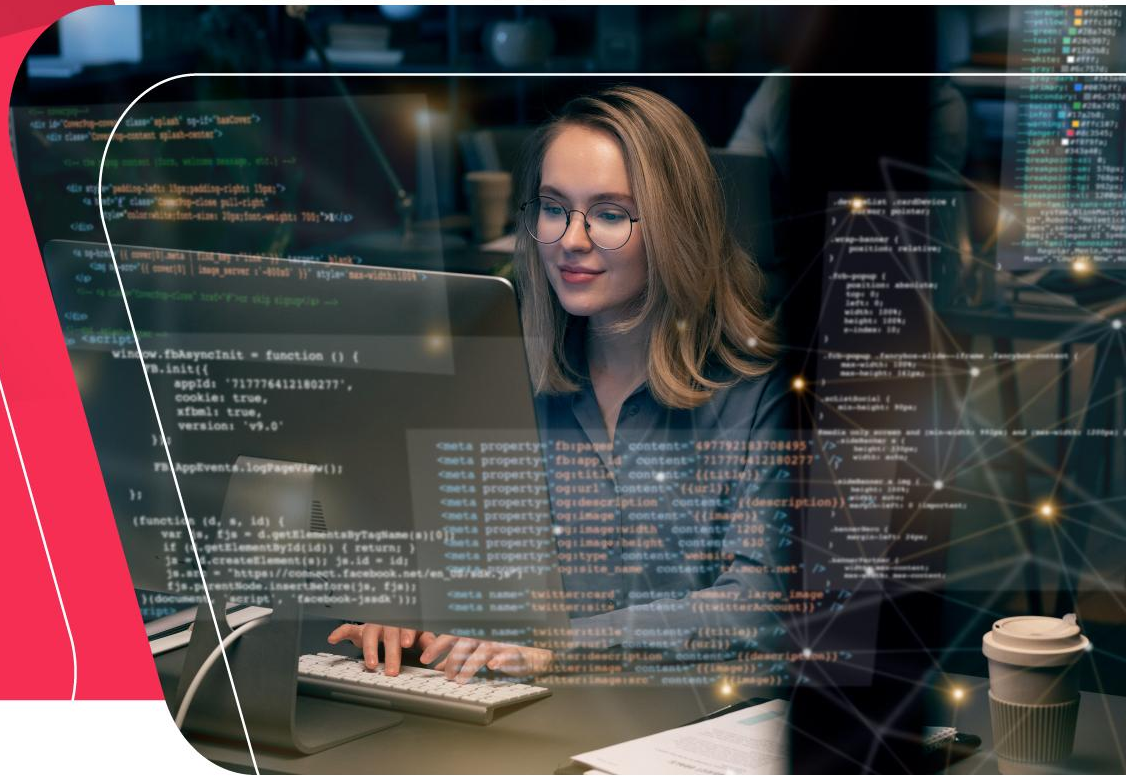
TIA-Academy

Java Bootcamp

Day 03



Technologies will be Use

- JDK 8/**11**/15
- JRE 8/**11**/15
- Writing Code using Notepad++ (For Windows) or Vim (For Linux and Mac)
- Compiling with Command Prompt (For Windows) or Terminal (For Linux and Mac)
- ***Forbidden of using IDE Based Development Tools as of Intellij IDEA or Eclipse***

TIA-Academy

Java Introduction

What is Java?

- Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. ***James Gosling*** is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.
- **Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Java Platforms / Editions

There are 4 platforms or editions of Java:

- Java SE (Java Standard Edition)

  It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

- Java EE (Java Enterprise Edition)

  It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

- Java ME (Java Micro Edition)

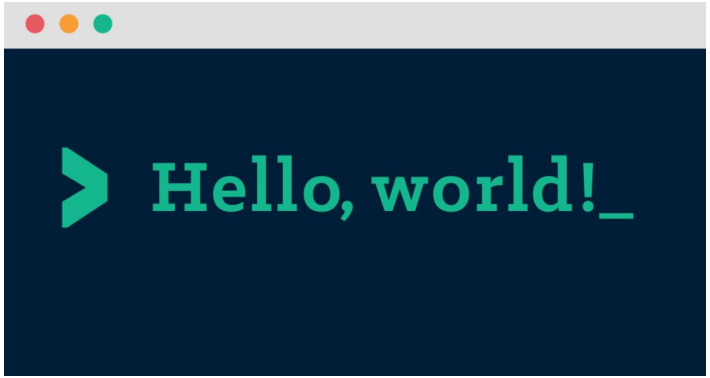
  It is a micro platform that is dedicated to mobile applications.

- JavaFX

  It is used to develop rich internet applications. It uses a lightweight user interface API.

Java Hello World Program

- In this tutorial, you will learn to write "Hello World" program in Java.
- A "Hello, World!" is a simple program that outputs Hello, World! on the screen.
- Since it's a very simple program, **it's often used to introduce a new programming language to beginner.**
- Let's explore how Java "Hello, World!" program works.



Java Hello World Program

```
// Your First Program

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

Output:

```
Hello, World!
```

How Java "Hello, World!" Program Works?

```
1.  // Your First Program
```

- In Java, **any line starting with // is a comment**.
- Comments are intended for users reading the code to understand the intent and functionality of the program.
- **It is completely ignored by the Java compiler** (an application that translates Java program to Java bytecode that computer can execute).

How Java "Hello, World!" Program Works?

2. `class HelloWorld { ... }`

- In Java, **every application begins with a class definition**.
- In the program, **HelloWorld is the name of the class**, and the class definition is:

```
class HelloWorld {
   ... .. ...
}
```

- For now, just remember that **every Java application has a class definition**, **and the name of the class should match the filename in Java.**

How Java "Hello, World!" Program Works?

3. `public static void main( String[] args) { ... }`

- This is the main method. Every application in Java must contain the main method. **The Java compiler starts executing the code from the main method.**
- How does it work? Good question. However, we will not discuss it in this article. After all, it's a basic program to introduce Java programming language to a beginner. We will learn the meaning **of public, static, void, and how methods work? in later chapters**.
- For now**, just remember that the main function is the entry point of your Java application**, and it's **mandatory in a Java program**. The signature of the main method in Java is:

```
public static void main(String[] args) {
... .. ...
}
```

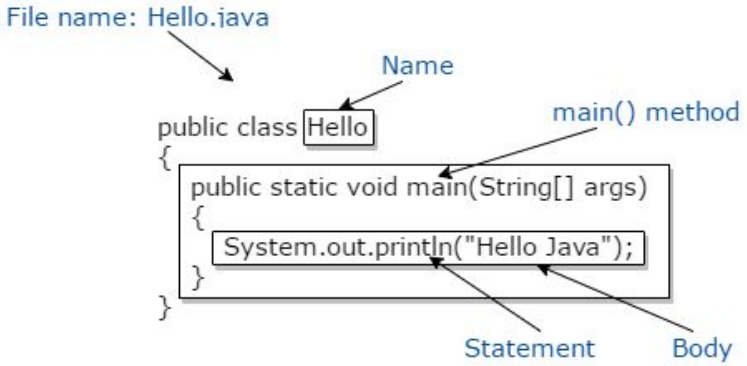
How Java "Hello, World!" Program Works?

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

- The code above is **a print statement. It prints the text Hello, World! to standard output (your screen).**
- The text inside the quotation marks is called String in Java.
- **Notice the print statement is inside the main function, which is inside the class definition.**

Summary

- Every valid Java Application must have a class definition that matches the filename (**class name and file name should be same**).
- The **main method must be inside the class definition.**
- The **compiler executes the codes starting from the main function**.



Summary

- This is a valid Java program that does nothing.
- Don't worry if you don't understand the meaning of class, static, methods, and so on for now. We will discuss it in detail in later chapters.

```
public class HelloWorld {
    public static void main(String[] args) {
        // Write your code here
    }
}
```

TIA-Academy

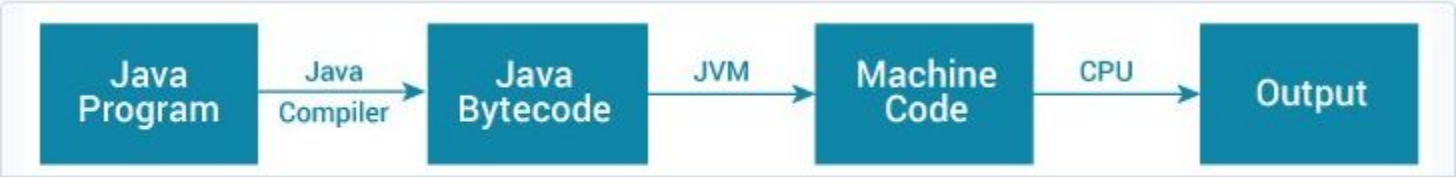
Java JDK, JRE and JVM

What is JVM?

- JVM (Java Virtual Machine) is an **abstract machine that enables your computer to run a Java program.**
- When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates **bytecode** into **native machine code (set of instructions that a computer's CPU executes directly).**

What is JVM?

- Java is a **platform-independent language**. It's because when you write Java code, it's ultimately **written for JVM** but not your physical machine (computer).
- Since JVM executes the Java bytecode which is platform-independent, Java is platform-independent.



Working of Java Program

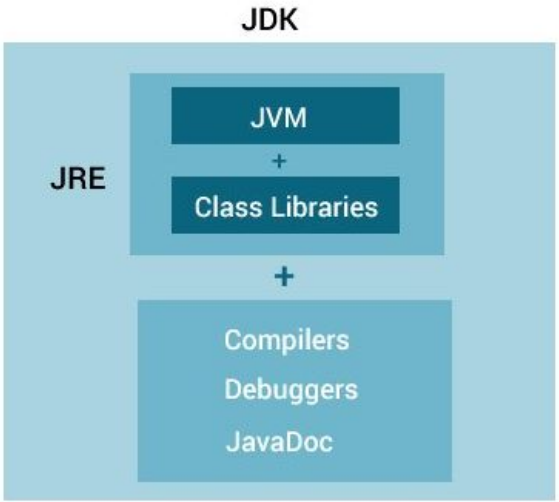
What is JRE?

- JRE (Java Runtime Environment**) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.**
- JRE is the superset of JVM. If you need to run Java programs, but not develop them, JRE is what you need.

What is JDK?

- JDK (Java Development Kit) is a **software development kit required to develop applications in Java.** When you download JDK, JRE is also downloaded with it.
- In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).

Relationship between JVM, JRE, and JDK.

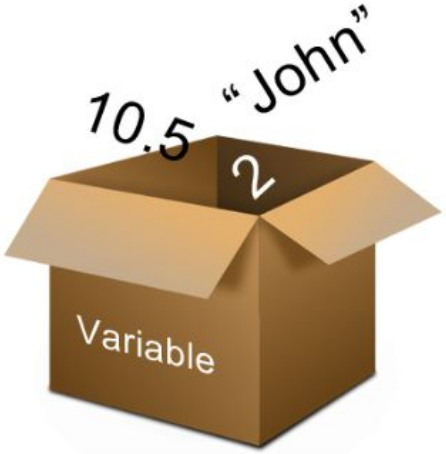


Relationship between JVM, JRE, and JDK

TIA-Academy

Java Variables, Identifiers and Literals

- A variable is a location in memory (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name **(identifier).**



Java Keywords

- Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler. For example:

```
int score;
```

- Here, **int is a keyword**. It indicates that the **variable score is of integer type** (32-bit signed two's complement integer).
- You cannot use keywords like **int, for, class, etc** as variable name (or identifiers) as they are part of the Java programming language syntax.

Java Keywords

Java Keywords List				
abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	goto	if	implements	import
instanceof	int	interface	long	native
new	package	private	protected	public
return	short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	try	void	volatile	while

- Here's the complete list of all keywords in Java programming.
- Beside these keywords, you cannot also use true, false and null as identifiers. It is because they are literals.

Java Identifiers

- Identifiers are the name given to variables, classes, methods, etc. Consider the following code;

```
int score;
```

- Here, score is a variable (an identifier). You **cannot use keywords as variable names**. It's because keywords have predefined meanings. For example,

```
int float;
```

- The above code is wrong. It's because **float is a keyword** and cannot be used as a variable name.

Rules for Naming an Identifier

- Identifiers **cannot be a keyword.**
- Identifiers are **case-sensitive**.
- It can have a sequence of letters and digits. However, it must begin with a letter, **$ or _**. The first letter of an **identifier cannot be a digit**.
- It's a **convention to start an identifier with a letter** rather and $ or _.
- **Whitespaces are not allowed**.
- Similarly, you cannot use symbols such as **@, #, and so on**.

Rules for Naming an Identifier

Here are some valid identifiers:

- score
- level
- highestScore
- number1
- convertToString

Rules for Naming an Identifier

Here are some invalid identifiers:

- class
- float
- 1number
- highest Score
- @pple

Create Variables in Java

- Here's how we create a variable in Java,

```
int speedLimit = 80;
```

- Here, **speedLimit is a variable of int data type** and **we have assigned value 80 to it.**
- The **int data type** suggests that the variable **can only hold integers**.

Create Variables in Java

- In the example, we have assigned value to the variable during declaration. However, it's not mandatory.
- You can declare variables and assign variables separately. For example,

```
int speedLimit;
speedLimit = 80;
```

- Note: Java is a **statically-typed language. It means that all variables must be declared before they can be used.**

Change values of variables

- The value of a variable can be changed in the program, hence the name variable. For example,

```
int speedLimit = 80;
... .. ...
speedLimit = 90;
```

- Here, initially, the value of speedLimit is 80. Later, we changed it to 90.

Change values of variables

- However, we cannot change the **data type of a variable in Java** within the same scope.
- *What is the variable scope?*
- Don't worry about it for now. Just remember that we can't do something like this:

```
int speedLimit = 80;
... .. ...
float speedLimit;
```

Rules for Naming Variables in Java

Java programming language has its own set of rules and conventions for naming variables. Here's what you need to know:

- Java is case sensitive. Hence, age and AGE are two different variables. For example,

```
int age = 24;
int AGE = 25;

System.out.println(age);   // prints 24
System.out.println(AGE);   // prints 25
```

Rules for Naming Variables in Java

- Variables must start with either a letter or an underscore, _ or a dollar, $ sign.

  For example,

```
int age;  // valid name and good practice
int _age;  // valid but bad practice
int $age;  // valid but bad practice
```

- Variable names cannot start with numbers. For example,

```
int 1age;  // invalid variables
```

Rules for Naming Variables in Java

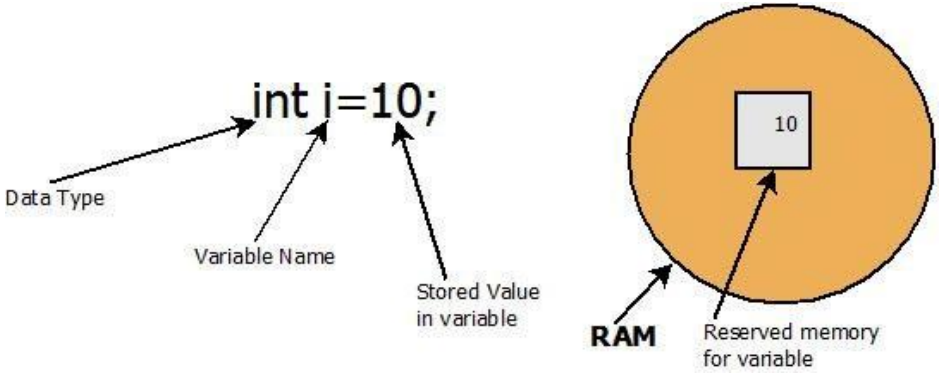
- Variable names can't use whitespace. For example,

```
int my age;  // invalid variables
```

- Here, is we need to use variable names having more than one word, use all lowercase letters for the first word and capitalize the first letter of each subsequent word. For example, myAge.

Rules for Naming Variables in Java

- When creating variables, choose a name that makes sense. For example, score, number, level makes more sense than variable names such as s, n, and l.
- If you choose one-word variable names, use all lowercase letters. For example, it's better to use speed rather than SPEED, or sPEED.



Variables Types

There are 3 types of variables in Java programming language:

1. Instance Variables (Non-Static Fields)
2. Class Variables (Static Fields)
3. Local Variables

```
public class A
{
    int a = 10 ;                    ---------- instance variable
    static int b = 20;              ---------- static variable
    void add ()
    {
        int c = 30,d;               ---------- local variable (d is local)
        d = a+b+c;
        System.out.println(d);
    }
    void mul()
    {
        int e = 40,f;
        f = a*b*e;
        System.out.println(f);
    }
}
```

Static (or class) Variable

- Static variables are also known as **class variable because they are associated with the class and common for all the instances of class**.
- For example, If I create three objects of a class and access this static variable, it would be common for all, **the changes made to the variable using one of the object would reflect when you access it through other objects.**

Static (or class) Variable

```
public class StaticVarExample {
   public static String myClassVar = "class or static variable";
   public static void main(String args[]){
      StaticVarExample obj = new StaticVarExample();
      StaticVarExample obj2 = new StaticVarExample();
      StaticVarExample obj3 = new StaticVarExample();

      //All three will display "class or static variable"
      System.out.println(obj.myClassVar);
      System.out.println(obj2.myClassVar);
      System.out.println(obj3.myClassVar);
```

```
      //changing the value of static variable using obj2
      obj2.myClassVar = "Changed Text";

      //All three will display "Changed Text"
      System.out.println(obj.myClassVar);
      System.out.println(obj2.myClassVar);
      System.out.println(obj3.myClassVar);
   }
}
```

Static (or class) Variable

- Output:

```
class or static variable
class or static variable
class or static variable
Changed Text
Changed Text
Changed Text
```

- As you can see all three statements displayed the same output irrespective of the instance through which it is being accessed. That's is why we can access the static variables without using the objects like this:

```
System.out.println(myClassVar);
```

Instance variable

- **Each instance(objects) of class has its own copy of instance variable.** Unlike static variable, instance variables have their own separate copy of instance variable.
- We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged.
- This shows that they have their own copy of instance variable.

Instance variable

```
public class InstanceVarExample {
   String myInstanceVar="instance variable";

   public static void main(String args[]){
        InstanceVarExample obj = new InstanceVarExample();
        InstanceVarExample obj2 = new InstanceVarExample();
        InstanceVarExample obj3 = new InstanceVarExample();

        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);

        obj2.myInstanceVar = "Changed Text";

        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);
   }
}
```

Instance variable

- Output:

```
instance variable
instance variable
instance variable
instance variable
Changed Text
instance variable
```

Local Variable

- These variables are declared inside method of the class. **Their scope is limited to the method which means that You can't change their values and access them outside of the method.**
- In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

Local Variable

```
public class VariableExample {
   // instance variable
   public String myVar="instance variable";
   public void myMethod(){
      // local variable
      String myVar = "Inside Method";
      System.out.println(myVar);
   }
   public static void main(String args[]){
      // Creating object
      VariableExample obj = new VariableExample();

      /* We are calling the method, that changes the
       * value of myVar. We are displaying myVar again after
       * the method call, to demonstrate that the local
       * variable scope is limited to the method itself.
       */
      System.out.println("Calling Method");
      obj.myMethod();
      System.out.println(obj.myVar);
   }
}
```

Local Variable

- Output:

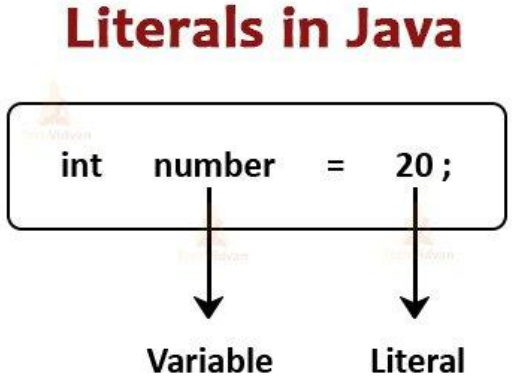
```
Calling Method
Inside Method
instance variable
```

- If not declared the instance variable and only declared the local variable inside method then the statement System.out.println(obj.myVar); would have thrown compilation error. As you cannot change and access local variables outside the method.

Java literals

- Literals are data used for representing fixed values. They can be used directly in the code. For example,

```
int number = 20;
float b = 2.5;
char c = 'F';
```



- Here, **20, 2.5**, and **'F'** are literals.
- Here are different types of literals in Java.

Boolean Literals

- In Java, boolean literals are used to initialize boolean data types. They can store two values: true and false. For example,

```
boolean flag1 = false;
boolean flag2 = true;
```

- Here, **false and true are two boolean literals**.

Integer Literals

An integer literal is a numeric value(associated with numbers) without any fractional or exponential part. There are 4 types of integer literals in Java:

1. binary (base 2)
2. decimal (base 10)
3. octal (base 8)
4. hexadecimal (base 16)

Integer Literals

```
// binary
int binNumber = 0b10010; // 0b represents binary

// octal
int octalNumber = 027;

// decimal
int decNumber = 34;

// hexadecimal
int hexNumber = 0x2F; // 0x represents hexadecimal
```

- In Java, **binary starts with 0b**, **octal starts with 0**, and **hexadecimal starts with 0x**.
- Note: Integer literals are used to initialize variables of integer types like **byte, short, int, and long.**

Floating-point Literals

- A floating-point literal is a numeric literal that has either **a fractional form or an exponential form**. For example,
- Note: The floating-point literals are used to initialize float and double type variables.

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

    double myDouble = 3.4;
    float myFloat = 3.4F;

    // 3.445*10^2
    double myDoubleScientific = 3.445e2;

    System.out.println(myDouble);  // prints 3.4
    System.out.println(myFloat);    // prints 3.4
    System.out.println(myDoubleScientific);   // prints 344.5
  }
}
```

Character Literals

- Character literals are unicode character enclosed inside single quotes. For example,

```
char letter = 'a';
```

- Here, a is the character literal.
- We can also use escape sequences as character literals. For example, \b (backspace), \t (tab), \n (new line), etc.

String literals

- A string literal is a sequence of characters enclosed inside double-quotes. For example,

```
String str1 = "Java Programming";
String str2 = "Programiz";
```

- Here, Java Programming and Programiz are two string literals.

TIA-Academy

ASSIGNMENT 01

TIA-Academy

ASSIGNMENT 02
(HOME ASSIGNMENT)

TIA-Academy

Thank You