**JDK (Java Developer Kit)**

* **Prgrm:**

**Class myFirstProgram**

**{**

**Public static void main (String args[])**

**{**

**System.out.println(“Hello World”);**

**}**

**}**

* **Class** is keyword.
* Then **name of class**.
* Name of class, variables and functions should be similar and appropriate. Names – per situation
* **{** : starting base
* **}** : ending base
* **{}** : boundary of class
* No semi – colon after class
* Main function is in boundary of class. In Java, everything should be written inside of class boundary **except 2 things:**

1. Inheritance Relationship
2. Packages(like namespace) declaration

* **main**: control entry point. Execution starts from here. Order of main function doesn’t matter. U can write it above or below but it should be inside the class boundary. In C++, main function was written below the class.
* **void:** Return type of main. Main function will return nothing.
* **public static**
* **public:** Access specifier. Can be accessed from anywhere
* **static:**

1. Do not have any association with object. They are associated with the class. Can be accessed without creating the object of class.
2. All objects share one copy of static.

* The main function can be called without creating object of the class.
* Parameter **(string args[]):** Array is used to accept **command line arguments** ( arguments that will be passed at run time, at execution time). Those arguments will be saved in this array.
* Body of function:
* Statement: **System.out.println(“Hello World”);**

This statement is equivalent to **cout <<**

**Cout ?? –** Instance (object) of console output stream

* **System:** class of Java. Java has itself provided written library.
* **out:** It is **static** variable in **System** class. out is an instance, an object.

Class System { static printStream out; }

out is declared static in this class, so it can be accessed with class name.

className.staticVariableName.

static variable – no link with object.

* **println**: It is a function of **printStream** class.

**class System**

**{**

**Static printStream out;**

**}**

**Class printStream**

**{**

**println()**

**{**

**}**

**}**

**System.out** (className.variableName)

**out.println** (objectName.functionName)

**out is an instance**

System.out.println();

**Stream:**

It is like a path, channel. It will read/ write data at endpoint from where we will connect it.

By default, stream is connected with console. So data will be write on console.

**Cout** – Not a function. Instance operator was overloaded in OOP.

* How did system class came into our scope? Where is System class written?
* By default, java will add a package in ur every java prgrm and that package is named as **java.lang.\*;**

Lang stands for language.

U don’t need to add this package specifically.

**System class is part of java.lang.\* package.**

* **Adding Package in Java:**

There is no #include in java. To import a package we use **import** keyword.

Java.io (I/P, O/P)

Java.net

Java.socket

Java.sql (DB activity0

Java.utils.\*

* **Why main function is public?**

So that it can be accessed from anywhere (from outside the class).

* **Why static?**

So that it can be called without creating instance of class.

* **Naming of java file:**

Name of file must be same as name of class.

What if there are multiple class?

1. Name of file must be same as name of class.
2. In case of multiple classes (and none of the class is public), name of file can be the name of any class.
3. In case of multiple classes (and one class is public), name of the file must be same as name of public class.
4. In case of multiple classes (and more than one class are public), what will be the scenario? HW

- While defining multiple classes in a single Java file you need to make sure that only one class among them is public. If you have more than one public classes in a single file a compile-time error will be generated.

* **Compilation Of Code:**

**javac myFirstProg.java**

* **javac:** java compilation
* Then space
* Then full name of file with extension
* Then enter

It will check if there is any syntax error. If there will be any syntax error in the file, it will tell the error. Otherwise, it will generate a special type of code and that will not be executable.

Special type of code**: Byte Code**

Main feature of Java – **To achieve platform independence** (byte code)

* **JAVA is a platform-independent language, the JVM is platform-dependent.**
* **After Compilation:**
* **java myFirstProg** – Command to run / execute byte code
* Then enter
* Hello World will be printed.
* **Byte Code? How it provides platform independence?**

Compile

Machine Code

C++ Source Code

**In C++,** when we compile source code, we get special code (binary, machine, executable code).

Why it is called **machine code?**

Because it is directly translated to underline CPU. It is generated for that on which it was compiled.

If we will try to run this machine code on linux, max, it will not run there. It will only run on underline CPU (for which it was executed). If we want to run on some other machine, then we need to take the source code on that machine, compile it and again generate the machine code.

**In Java,** when we compile source code, a special type of code known as byte code will be generated.

**Byte Code** can run on Win, Linux, Unix, mac etc. It can run on any OS with just one constraint of only one software and that is **JVM (Java Virtual Machine).**

We can run it on any machine just there should be JVM (compatible to that machine) there.

Byte code is being generated specific to a machine.

JVM is being generated for specific software.

**JVM**

**Byte Code**

compile

**Java Source Code**

**JVM** – Sub part of **JDK**

All libraries etc are already available in JVM**.**

Java provides **platform independence** feature and can be runned on any machine just with **compatible of JVM.**

Java provides JVM specific to every OS. If JVM is available, then we can run code on machine directly.

C – We need to take source code again and again on other machine and then we need to compile it.

Java – Just use byte code. No need to compile. Other machine just needs to have JVM. Directly execute.

**Q:** We are on one machine. Machine will not be changed. C++, java approach. Which is better?

**Drawback in Java Approach:**

In C++, when machine code generated, it is directly executed.

In java, byte code will go to JVM. JVM will interpret it. Then machine code. It is getting further converted to machine code from byte code. There is gap. It is **taking time.**

To capture performance limiter, javauses **JIT (Just In Time) compiler.**

**HW: JIT Compiler, JRE (java runtime environment)**

* **5 Stages:**

Java code goes through these five stages from writing the code to its execution.

1. Edit
2. Compile
3. Load
4. Verify
5. Execute
6. **Edit:**

Program is in editing phase until we save it.

1. **Compile:**

When we give command of javac, it is in compilation phase.

After compilation, it will generate byte code. Byte code will go and be saved on hard drive (sec-drive).

Until bytecode doesn’t go to hard drive, we are in compilation phase

When we give command to execute javac, then 3 phases come

1. **Load:**

Load into RAM.

JVM loads a service **class loader** into main memory.

1. **Verify:**

Then JVM services **byte code verifier.**

It can happen that u are using online byte code and it can happen to delete everything like maybe u are using some 3rd party application. It can be a virus. It is verified in this phase.

1. **Execute:**

If there is no issue, there is no virus, then it will generate a service named **interpreter (it interprets byte code to machine code).**

When byte code is generated, it just looks at byte code. It doesn’t look at source code.

* In java, when we compile a source code, we get a byte code for each class. If there are 4 classes in a source code, we will get 4 byte codes. One separate byte code for each class. Source code generates byte code and then execution from byte code.
* **4 Access Specifiers In Java:**

1. Package **(by default)**
2. Public
3. Private
4. Protect

**Package** – current folder (folder in which file is)

**Public** – Outside the folder access