**What is Java**

**Section1, Lecture1**

Java is a programming language first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed. Java is fast, secure and reliable. It is one of the most used programming languages.

**Java Version History**

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta 1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)
7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

**Features of Java**

There is given many features of Java. The Java features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed
13. **Simple**

Java is easy to learn and its syntax is quite simple, clean and easy to understand. The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

E.g.- Pointers and Operator overloading are not there in Java but they are important part of C++.

1. **Object-Oriented**

In Java, everything is Object which has some data and behavior. Java can be easily extended as it is based on Object Model.

1. **Portable**

Java byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types.

1. **Platform independent**

Unlike other programming languages such as C, C++ etc. which are compiled in to platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this byte code format also provide security. Any machine with Java Runtime Environment can run Java Programs.

1. **Secure**

When it comes to security, Java is always the first choice. With java secure features it enables us to develop virus free, temper free system. Java program always run in Java runtime environment with almost null interaction OS, hence it is more secure.

1. **Robust**

Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled exceptions by introducing automatic ***Garbage Collector*** and ***Exception Handling***.

1. **Architecture neutral**

Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a java program is easy to interpret on any machine.

1. **High Performance**

Java is an interpreted language, so it will never be as fast as complied language like C or C++. But Java enables high performance with the use of just-in-time compiler.

1. **Multithreaded**

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn’t occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications.

1. **Distributed**

We can create distributed applications in Java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

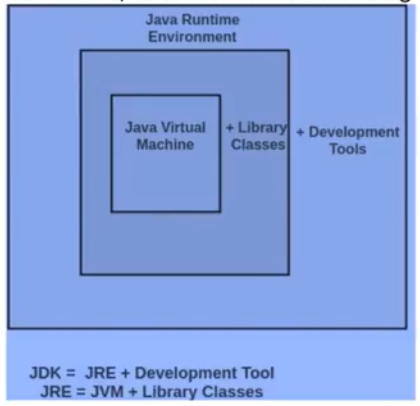
**JDK, JRE and JVM**

**Section1, Lecture2**

* **JDK – Java Development Kit** is kit which provides the environment to ***develop and execute (run)*** the Java program. JDK is a kit (or package) which includes two things –
* **JRE – Java Runtime Environment** is an installation package which provides environment to ***only run (not develop)*** the java program (or application) onto your machine. JRE is only used by them who only wants to run the Java programs i.e. end users of your system.
* **JVM – Java Virtual Machine** is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever java program you run using JRE or JDK goes into JVM, and JVM is responsible for ***executing the Java program line by line*** hence it is also known as interpreter.

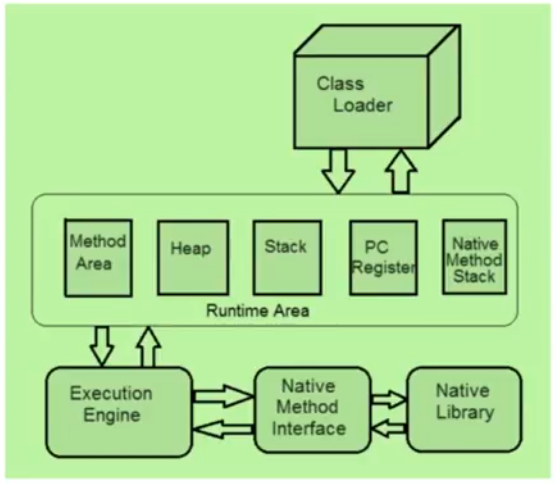
**Note:** JDK is only used by Java developers.

* **Difference between JDK, JRE and JVM.** To understand the difference between these three, let us consider the following diagram.



**JVM Architecture**

**Section1, Lecture3**



1. **Method Area:**

* Java Virtual machine method area can be used for storing all the class code and method code.
* All classes bytecode is loaded and stored in this runtime area, and all static variables are created in this area.

1. **Heap Memory:**

* JVM Heap Area can be used for storing all the objects that are created.
* It is the main memory of JVM, all objects of classes:- non static variables memory are created in this runtime area.
* This runtime area memory is finite memory.
* This area can be configured at the time of setting up of runtime environment using nonstandard option like

Java-xms <size> class-name

* This can be expandable by its own, depending upon the object creation.
* Method Area and the Heap area both are shareable memory areas.

1. **Java Stack Area:**

* JVM stack area can be used for storing the information of the methods. That is under execution. The java stack can be considered as the combination of stack frames where every frame will contain the state of a single method.
* In this runtime area all java methods are executed.
* In this runtime JVM by default creates two threads, they are –
* Main method
* Garbage Collection thread

Main thread responsible to execute java methods start with main method.

Also, responsible to create objects in heap area if it finds ***new*** keyword in any method logic.

Garbage collection thread is responsible to destroy all unused objects from heap area.

1. **PC Register (program counter) area:**

The PC register JVM will contain address of the next instruction that have to be executed.

1. **Java Native stack:**

Java native stack area is used for storing non-java coding available in the application. The non-java code is called as native code.

1. Execution Engine:

* The execution engine of JVM is responsible for executing the program and it contains two parts.

1. Interpreter
2. JIT Compiler (Just in Time Compiler)

* The Java code will be executed by both interpreter and JIT compiler simultaneously which will reduce the execution time and then by providing high performance. The code that is executed by JIT complier is called as HOTSPOTS (company).

**Variables and Data Types in Java**

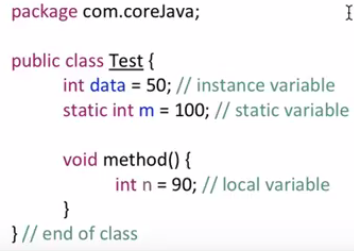
**Section1, Lecture4**

Variable is the name of memory location. There are three types of variables in java: local, instance and static.

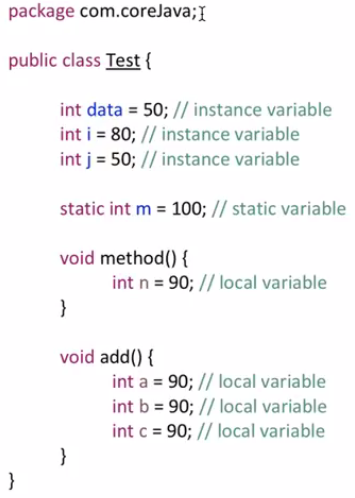
There are two types of data types in java: primitive and non-primitive.

***Types of Variables:-*** There are three types of variables in java –

1. **Local variable –** A variable which is declared inside the method is called local variable.
2. **Instance variable –** A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static.
3. **Static variable –** A variable that is declared as static is called static variable. It cannot be local.

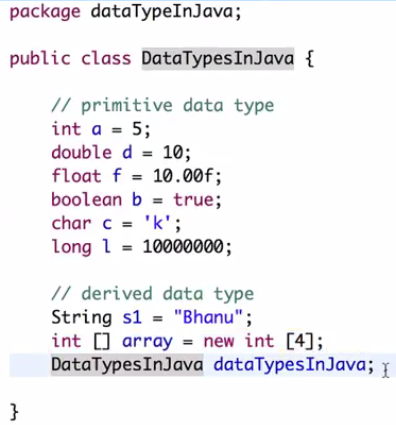


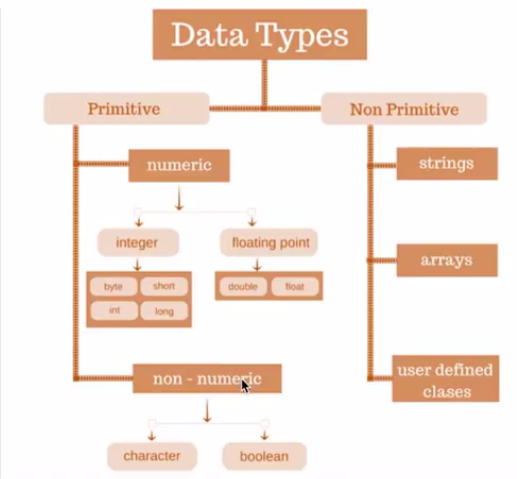
---------------------------------------------------------------

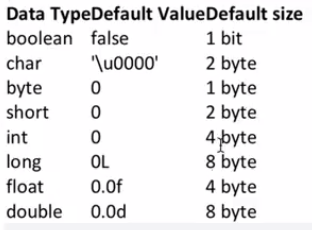


**Data Types in Java –** Data types represent the different values to be stored in a variable. In Java, there are two types of data types:

* Primitive Data types
* Non-primitive Data types







**Increment and Decrement operators in Java**

**Section2, Lecture5**

**Syntax:**

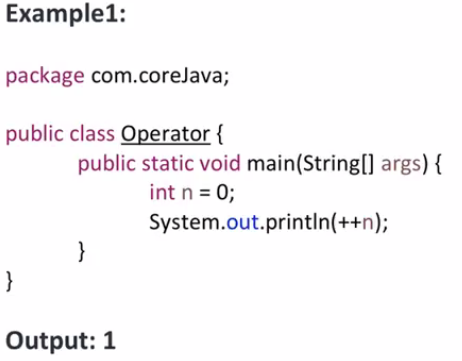
++ Increment operator: increments a value by 1.

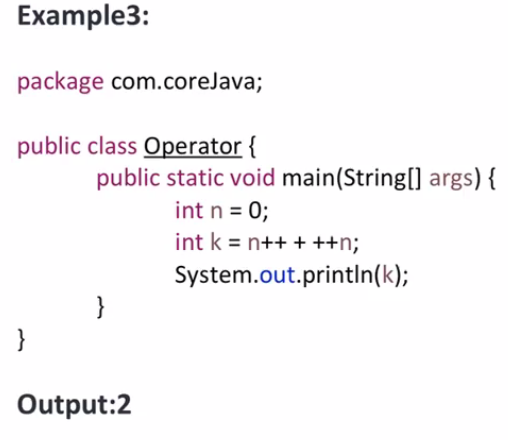
-- Decrement operator: decrements a value by 1.

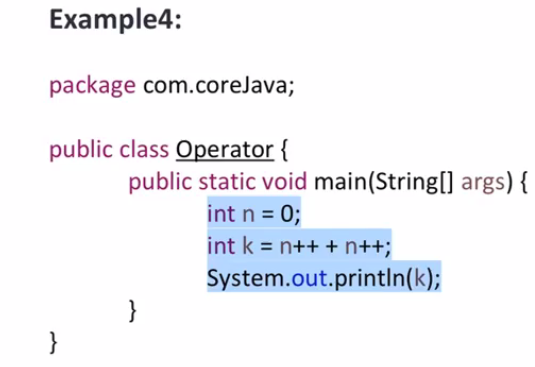
**What is Pre and Post Increment?**

Pre-Increment: First increase value of variable then use in program.

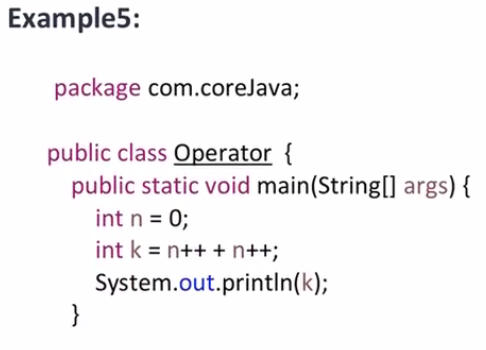
Post Increment: First use the value in program then increase.



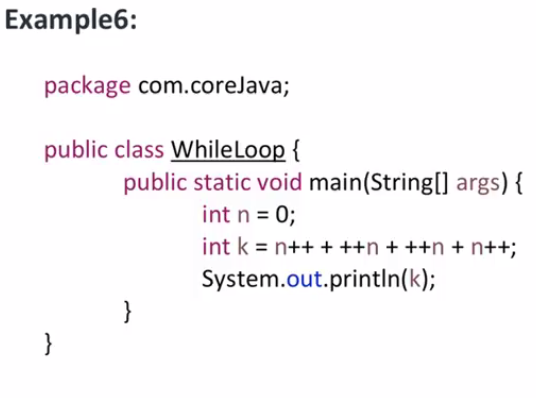




**Output: 1**



**Output: 1**



**Output: 8**

**Control Statements in Core Java**

**Java If Statement**

**Section3, Lecture6**