UNIT-I

Introduction to Object Oriented Programming (OOP):

Need of Object Oriented Programming (OOP):

All computer programs consist of two elements: code and data. Furthermore, a program can be conceptually organized around its code or around its data. That is, some programs are written around —what is happening and others are written around —who is being affected. These are the two paradigms that govern how a program is constructed.

The first way is called the procedure-oriented programming approach. This approach characterizes a program as a series of linear steps (that is, code). The procedure-oriented approach can be thought of as code acting on data. Procedural languages such as C employ this model to considerable success. Problems with this approach appear as programs grow larger and more complex. To manage increasing complexity, the second approach, called object-oriented programming, was conceived.

Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code. As you will see, by switching the controlling entity to data, you can achieve several organizational benefits.

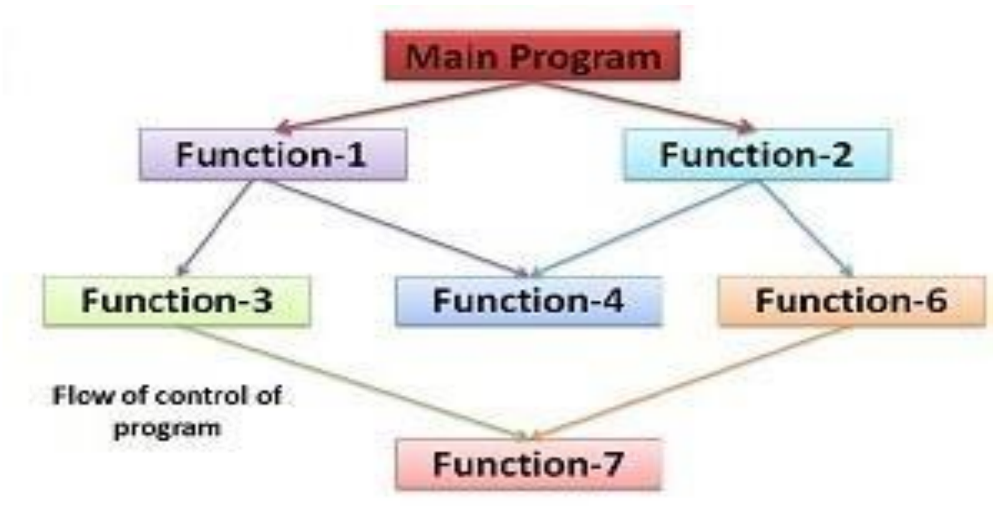
Procedure Oriented Programming (POP):

In this approach, the problem is always considered as a sequence of tasks to be done. A number of functions are written to accomplish these tasks. Here primary focus On—Functions and little attention on data.

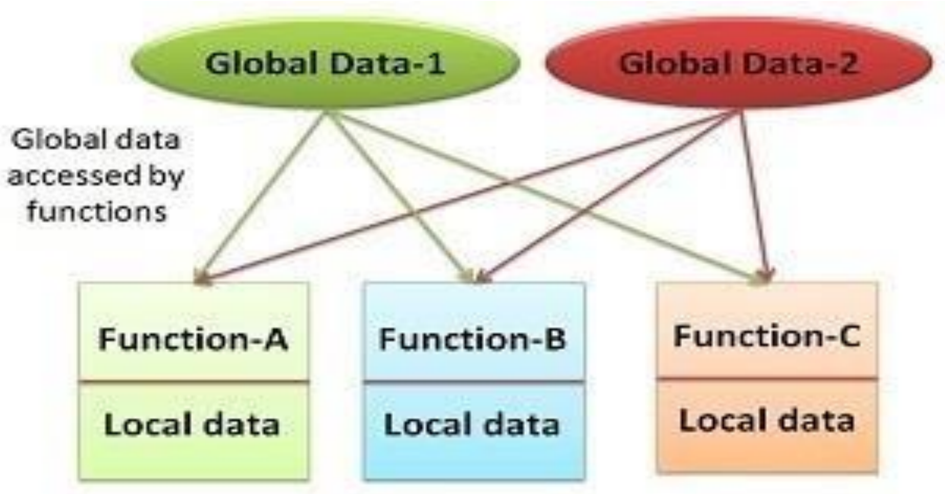
There are many high level languages like COBOL, FORTRAN, PASCAL, C used for conventional programming commonly known as POP.

POP basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions.

In a multi-function program, many important data items are placed as global so that they may be accessed by all the functions. Each function may have its own local data. Global data are more vulnerable to an in advent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we should also revise all the functions that access the data. This provides an opportunity for bugs to creep in.



Structure of procedure oriented program



Drawback: It does not model real world problems very well, because functions are actionoriented and do not really corresponding to the elements of the problem.

Characteristics of Procedure oriented Programming:

- Emphasis is on doing actions.
- Large programs are divided into smaller programs known as functions.
- Most of the functions shared global data.
- Data move openly around the program from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

Object Oriented Programming (OOP):

Object Oriented Programming allows us to decompose a problem into a number of entities called objects and then builds data and methods around these entities.

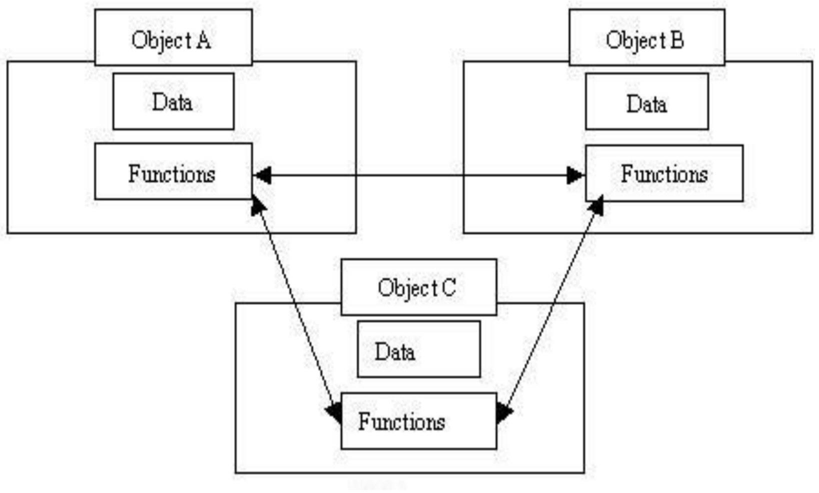
Object Oriented Programming is an approach that provides a way of modularizing programs by creating portioned memory area for both data and methods that can used as templates for creating copies of suchmodules on demand.

That is, an object a considered to be a partitioned area of computer memory that stores data and set of operations that can access that data. Since the memory partitions are independent, the objects can be used in a variety of different programs without modifications.

Characteristics of Object Oriented Programming:

- ❖ Emphasis on data.
- ❖ Programs are divided into what are known as objects.
- ❖ Data structures are designed such that they characterize the objects.
- ❖ Methods that operate on the data of an object are tied together.
- ❖ Data is hidden.
- ❖ Objects can communicate with each other through methods.
- ❖ Reusability.
- ❖ Follows bottom-up approach in program design.

Organization of Object Oriented Programming:



Differences between Procedure Oriented Programming and Object Oriented Programming:

Procedure Oriented Programming	**Object Oriented Programming**
1. In procedural programming, the program is divided into small parts called ***functions***.	1. In object-oriented programming, the program is divided into small parts called ***objects***.
2. In procedural programming, data moves freely within the system from one function to another.	2. In OOP, objects can move and communicate with each other via member functions.
3. It is less secure than OOPs.	3. Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
4. It follows a top-down approach.	4. It follows a bottom-up approach.
5. There are no access modifiers in procedural programming.	5. The access modifiers in OOP are named as private, public, and protected.
6. It gives importance to functions over data.	6. It gives importance to data over functions.
7. There is no code reusability present in procedural programming.	7. It offers code reusability by using the feature of inheritance.
8. It is not appropriate for complex problems.	8. It is appropriate for complex problems.
9. Examples of Procedural programming include C, Fortran, Pascal, and VB.	9. The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

Principles of Object Oriented Languages (or) OOP Principles:

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

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

Object:

- Any entity that has state and behavior is known as an object.
- For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class.
- An object contains an address and takes up some space in memory.

- An object is a run time entity
- Each object associated with data and methods to manipulate the data

Class:

- Collection of objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object.
- Class doesn't consume any space.
- Class defines the behaviour of an object.
- We can create any number of objects for a class.

Encapsulation:

Wrapping up of data and methods into a single unit is called as Data Encapsulation.

Example for encapsulation is a class

By using encapsulation, security is provided to the data.

Abstraction:

Data Abstraction refers to the act of representing the essential features without including the background details (or) explanations.

(or)

Hiding unnecessary data from the user.

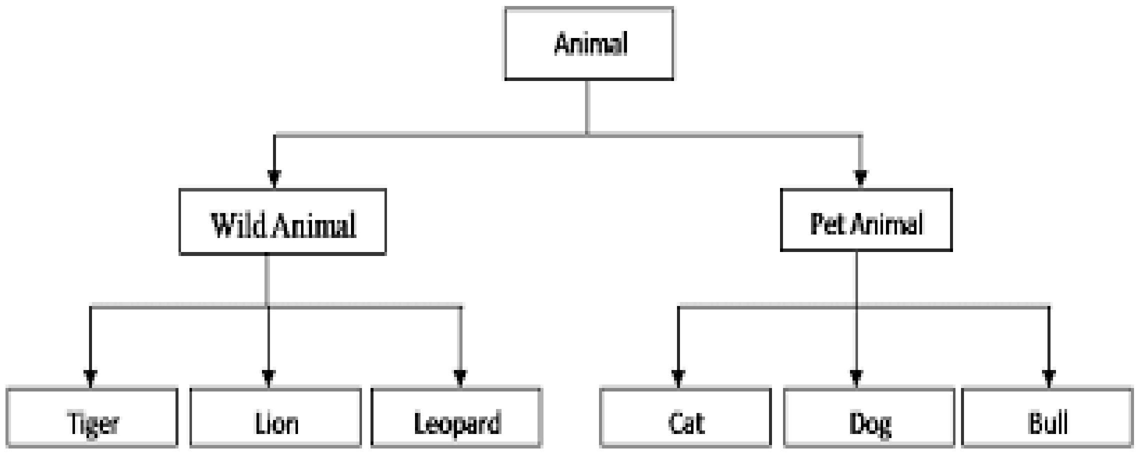
Inheritance:

Inheritance is the process by which objects of one class acquires the properties of objects of another class.

In inheritance, the derived class (child class) obtains the characteristics of the parent class.

By using inheritance, reusability can be achieved in Object Oriented Programming languages.

Example:



Polymorphism:

Polymorphism is a greek term, which means an ability to take more than one form.

Example:

Operation exhibit different behaviours at different instances.

Consider the addition operator '+'. If we pass integers as operands, then the operator will give the sum of 2 integers.

Here the operator + performs different operations depending on the type of data. This is called as operator overloading.

In JAVA, we use method overloading and method overriding to achieve polymorphism.

Applications of OOP:

- Computer graphics applications
- Object-oriented database
- User-interface design such as windows
- Real-time systems
- Simulation and modeling
- Client-Server System
- Artificial Intelligence System
- CAD/CAM Software
- Office automation system

History of JAVA:

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

Initially it was designed for small, embedded systems in electronic appliances like set-top boxes.

Firstly, it was called "Green talk" by James Gosling, and the file extension was .gt.

In 1993, it was named as “OAK”.

In 1995, Sun Micro Systems conducted a public conference and released JAVA’s first version.

Java Version History

Many java versions have been released till now. The current stable release of Java is Java SE 10.

JDK Alpha and Beta (1995)

JDK 1.0 (23rd Jan 1996)

JDK 1.1 (19th Feb 1997)

J2SE 1.2 (8th Dec 1998)

J2SE 1.3 (8th May 2000)

J2SE 1.4 (6th Feb 2002)

J2SE 5.0 (30th Sep 2004)

Java SE 6 (11th Dec 2006)

Java SE 7 (28th July 2011)

Java SE 8 (18th Mar 2014)

Java SE 9 (21st Sep 2017)

Java SE 10 (20th Mar 2018)

Java SE 11 (September 2018)

Java SE 12 (March 2019)

Java SE 13 (September 2019)

Java SE 14 (Mar 2020)

Java SE 15 (September 2020)

Java SE 16 (Mar 2021)

Java SE 17 (September 2021)

Java SE 18 (March 2022)

Features of JAVA (or) JAVA buzz words:

1. Simple
2. Object Oriented
3. Robust
4. Architecture Neutral
5. Multithreaded
6. Interpreted and High Performance
7. Secure
8. Portable
9. Distributed
10. Dynamic

1. **Simple:**
    - JAVA language is designed to be very simple for programmers to learn and use effectively.
    - Many of the confusing concepts in C/C++ like pointers, operator overloading etc. are eliminated from JAVA language.
2. **Object Oriented:**
    - JAVA language supports all the object oriented programming principles.
    - JAVA is a pure Object Oriented Programming language.
    - In JAVA language, everything is represented as an object. Even primary data types like integers, floating point numbers are also represented as objects.
3. **Robust:**
   Robust means strong.
   JAVA is said to be robust language because of two reasons. They are
   a) Memory Management
   b) Exception Handling

   a) **Memory Management:**
      - JAVA language provides implicit memory management. It avoids writing much code of memory management by the programmer.
      - When JAVA software starts, it runs a garbage collector thread, it takes care of garbage process.
      - Objects which are not required in further process of a program are treated as garbage objects. Garbage collector removes garbage objects while the execution is going on.

   b) **Exception Handling:**
      - Exception is an abnormal condition in a program.
      - JAVA language provides an efficient mechanism for handling exceptions without trashing the program.

4. **Architecture Neutral:**
    - JAVA compiler generates byte code that must be again converted into machine code at execution time.
    - This machine code conversion can be done by JVM.
    - Byte code is architecture neutral. That means, any operating system can

execute the byte code without bothering its architecture.

5. **Multithreaded:**
    - JAVA language supports multithreaded programming which allows to write programs that do many things simultaneously.
6. **Interpreted and High Performance:**
    - Generally languages like C and C++ are compiled languages but JAVA is both compiled and interpreted language.
    - At compilation stage, the JAVA source code is converted into byte code.
    - At run time, the JVM converts byte code into machine understandable code.
    - By the use of interpreter in JVM, JAVA language is slower in its execution. To enhance the performance of the program, Just In Time(JIT) compiler is used in JVM.
7. **Secure:**
    - By downloading C/C++ programs from internet, we obtain .exe files.
    - .exe file is the main source for getting virus into the system. So by downloading C/C++ programs, there is a chance to get virus into our systems.
    - By downloading JAVA programs from internet, we obtain .class files.
    - .class file is the solution for virus. So by downloading JAVA programs, there is no way of getting virus into the system.
8. **Portable:**
    - Portability of JAVA comes from its byte code.
    - A .class file obtained from the compilation on one platform can be easily ported or carried for execution on different platform.
9. **Distributed:**
    - JAVA language supports distributed programming so that we can easily write the networking programs using JAVA language.
    - JAVA language internally supports 2 important protocols for network programming i.e., TCP/IP.
      TCP-Transmission Control Protocol
      IP-Internet Protocol
10. **Dynamic:**
    - JVM carries lot of run time information of the program and also the objects of the program.
    - JVM dynamically loads .class files.

Simple Program:

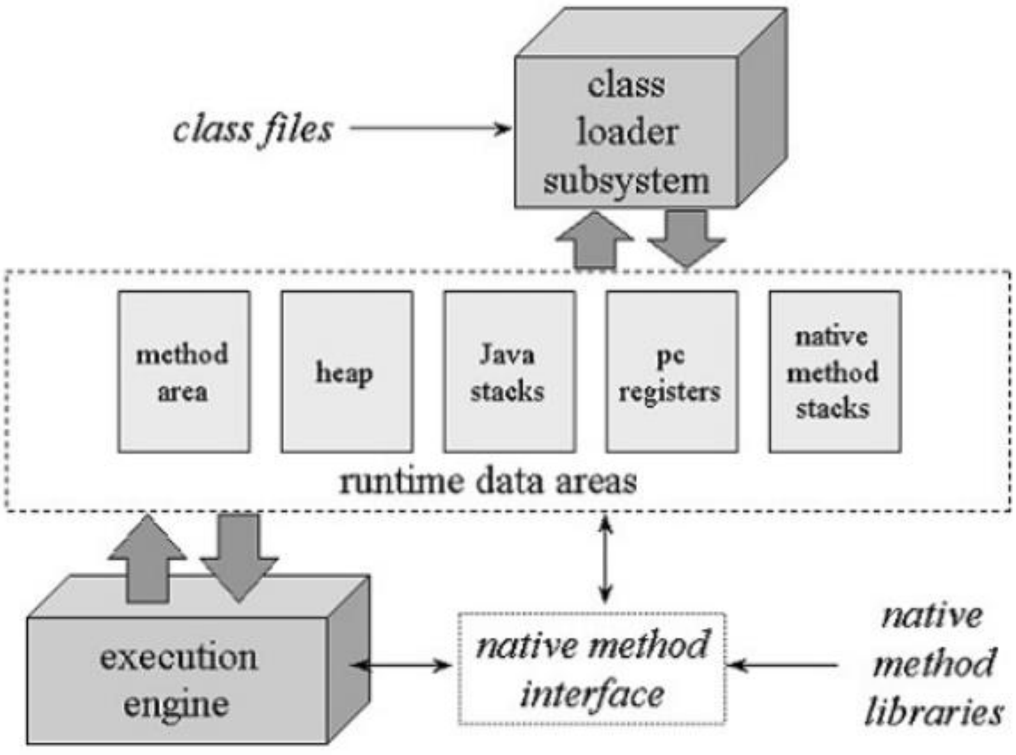
```
class Simple
{
  public static void main (String args[])
{
 System.out.println (“Hello VIT”);
}
}
```

- Here class is a keyword which is used for defining a class.
- Simple is a name of the class.
- To make main method available to outside of the class, we use public keyword.
- static keyword specifies that the main method is called by the JVM without instantiating the class.
- void represents the main method does not return any value.
- The parameters which are passed to main method are called as command line arguments. In JAVA language, main method takes an array of string objects as parameters.
- System is the one of the standard (or) predefined class which is used to perform some I/O operations.
- System.out represents the standard output device i.e., monitor.
- System.in represents the standard input device i.e., keyboard.
- println is the method which is used to display the output on the output screen line by line.

JAVA Virtual Machine (JVM):

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed.

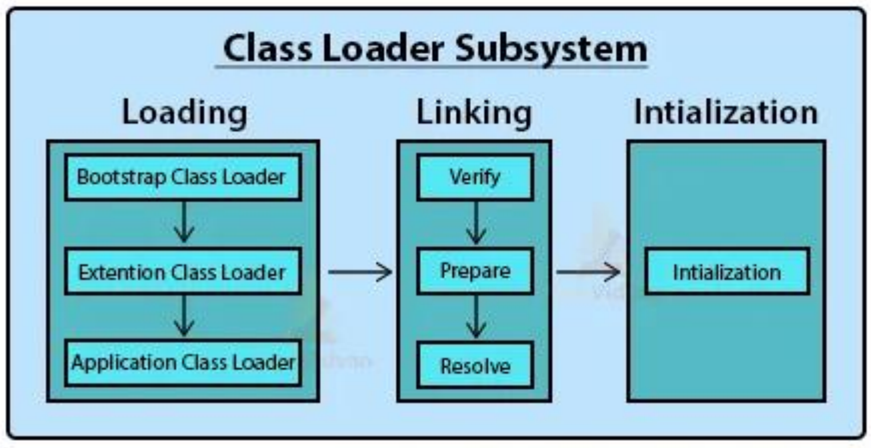
JVM Architecture:



JVM Diagram

Classloader Sub System:

- It is responsible for finding and loading class files into JVM.
- Classloader Sub System performs loading, linking and initialization.



Loading:

- It is used for finding and importing a class file.
- To load a class file, JVM contains 3 types of class loaders. They are
  a) Bootstrap Class Loader
  b) Extension Class Loader
  c) Application Class Loader

- Bootstrap Class Loader is responsible for loading JAVA API classes which are available in rt.jar file.
- Extension Class Loader is responsible for loading the classes from JRE's extension directories (jre/lib/ext)
- Application Class Loader is responsible for loading the classes found on path which maps to the class path environment variable.
- If all the class loaders are failed to load the required class file then JVM will generate "Class Not Found" exception.

Linking:

- It is used to perform verification, preparation and resolution.
- Verification-ensuring the correctness of the imported byte code.
- Preparation-Allocating memory for static variables and initializing the memory with default values.
- Resolution-Transforming symbolic references from the class into direct references.

Initialization:

- It is used to invoke the java code that initializes static variables to their proper starting values.

Run time Data Areas:

- Different areas into which the byte code is loaded are called as Run Time Data Areas.
- JVM creates 5 run time data areas to load the byte code. They are
  a) Method area
  b) Heap area
  c) Java Stack area
  d) PC register
  e) Native Method stack

Method area:

- All classes' byte code is loaded and stored in this run time area. All static variables are created and stored in this area.

Heap area:

- This area can be used for storing all the objects that are created. It is the main memory of JVM.
- This can be expanded by its own depending on the object creation.
- Method area and Heap area are sharable memory areas.

Java Stack area:

- This area can be used for storing the information of the methods.
- Java Stack can be considered as the combination of stack frames where every frame contains the state of a single method.
- In this run time area, JVM by default creates one thread i.e, main thread.
- main thread is responsible to execute main method.
- main thread is also responsible for creating objects in Heap area.

PC register:

- This area will contain the address of next instruction that have to be executed.

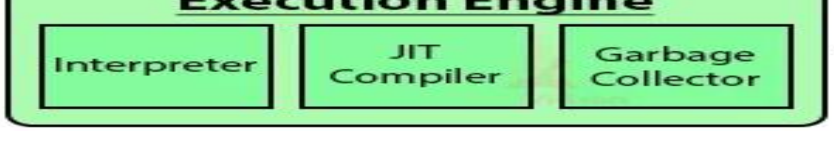
Native Method Stack:

- This area contains all the native methods used in application.

Execution Engine:

It is responsible for executing the program. It contains 3 parts.

a) Interpreter
b) Just In Time (JIT) compiler
c) Garbage Collector



- Interpreter is used to read the byte code and then execute the instructions.
- JIT compiler is used to improve the performance.
- Garbage Collector is responsible to destroy all the unused objects from Heap area.

Comments in JAVA:

- Comments are used to make the program more readable by adding the details of the code.
- There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1. Single Line Comment:

- The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements.
- Single line comments starts with two forward slashes **(//)**.

Syntax:

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

Example:

// This is a program to calculate area of triangle.

2. **Multi Line Comment:**

- The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time.
- Multi-line comments are placed between /* and */.

Syntax:

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

Example:

/* This is a program to calculate average marks of 60 students available in a class by using classes and objects.*/

3. **Documentation Comment:**

- Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API.
- These APIs are needed for reference, i.e., which classes, methods, arguments, etc., are used in the code.
- To create documentation API, we need to use the **javadoc tool**. The documentation comments are placed between /** and */.

Syntax:

```
/**
*
*We can use various tags to depict the parameter
*or heading or author name
*We can also use HTML tags
*
*/
```

Escape Sequences in JAVA:

- A character with a backslash (\) just before it is an escape sequence or escape character. We use escape characters to perform some specific task.

In JAVA, there is a total of eight escape sequences that are described in the following table.

Escape Characters	**Description**
\t	It is used to insert a **tab** in the text at this point.
\'	It is used to insert a **single quote** character in the text at this point.
\"	It is used to insert a **double quote** character in the text at this point.
\r	It is used to insert a **carriage return** in the text at this point.
\\	It is used to insert a **backslash character** in the text at this point.
\n	It is used to insert a **new line** in the text at this point.
\f	It is used to insert a **form feed** in the text at this point.
\b	It is used to insert a **backspace** in the text at this point.

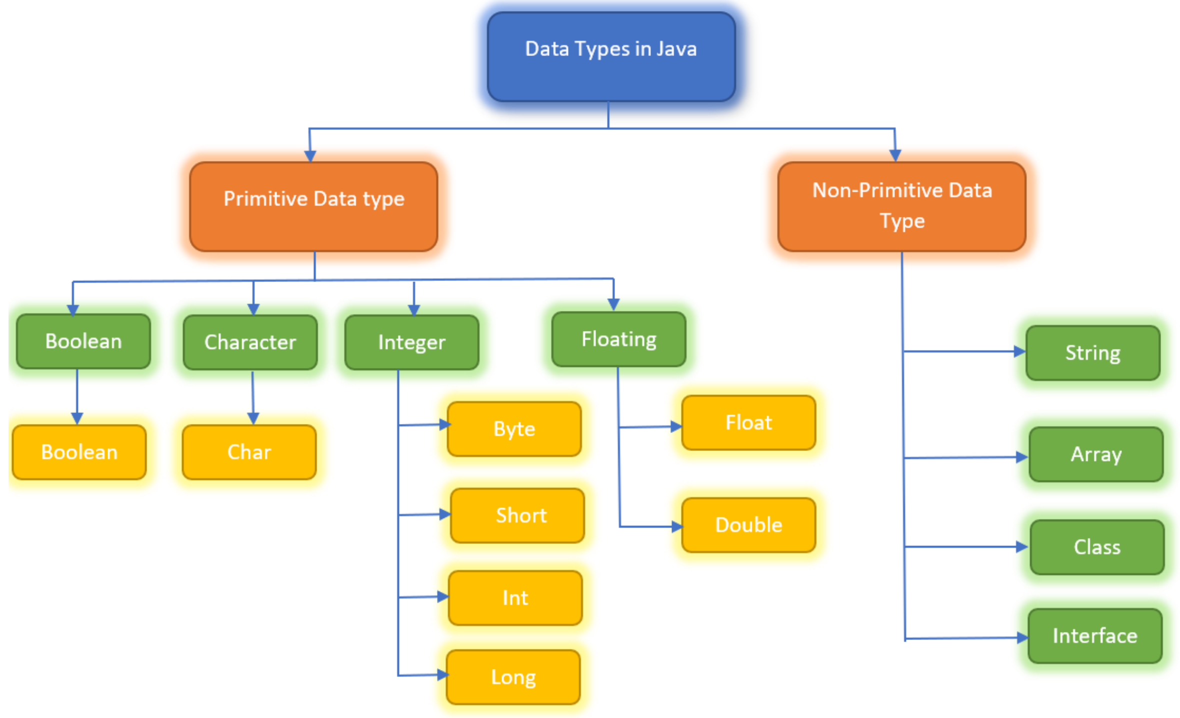
Data types in JAVA:

Data types are used to specify what type of values are stored in variables.

In JAVA language, data types are categorized into 2 types.

They are

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



Primitive Data types:

- In JAVA language, primitive data types are the building blocks of data manipulation.
- In JAVA language, primitive data types are categorized into 4 groups. They are
  1) Integers
  2) Floating point numbers
  3) Characters
  4) Boolean

Integers:

- These are used to represent signed whole numbers.
- JAVA language defines four integer types such as byte, short, int and long.

byte- variables of type byte are especially useful when working with a stream of data from a network or file. This is a signed 8-bit type that has a range from -2^7 to $+2^7-1$. Its default value is 0.

Example:

byte b;

short – It is probably the least used data type. It is a signed 16-bit type that has a range from -2^{15} to $+2^{15}-1$. Its default value is 0.

Example:

short s;

int – It is widely used data type for storing integer values. It is a signed 32-bit type that has a range from -2^{31} to +2^{31}-1. Its default value is 0.

Example:

int a;

long – It is used for occasions where an int type is not large enough to hold the desired value. It is a signed 64-bit type that has a range from -2^{63} to +2^{63}-1. Its default value is 0.

Example:

long l;

Floating point numbers:

- These are used to represent numbers with fractional precision.
- In JAVA language, there are 2 kinds' o f floating point types such as float and double for representing single and double precision numbers.

float – the type float specifies a single precision value that uses 32 bits of storage. Range of float type is 1.4e-045 to 3.4e+038. By using float type, it is possible to store upto 7 digits after decimal point.

Example:

float f;

double – It specifies double precision values that uses 64 bits of storage. Range of double type is 4.9e-324 to 1.8e+308. By using double type, it is possible to store upto 15 digits after decimal point.

Example:

double d;

Characters:

- In JAVA language, the data type used to store characters is char.
- JAVA language uses UNICODE to represent characters.
- UNICODE defines a fully international character set that can represent all of the characters found in all human languages.
- In JAVA language, character is a 16 bit type. The range of char is 0 to 2^{16} -1.

Example:

char ch;

Boolean:

- JAVA language has a primitive type called Boolean for storing logical/ boolean values.
- It can have only one of two possible values true or false.

- It represents one bit of information.
- The default value is false.

Example:

boolean b;

Keywords in JAVA:

- Keywords are also known as reserved words.
- Keywords are particular words that act as a key to a code.
- These are predefined words by Java so they cannot be used as a variable or object name or class name.

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		

Identifiers:

Identifier refers to name of a variable, method, class etc.

Rules for writing identifiers in JAVA:

- An identifier can consist of Capital letters A-Z, lowercase letters a-z, digits 0-9, and two special characters such as underscore and dollar Sign.
- The first character must be a letter.
- Blank spaces cannot be used in identifiers.
- Java keywords cannot be used as identifiers.
- Identifiers are case-sensitive.

Variables:

- Variable in Java is a data container that saves the data values during Java program execution.
- Every variable is assigned a data type that designates the type and quantity of value it can hold.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location.
- All the operations done on the variable affect that memory location.
- In Java, all variables must be declared before use.

Declaration of variables:

To declare a variable in JAVA language, the following syntax is used.

Syntax:

```
datatype var1, var2, var3, ........... varn;
```

Here datatype represents any one of the data type in JAVA language.

Example:

int a, b, c;

Initialization of variables:

The process of assigning values to variables is called as initialization of variables.

To initialize variables, the following syntax is used.

Syntax:

```
var_name=value;
```

Example:

a=10;

It is also possible to initialize the variables at the time of declaration as follows.

Syntax:

```
datatype var_name=value;
```

Example:

int a=10;

Reading input from the keyboard:

- In JAVA language, in order to read input from keyboard at runtime, we use Scanner class which is available in java.util package.
- Scanner class is used to read the input of primitive types like int, double, long, short, float, and byte.

Syntax:

```
Scanner sc=new Scanner(System.in);
```

The above statement creates scanner object and we are attaching standard input device (key board) to scanner object. So that scanner object is able to read the input from the keyboard.

Java Scanner class provides the following methods to read different primitives types:

Method	**Description**
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.

Write a Program to perform addition of two numbers.

```
//program to perform addition of two numbers
import java.util.*;
class Add
{
  public static void main(String args[])
{
    int a, b, c;
   Scanner sc=new Scanner(System.in);
  System.out.println(“enter a, b values”);
  a=sc.nextInt();
 b=sc.nextInt();
  c=a+b;
 System.out.println(“Result=”+c);
}
}
```

Write a program to find the area of triangle.

```
//program to find the area of a triangle
import java.util.*;
class Area
{
   public static void main(String args[])
  {
     float b,h,area;
    Scanner sc=new Scanner(System.in);
   System.out.println(“enter b, h values”);
   b=sc.nextFloat();
  h=sc.nextFloat();
   area=0.5*b*h;
   System.out.println("Area="+area);
}
}
```

Literals in JAVA:

Any constant value which can be assigned to the variable is called literal/constant.

Types of Literals in Java

There are majorly five types of literals in Java:

1. Integer Literals
2. Floating point Literals
3. Character Literals
4. Boolean Literals
5. String Literals

Integer Literals:

Integer literals are sequences of digits. There are three types of integer literals:

- **Decimal Integer**: These are the set of numbers that consist of digits from 0 to 9. It may have a positive (+) or negative (-) Note that between numbers commas and non-digit characters are not permitted. For example, 5678, +657, -89, etc.

```
int decVal = 26;
```

- **Octal Integer**: It is a combination of number has digits from 0 to 7 with a leading 0. For example, 045, 026,

```
int octVal = 067;
```

- **Hexa-Decimal**: The sequence of digits preceded by 0x or 0X is considered as hexadecimal integers. It may also include a character from a to f or A to F that represents numbers from 10 to 15, respectively. For example, 0xd, 0xf,

```
int hexVal = 0x1a;
```

- **Binary Integer**: Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later). Prefix 0b represents the Binary system. For example, 0b11010.

int binVal = 0b11010;

Floating Point Literals:

The vales that contain decimal are floating literals. In Java, float and double primitive types fall into floating-point literals. Keep in mind while dealing with floating-point literals.

- Floating-point literals for float type end with F or f. For example, 6f, 8.354F, etc. It is a 32-bit float literal.
- Floating-point literals for double type end with D or d. It is optional to write D or d. For example, 6d, 8.354D, etc. It is a 64-bit double literal.
- It can also be represented in the form of the exponent.

Floating point literal in decimal form:

float length = 155.4f;

double literal in decimal form:

double interest = 99658.445;

double literal in Exponent form:

double val= 1.234e2;

Character Literals:

A character literal is expressed as a character or an escape sequence, enclosed in a single quote ('') mark. It is always a type of char.

For example, 'a', '%', '\u000d', etc.

Boolean Literals:

Boolean literals are the value that is either true or false. It may also have values 0 and 1. For example, true, 0, etc.

boolean isEven = true;

String Literals:

String literal is a sequence of characters that is enclosed between double quotes ("") marks. It may be alphabet, numbers, special characters, blank space, etc.

For example, "Jack", "12345", "\n", etc.

Operators:

Operator is a symbol which is used for performing an operation between operands.

In JAVA language, there are 7 types of operators.

1) Arithmetic Operators
2) Relational Operators
3) Logical Operators
4) Assignment Operators
5) Increment and Decrement Operators (Unary Operators)
6) Bit-wise operators
7) Conditional Operator

Arithmetic Operators:

These operators are used to perform arithmetic operations like addition, subtraction, multiplication and division.

Symbol	**Meaning**
+	Addition
-	Subtraction
*	Multiplication
/	Division (gives quotient of the division)
%	Modulus (gives remainder of the division)

An expression which contains arithmetic operators is called as an arithmetic expression.

Example:

Consider int a=10, b=5;

a+b=15

a-b=5

a*b=50

a/b=2

a%b=0

Relational Operators:

These operators are used to compare two things.

JAVA language provides the following relational operators.

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Is equal to
!=	Not equal to

An expression which contains a relational operator is called as relational expression.

A relational expression always gives either true or false.

A relational expression is in the form

a.e-1 relop a.e-2

Here a.e-1 and a.e-2 are used to represent variables or values or expressions.

relop represents a relational operator.

Example:

int a=10, b=20;

a<b gives true

a<=b gives true

a>b gives false

a>=b gives false

a==b gives false

a!=b gives true

Logical Operators:

Logical operators are used to combine two or three conditions.

JAVA Language provides 3 logical operators.

They are

a) Logical AND (&&)

b) Logical OR (||)

c) Logical NOT (!)

a)Logical AND (&&): This operator is used to combine two conditions. If both the conditions are true then the result will be true otherwise false.

op-1	**op-2**	**op-1&&op-2**
Non-zero	Non-zero	Non-zero
Non-zero	Zero	Zero
Zero	Non-zero	Zero
Zero	Zero	Zero

An expression which contains logical operator is called as logical expression.

Logical expression gives either true or false.

Example:

int a=10, b=20, c=30;

(a<b)&&(b<c) gives true

(a<b)&&(b>c) gives false

(a>b)&&(b<c) gives false

(a>b)&&(b>c) gives false

b) Logical OR(||): If one of the conditions is true then this operator will give true otherwise give false.

op-1	**op-2**	**op-1||op-2**
Non-zero	Non-zero	Non-zero
Non-zero	Zero	Non-zero
Zero	Non-zero	Non-zero
Zero	Zero	Zero

Example:

int a=10, b=20, c=30;

(a<b)||(b<c) gives true

(a<b)||(b>c) gives true

(a>b)||(b<c) gives true

(a>b)||(b>c) gives false

c) Logical NOT(!): This operator converts the true value to false and false value to true.

Example:

int a=10, b=20;

!(a<b) gives false

!(a>b) gives true

!(b>a) gives false

!(b<a) gives true

Assignment Operators:

In JAVA Language, Assignment operator (=) is used to assign a value to the variable.

Example:

int a=10;

a=b=c=20;

Short hand Assignment operator:

An expression is in the form

var=var op exp

The above expression can be written as var op= exp

Here op= is called as short hand assignment operator.

Example:

a=a+30 can be written as a+=30;

short hand assignment operator examples are +=, -=, *=, /=, %= etc.

Increment and decrement operators:

These operators are called as unary operators because these operators are applied to only one operand.

Increment operator:

Increment operator increments the operand value by 1.

Symbol for increment operator is ++.

Depends on the position of increment operator, there are 2 types of incrementations.

a) Pre incrementation

b) Post incrementation

a) Pre incrementation: In this, increment operator is placed before the operand.

Syntax:

```
++operand
```

In pre incrementation, incrementation is done first and any other operation is done next.

Example:

```
int a=10, b;

b=++a;

System.out.println("a="+a+" "+"b="+b);
```

Output: a=11   b=11

b) Post incrementation: In this, increment operator is placed after the operand.

Syntax:

```
operand++;
```

In post incrementation, any other operation is done first and incrementation is done next.

Example:

```
int a=10, b;

b=a++;

System.out.println("a="+a+" "+"b="+b);
```

Output: a=11   b=10

Decrement operator:

Decrement operator decrements the operand value by 1.

Symbol for decrement operator is --.

Depends on the position of decrement operator, there are 2 types of decrementations.

a) Pre decrementation

b) Post decrementation

a) Pre decrementation: In this, decrement operator is placed before the operand.

Syntax:

```
    --operand
```

In pre decrementation, decrementation is done first and any other operation is done next.

Example:

```
int a=10, b;

b=--a;

System.out.println("a="+a+" "+"b="+b);
```

Output: a=9   b=9

b)Post decrementation: In this, decrement operator is placed after the operand.

Syntax:

```
operand--;
```

In post decrementation, any other operation is done first and decrementation is done next.

Example:

```
int a=10, b;

b=a--;

System.out.println("a="+a+" "+"b="+b);
```

Output: a=9   b=10

Bit-wise operators:

These operators are applied to only bits (i.e., 0's and 1's).

If we want apply bit-wise operators to any numbers, first that number is converted into binary number and then apply bit-wise operator on binary number.

JAVA Language provides the following bit-wise operators.

Operator	**Meaning**
&	Bit-wise AND
|	Bit-wise OR
^	Bit-wise Ex-OR
~	Bit-wise complement
<<	Bit-wise Left Shift
>>	Bit-wise Right Shift
>>>	Zero-fill Right Shift (or) Unsigned Right Shift

Truth table for the Bit-wise AND, Bit-wise OR, Bit-wise Ex-OR and Bit-wise complement.

A	**B**	**A&B**	**A|B**	**A^B**	**~A**	**~B**
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Bit-wise Left Shift Operator(<<): This operator shits the bits towards left side by specified number of times.

Syntax:

```
var<<num;
```

Here var represents a variable name and num represents an integer which specifies how many times the bits to be shifted towards left side.

Example:

int a=10;

a<<2;

a=00001010

a<<2=00101000

Bit-wise Right Shift Operator(>>): This operator shifts the bits towards right side by specified number of times.

Right Shift operator preserves the sign bit.

Syntax:

```
var>> num;
```

Example:

int a=10;

a>>2;

a-00001010

a>>2-00000010

zero-fill Right Shift (or) Unsigned Right Shift Operator:

This operator also shifts the bits towards right side by specified number of times but this operator always fills the sign bit with zero.

Syntax:

```
var>>> num;
```

Example:

int a=-8;

-8>>>2

a-11111000

a>>>2-00111110

Conditional Operator:

This operator is also called as ternary operator.

This is a simplified form of if-else statement.

Syntax:

```
var=exp1?exp2:exp3;
```

Here exp1 represents a condition. If the condition evaluates to true then exp2 is evaluated and assigned to var.

If condition evaluates to false then exp3 is evaluated and assigned to var.

Example:

int a=5,b=3,max;

max=(a>b)?a:b;

Output: max=5

Expressions:

Expression is a combination of operands and operators that reduces to a single value.

Evaluation of expressions:

- If an expression contains parenthesis, then always parenthesized sub expression is evaluated first.
- If parenthesis is nested then the inner most parenthesized expression is evaluated first.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
- Associativity rule is applied when two or more operators are having same priority in an expression.
- Arithmetic expressions are always evaluated from left to right according to operators' priority.

Precedence and associativity table:

Precedence	**Operator**	**Type**	**Associativity**
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension	Left to right

		Bitwise right shift with zero extension	
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Example1:

Evaluate expression a-b/3+c*2-1 when a=9, b=12, c=3.

a-b/3+c*2-1

9-12/3+3*2-1

9-4+3*2-1

9-4+6-1

5+6-1

11-1

10

Example2:

Evaluate r=a/b+c*d-e&&f-!g/h where a=1, b=2, c=3, d=4, e=5, f=6, g=7, h=8

1/2+3*4-5&&6-!7/8

1/2+3*4-5&&6-0/8

0+3*4-5&&6-0/8

0+12-5&&6-0/8

0+12-5&&6-0

12-5&&6-0

7&&6-0

7&&6

1

Type Conversion and Casting:

Type conversion is nothing but converting from one data type to another data type.

In JAVA language, type conversion can be done in 2 ways.

Those are

1) Implicit Type Conversion
2) Explicit Type Conversion

Implicit Type Conversion:

This type conversion is also called as widening conversion.

When one type of data is assigned to another type of variable, JAVA's automatic type conversion takes place if the following conditions are met.

i) Two types should be compatible
ii) Destination type is larger than source type.

Example:

1. Converting from byte to int
2. Converting from float to int
3. Converting from char to int

Explicit Type Conversion:

This type conversion is also called as narrowing conversion.

When two types are incompatible, we can use explicit type conversion.

Explicit type conversion can b e performed by using the following syntax

Syntax:

```
(Cast-type) expression;
```

Here cast-type represents any one of the predefined data type and expression represents a value or a variable or an expression.

Example:

```
int x=(int)23.5;

float ratio=(float)male/female;
```

Program to demonstrate Explicit Type Conversion.

```
class Conversion
{
  public static void main(String args[])
 {
    int i=257;
    byte b;
   double d=323.142;
    b=(byte)i;
    System.out.println(b+" "+i);
    b=(byte)d;
    System.out.println(b+" "+d);
}
}
```

Type Promotions in Expressions

While evaluating expressions, the intermediate value may exceed the range of operands and hence the expression value will be promoted. Some conditions for type promotion are:

1. Java automatically promotes each byte, short, or char operand to int when evaluating an expression.
2. If one operand is long, float or double the whole expression is promoted to long, float, or double respectively.

Control Statements:

Control statements are used to control the flow of execution in a program.

Control statements are categorized into 3 types.

i) Selection Statements
ii) Iteration Statements (or) looping statements
iii) Jump Statements

Selection Statements:

These statements are used to select one set of statements that are to be executed based on the outcome of the condition.

C-language provides 2 selection statements.

a) if statement
b) switch statement

if statement: Generally, if statement is available in 4 formats.

1. Simple if statement
2. if-else statement
3. Nested if-else statement
4. else-if ladder.

Simple if statement:

This is a one-way selection statement or decision-making statement.

Syntax:

```
if(condition)
{
   Statement block;
}
Statement-x;
```

Here the condition is evaluated first. If the condition evaluates to true, then statement block flowed by statement-x will be executed.

If the condition evaluates to false then the control is transferred to statement-x and statement-x is executed.

Flow-chart for Simple if statement:

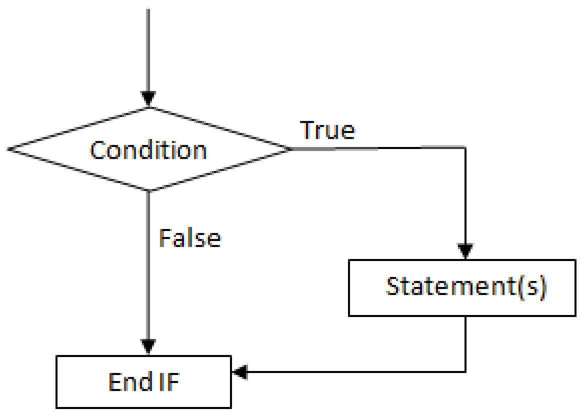


fig: Flowchart for if statement

Example:

```
if (10<20)

System.out.println ("Hello")

System.out.println  ("Bye");
```

Output:

```
Hello

Bye
```

Write a program to illustrate the use of simple if statement.

```
//program to illustrate the use of simple if statement

import java.util.*;

class Result

{

  public static void main(String args[])

{

  int a, b, c, d;

float result;

Scanner sc=new Scanner(System.in);
System.out.println(“enter a,b,c,d values”);
a=sc.nextInt();
b=sc.nextInt();
c=sc.nextInt();
d=sc.nextInt();
if((c-d)!=0)
{
   Result=(float)(a-b)/(c-d);
System.out.println(“result is ”+result);
}
System.out.println(“End of the program”);
}
```

Write a program to find the smallest of two numbers using simple if statement.

```
//program to find the smallest of two numbers
import java.util.*;
class Smallest
{
public static void main(String args[])
{
   int a,b;
  Scanner sc=new Scanner(System.in);
System.out.println(“enter a,b values”);
a=sc.nextInt();
b=sc.nextInt();
if(a<b)
System.out.println(a+“ is the smallest number”);
if(b<a)
System.out.println(b+"is the smallest number");

}
```

if-else statement:

It is an extension of simple if statement.

This statement is a 2-way selection statement or decision-making statement.

Syntax:

```
if(condition)
{
  statement block-1;
}
else
{
 statement block-2;
}
Statement-x;
```

Here the condition is evaluated first. If the condition evaluates to true then statement block-1 is executed.

If the condition evaluates to false then statement block-2 is executed.

After execution of either statement block-1 or statement block-2, the control is transferred to statement-x;

Flow-chart for if-else statement:

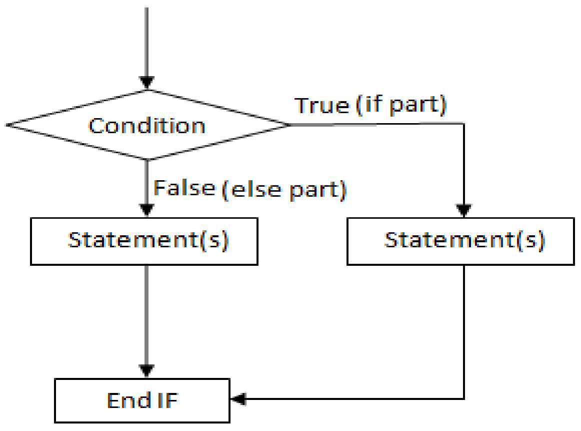


fig: Flowchart for if … else statement

Example:

```
int a=10,b=20;
if(a==b)
System.out.println("a and b are equal");
else
System.out.println("a and b are not equal");
```

Output: a and b are not equal.

Write a program to find the largest of two numbers using if-else statement.

```
//program to find the largest of two numbers using if-else statement
import java.util.*;
class Largest
{
public static void main(String args[])
{
   int a,b;
  Scanner sc=new Scanner(System.in);
System.out.println("enter a,b values");
a=sc.nextInt();
b=sc.nextInt();
  if(a>b)
System.out.println(a+" is largest number");
 else
System.out.println(b+"is largest number");
}
```

Write a program to check whether the given number is even or odd.

```
//program to check whether the given number is even or odd
import java.util.*;
class EvenOdd
```

```
{
public static void main(String args[])
{
    int n;
   Scanner sc=new Scanner(System.in);
   System.out.println("enter a number");
  n=sc.nextInt();
  if(n%2==0)
System.out.println(n+"is even number");
 else
System.out.println(n+"is odd number");
}
```

Write a program to check whether the given number is positive or negative.

```
//program to check whether the given number is positive or negative
import java.util.*;
class Number
{
public static void main(String args[])
{
   int n;
   Scanner sc=new Scanner(System.in);
System.out.println("Enter n value");
n=sc.nextInt();
  if(n>=0)
System.out.println(n+" is positive number");
 else
System.out.println(n+"is negative number");
```

}

}

Nested if-else statement:

When a series of decisions are involved, nested if-else statement is used.

Nested if-else statement means writing one if-else with in another if-else statement.

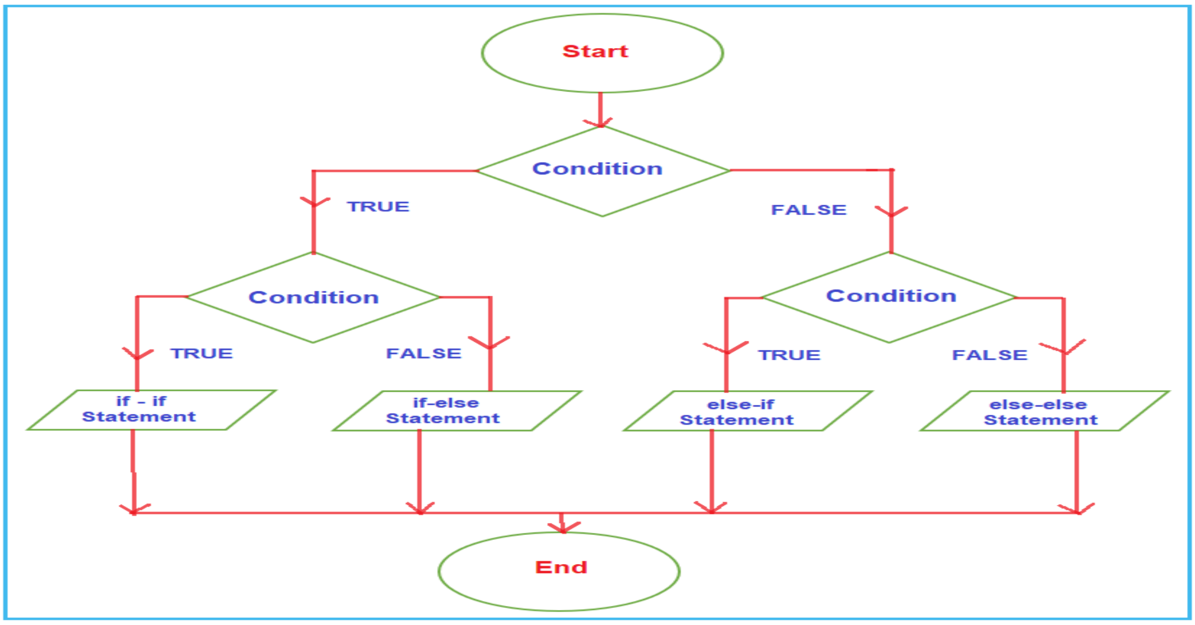
Syntax:

```
if(condition1)
{
  if(condition2)
 {
      Statement block-1;
 }
 else
  {
     Statement block-2;
  }
}
else
{
   if(condition3)
   {
Statement block-3;
    }
  else
  {
    Statement block-4;
   }
}
```

Here condition1 is evaluated first. If the condition1 evaluates to true then the condition2 is evaluated. If the condition2 is also true then statement blolck-1 is executed otherwise statement block-2 is executed.

If condition1 evaluates to false then condition3 is evaluated. If the condition3 evaluates to true then statement block-3 is executed otherwise statement block-4 is executed.

Flow-chart for Nested if-else statement:



Write a program to find the largest of three numbers.

```
//program to find the largest of three numbers
import java.util.*;
class Large
{
public static void main(String args[])
{
  int a,b,c;
  Scanner sc=new Scanner(System.in);
 System.out.println("enter a,b,c values");
 a=sc.nextInt();
 b=sc.nextInt();
 c=sc.nextInt();
  if(a>b)
  {
      if(a>c)
         System.out.println(a+" is largest number");
     else
```

```
          System.out.println(b+“is largest number”);
   }
   else
   {
        if(b>c)
             System.out.println(b+“is largest number”);
        else
             System.out.println(c+“ is largest number”);
  }
}
}
```

else-if ladder:

When multi-path decisions are involved, else-if ladder is used.

Syntax:

```
if(condition1)
      statement-1;
else if(condition2)
      statement-2;
else if(condition3)
     statement-3;
.
.
.
else if(condition-n)
    statement-n;
else
     statement-x;
```

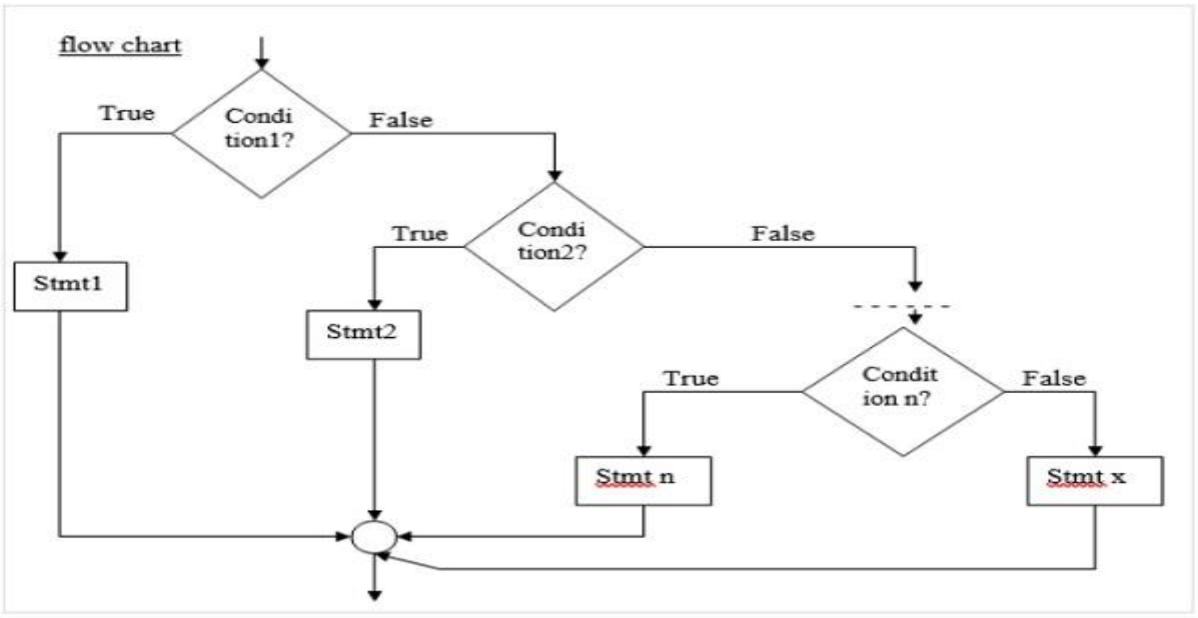
Here condition1 is evaluated first. If condition1 evaluates to true then statement-1 is executed otherwise condition2 is evaluated.

If condition2 evaluates to true then statement-2 is executed otherwise condition3 is evaluated.

This process is repeated till the condition-n.

If none of the conditions become true then the final else block i.e., statement-x will be executed.

Flow-chart for else-if ladder:



Write a program to find the roots of quadratic equation.

```
//program to find the roots of quadratic equation

import java.util.*;

class Roots

{

public static void main(String args[])

{

  double a,b,c,d,r1,r2;

Scanner sc=new Scanner(System.in);

System.out.println("enter a,b,c values");l

a=sc.nextDouble();

b=sc.nextDouble();

c=sc.nextDouble();

   d=b*b-4*a*c;

  if(d==0)
```

```
 {
System.out.println("roots are real and equal");
     r1=r2=-b/(2*a);
System.out.println("Root1="+r1+" "+"Root2="+ r2);
 }
else if(d>0)
 {
System.out.println("roots are real and distinct");
    r1=((-b+Math.sqrt(d))/(2*a));
    r2-((-b-Math.sqrt(d))/(2*a));
System.out.println("Root1="+r1+"  "+"Root2="+ r2);
 }
else
System.out.println("Roots are imaginary");
}
}
```

Write a program that accepts a number from 1 to 7 as input and display the week days depending on the input number.

```
//program to display week days depending on the input value
import java.util.*;
class WeekDay
{
public static void main(String args[])
{
   int n;
  Scanner sc=new Scanner(System.in);
 System.out.println("enter n value");
n=sc.nextInt();
```

```
  if(n==1)
   System.out.println(“Monday”);
 else if(n==2)
    System.out.println(“Tuesday”);
 else if(n==3)
     System.out.println(“Wednesday”);
  else if(n==4)
         System.out.println(“Thursday”);
  else if(n==5)
        System.out.println(“Friday”);
  else if(n==6)
      System.out.println(“Saturday”);
 else if(n==7)
      System.out.println(“Sunday”);
else
      System.out.println(“Invalid number”);
}
}
```

Switch Statement:

C-language provides a built-in multi-way selection statement known as switch statement.

switch statement is used to display the output in menu format.

Syntax:

```
switch(expression)
{
   case value-1: block-1;
breakl;
   case value-2: block-2;
                 break;
   case value-3: block-3;
                 break;
  .
  .
  .
  .
  case value-n: block-n;
                break;
  default: default-block;
}
Statement-x;
```

Here switch is a keyword, expression should evaluate to either integer or character constants.

value-1,value-2,…….value-n represents integers or character constants and are called as case labels.

block-1,block-2,……..block-n represents a set of zero or more statements.

break statement at the end of each case statement signals that the end of the case statement and causes an exit from switch statement, transferring control to statement-x;

When the switch statement is executed, the value of the expression is compared against the values value-1, value-2, ……value-n.

If a case is found whose value matches with the value of the expression, then the block of statements that follows the case are executed.

Default statement is an optional case. If none of the values matches with the expression value then the default block is executed.

Default statement can be written anywhere in switch statement.

Flow-chart for switch statement:

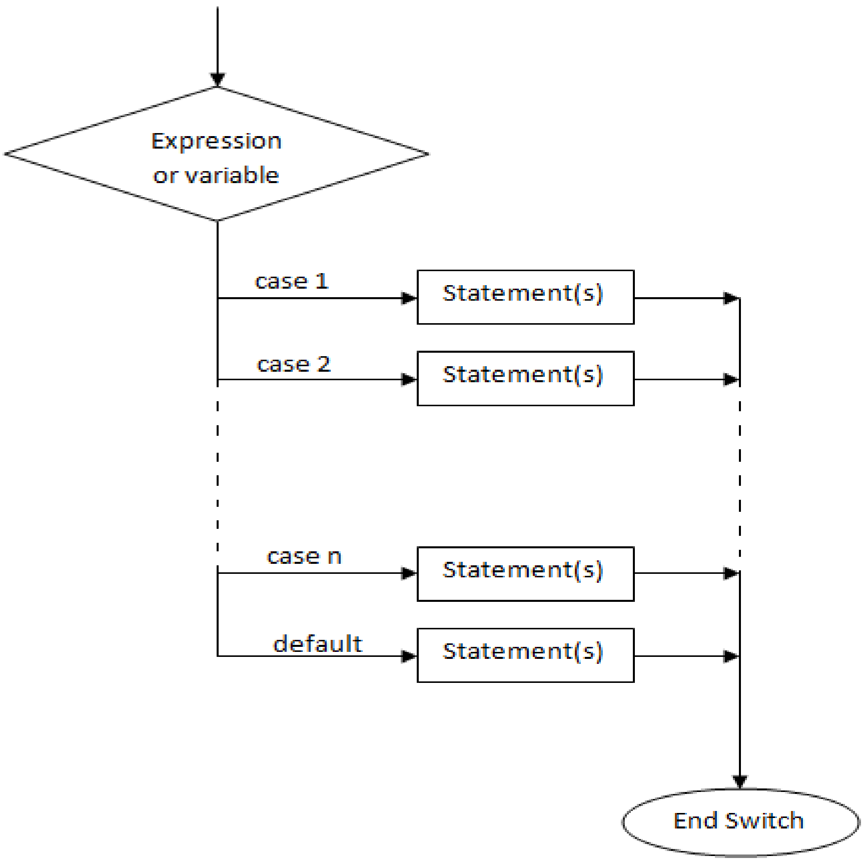


fig: Flowchart for switch case statement

Example1:

```
int a=1;
switch(a)
{
   case 0: System.out.println("zero");
              break;
   case 1: System.out.println("one");
              break;
   case 2: System.out.println("two");
              break;
  default: System.out.println("last statement");
}
```

Output: one

Example2:

```
int a=4;
switch(a)
{
    case 0: System.out.println("zero");
              break;
    case 1: System.out.println("one");
              break;
    case 2: System.out.println("two");
              break;
   default: System.out.println("last statement");
}
```

Output: last statement

Write a program that accepts a number from 1 to 7 as input and display the week days depending on the input number.

```
//program to display week days depending on the input value
import java.util.*;
class WeekDay
{
public static void main(String args[])
{
   int n;
   Scanner sc=new Scanner(System.in);
System.out.println(“enter n value”);

n=sc.nextInt();

  switch(n)

  {

           case 1:  System.out.println(“Monday”);

                        break;

           case 2: System.out.println(“Tuesday”);

                        break;

          case 3: System.out.println(“Wednesday”);

                       break;

        case 4: System.out.println(“Thursday”);

                     break;

        case 5: System.out.println(“Friday”);

                    break;

      case 6: System.out.println(“Saturday”);

                   break;

      case 7: System.out.println(“Sunday”);

                  break;

     default: System.out.println(“Invalid number”);

}

}

}
```

Write a program that accepts two integers and perform arithmetic operations by taking user choice using switch statement.

```
//program to perform arithmetic operations

import java.util.*;

class Arithmetic

{
```

```
public static void main(String args[])
{
  int a,b,c,ch;
  Scanner sc=new Scanner(System.in);
 System.out.println("enter a,b values");
 a=sc.nextInt();
b=sc.nextInt();
System.out.println("1.Addition");
System.out.println("2. Subtraction");
System.out.println ("3. Multiplication");
System.out.println ("4. Division");
System.out.println("5. Exit");
System.out.println("enter your choice");
ch=sc.nextInt();
 switch(ch)
{
   case 1: c=a+b;
               System.out.println("Addition="+c);
               break;
  case 2: c=a-b;
               System.out.println("Subtraction="+c);
              break;
  case 3: c=a*b;
              System.out.println("Multiplication="+c);
              break;
  case 4: c=a/b;
              System.out.println("Division="+c);
              break;
 case 5: System.exit(0);
             break;
  default: System.out.println("Invalid operation");
}
}
}
```

Iteration or looping statements:

Iterative statements are used to execute a set of statements repeatedly for a specific number of times or until the condition is satisfied.

Each iteration statement uses the following things.

Initialization:

In this, the values are assigned to control variables.

A variable which is used in looping statements is called as control variable.

Example:

i=0,n=10,a=5;

Condition:

Condition represents a relational expression.

Example:

a>b, i<=10, n!=0

Updation:

In this, the value of the control variable is updated by using either incfrement operator or decrement operator.

Example:

i++, n--, n++;

JAVA language provides 3 iteration statements.

a) While loop
b) do-while loop
c) for loop

while loop:

while loop is also called as entry-controlled loop or pretest loop or condition-controlled loop because the condition is checked at the entry point of while loop.

Syntax:

```
initialization;
while(condition)
{
  statement-1;
  statement-2;
  .
  .
  .
  statement-n;
  updation;
}
```

Here the condition is evaluated first. If condition evaluates to true then the statements with in the while block are executed.

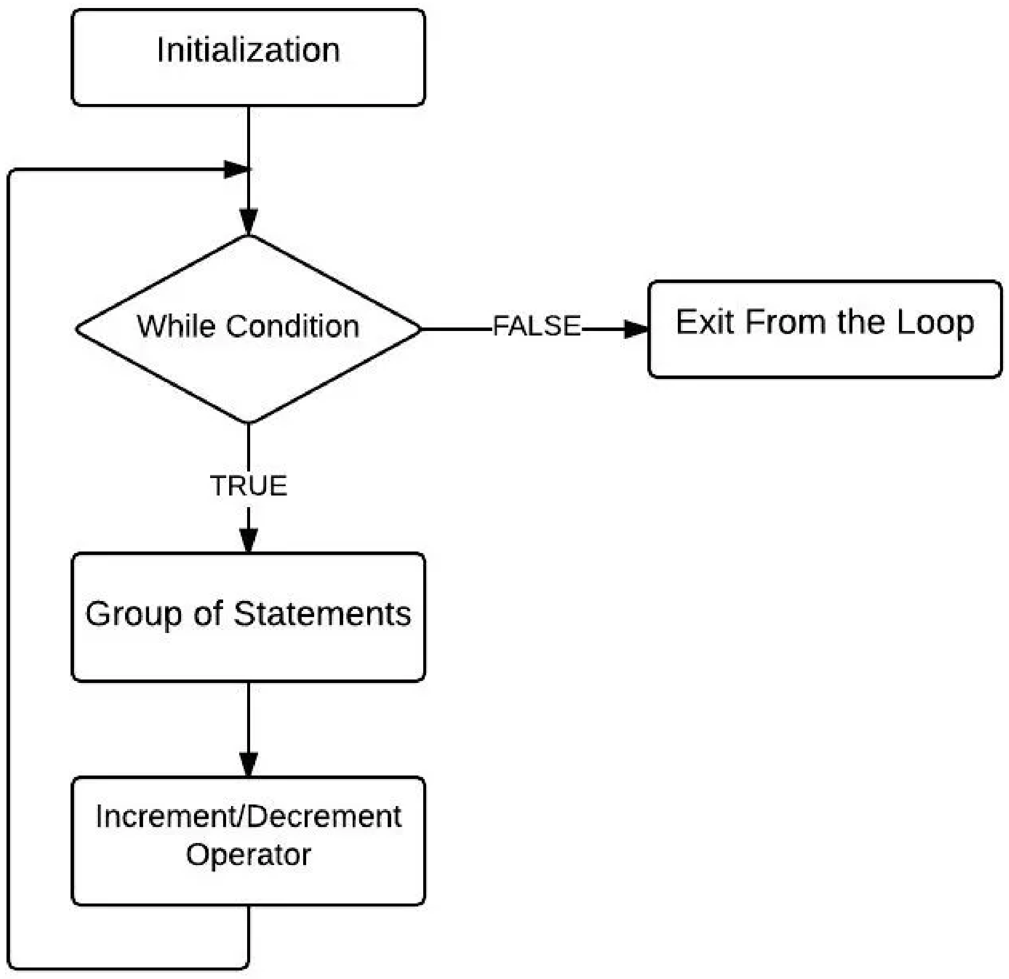
After execution of all the statements in while block, control transferred to condition.

If condition is again true then the statements in the while block are executed again.

This process is repeated until the condition becomes false.

Once the condition becomes false, the control is transferred to the statement after the while loop.

Flow-chart for while loop:



Write a program to display the numbers from 1 to 10 .

```
//program to display the numbers from 1 to 10

class Display

{

public static void main(String args[])

{

  int i;

  i=1;

  while(i<=10)

   {

        System.out.println(i);

          i++;
  }
}
}
```

Write a program to find the sum of first n numbers.

```
//program to find the sum of first n numbers
import java.util.*;
class Sum
{
public static void main(String args[])
{
 int n, i, sum=0;
 Scanner sc=new Scanner(System.in);
 System.out.println("enter n value");
 n=sc.nextInt();
i=1;
while(i<=n)
{
 sum=sum+i;
i++;
}
System.out.println("sum="+sum);
}
}
```

Write a program to find the sum of squares of first n numbers.

```
//program to find the sum of squares of first n numbers
import java.util.*;
class Sum
```

```
{
public static void main(String args[])
{
  int n, i, sum=0;
  Scanner sc=new Scanner(System.in);
  System.out.println("enter n value");
  n=sc.nextInt();
   i=1;
  while(i<=n)
{
  sum=sum+i*i;
i++;
}
System.out.println("sum="+sum);
}
}
```

Write a program to find the sum of individual digits of a given number.

```
//program to find the sum of individual digits of a given number
import java.util.*;
class SumOfDigits
{
public static void main(String args[])
{
int n, sum=0, digit;
 Scanner sc=new Scanner(System.in);
 System.out.println("enter n value");
 n=sc.nextInt();
 while(n!=0)
 {
   digit=n%10;
   sum=sum+digit;
  n=n/10;
}
System.out.println("sum="+sum);
}
}
```

Write a program to find the reverse of a given number.

```
//program to find the reverse of a given number
import java.util.*;
class Reverse
{
public static void main(String args[])
{
  int n, rev=0, digit;
  Scanner sc=new Scanner(System.in);
  System.out.println("enter n value");
  n=sc.nextInt();
 while(n!=0)
 {
   digit=n%10;
   rev=rev*10+digit;
  n=n/10;
}
System.out.println("Reverse of given number is "+rev);
}
}
```

Write a program to check whether the given number is palindrome or not.

```
//program to check whether the given number is palindrome or not
import java.util.*;
class Palindrome
{
public static void main(String args[])
{
    int n, rev=0, digit,temp;
   Scanner sc=new Scanner(System.in);
   System.out.println("enter n value");
  n=sc.nextInt();
   temp=n;
  while(n!=0)
  {
     digit=n%10;
     rev=rev*10+digit;
    n=n/10;
}
if(rev==temp)
      System.out.println(temp+"is palilndrome");
else
     System.out.println(temp+" is not palindrome");
}
}
```

Write a program to check whether the given number is Armstrong number or not.

```
//program to check whether the given number is Armstrong number or not
import java.util.*;
class Armstrong
{
public static void main(String args[])
{
   int n, sum=0, digit,temp;
   Scanner sc=new Scanner(System.in);
  System.out.println("enter n value");
  n=sc.nextInt();
   temp=n;
  while(n!=0)
  {
    digit=n%10;
sum=sum+(digit*digit*digit);
   n=n/10;
}
if(sum==temp)
      System.out.println(temp+" is Armstrong number");
else
     System.out.println(temp+" is not Armstrong number");
}
}
```

Write a program to find the factorial of a given number.

```
//program to find the factorial of a given number
import java.util.*;
```

```
class Factorial
{
public static void main(String args[])
{
  int n, i, fact=1;
  Scanner sc=new Scanner(System.in);
 System.out.println(“enter n value”);
n=sc.nextInt();
i=1;
  while(i<=n)
  {
    fact=fact*I;
  }
System.out.println(“Factorial of”+n+” is”+fact);
}
}
```

Write a program to print the Fibonacci series upto the given number.

```
//program to print the Fibonacci series upto the given number
import java.util.*;
class Fibonacci
{
public static void main(String args[])
{
   int n, a=0, b=1, c;
   Scanner sc=new Scanner(System.in);
  System.out.println(“enter n value”);
  n=sc.nextInt();
System.out.println("Fibonacci series is");
System.out.println(a+" "+b);
 c=a+b;
 while(c<=n)
 {
     System.out.println(c);
     a=b;
     b=c;
     c=a+b;
 }
}
}
```

do-while loop:

do-while loop is also called as exit controlled loop or posttest loop.

Syntax:

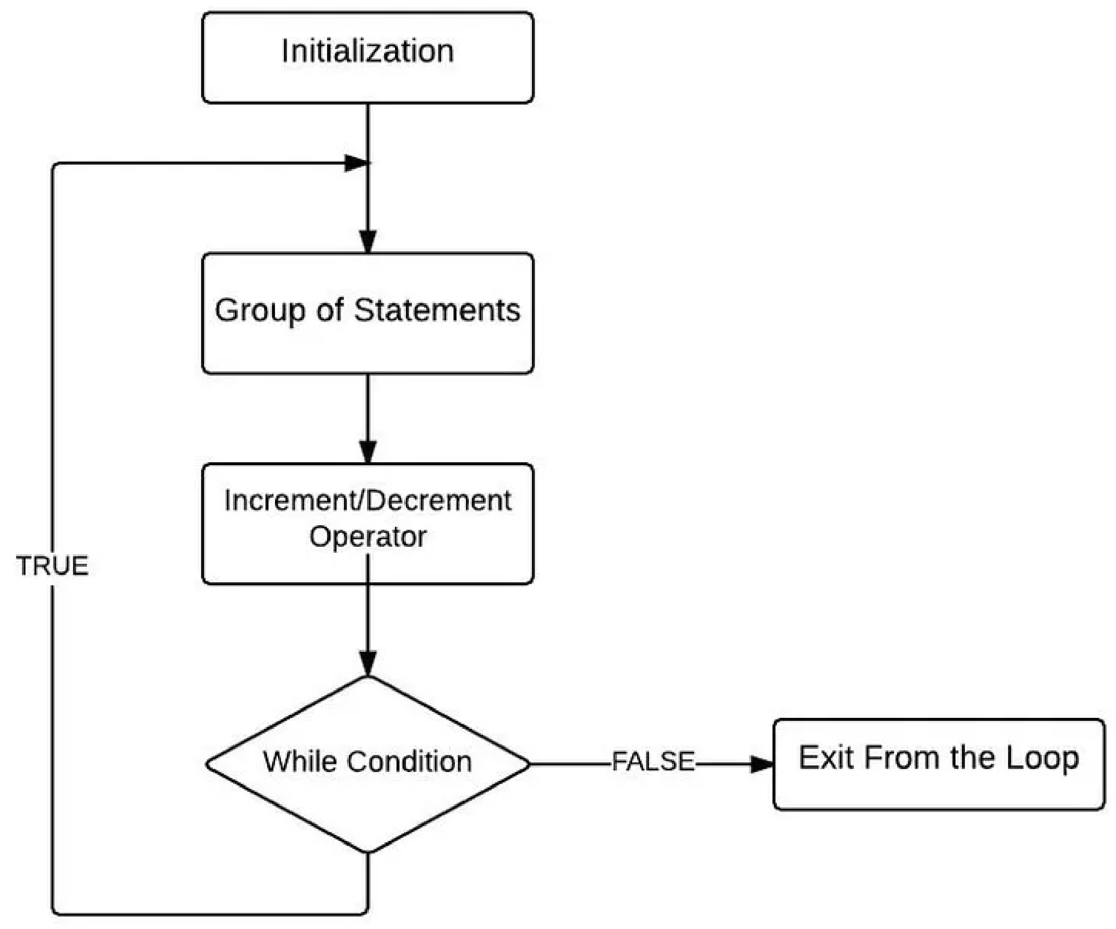
```
initialization;
do
{
  statement-1;
  statement-2;
  statement-3;
  .
  .
 statement-n;
 updation;
} while(condition);
```

Here the set of statements available in do-while block are executed once without checking any condition.

After execution of set of statements one time, control is transferred to condition. If the condition evaluates to true then the set of statements in do-while block are executed again.

This process is repeated until condition becomes false.

Flow-chart for do-while loop:



Write a program to display the numbers from 1 to 10 using do-while loop.

```
//program to display the numbers from 1 to 10
class Display
{
public static void main(String args[])
{
  int i;
i=1;
  do
  {
     System.out.println(i);
      i++;
  } while (i<=10);
}
}
```

Write a program to display the odd numbers from 1 to 50 using do-while loop.

```
//program to display odd numbers from 1 to 50
class DisplayOdd
{
public static void main(String args[])
{
   int i;
     i=1;
   do
   {
         System.out.println(i);
           i+=2;
   } while(i<=50);
}
}
```

Write a program to display the multiplication table of a given number.

```
//program to display the multiplication table of a given number
import java.util.*;
class MultiplicationTable
{
public static void main(String args[])
{
  int n,i;
  Scanner sc=new Scanner(System.in);
 System.out.println("enter n value");
 n=sc.nextInt();
i=1;
  do
  {
      System.out.println(i+"*"+n+"="+(n*i));
      i++;
 } while(i<=20);
}
}
```

for loop:

for loop is also called as entry-controlled loop or pretest loop.

Syntax:

```
for(initialization;condition;updation)
{
   //body of the loop
}
```

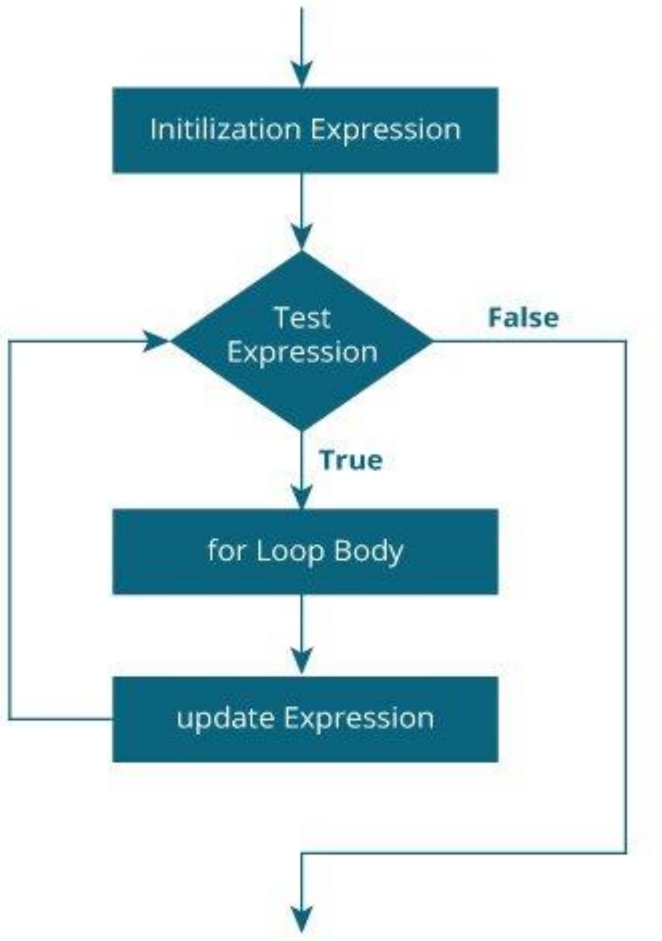
Here initialization is done first and initialization part is executed only one time during the execution of for loop.

After initialization, control is transferred to condition. If the condition is true then the body of the loop is executed. After execution of body of the loop, control is transferred to updation.

Then control is transferred to condition. If the condition is again true then body of the loop is executed again.

This process is repeated until the condition becomes false.

Flow-chart for for loop:



Write a program to print the numbers from 1 to 10.

```
//program to display the numbers from 1 to 10
class Display
{
Public static void main(String args[])
{
  int i;
  for(i=1;i<=10;i++)
  {
     System.out.println(i);
```

```
   }
}
}
```

Write a program to check whether the given number is prime number or not.

```
//program to check whether the given number is prime number or not
import java.util.*;
class Prime
{
public static void main(String args[])
{
    int n, i, count=0;
   Scanner sc=new Scanner(System.in);
  System.out.println("enter n value");
 n=sc.nextInt();
    for(i=1;i<=n;i++)
   {
      if(n%i==0)
           count++;
   }
if(count==2)
  System.out.println(n+" is prime number");
else
   System.out.println(n+"is not a prime number");
}
}
```

Write a program to check whether the given number is perfect number or not.

```
//program to check whether the given number is perfect number or not
import java.util.*;
class Perfect
{
public static void main(String args[])
{
    int n, i, sum=0;
    Scanner sc=new Scanner(System.in);
   System.out.println("enter n value");
  n=sc.nextInt();
    for(i=1;i<n;i++)
   {
      if(n%i==0)
          sum=sum+i;
   }
if(sum==n)
     System.out.println(n+" is perfect number");
else
     System.out.println(n+"is not a perfect number");
}
}
```

For-Each version of the for loop:

The Java for-each loop or enhanced for loop is introduced since J2SE 5.0.

It provides an alternative approach to traverse the array or collection in Java.

It is mainly used to traverse the array or collection elements.

Syntax:

```
for(type iter_var:collection)
      statement_block;
```

Here type specifies the type of the elements available in collection, iter_var represents a variable that receives elements from collection one at a time, from beginning to end and collection represents a collection of elements.

Example:

```
int a[]={1,2,3,4,5};

for(int x:a)

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

Nested for loop:

Writing one for loop with in another for loop is called as nested for loop.

Syntax:

```
for(initialization;condition1;updation1)
{
  for(initialization2;condition2;updation2)
  {
    //body of the inner loop
  }
}
```

Here initialization in outer for loop is executed first then control is transferred to condition1. If condition1 is true then the control is transferred to inner for loop.

In inner loop, initialization2 is executed first and then the control is transferred to condition2. If condition2 is true then the body of the inner for loop is executed. After execution of body of the inner for loop, control is transferred to updation2.

After updation, control is transferred to condition2 and it is evaluated.

This process is repeated until condition2 becomes false.

When the condition2 is false, the control is transferred to updation1. After updation, control is transferred to condition1 and it is evaluated.

This process is repeated until condition1 becomes false.

Write a program to print all the prime numbers between 2 and n.

```
//program to print all the prime numbers between 2 and n
import java.util.*;
class Prime
{
public static void main(String args[])
{
  int n, i, j, count;
 Scanner sc=new Scanner(System.in);
 System.out.println("enter n value");
 n=sc.nextInt();
   for(i=2;i<=n;i++)
   {
      count=0;
      for(j=1;j<=i;j++)
     {
        if(i%j==0)
            count++;
     }
    if(count==2)
          System.out.println(i);
  }
}
}
```

Write a program to display the following number pattern.

1

12

123

1234

12345

```
//program to display the number pattern
import java.util.*;
class Pattern1
{
public static void main(String args[])
{
   int n,i, j;
   Scanner sc=new Scanner(System.in);
   System.out.println("enter number of rows");
  n=sc.nextInt();
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=i;j++)
    {
       System.out.print(j);
     }
System.out.println();
 }
}
}
```

Write a program to display the following number pattern.

1

22

333

4444

55555

```
//program to display the number pattern
import java.util.*;
class Pattern2
{
public static void main(String args[])
{
    int n, i, j;
    Scanner sc=new Scanner(System.in);
   System.out.println("enter number of rows");
  n=sc.nextInt();
   for(i=1;i<=n;i++)
   {
      for(j=1;j<=i;j++)
      {
           System.out.println(i);
       }
         System.out.println();
  }
}
}
```

Write a program to display the following number pattern.

54321

4321

321

21

1

```
//program to display the number pattern
import java.util.*;
class Pattern3
{
public static void main(String args[])
{
   int n, i, j;
  Scanner sc=new Scanner(System.in);
 System.out.println("enter number of rows");
 n=sc.nextInt();
  for(i=n;i>=1;i--)
  {
     for(j=i;j>=1;j--)
     {
     System.out.print(j);
      }
     System.out.println();
 }
}
```

Jump Statements:

These statements are used to transfer the control from one point to another point in a program.

Some of the jump statements in C language are

1. break statement
2. continue statement

break statement:

break statement is used for 2 purposes.

i) To come out of switch statement
ii) An early exit from loops.

Syntax:

```
break;
```

Example:

```
int i=0;

while(i<=10)

{

i++;

  if(i==6)

    break;

System.out.println(i);

}
```

In nested loops, if the break statement is used in inner loop, then that break statement causes an early exit only from inner loop.

break statement is also used to terminate infinite loops.

While(1)
{
   -------------- 	
     ---------- ---
     -------------- 	
    break;
     ------------- 	
      ------------ 	
}
	do
	{
	   ------------ 	
	   ----------- 	
	    ---------- 	
	     ---------- 	
	     break;
	     ---------- 	
	     --------- 	
	}while(1);
		for(;;)
		{
		   ---------------- 	
		    --------------- 	
		    --------------- 	
		    break;
		    -------------- 	
		    ------------- 	
		}

continue statement:

continue statement is used to perform the next iteration of the loop by skipping some statements in loop.

Syntax:

```
continue;
```

Example1:

```
int i=0;
while(i<10)
{
  i++;
 if(i==6)
   continue;
System.out.println(i);
}
```

Example2:

```
int i=0;
while(i<10)
{
i++;
    if(i<=5)
```

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

Arrays:

An array is a collection of homogeneous elements that are stored under a single name.

In JAVA language, arrays are categorized into two types. They are

1) One dimensional arrays
2) Multi dimensional arrays

One dimensional arrays:

In JAVA language, array creation is a two step process.

In first step, we have to declare the name of the array with data type.

Syntax:

```
datatype array_name[];
```

Or

```
datatype[] array_name;
```

In Second step, memory is allocated for the array using new operator.

Syntax:

```
array_name = new datatype[size];
```

Here size represents the number of elements to be stored in an array.

After allocating memory, all the array elements are initialized to zero by default.

We can combine the above two steps into a single statement as follows.

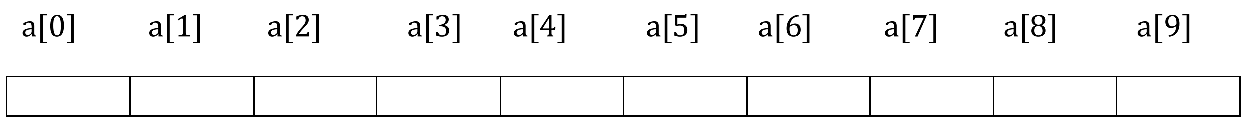
```
datatype array_nmae[]=new datatype[size];
```

Example:

int a[]=new int[10];

After creating an array, array elements are accessed using index value.

In arrays, index value is starting from 0 to size-1.



The above array can be initialized as follows

```
a[0]=1;

a[1]=2;

a[2]=3;

a[3]=4;

a[4]=5;

a[5]=6;

a[6]=7;

a[7]=8;

a[8]=9;

a[9]=10;
```

Arrays can also be initialized at the time of declaration.

Example:

```
int a[]={1,2,3,4,5};

int a[]=new int[] {1,2,3,4,5};
```

To initialize the array elements at runtime, we use for loop.

Example:

```
int a[]=new int[5];

for(int i=0;i<5;i++)

{

    a[i]=sc.nextInt();

}
```

Write a program to read and display array elements.

```
//Program to read and display array elements
import java.util.*;
class A
{
   public static void main(String args[])
   {
         int n,i;
        Scanner sc=new Scanner(System.in);
      System.out.println(“enter the size of the array”);
      n=sc.nextInt();
      int a[]=new int[n];
      System.out.println(“enter array elements”);
      for(i=0;i<n;i++)
      {
          a[i]=sc.nextInt();
      }
     System.out.println(“Array elements are”);
      for(i=0;i<n;i++)
     {
      System.out.println(a[i]);
    }
}
}
```

Write a program to find the sum and average of array elements.

```
//Program to find the sum and average of array elements
import java.util.*;
```

```
class SumAvg
{
    public static void main(String args[])
    {
         int n,i,sum=0;
         float avg;
        Scanner sc=new Scanner(System.in);
        System.out.println(“enter the size of the array”);
       n=sc.nextInt();
       int a[]=new int[n];
        System.out.println(“enter array elements”);
       for(i=0;i<n;i++)
      {
          a[i]=sc.nextInt();
       }
    for(i=0;i<n;i++)
    {
        sum=sum+a[i];
      }
   avg=sum/n;
 System.out.println(sum+” “+avg);
}
}
```

Write a program for sorting a list of elements in ascending order.

```
//Program for sorting a list of elements in ascending order.
import java.util.*;
 class Sorting
```

```
{
    public static void main(String args[])
     {
          int n,i,temp;
          Scanner sc=new Scanner(System.in);
         System.out.println("enter the size of array");
        n=sc.nextInt();
       int a[]=new int[n];
      System.out.println("enter elements into array");
      for(i=0;i<n;i++)
     {
        a[i]=sc.nextInt();
     }
     for(i=0;i<n;i++)
      {
           for(j=i+1;j<n;j++)
           {
              if(a[i]>a[j])
              {
                   temp=a[i];
                 a[i]=a[j];
                a[j]=temp;
            }
}
}
System.out.println("After sorting elements are");
 for(i=0;i<n;i++)
{
    System.out.println(a[i]);
}
}
}
```

Multi-dimensional Arrays:

An array with more than one dimension is called as multi dimensional array.

Multi dimensional array is nothing but array of arrays.

Two-dimensional array:

When we want to store the data in tabular form or matrix form, two-dimensional array is used.

To create two-dimensional array in JAVA language, the following syntax is used.

Syntax:

```
datatype array_name[][]=new datatype[row_size][col_size];
```

Example:

int a[][]=new int[2][2];

It is also possible to create two-dimensional array without specifying the column size.

Example:

int a[][]=new int[2][];

Write a program for addition of two matrices.

```
//Program for addition of two matrices.
import java.util.*;
 class Matrix
{
  public static void main(String args[])
  {
   int r1,c1,r2,c2i,j;
   Scanner sc=new Scanner(System.in);
   System.out.println("enter row and column sizes of first matrix");
   r1=sc.nextInt();
   c1=sc.nextInt();
   int a[][]=new int[r1][c1];
    System.out.println("enter elements of first matrix");
    for(i=0;i<r1;i++)
   {
     for(j=0;j<c1;j++)
     {
         a[i][j]=sc.nextInt();
      }
  }
  System.out.println("enter row and column sizes of second matrix");
   r2=sc.nextInt();
   c2=sc.nextInt();
   int b[][]=new int[r2][c2];
    System.out.println("enter elements of first matrix");
    for(i=0;i<r2;i++)
   {
     for(j=0;j<c2;j++)
     {
         b[i][j]=sc.nextInt();
      }
  }
if((r1==r2)||(c1==c2))
```

```
{
  int c[][[]=new int[r1][c1];
 for(i=0;i<r1;i++)
 {
   for(j=0;j<c1;j++)
  {
     c[i][j]=a[i][j]+b[i][j];
    }
 }
System.out.println("Resultant matrix is");
for(i=0;i<r1;i++)
{
 for(j=0;j<c1;j++)
 {
   System.out.print(c[i][j]+" ");
  }
System.out.println();
}
}
else
System.out.println("Matrix addition is not possible");
}
}
```