Unit – 1
Basic of JAVA

Content

- Features of Java
- Byte Code and Java Virtual Machine
- JDK
- Data types
- Operator
- Control Statements – If , else, nested if, if-else ladders, Switch, while, do-while, for, for-each, break, continue.

What is Java

- Java is a high level, robust, object-oriented and secure programming language.
- Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).
- The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: J2EE for Enterprise Applications, J2ME for Mobile Applications.
- The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be **Write Once, Run Anywhere.**

- **Application**
- According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:
  - Desktop Applications such as acrobat reader, media player, antivirus, etc.
  - Web Applications such as irctc.co.in, javatpoint.com, etc.
  - Enterprise Applications such as banking applications.
  - Mobile
  - Embedded System
  - Smart Card
  - Robotics
  - Games, etc.

- **Standalone Application**
  - Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.
- **Web Application**
  - An application that runs on the server side and creates a dynamic page is called a web application.
    Currently, Servlet, JSP Struts, Spring, Hibernate etc. technologies are used for creating web applications in Java.
- **Enterprise Application**
  - An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.
- **Mobile Application**
  - An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

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

- Java ME (Java Micro Edition)
  - It is a micro platform which is mainly used to develop mobile applications.

- JavaFX
  - It is used to develop rich internet applications. It uses a light-weight user interface API.

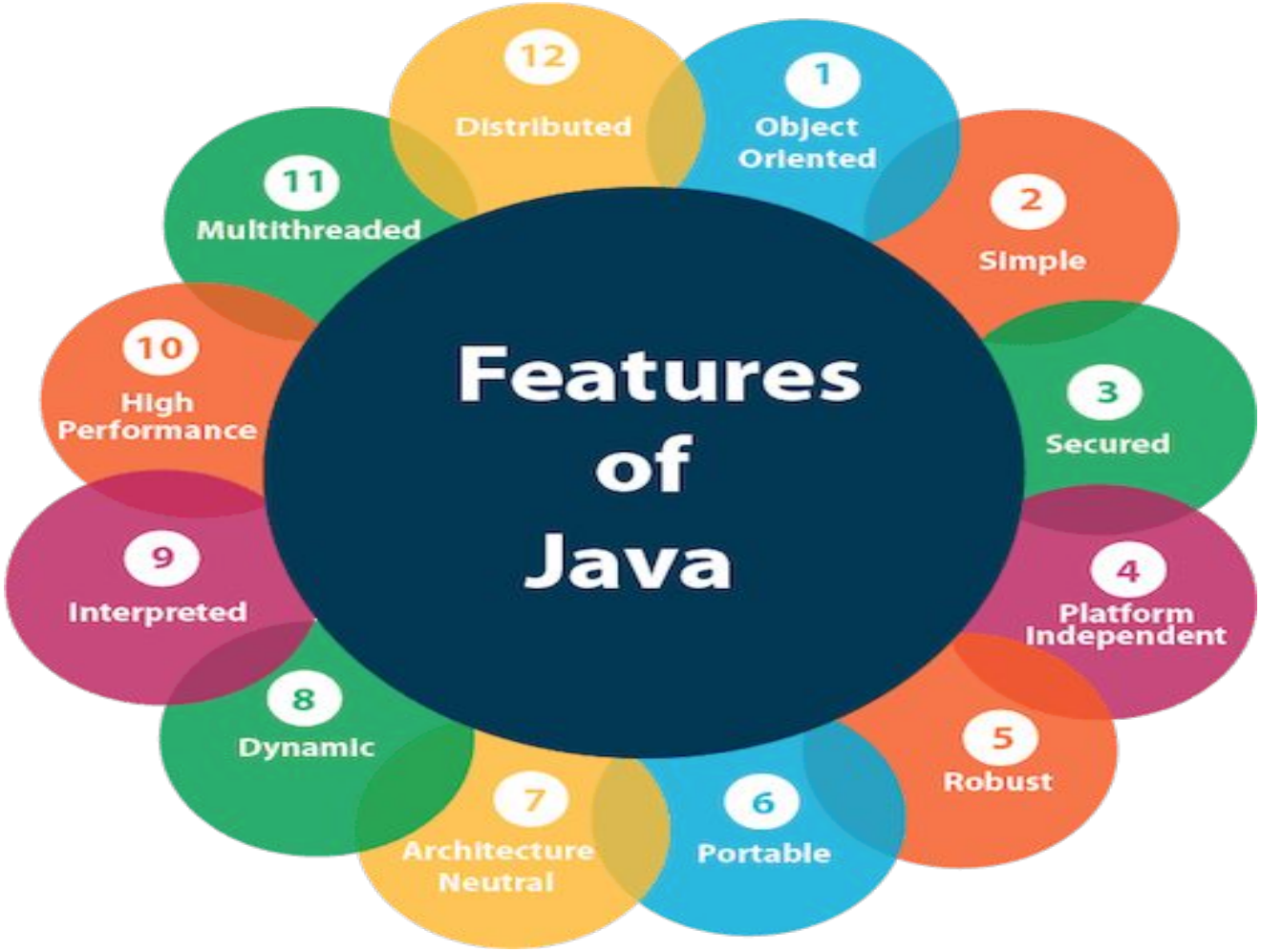
History of Java

- James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.
- Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.
- On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).
- On 8 May, 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Tools You Will Need

- For performing the examples discussed in this tutorial, you will need a Pentium 200-MHz computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).
- You will also need the following softwares –
  - Linux 7.1 or Windows xp/7/8 or above operating system
  - Java JDK
  - Microsoft Notepad or Netbeans

Features of Java



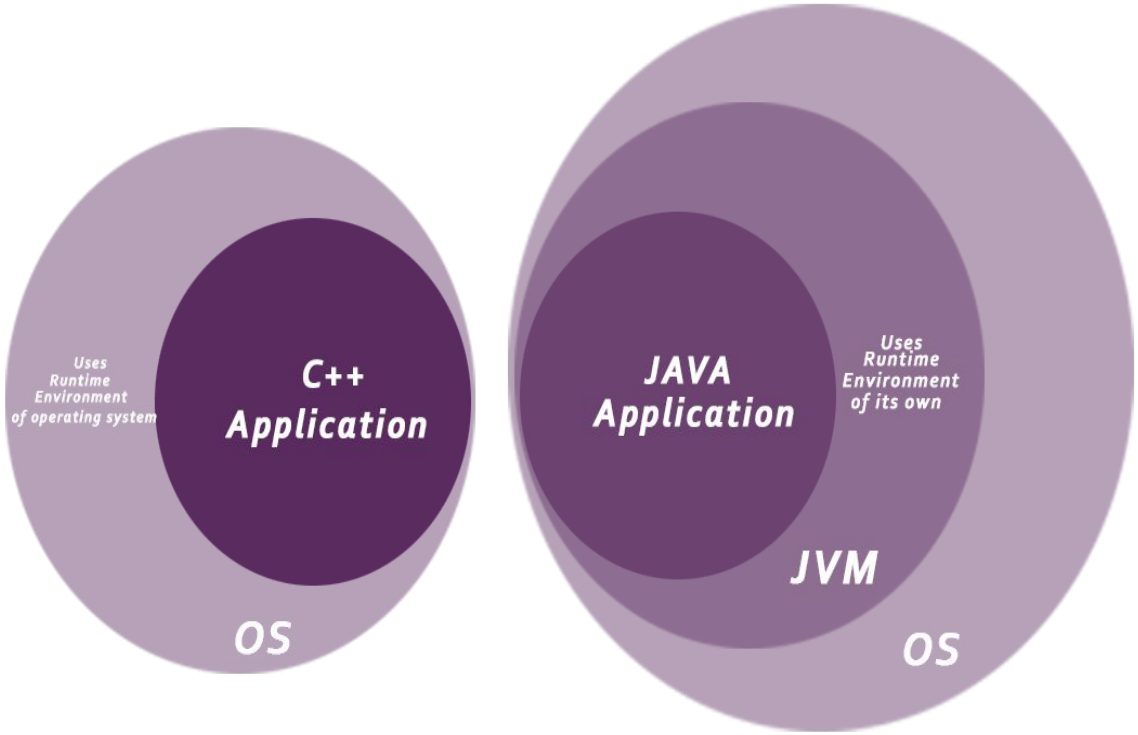
- Java is –
- **Simple** :
  - Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
  - Java syntax is based on C++ (so easier for programmers to learn it after C++).
  - Java has removed many complicated and rarely-used features, for example, explicit pointers etc.
  - There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

- **Object-oriented**
  - Java is an object oriented programming language.
  - Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
- Basic concepts of OOPs are:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

- **Platform Independent**
  - Java is platform independent because it is different from other languages like C, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language.
  - A platform is the hardware or software environment in which a program runs.
  - Java provides a software-based platform.
- It has two components:
  - Runtime Environment
  - API(Application Programming Interface)

  - Java code can be run on multiple platforms, for example, Windows, Linux, Mac/OS, etc.
  - Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., **Write Once and Run Anywhere(WORA)**

- **Secured**
  - Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
  - **No explicit pointer**
  - **Java Programs run inside a virtual machine sandbox**



- **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

- **Robust**
  - Robust simply means strong. Java is robust because:
    - It uses strong memory management.
    - There is a lack of pointers that avoids security problems.
    - There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
    - There are exception handling and the type checking mechanism in Java. All these points make Java robust.

- **Architecture-neutral**
  - Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
  - In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.
- **Portable**
  - Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

- **High-performance**
  - Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

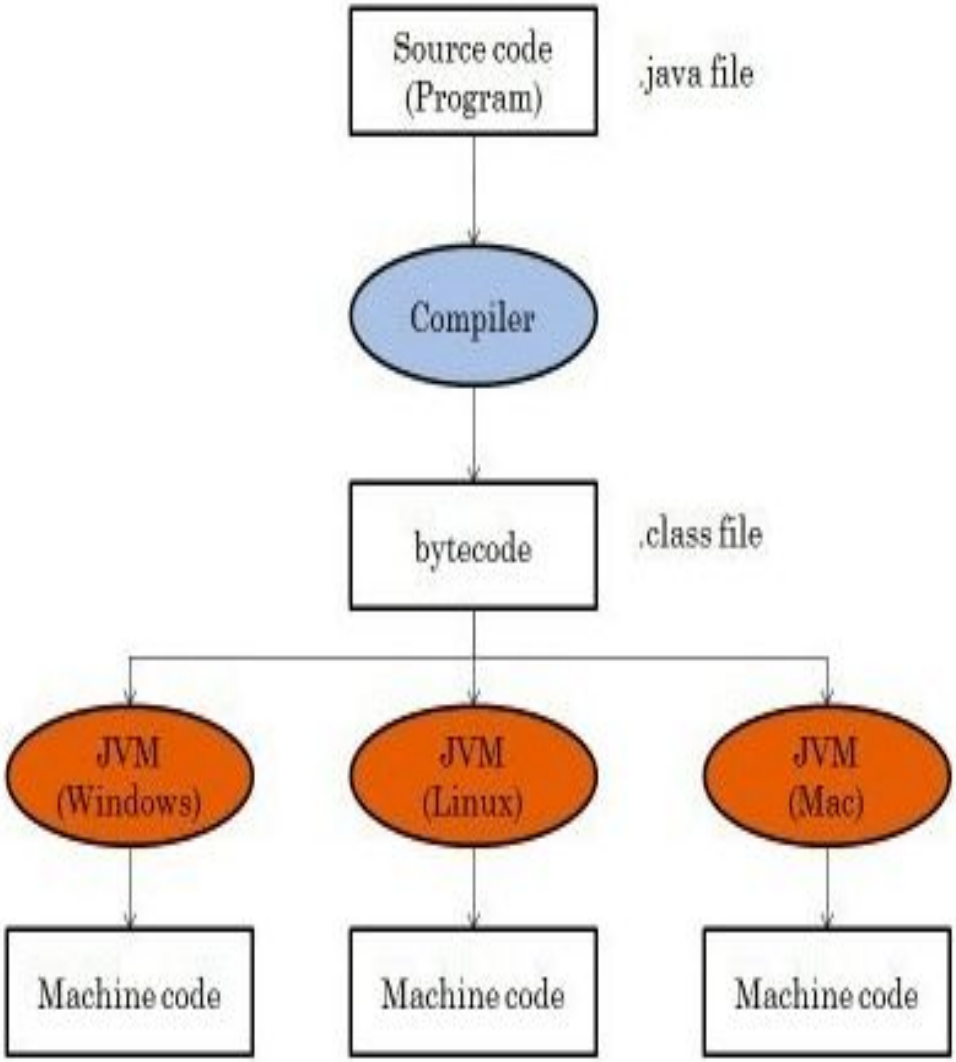
- **Distributed**
  - Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

- **Multi-threaded**
  - 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, etc.

- **Dynamic**
  - Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand.
  - Java supports dynamic compilation and automatic memory management (garbage collection).

What is Java Bytecode

- Java bytecode is the instruction set for the Java Virtual Machine. As soon as a java program is compiled, java bytecode is generated.
- java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.
- Bytecode is the compiled format for Java programs. Once a Java program has been converted to bytecode, it can be transferred across a network and executed by Java Virtual Machine (JVM). Bytecode files generally have a .class extension

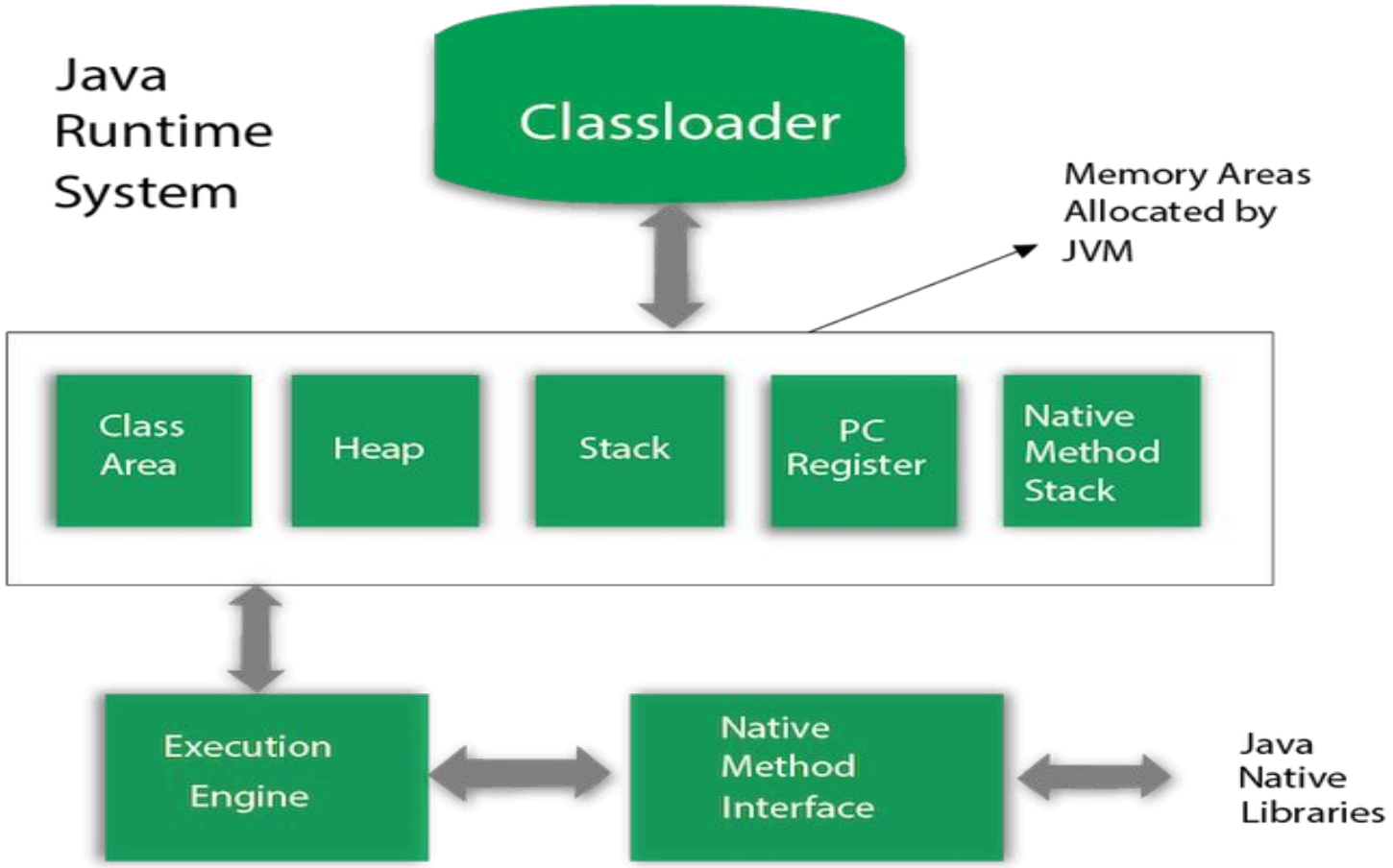

Source code
(Program)
.java file
Compiler
bytecode
.class file
JVM
(Windows)
JVM
(Linux)
JVM
(Mac)
Machine code
Machine code
Machine code

Java Virtual Machine

- JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
- JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).
  - **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle
  - **An implementation** Its implementation is known as JRE (Java Runtime Environment).
  - **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created

- The JVM performs following operation:
  - Loads code
  - Verifies code
  - Executes code
  - Provides runtime environment
- JVM provides definitions for the:
  - Memory area
  - Class file format
  - Register set
  - Garbage-collected heap
  - Error reporting

JVM Architecture



Classloader

- Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

  - **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
  - **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
  - **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

- Class(Method) Area
  - Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.
- Heap
  - It is the runtime data area in which objects are allocated.
- Stack
  - Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.
  - Each thread has a private JVM stack, created at the same time as thread.
  - A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
- Program Counter Register
  - PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

- Execution Engine
- It contains:
  - **A virtual processor**
  - **Interpreter:** Read bytecode stream then execute the instructions.
  - **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

- Java Native Interface

  - Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

JDK: Java Development Kit

- JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop java applications and applets. It physically exists. It contains JRE + development tools.
- JDK is an implementation of any one of the below given Java Platforms released by Oracle corporation:
  - Standard Edition Java Platform
  - Enterprise Edition Java Platform
  - Micro Edition Java Platform
- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) etc. to complete the development of a Java Application.

Java Variables

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.
- Variable is a 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.
- Example: **int** data=50;//Here data is variable

- Types of Variables
  - local variable
  - instance variable
  - static variable

- Local Variable
  - A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

- Instance Variable
  - A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
  - It is called instance variable because its value is instance specific and is not shared among instances.

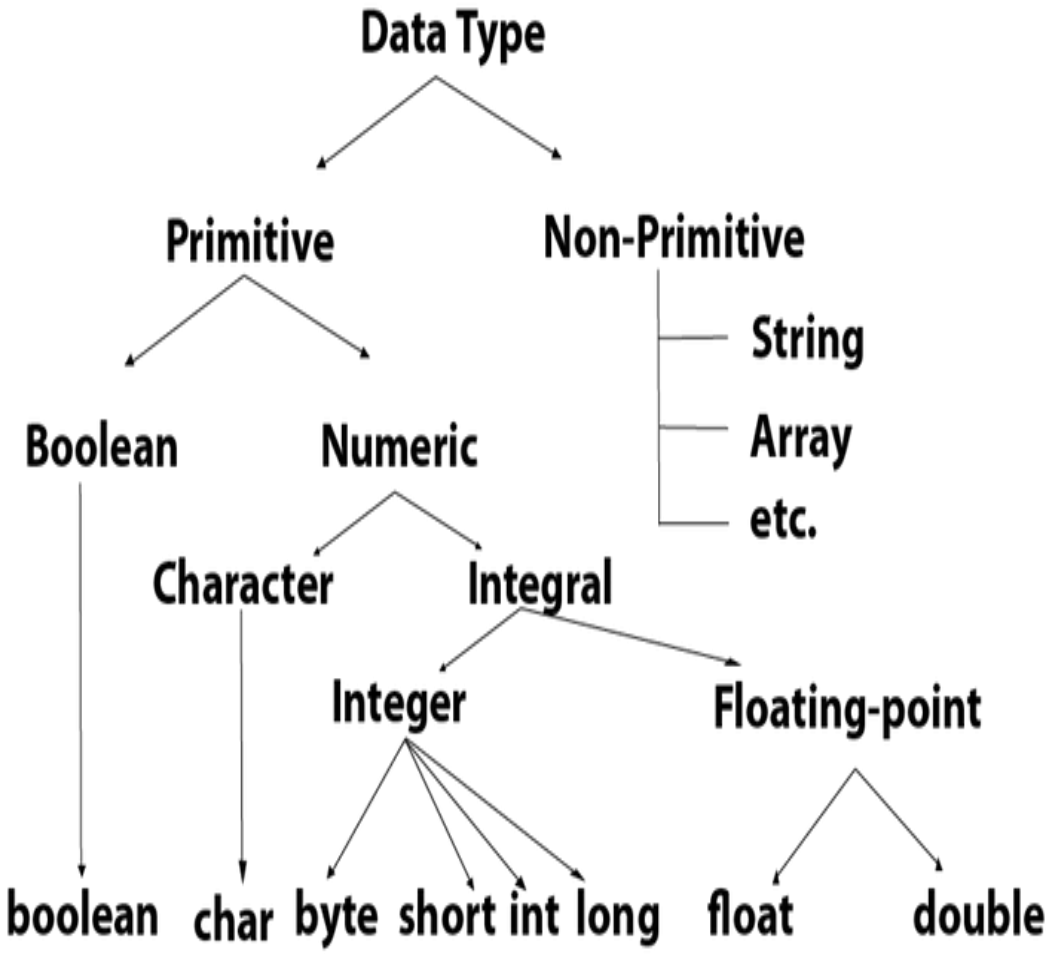
- Static variable
  - A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

- Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:
- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays



- **Boolean Data Type**
  - The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.
  - The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.
  - **Example:** Boolean one = false

- **Byte Data Type**
  - The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.
  - The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.
  - **Example:** byte a = 10, byte b = -20

- Short Data Type
  - The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
  - The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.
  - **Example:** short s = 10000, short r = -5000
- Int Data Type
  - The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.
  - The int data type is generally used as a default data type for integral values unless if there is no problem about memory.
  - **Example:** int a = 100000, int b = -200000

- Long Data Type
  - The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive).
  - Its default value is 0. The long data type is used when you need a range of values more than those provided by int.
  - **Example:** long a = 100000L, long b = -200000L
- Float Data Type
  - The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.
  - **Example:** float f1 = 234.5f
- Double Data Type
  - The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.
  - **Example:** double d1 = 12.3

- Char Data Type
- The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.
- **Example:** char letterA = 'A'

Operators in Java

- **Operator** in Java is a symbol which is used to perform operations. For example: +, -, *, / etc.

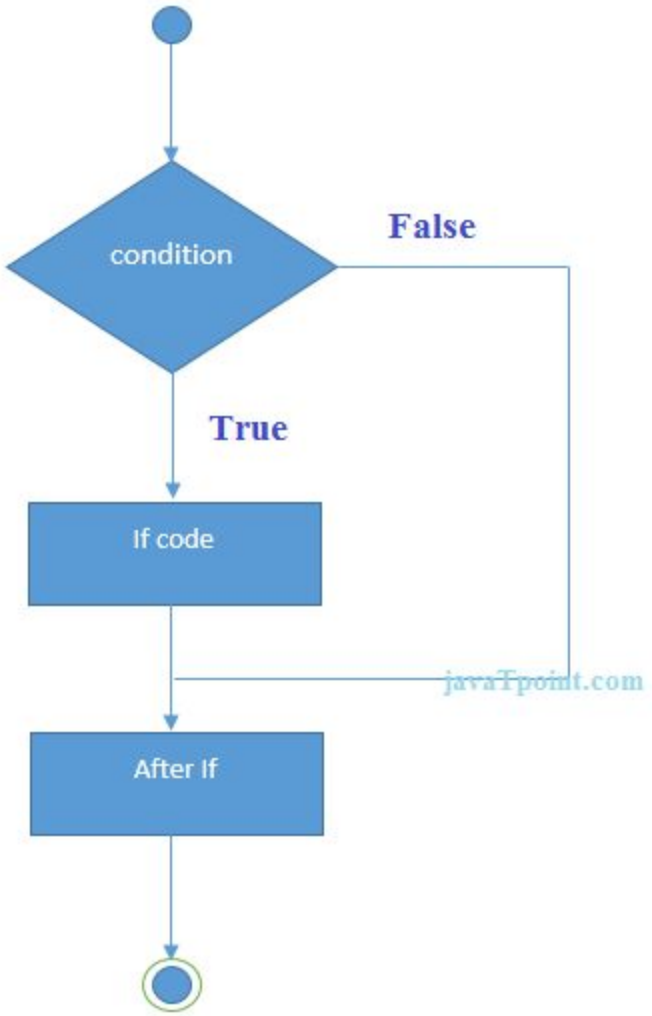
  - Unary Operator,
  - Arithmetic Operator,
  - Shift Operator,
  - Relational Operator,
  - Bitwise Operator,
  - Logical Operator,
  - Ternary Operator,
  - Assignment Operator.

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java If-else Statement

- The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in Java.
  - if statement
  - if-else statement
  - if-else-if ladder
  - nested if statement

- **Java if Statement**
- The Java if statement tests the condition. It executes the *if block* if condition is true.




```
if(condition){
//code to be executed
}
```

```
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");

   }
}
}
```

- **Java if-else Statement**
- The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

```
if(condition){
//code if condition is true
}
else{
//code if condition is false
}
```

```
public class IfElseExample {

public static void main(String[] args) {

   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not

   if(number%2==0){
      System.out.println("even number");
   }
   else{
      System.out.println("odd number");
   }
}
}
```

Example: Check given number is Leap Year or not

- **Java if-else-if ladder Statement**
- The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true

}else if(condition2){
//code to be executed if condition2 is true

}
else if(condition3){
//code to be executed if condition3 is true

}
...
else{
//code to be executed if all the conditions
    are false
}
```

```
public class IfElseIfExample {
public static void main(String[] args) {
   int marks=65;

   if(marks<60){
      System.out.println("fail");
   }

   else if(marks>=60 && marks<70){
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");
   }else if(marks>=90 && marks<100){
      System.out.println("A+ grade");
   }else{
      System.out.println("Invalid!");
   }
}
}
```

Write down Program to check POSITIVE, NEGATIVE or ZERO:

- **Java Nested if statement**
- The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed

    }
}
```

```
public class JavaNestedIfExample {
public static void main(String[] args) {
   //Creating two variables for age and w
eight
   int age=20;
   int weight=80;
   //applying condition on age and weigh
t
   if(age>=18){
       System.out.println(“Age”+age);
      if(weight>50){
         System.out.println("You are eligib
le to donate blood");
      }
   }
}}
```

- Switch
- The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.
- The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long.

- **Points to Remember**
  - There can be *one or N number of case values* for a switch expression.
  - The case value must be of switch expression type only. The case value must be *literal or constant*.
  - Each case statement can have a ***break*** *statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
  - The case value can have a *default label* which is option

- **Syntax**

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all
    cases are not matched;

}
```

```
public class SwitchExample {
public static void main(String[] args) {
   //Declaring a variable for switch expression
   int number=20;
   //Switch expression
   switch(number){
   //Case statements
   case 10: System.out.println("10");
   break;
   case 20: System.out.println("20");
   break;
   case 30: System.out.println("30");
   break;
   //Default case statement
   default:System.out.println("Not in 10, 20 or
30");
   }
}
}
```

- Program to check Vowel or Consonant: If the character is A, E, I, O, or U, it is vowel otherwise consonant
- Program printing month name for the given number

- Loops in Java
  - In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true. There are three types of loops in Java.
    - for loop
    - while loop
    - do-while loop

- Java For Loop
  - The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.
  - There are three types of for loops in java.
    - Simple For Loop
    - For-each or Enhanced For Loop
    - Labeled For Loop

Java Simple For Loop

- A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

- **Initialization**: It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
- **Condition**: It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
- **Statement**: The statement of the loop is executed each time until the second condition is false.
- **Increment/Decrement**: It increments or decrements the variable value. It is an optional condition.

Syntax:

```
for(initialization;condition;incr/decr){
//statement or code to be executed
}
```

```
public class ForExample {
public static void main(Strin
    g[] args) {
    //Code of Java for loop

    for(int i = 1;i<=10;i++){
        System.out.println(i);
    }
}
}
```

```
public class NestedForExample {

public static void main(String[] a
rgs) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);

}//end of j
}//end of i
}
}
```

```
i=1: 3
i=2: 3
```

- Java for-each Loop
  - The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.
  - It works on elements basis not index. It returns element one by one in the defined variable.
- **Syntax:**

```
for(Type var:array){
//code to be executed
}
```

```
public class ForEachExample {
public static void main(String[] args) {
    //Declaring an array
    int arr[]={12,23,44,56,78};
    //Printing array using for-each loop
    for(int i:arr){
        System.out.println(i);
    }
}
}
```

- Java Labeled For Loop
  - We can have a name of each Java for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.
  - Usually, break and continue keywords breaks/continues the innermost for loop only.
- **Syntax:**

```
labelname:
for(initialization;condition;incr/decr){
//code to be executed
}
```

Example:

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample {
public static void main(String[] args) {
   //Using Label for outer and for loop
aa:
    for(int i=1;i<=3;i++){
      bb:
        for(int j=1;j<=3;j++){
          if(i==2&&j==2){
            break bb;
          }
          System.out.println(i+" "+j);
        }
    }
}
}
```

Java Infinitive For Loop

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Syntax:

```
for(;;){
//code to be executed
}
```

Example:

```
//Java program to demonstrate the use of infinite for loop
//which prints an statement
public class ForExample {
public static void main(String[] args) {
   //Using no condition in for loop
   for(;;){
      System.out.println("infinitive loop");
   }
}
}
```

- Java While Loop
  - The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

```
while(condition){
//code to be executed
}
```

```
public class WhileExample {
public static void main(String[] args
) {
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   }
}
}
```

- **Java Infinitive While Loop**
  - If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

```
while(true){
//code to be executed
}
```

Example:

```
public class WhileExample2 {
public static void main(String[] args) {
   while(true){
      System.out.println("infinitive while loop");
   }
}
}
```

- Java do-while Loop
  - The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.
  - The Java *do-while loop* is executed at least once because condition is checked after loop body.
- **Syntax:**

```
do{
//code to be executed
}while(condition);
```

```
public class DoWhileExample {
public static void main(String[] args
) {
   int i=1;
   do{
      System.out.println(i);
   i++;
   }while(i<=10);
}
}
```

- Java Break Statement
  - When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
  - The Java *break* statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.
  - We can use Java break statement in all types of loops such as for lop, while loop and do-while loop.

```
public class BreakExample {
public static void main(String[] arg
    s) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   }
}
}
```

```
public class BreakWhileExample {
public static void main(String[] args
) {
   //while loop
   int i=1;
   while(i<=10){
      if(i==5){
         //using break statement
         i++;
         break;//it will break the loop

      }
      System.out.println(i);
      i++;
   }
}
}
```

- Java Continue Statement
  - The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
  - The Java *continue statement* is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only.
  - We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

```
public class ContinueExample {

public static void main(String[]
     args) {
   //for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //using continue statem
    ent
         continue;
      }
      System.out.println(i);
   }
}
}
```

```
public class ContinueWhileExample {

public static void main(String[] args)
{
   //while loop
   int i=1;
   while(i<=10){
      if(i==5){
         //using continue statement
         i++;
         continue;//it will skip the rest
statement
      }
      System.out.println(i);
      i++;
   }
}
}
```

- Java Comments
  - The Java comments are the statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code.
  - Types of Java Comments
    - Single Line Comment
    - Multi Line Comment
    - Documentation Comment

1) Java Single Line Comment

The single line comment is used to comment only one line.

Syntax:

```
//This is single line comment
```

Example:

```
public class CommentExample1 {
public static void main(String[] args) {
   int i=10;//Here, i is a variable
   System.out.println(i);
}
}
```

2) Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

```
/*
This
is
multi line
comment
*/
```

Example:

```
public class CommentExample2 {
public static void main(String[] arg
s) {
/* Let's declare and
 print variable in java. */
   int i=10;
   System.out.println(i);
}
}
```

Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

```
/**
This
is
documentation
comment
*/
```

Example:

```
/** The Calculator class provides methods to get addition and subtraction of given 2 nu
    mbers.*/
public class Calculator {
/** The add() method returns addition of given numbers.*/
public static int add(int a, int b){return a+b;}
/** The sub() method returns subtraction of given numbers.*/
public static int sub(int a, int b){return a-b;}
}
```

- Compile it by javac tool:

  javac Calculator.java
- Create Documentation API by javadoc tool:

  javadoc Calculator.java

- **Understanding public static void main(String[] args) in Java**
- n Java programs, the point from where the program starts its execution or simply the entry point of Java programs is the **main()** method. Hence, it is one of the most important methods of Java and having proper understanding of it is very important.
- **Most common syntax of main() method:**

```
class Demo {
   public static void main(String[] args)
   {
      System.out.println(“Good Morning");
   }
}
```

- **Public:** It is an *Access modifier*, which specifies from where and who can access the method. Making the *main()* method public makes it globally available. It is made public so that JVM can invoke it from outside the class as it is not present in the current class.
- **Static:** It is a *keyword* which is when associated with a method, makes it a class related method. The *main()* method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the *main()* method by the JVM.
- **Void:** It is a keyword and used to specify that a method doesn't return anything. As *main()* method doesn't return anything, its return type is *void*. As soon as the *main()* method terminates, the java program terminates too. Hence, it doesn't make any sense to return from *main()* method as JVM can't do anything with the return value of it.
- **main:** It is the name of Java main method. It is the identifier that the JVM looks for as the starting point of the java program. It's not a keyword.
- **String[] args:** It stores Java *command line arguments* and is an array of type *java.lang.String* class. Here, the name of the String array is *args* but it is not fixed and user can use any name in place of it.