what larger glass is holding. Hence, if you are moving from larger to smaller glass, then you need to explicitly say that it can actually hold it.

```
long l = 1000;
int i = (int) l; // not allowed without the explicit type casting
```

Let's connect the dots

```
public double calculateAverageOfTwoNumbers(int firstNumber, int secondNumber) {
    double average = (firstNumber + secondNumber)/2;
    return average;
}
```

- `Methods` perform work – hence their names will look like verbs
- `Variables` store values – hence their names will look like nouns
- Every variable has a `data type` – which defines which type of data it will store
- Functions/Methods may or may not return anything. If they do not return anything, the return type is `void`.
- If functions/methods `return a value` (and they can only return one value), then that has to be designated by a data type as well.
- When we call a method, we pass `parameters`.
- Parameters passed into method call are received by method as `arguments`

Let us practice below

1. Write your first program, and print your own favorite line.
2. WAP (Write a Program) to do the following using methods to implement business/calc logic:
    i. Sum of the two numbers
    ii. Average of the two numbers.
    iii. The difference between the two numbers (use `Math.abs()`)

APPENDIX

What is Java? And why is it so popular

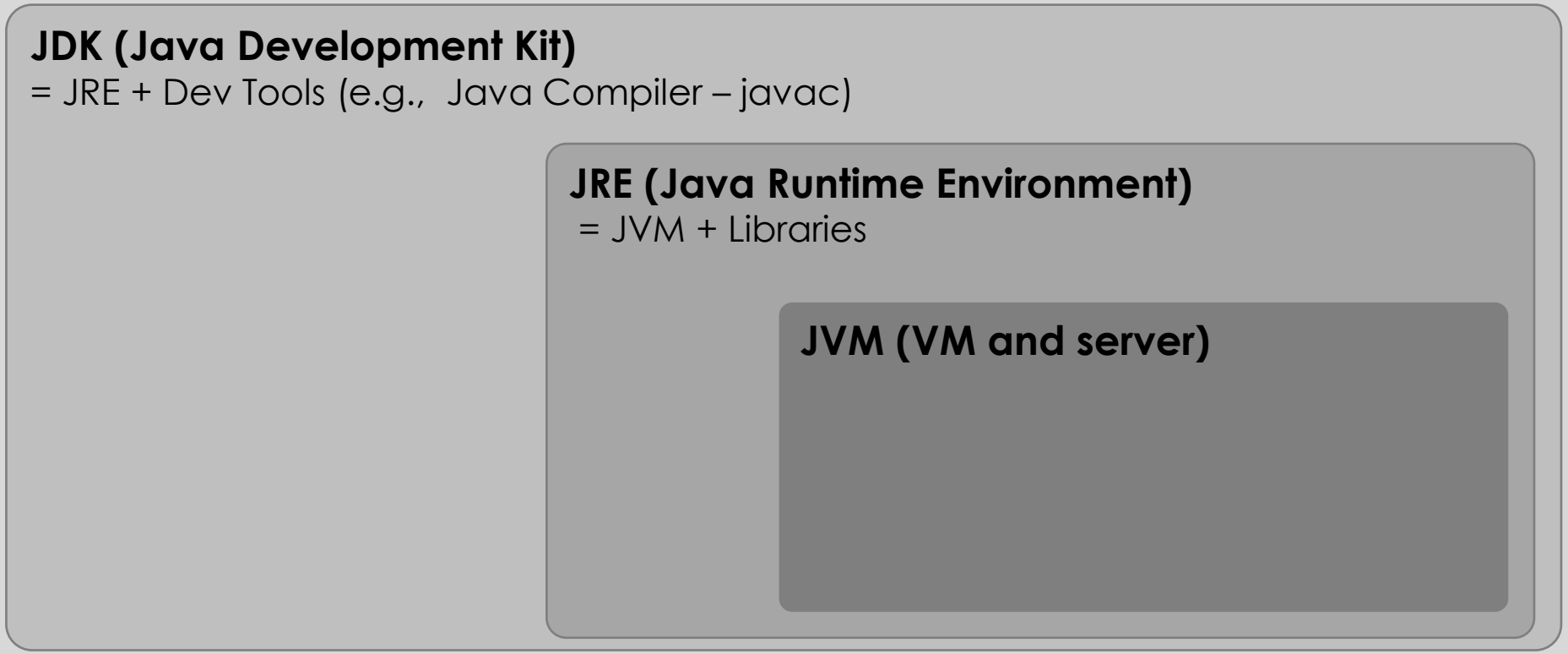
- Java – A programming language and a platform (JRE)
- Language with maximum jobs in the market (Search for "most popular programming languages 2020")
- **JDK and JRE** – what is the difference. What is JVM

JDK = JRE + Development Tools (Javadoc, Javac)
JRE = JVM + Libraries

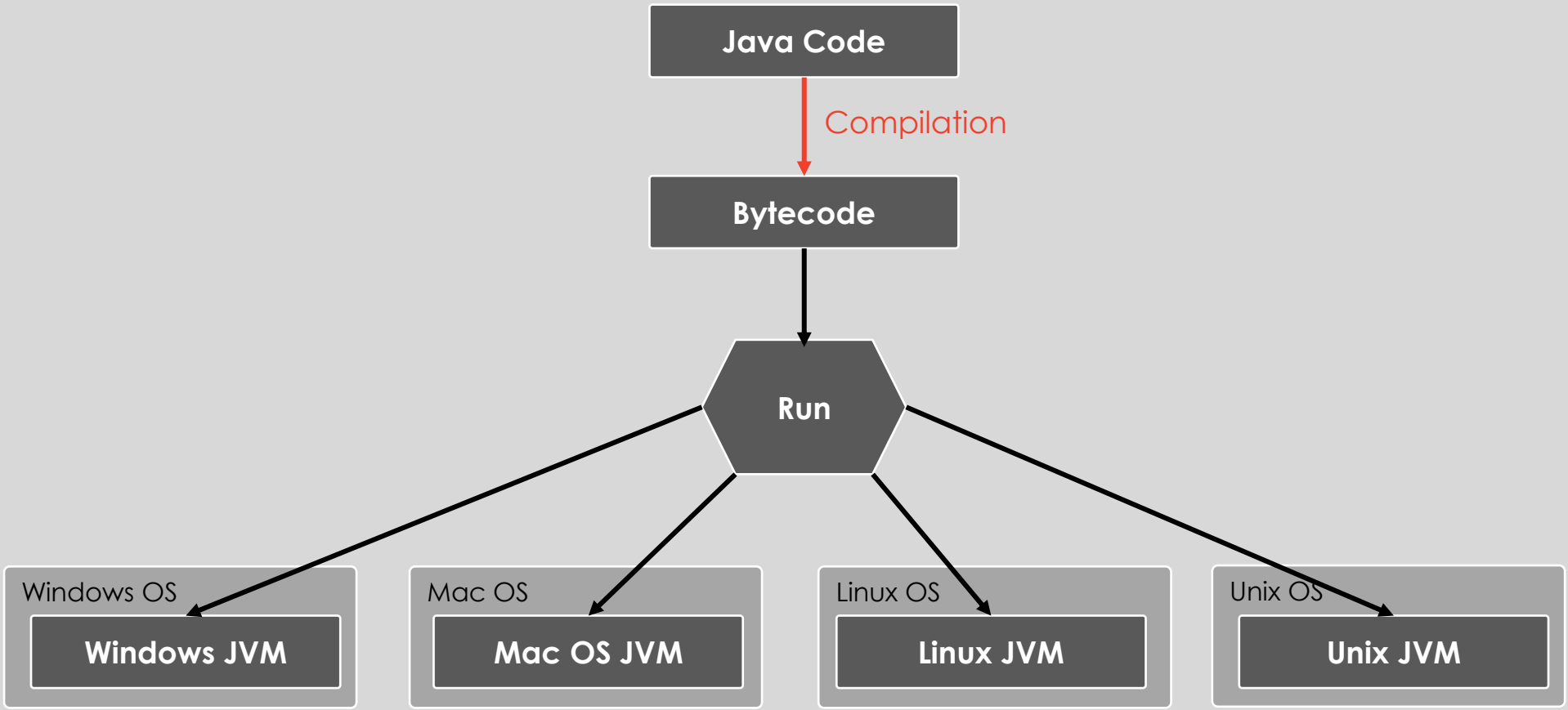
- **James** Bond + Ryan **Gosling** – invented Java in 1995
- Java 1, 1.1, 1.2…..1.4, Java 5, 6, 7, **8**, 9, 10, **11**, 12 … 16 (Java 8 and 11 are **LTS** versions)
- Started under Sun Microsystems, and now under Oracle (since 2009).
- Popularity of Java

  - **Platform Independent**, Object Oriented, Fast, Secure (OS >> JVM >> Programs)

JDK, JRE and JVM



Platform Independence in Java



How is Java different from C++

Both are object oriented but there are some major differences between the two.

Aspect	**C++**	**Java**
Security	Lower (usage of pointers)	Higher (no pointers, JVM is used)
Type of Programming	System	Application
Platform	Dependent (only compiler)	Independent (compiler + interpreter)
Supports goto, operator overloading	Yes (program becomes difficult to follow)	No (keeps things simple, yet effective)
Parameter passing	Call by Value & Call by Reference	Only Call by Value
Thread Support	No built in support	Built in thread support