Unit -1

Introduction to Java: Introduction to Object Oriented Paradigm, Concepts of OOP, Applications of OOP, History of Java, Java Features, JVM, Program Structure. Variables, Primitive Data Types, Constants, Java String Class, Expressions, Primitive type conversion and Casting, Control Structures.

Object Oriented Paradigm :

Introduction to Object Oriented Programming:

Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects. Objects collaborate by sending messages to each other. An object is an entity that possess both state (called properties/attributes, in program defined as variables) and behaviour (the functionalities, in program defined as functions)

Principles of Object-Oriented Programming

- Class
- Object
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Class:

A class is a logical entity. It is blueprint from which individual objects are created. It represents the set of properties or methods that are common to all objects of one type.

Object:

Objects are the key for Object Oriented Programming. An Object is called the instance for a class (instantiation /initialization / memory allocation for a class) An Object possess two characteristics

State (Properties)

Behaviour (functions)

An Object stores its state in fields (Variables) and exposes its behaviour through methods.

Methods operate for functionalities which uses the state of object internally and serves as primary mechanism for object to object communication.

Example for Class and Object

Let’s take familiar entities Student and Teacher

First identify what are the properties and behaviour of these entities Student entity

State / properties include

Roll Number (String)

Name (String)

Year

Age (integer)

Marks (integer)

Branch (String)

Result (String) Behaviour / methods include

listenClasses()

onlineExam()

playGames()

let’s take a function writeExam

onlineExam()

{

// Use the properties, year and branch to retrieve respective online bits from data base

// allow the student to select their answers

// generate the result and update student’s property marks.

}

From the above function we can clearly observe that methods perform some actions using any properties if needed (like year and branch) and updates and property if needed (like marks).

Abstraction:

Abstraction is the process of hiding complexity (internal implementation) and showing essential information. For example, if you want to drive a car, you don't need to know about its internal workings. The same is true of Java classes. You can hide internal implementation details by using abstract classes or interfaces. On the abstract level, you only need to define the method signatures (name and parameter list) and let other classes implement these interfaces in their own way.

Encapsulation:

Binding the state and behaviour together into a single unit is known as encapsulation. In encapsulation, the variables / data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.

For example, if we write a single C program for defining state and behaviour of different entities (say student and teacher) it will be like

```
main ()
{
// following represent entity student properties
char [30] Rollnum;
char [30] name; int marks;
// following represent entity teacher properties char [30] name;
int empId;
char [20] PFnum;

}
// functions for entity student include listenClasses()
{

…. // implementation
}

writeExams()
{

… // implementation
}



// functionalities for entity teacher, include examEvaluation()
{
```

….. // implementation
}

postAttendence()
{

…. // implementation
}

In this approach maintaining, all entities state and behaviour together, programmer may willingly or unwillingly can access one entity variables under another entity functionalities.

With the help of classes we can eliminate such scenarios and bind properties and its related functions into single unit class also known as Encapsulation

class Student
{

// following represent entity student properties char [30] Rollnum;
char [30] name; int marks;
// functions for entity student include listenClasses()
{

…. // implementation
}

writeExams()
{

… // implementation
}

} // end of class Student.

Class Teacher
{
// following represent entity teacher properties char [30] name;

```
int empId;
char [20] PFnum;

}
// functions for entity student include listenClasses()
{

…. // implementation
}
writeExams()
{

… // implementation
}


// functionalities for entity teacher, include examEvaluation()
{

….. // implementation
}


postAttendence()
{

…. // implementation
}

} //end of class Teacher.
```

Applications of Object Oriented Programming User interface design such as windows, menu. Real Time Systems

Simulation and Modeling Object oriented databases AI and Expert System

Neural Networks and parallel programming

Decision support and office automation systems etc.

History of Java:

Java is invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems. in 1991. Most of the Java characteristics are inherited from C and C++ language. It was first named as Greentalk later called as “Oak” and was finally named as “Java” in 1995.

Other languages have the problem that they are designed to compile the code for a specific platform. To overcome this, Gosling and others started working on a portable and platform-independent language, this leads to the creation of Java.

Java had an extreme effect on the Internet by the innovation of a new type of networked program called the Applet. An applet is a Java program that is designed to be transmitted over the internet and executed by the web browser that is Java-compatible. Applets are the small program that is used to display data provided by the server, handle user input, provide a simple functions.

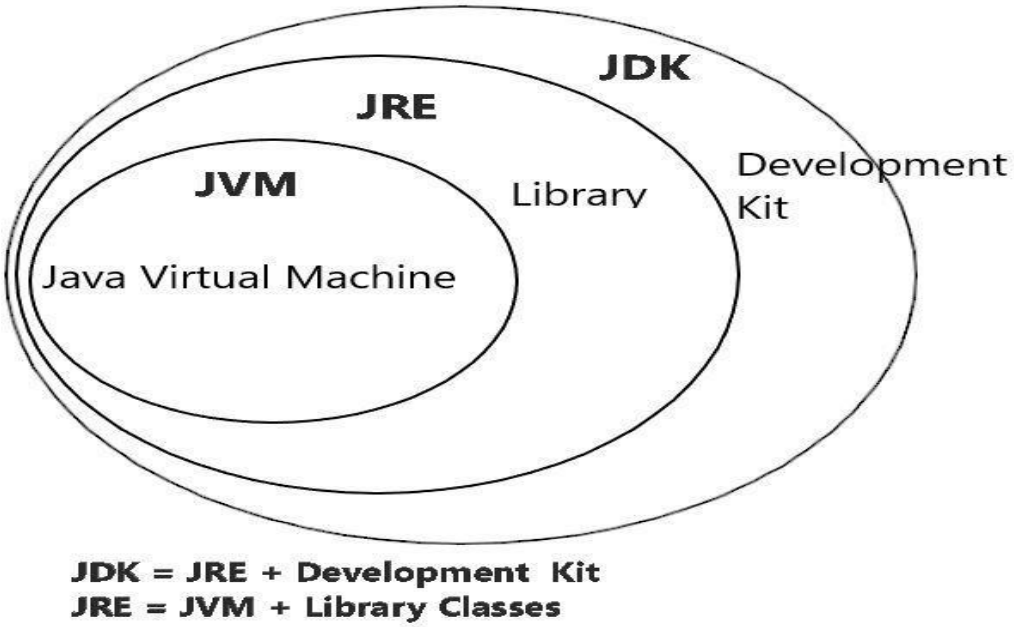
JDK :

JDK is a software development environment used for making applets and Java applications. The full form of JDK is Java Development Kit. Java developers can use it on Windows, macOS, Solaris, and Linux. JDK helps to code and run Java programs.

JRE :

JRE is a software which is designed to provide runtime environment for java applications.

It contains the class libraries, loader class, and JVM. In simple terms, if you want to run Java program you need JRE. If you are not a programmer, you don't need to install JDK, but just JRE to run Java programs. As all JDK versions comes bundled with Java Runtime Environment, so you do not need to download and install the JRE separately in your PC. The full form of JRE is Java Runtime Environment.



Evaluation of Java Versions

Java SE Version	Version Number	Release Date
JDK 1.0	1.0	January 1996
JDK 1.1	1.1	February 1997
J2SE 1.2	1.2	December 1998
J2SE 1.3	1.3	May 2000
J2SE 1.4	1.4	February 2002
J2SE 5.0	1.5	September 2004
Java SE 6	1.6	December 2006
Java SE 7	1.7	July 2011
Java SE 8	1.8	March 2014
Java SE 9	9	September, 21st 2017

Java SE 10	10	March, 20th 2018
Java SE 11	11	September, 25th 2018
Java SE 12	12	March, 19th 2019
Java SE 13	13	September, 17th 2019
Java SE 14	14	March, 17th 2020
Java SE 15	15	September, 15th 2020
Java SE 16	16	March, 16th 2021
Java SE 17	*17*	*Expected on Sept. 2021*

Among these versions only Java 8 and Java 11 have LTS (Long Term Service). Java 8 is the default and recommended version to download

what is LTS ?

A Java LTS (long-term support) release is a version of Java that will remain the industry standard for several years. Java 8 which was released in 2014, will continue to receive updates until 2020, and extended support will end by 2025. This gives plenty of OS vendors like Microsoft and Red Hat the time to repackage their releases with Java 8, time for application developers to update their applications to take full advantage of Java 8 features. At this time, the only other Java version that have LTS service is Java 11

Java Features

1) **Simple:**

The Java programming language is easy to learn. Java is similar to C/C++ but it removes the drawbacks and complexities of C/C++ like pointers and multiple inheritances. So one having knowledge on these languages will find Java familiar and easy to learn.

Object-Oriented programming language:

Java is a object-oriented programming language. It has all OOP features such as

- Object
- Class
- Inheritance

- Polymorphism
- Abstraction
- Encapsulation

Robust:

Java uses strong memory management techniques so that there is no improper memory assignment during the running of a program. The unreferenced objects still being in the memory led to the wastage of space. Java's garbage collector solves the problem it will delete the objects which are not used or not referenced anymore by the program.

Secure:

The Java platform is designed with security features built into the language, You never hear about viruses attacking Java applications. Memory access via pointer and performing pointer arithmetic is unsafe, so Java has no support for pointers to provide more security.

High Performance:

Java is an interpreted language, so it cannot be as fast as a compiled language like C or C++. But, Java achieves high performance with the use of just-in-time compiler.

Java is Multithreaded:

With this feature, Java supports "Multitasking". Multitasking is when multiple jobs are executed simultaneously. Multitasking improves CPU and Main Memory Utilization.

Distributed

In the era of Internet, applications need to run in distributed environment. This is possible in java applications since the programmer can use the TCP/IP protocols in the code. Java offers Remote Method Invocation (RMI) package to implement such interfaces in a multi-user application.

Java is Platform Independence:

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java follows write-once, run-anywhere principle.

On compilation Java program is compiled into bytecode.

This bytecode is platform independent and can be run on any machine.

JVM (Java Virtual Machine)

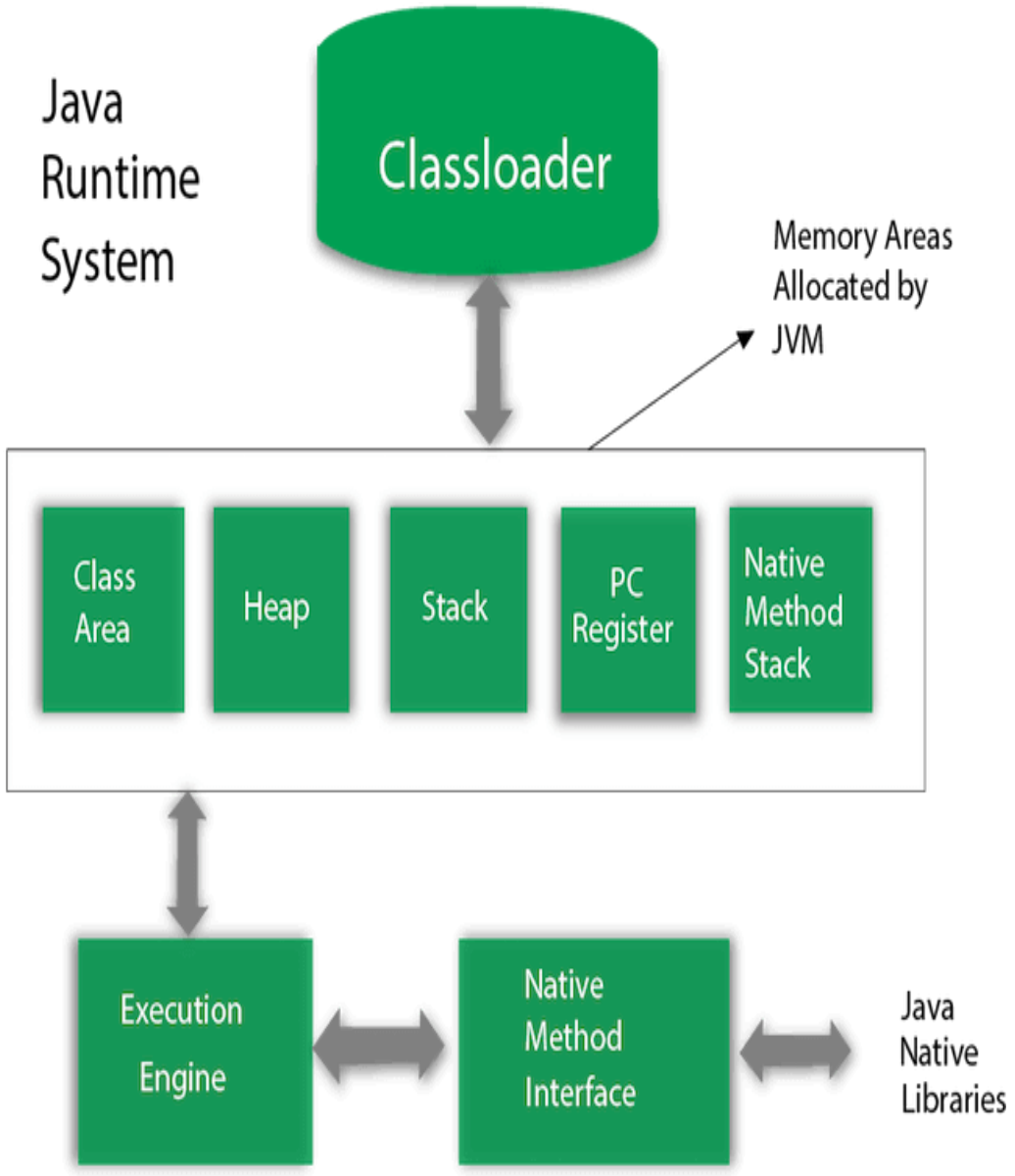
JVM (Java Virtual Machine) is a specification that provides runtime environment in which java bytecode can be executed. JVMs are platform dependent. The JVM will be provided separately for different machine languages(OS)

functionalities performed by the JVM

1. Loads code
2. Verifies code
3. Executes code
4. Provides runtime environment

JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc.



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. It is also known as Applicationclassloader.

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.

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.

Native Method Stack

It contains all the native methods used in the application.

Execution Engine

The execution engine is the Central Component of the java virtual machine(JVM). It communicates with various memory areas of the JVM. Each thread of a running application is a distinct instance of the virtual machine's execution engine. Execution engine executes the byte code which is assigned to the run time data areas in JVM via class loader. Java Class files are executed by the execution engine.

ExecutionEnginecontainsthreemaincomponentsforexecutingJavaClasses.Theyare:

Interpreter: It reads the byte code and interprets(convert) into the machine code(native code) and executes them in a sequential manner.

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.

Profiler: This is a tool which is the part of JIT Compiler is responsible to monitor the java bytecode constructs and operations at the JVM level.

Garbage Collector: This is a program in java that manages the memory automatically. It is a daemon thread which always runs in the background. This basically frees up the heap memory by destroying unreachable methods.

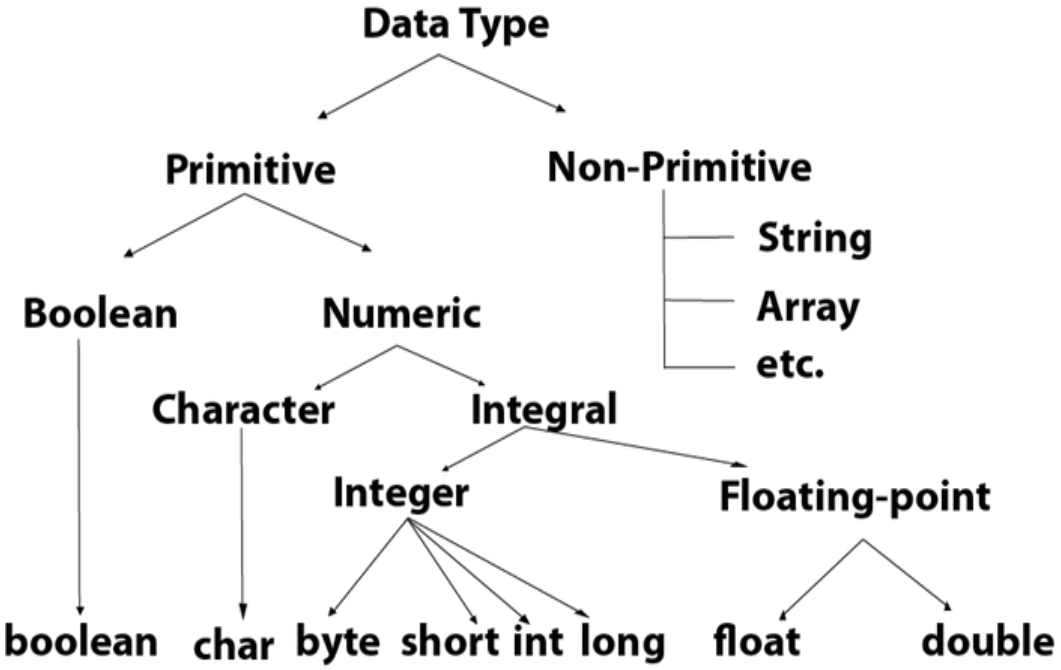
Java Native Interface

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.

Data Types

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

1) Primitive data types
2) Non-primitive data type



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Note: char default value '\u0000' indicates nul

Variables and Data Types in Java

Variable is a reference for memory location. There are three types of variables in java - local, instance and static.

Types of Variable :

There are three types of variables in java: o local variable

1) instance variable
2) static variable
3) Local Variable

A variable which is declared inside the method is called **local variable**. The scope and lifetime are limited to the method itself. the arguments of a function will also be treated as local variables to that method

```
void m(int a, int b)
{
int sum = a+b  // here a,b and c all three are local variables to function/method – m
}
```

Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by

the garbage collector of Java when there are no more reference to that object.

Static variable

A variable that is declared as static is called static variable. It cannot be local variable. It is a variable which belongs to the class and not to object(instance). These variables will be initialized first, before the initialization of any instance variables.

A single copy to be shared by all instances of the class

A static variable can be accessed directly by the class name and doesn't need any object

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 A
```

Constants in Java

A constant is a variable which cannot have its value changed after declaration. It uses the 'final' keyword.

Syntax

final dataType variableName = value; final int a =10;

static final dataType variableName = value; static final int b = 40;

Operators in java

Operator in java is a symbol that is used to perform operation over the operands. For example: +, -, *, / etc.

There are many types of operators in java which are given below: o Unary Operator,

1. Arithmetic Operator,
2. shift Operator,
3. Relational Operator,
4. Bitwise Operator,
5. Logical Operator,
6. Ternary Operator and
7. Assignment Operator

Operator Precedence

Operators	Precedence
postfix	*expr++ expr--*
unary	*++expr --expr +expr -expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Expressions

Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable.

Expressions can be built using values, variables, operators and method calls.

Types of Expressions -

While an expression frequently produces a result, it doesn't always. An expression can be Those that produce a value, i.e. the result of (1 + 1)

Those that assign a variable, for example (v = 10)

Java Type casting and Type conversion

Implicit type casting/Automatic Type Conversion -

Also known as widening conversion takes place when two data types can be automatically converted without any loss of data. This happens

when:

The two data types are compatible.

When we assign value of a smaller data type to a bigger data type.

For Example,

In java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char (or) 15boolean. Also, char and 15boolean are not compatible with each other.

type conversion/Implicit type casting example int a; short b = 50;

a = b; // conversion from b to a, smaller type to larger

Narrowing or Explicit Conversion -

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing. This is useful for incompatible data types where automatic conversion cannot be done.Here, target-type specifies the desired type to convert the specified value to.

example,

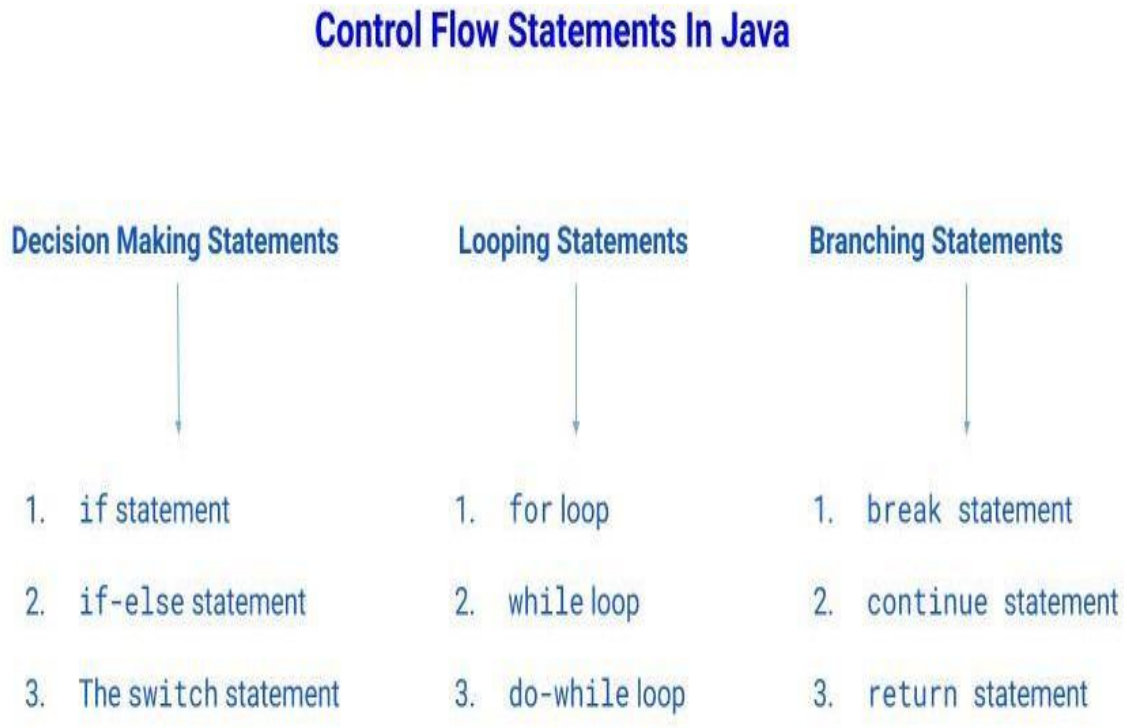
```
int a = 10;
byte b;
b = (byte) a;       //we are explicitly mentioning to convert a of type int to byte and then assign to b
```

Control Flow Statements

The control flow statements let you control the flow of the execution in your program. Java programming language, supports decision making, branching, looping, and adding conditional blocks.

The control flow statements can be categorized into

1) Decision Making Statements
2) Looping Statements
3) Branching Statements



Decision Making Statements

decision-making statements are used when we have to change the flow of execution based on a condition

There are three types of decision-making statements.

1) if statement
2) if-else statement
3) The switch statement

if statement

if statement is the most used decision-making statement in the java programming language.

syntax :

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

if-else statement

it is similar to the if statement, here we will add a block of statements to be executed when condition fails, which will be written under else case. If the value of the condition statement is true, then if block will be executed, otherwise the else block will be executed.

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

Nested if- else

In Nested if – else we can add one if-else block in another if-else block. In following we are adding an inner if-else block in the outer if block.

```
if(condition 1)
{
code to be executed
}
else
{
if(condition 2)
{
code to be executed -
}
else
{
code to be executed
}
}
```

Switch statement

In the switch statement, there could be several execution paths, each block the control will be transferred based on case value

```
switch(week)
{
case 1:
printf("Monday"); break;
case 2:
printf("Tuesday"); break;
case 3:
printf("Wednesday"); break;
case 4:
printf("Thursday"); break;
case 5:
printf("Friday"); break;
case 6:
printf("Saturday"); break;
case 7:
printf("Sunday"); break;
default:
printf("Invalid input! Please enter week number between 1-7.");
}
```

In the above example switch will receive an integer value in variable week, based on week value corresponding case statements will be executed. if value doesn't match with any case value the default will be executed.

Looping Statements

In looping statements, we are making a decision and executing the block of code multiple times. Until the condition is true, we are looping over the block of the code.

Each time we will check if the result of our decision statement is true or not, until and unless the result is true, we will execute the block of the code.

We can classify the lopping statements as follows:

1) for loop
2) while loop
3) do-while loop

1) **for loop**

In the for loop, we are going to check the value of the condition statement. If the value is true, the block of the code will be executed. After the successful execution of the code block, control again goes to the condition statement. Now, if the value is true, again the block of the code will be executed. Also, we are declaring one variable which stores the number of iteration. Each time the for loop runs, the value of i will be increased or decreased.

```
for(init variable declaration ; condition ; increment /decrement){

// code to be executed
}
```

while loop

In a while loop, we do apply the initialization, condition statement and an increment or decrement operator like the for loop but the syntax differs

```
init variable declaration while (condition){

// code to be executed
// increment or decrement statement
}
```

Example :

```
int i = 1; while(i<=5)  {
System.out.println( i ); i++;
}
System.out.print("End of while loop");
```

In the first line of the code, we are initializing a variable i of integer type with the value 1. In the next statement, we are checking the condition, if the value of the condition statement is true, the block of code will be executed.

When looking at the block of the code, at the end of the block you can see an increment operator, the value of i will increase by one, and the control goes to the condition statement. If the value of i is less than the 5, the block of code executes repeatedly.

When the value of the condition statement is false, the control goes to the next line of code after the while loop.

do -while loop

In the while loop, we are checking the condition statement first and then executing the block of code. But in the do-while loop, we are first executing the block of code and then checking the condition. If the value of the condition statement is true, the control goes at the starting of the code block, and the whole block of the code will be executed. Once the value of the condition statement is false, the control goes to the next line of code after the do-while loop.

```
init variable declaration do {
//code to be executed
} while ( condition statement);
```

The difference between the while and do-while loop is, in a while loop, we check the condition and executes the block of code, but in the do-while loop, we first execute the block of code and then check the condition.

Branching Statements

1. break statement
2. continue statement
3. return statement

break statement

break statement terminates the control flow. Usually, we do use the break statement to terminate the flow of for, while and do-while loop.

```
for(int i = 0; i<=5; i++)
{
if(i == 4) { break;
}
System.out.println( i );
}
```

The above code will print number from 1 to 3 and when i value becomes 4, the break statement will be executed and terminates from loop

continue statement

continue statement skips the current iteration of the for, while and do-while loop. The simple continue statement skips the iteration of the loop and sends the control back to the condition statement. The code after the continue statement will not be executed for the current iteration

```
for(int i = 0; i<=5; i++)
{
if(i == 4) { continue;
}
System.out.print( i );
}
```

The above loop will print numbers from 1 to 3, when i equals 4, the following statements will be skipped and controller will go back to the condition to execute next iteration of statements.It will print 1 2 4 5

4. return statement

In general return statement will be last line of the method. The return statement exits from the current method and control flow return to the line from which the method was invoked.

```
void m1()
{
int a = 5; int b = 6; int sum;
sum = add(a,b);
System.out.println(“result is ”+ sum)
}

int add(int I, int j)
{
int s = i+j;
return s;
}
```

In the above example when add function is called, controller will reach the add function, after executing statements of add, the return statement will return the controller back to calling method, using return we can even return values to calling method.