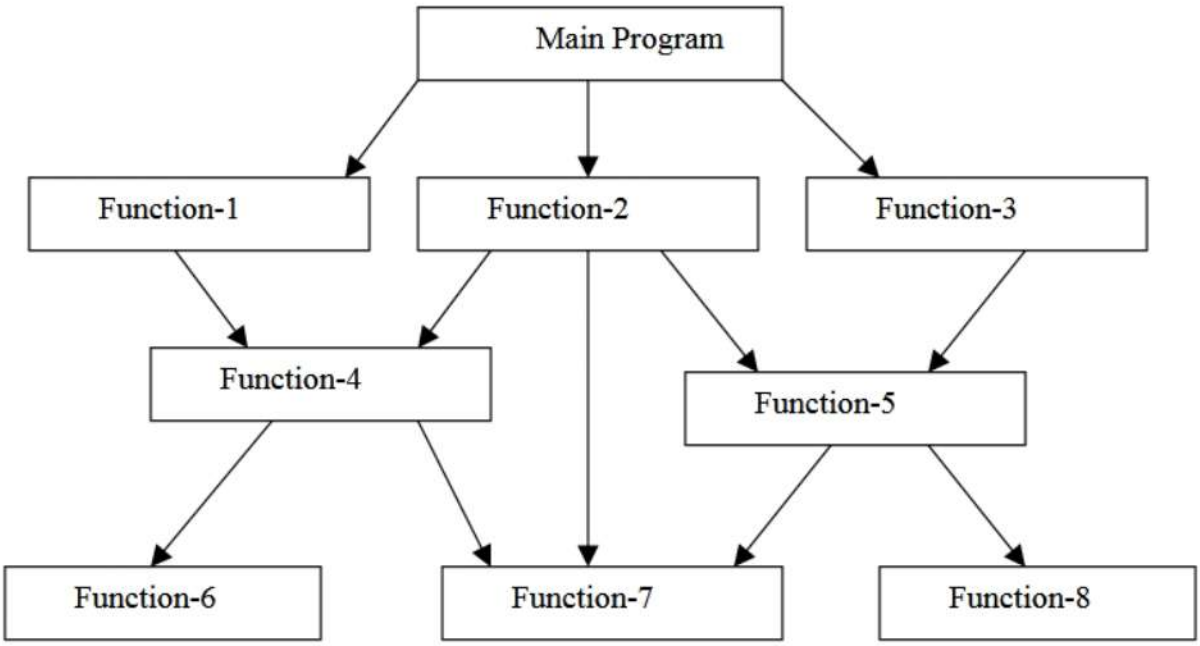
# UNIT-I: Introduction

## Procedure-Oriented Programming vs Object-Oriented Programming:

### Procedure-Oriented Programming Approach:

Procedure oriented programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as procedures or functions. In the procedure oriented approach, the primary focus is on functions.

A typical structure for procedural programming is shown in the following figure.




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 inadvertent 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 also need to revise all functions that access the data.

Some Characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.

• Employs top-down approach in program design.

**Object-Oriented Programming Approach:**

Object-oriented programming (OOP) is a programming paradigm based upon objects

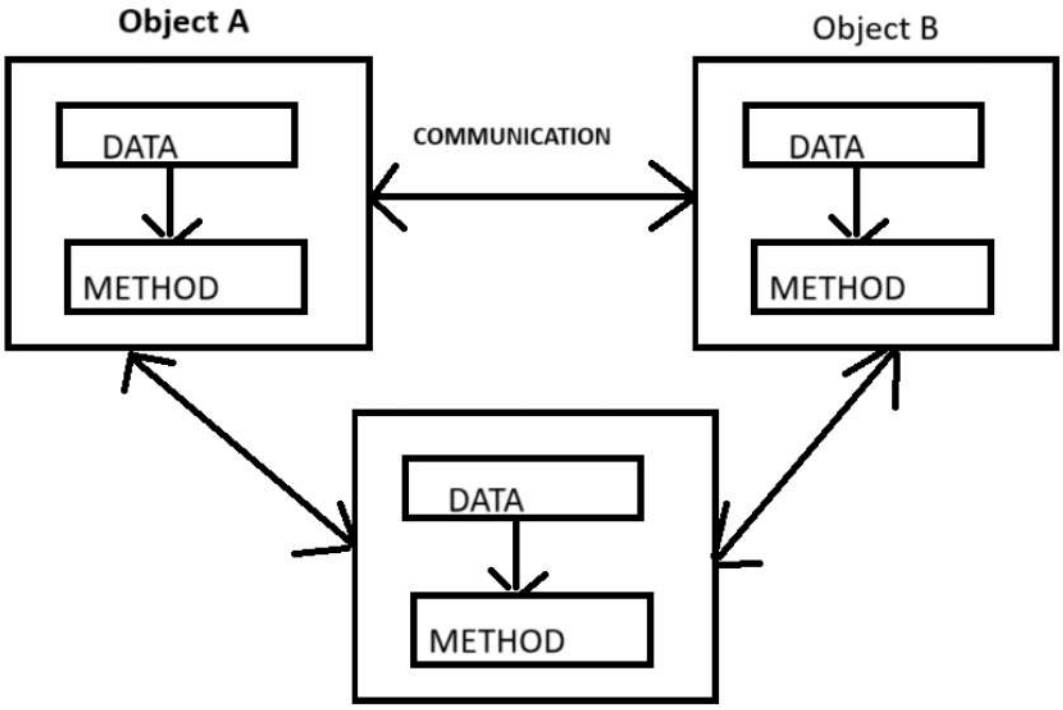
Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Programs organized around objects, grouped in classes. An object consists of data and operations that can be performed on that data called methods.

Object-oriented programming lets you **extend a program** without having to touch previously tested, working code.

Object-oriented programming provides great **flexibility, modularity and reusability** through encapsulation, inheritance, polymorphism and abstraction

OOP treats data as a critical element in the program development and does not allow it to flow freely around the system.




Some of the features of object oriented programming are:

• Emphasis is on data rather than procedure.

• 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 ties together in the data structure.

• Data is hidden and cannot be accessed by external methods.

• Objects may communicate with each other through methods.

• New data and methods can be easily added whenever necessary.

• Follows bottom up approach in program design

**Basic Concepts of Object Oriented Programming and OOPs Principles:**

The Basic Concepts that are extensively used in Object Oritned Programming are

- Objects
- Classes
- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

**Object:**

An object is anything that really exists in the world and can be distinguished from others.Every object has properties(or attributes) and can perform certain actions(or behaviour)

Ex:

A MotorCycle object can have make, color, enginestate etc. and can perform actions like startengine, accelerate, stop etc.

**Class:**

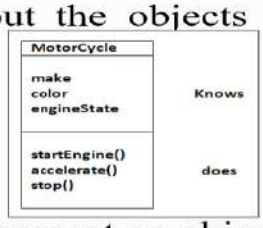
It is possible that some objects may have similar properties and actions. Such objects belong to same category called a class. While writing a program in an object-oriented language, define classes of objects. Then instances of these classes called objects were created.

**A class is a template for creating multiple objects with similar features**. An object is an instance of a class that has some state and behavior.

➢ When you design a class, think about the objects that will be created from that class type.

➢ Think about:

- things the object knows
- things the object does

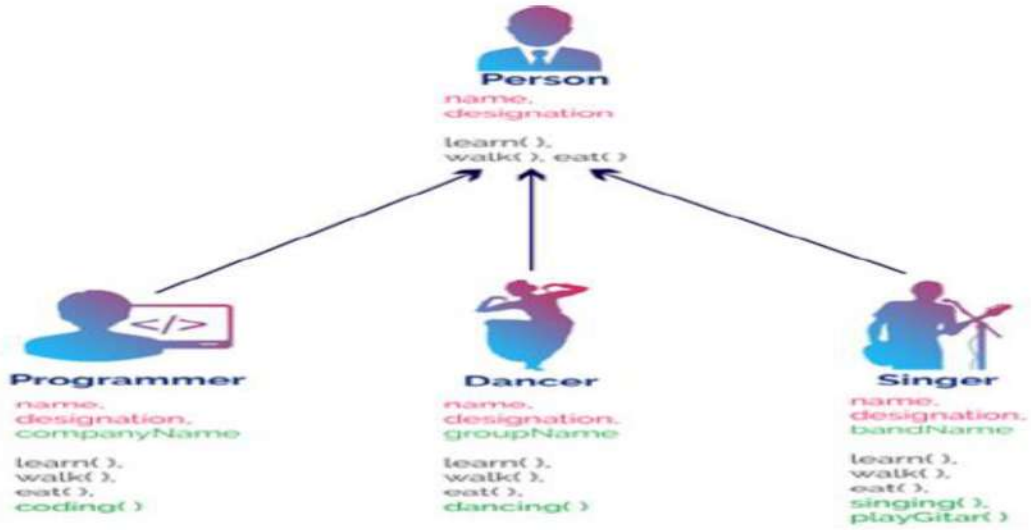



➢ Things an object knows (attributes) represent an object's state (the data)

➢Things an object can do are called methods represent object's behavior

**Inheritance:**

Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance. Inheritance helps in **software re-use**, because programmers can create methods that do a specific job exactly once and also **Redundancy of the code is reduced.**




**Polymorphism:**

Polymorphism means the ability to take more than one form. Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. Polymorphism is extensively used in implementing inheritance.

**Encapsulation:**

Wrapping up of data and methods that manipulate the data into a single unit is known as encapsulation.The Data is not accessible to the outside world and only the methods(which are wrapped) can access it and that **keeps data both safe from outside interference and misuse.**

**Abstraction:**

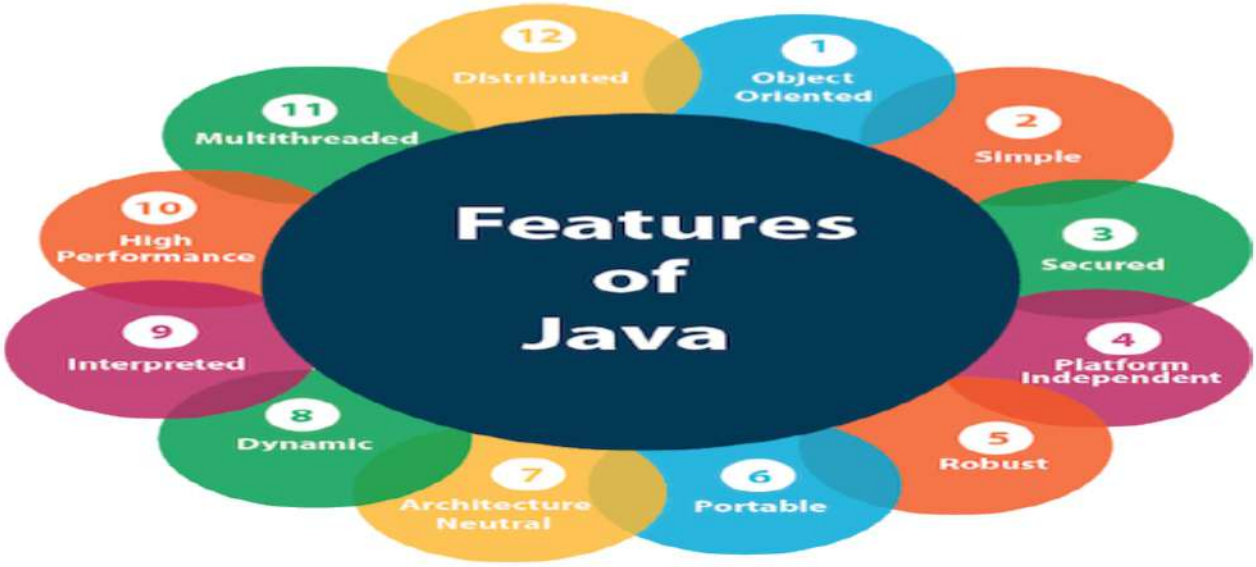
Abstraction is a process of **hiding all the unnecessary implementation details and exposing only the required functionalities**

## Introduction to Java:

- Java is a High level, General purpose, object oriented and platform Independent and popular **Programming Language**
- Java was conceived by James Gosling, Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan at Sun Microsystems, Inc. in 1991
- This language was initially called "Oak," but was renamed "Java" in 1995.
- On January 23rd 1996, JDK 1.0 version was released. Today more than 10 million developers use Java and more than 15 billion devices run· Java.
- It is acquired by oracle corporation in 2010.

## Features of Java ( Java Buzzwords):

The following are the features of Java




### 1. Object Oriented Language:

Java is an Object Oriented Programming Language. Object-oriented programming (OOP) is a programming paradigm based upon objects and classes

Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

Programs organized around objects, grouped in classes. An object consists of data and operations that can be performed on that data called methods.

Object-oriented programming lets you **extend a program without having to touch previously tested, working code**

Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Basic concepts of OOP includes

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

**2. simple:**

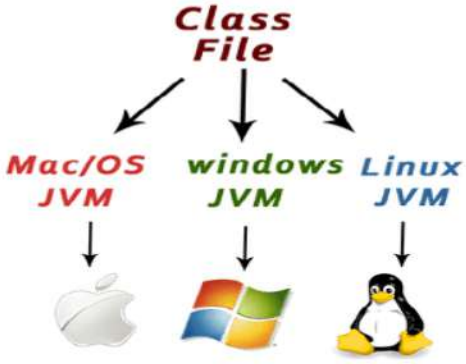
- Java is very easy to learn, and its syntax is simple, clean and easy to understand.
- Java language is a simple programming language because
    - Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
    - There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.
    - Java contains rich set of API (Application Programming Interface) in order to develop various applications to meet industry requirements

**3. Secured:**

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

**4. Platform Independent:**

- Java is platform independent programming Language, because, Java allows its applications to compile on one operating system and to execute on another operating system.
- Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.
- Java code is compiled by the compiler and converted into bytecode.




- This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

**5. Robust**

- Java programs are strong and they don't crash easily like a C or C++ program.
- Java is robust because
  - Java has got excellent inbuilt exception handling features.
  - It uses strong memory management.
  - Java provides automatic garbage collection

**6. Portable:**

Java Programs can be easily moved from one computer system to another. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java Programs.

**7. Architecture-Neutral:**

- It is a property in which the language or a technology runs based on the architecture of the processor.
- The languages like C and C++ runs based on the architecture of the processor. Those languages are called architecture dependent languages.
- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
- In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java

**8. Dynamic:**

- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand.

**9. Interpreted:**

- Java enables the creation of cross-platform programs by compiling into an intermediate representation called Java bytecode. This byte code can be downloaded and interpreted by the interpreter in JVM. So, in java, both compiler and Interpreter will be used for the execution.

**10. High Performance:**

JAVA is high performance programming language due to its rich set of features like Platform independent, Arch Nuetral, Portable, Robust, Dynamic,.....

- Java Performance is impressive mainly due to use of intermediate bytecode.
- Java architecture is also designed to reduce overheads during runtime. Further incorporation of multithreading enhances the overall execution speed of the program

**11. Multi-threaded:**

- Multi-threading is the ability of an application to execute more than one task at a time.
- Java programs can deal with many tasks at once by defining multiple threads.
- JVM uses several threads to execute different blocks of code to execute several tasks at once. Creating multiple threads is called 'multithreaded'.

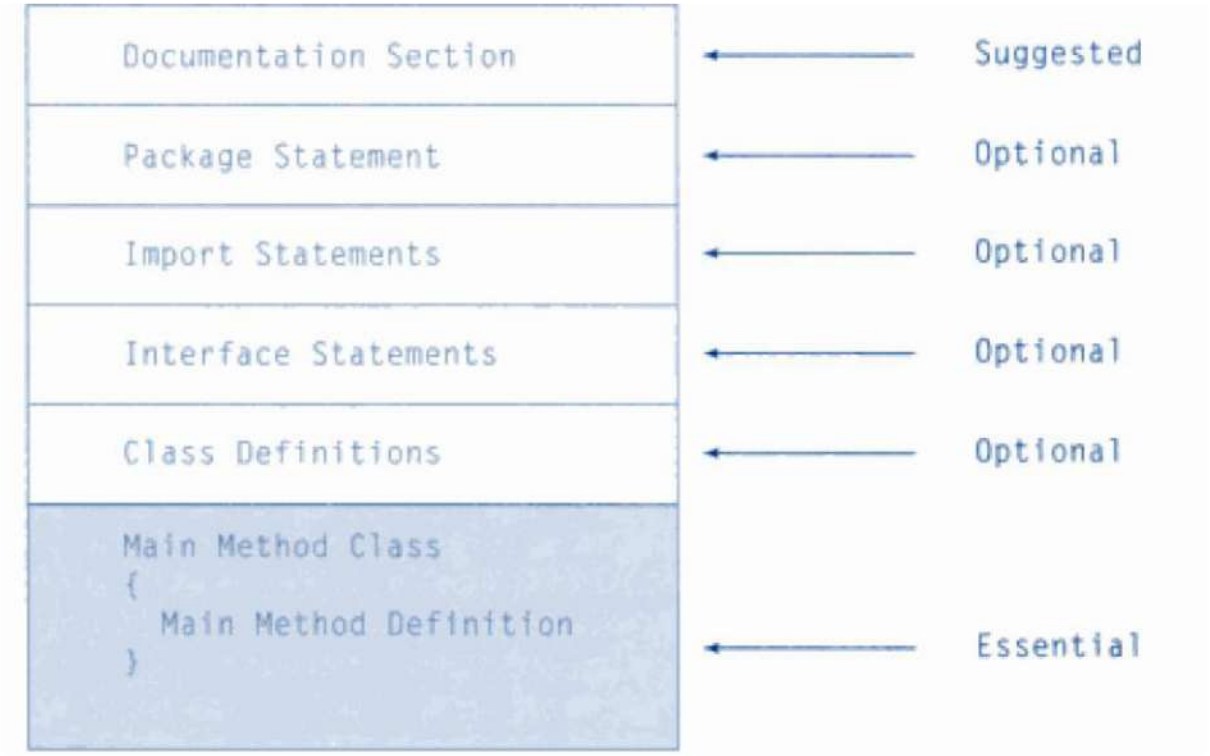
**12. Distributed**

Java is considered as a distributed language because it has inbuilt support for developing distributed systems and applications.

Java provides a comprehensive networking API, making it easy to develop applications tat communicate over networks, which is essential for distributed systems.

**Program Structure in Java:**

Every Java program may contain one or more sections as shown below.




**Documentation Section:**

Java documentation is crucial for understanding and maintaining code. The documentation section typically includes comments and JavaDoc comments to describe the functionality, parameters, return values, and other details of the classes, methods, and variables.

**Types of Comments in Java**

**a. Single-line Comments**:

These comments are for marking a Single line as a comment. These comments start with double slash symbol / / and after this, whatever is written till the end of the line is taken as a comment

Ex:

```
// program that displays Welcome to java
```

**b. Multi-line Comments**:

These comments are used for representing several lines as comments. These comments start with / * and end with */. In between / * and */, whatever is written is treated as a comment.

Ex:

/* this is

multiline comment*/

**c. JavaDoc Comments**:

**Javadoc** is a tool that generates HTML documentation from comments in your Java source code. These comments, called **javadoc comments**, are special comments that follow a specific format.

A javadoc comment starts with /** and ends with */

Ex:

```
/**
 * This class represents a simple calculator.
 *
 * @author Your Name
 * @version 1.0
 */
public class Calculator {
    // ...
}
```

**Package Statement Section:**

Package statement declares a package name and informs the compiler that the classes and interfaces defined here belongs to that package.

Ex:

The statement

package com.example.myapp;

declares that the classes and interfaces which are defined are part of the com.example.myapp package

**Import Statement Section:**

Import statement instructs the interpreter to load the classes and interfaces from the packages.

Ex:

The statement

```
import java.util.Scanner;
```

allows the program to use the Scanner class from the java.util package.

The main intention of "import" statement is to make available classes and interfaces of a particular package into the present JAVA file in order to use in present java file

**Interface Statement Section:**

An interface is like a class containing constants, abstract methods, default methods, static methods, and nested types. Interfaces cannot contain instance fields or constructors.

**Class Definition Section:**

A Java program must contain at least one class definition. The class declaration includes the class name, the class body, and optionally extends and implements clauses if the class is inheriting from a superclass or implementing interfaces.

**Main Method class section:**

Java program must have a main method in at least one of its classes, and that main method is the entry point for the program's execution.

The main method is the entry point for any Java program, and it's called when the program starts execution. It's where the program's logic begins, and it's typically where you'll find the code that creates objects, calls methods, and performs other tasks to get the program running.

Ex:

```
// This program prints Welcome to Java!
public class Welcome
{
   public static void main(String[] args)
   {
            System.out.println("Hello, world!");
   }
}
```

Output:

```
Hello, world!
```

**Note:**

1) The main method is a special method in Java that serves as the entry point for any standalone Java application. It serves as the initial point from where the Java Virtual Machine (JVM) begins executing the application. Without a main method, a Java program cannot run.

The signature of the main method in Java is:

```
public static void main(String[] args)
{
    //statements
}
```

Explanation of components:

a) **public:** This keyword makes the method accessible from anywhere.

b) **static:** This keyword allows the method to be called without creating an instance of the class. Since the JVM needs to call the main method before any objects exist, it must be static.

c) **void:** This indicates that the main method doesn't return any value.

d) **main:** This is the specific name of the method that the JVM looks for.

e) **String[] args:** This parameter is an array of strings that can be used to pass command-line arguments to the program.

Note:

2) It is not possible to declare more than one package declaration statement with in a single java file and package declaration statement must be first statement.

3) To provide package names, JAVA has given a convention like to include our company domain name in reverse in package names.

Ex: com.bytexl;

4) In a single java file, we can provide atmost one package declaration statement, but, we can provide any number of import statements.

import java.io.*;

import java.util.*

import java.sql.*;

## Compiling/running a program:

Before you run your programs, you must *compile* them. Java Development Kit (JDK) includes a Java compiler. The Java compiler converts your source code into a format named *byte code* that can be executed on many different kinds of computers. bytecode can be executed by the Java Virtual Machine (JVM).

### 1. Writing the Java Source Code

First, write the Java program in a text editor or an Integrated Development Environment (IDE) and save it with a .java extension. For example, let's create a simple Java program.

**File: MyFirstApp.java**

```
public class MyFirstApp {

   public static void main (String[] args) {
     System.out.println("I Rule!");
     System.out.println("The World");
   }

}
```

## 2. Compiling the Java Source Code

Compiling is the process of transforming the Java source code into bytecode, which the JVM can execute. The Java compiler (javac) performs this task.

Steps to Compile:

- Open a terminal or command prompt.
- Navigate to the directory where your .java file is located.
- Run the javac command followed by the file name as below

```
javac MyFirstApp.java
```

If there are no errors in your code, the compiler will produce a bytecode file named **MyFirstApp.class** in the same directory.

## 3. Running the Compiled Java Program

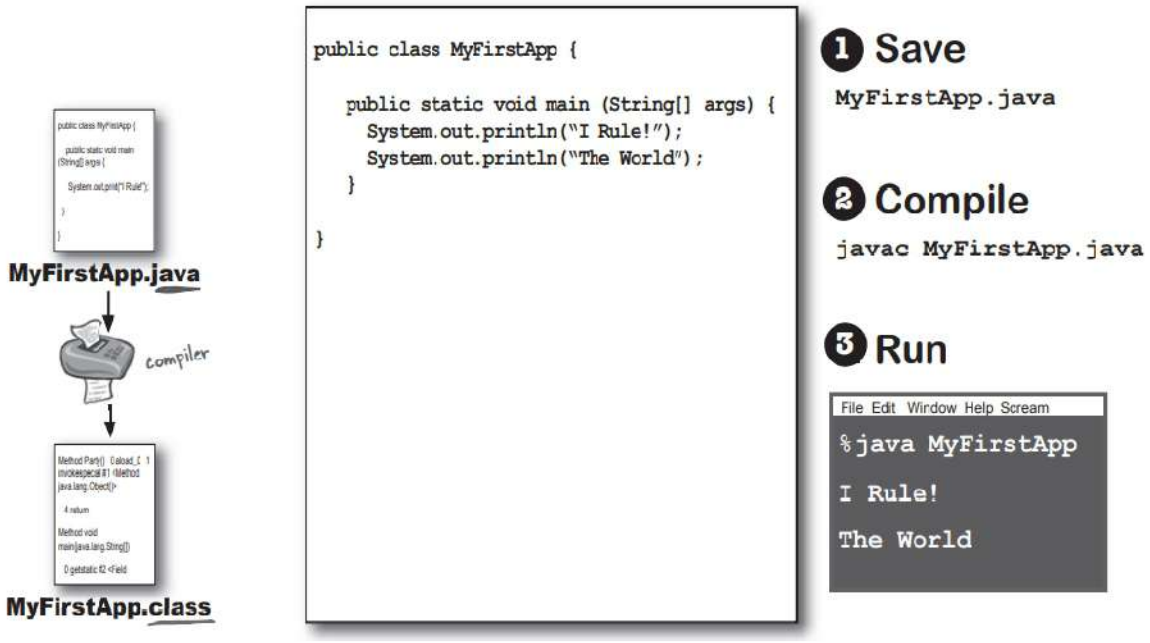
To run the compiled bytecode, you use the Java interpreter (java), specifying the name of the class containing the main method. Do not include the .class extension.

Steps to Run:

- Navigate to the directory containing your .class file.
- Run the java command followed by the class name.

```
java MyFirstApp
```

This command tells the JVM to load the MyFirstApp class and start executing from its main method. It is illustrated in the following figure.



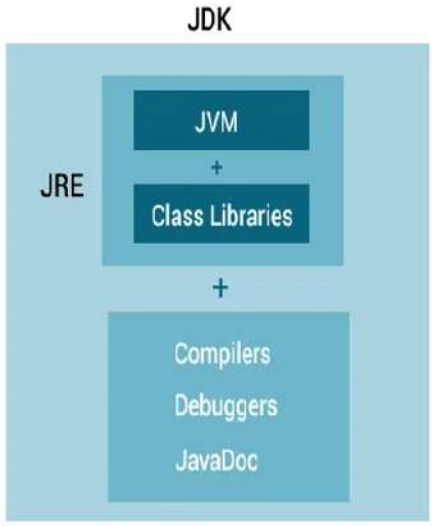

## Java JDK, JRE and JVM:

JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.

In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).


JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.


**JVM (Java Virtual Machine)**




When you run the Java program, Java compiler first compiles your Java code to bytecode( .class file).

Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

**Tokens:**

In Java, tokens are the smallest elements of a program that are meaningful to the compiler. The Java compiler uses tokens to understand and interpret the code.

Types of tokens in java:

The various types of java tokens are

- Keywords
- Identifiers
- Literals
- Operators
- Special symbols

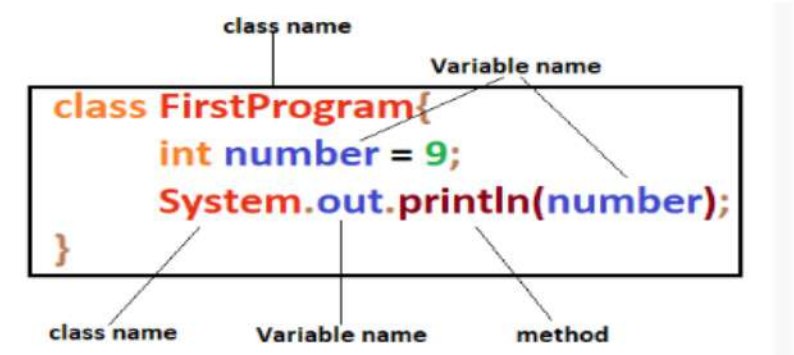
Keywords:

Keywords are reserved words that have a predefined meaning in Java. They cannot be used as identifiers (names for variables, classes, methods, etc.). Java keywords include:

| | | | |
|---|---|---|---|
| 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:

Identifiers are the user defined names being given to various programming elements such as variables, objects, classes, methods, packages, interfaces etc.




Identifier rules:

They can have alphabets, digits, underscore and dollar symbol

They must not begin with digit

Keywords cannot be used as identifiers

whitespace characters are also not allowed.

Note:

a) java is a case sensitive language i.e., uppercase and lowercase characters are treated as different
b) The keywords const and goto are reserved, even though they are not currently used. true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

Literals:

Any constant value in the program is called a literal. Literals in Java are used to represent value or assign variables value.

Ex:- 9, 9.5, 'a', "a", true, false, null etc.

Special Symbols(or separators):

Special symbols in Java are a few characters which have special meaning known to Java compiler and cannot be used for any other purpose.

**Parentheses**: `()` - Used for grouping expressions and parameter lists in method definitions.

**Braces**: `{}` - Used to define blocks of code, such as methods, classes, and loops.

**Brackets**: `[]` - Used for array declaration and access.

**Semicolon**: `;` - Used to terminate statements.

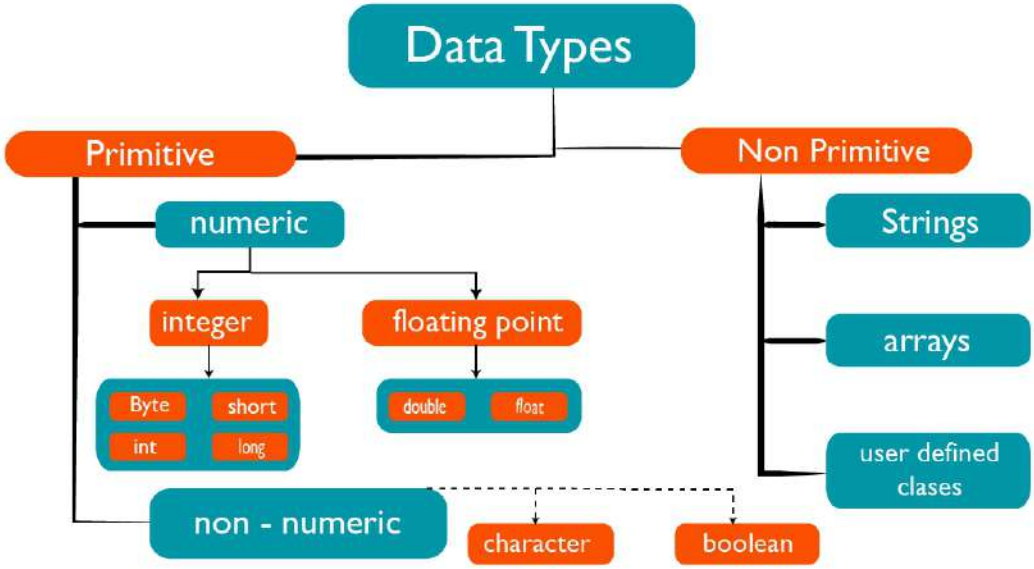
**Comma**: `,` - Used to separate items in a list.

**Period**: `.` - Used to access members of a class or package.

**Datatypes:**

A data type specifies the type of data that a variable can hold in a programming language. It defines the operations that can be performed on the data, the way data is stored, and the range of values that can be stored. In Java, data types are crucial for defining variables and for determining the behaviour of operations performed on these variables.

In java, the datatypes are classified as shown in the following figure.




primitive data types are the most basic data types and are not objects. They represent simple fundamental values and are built into the language. Java provides eight primitive data types which are given below.

**Integer Data types**:

Integer data types are used to store whole numbers i.e., numbers that do not have any decimal point or fractional point. In java, Integer datatypes are again divided into four types byte,short,int and long. The difference between these datatypes is the number of bytes to occupy and the range of values which are given as follows.

| *Type* | *Size* | *Minimum value* | *Maximum value* |
|---|---|---|---|
| byte | One byte | -128 | 127 |
| short | Two bytes | -32, 768 | 32, 767 |
| int | Four bytes | -2, 147, 483, 648 | 2, 147, 483, 647 |
| long | Eight bytes | -9, 223, 372, 036, 854, 775, 808 | 9, 223, 372, 036, 854, 775, 807 |

**Floating point datatypes:**

These datatypes are used to store the numbers with decimal point.In java, floating point datatypes are again divided into two types float and double.

The difference between these datatypes is the number of bytes to occupy and the range of values which are given as follows

| *Type* | *Size* | *Minimum value* | *Maximum value* |
|---|---|---|---|
| float | 4 bytes | 3.4e-038 | 3.4e+038 |
| double | 8 bytes | 1.7e-308 | 1.7e+308 |

**char datatype:**

The char data type is used to represent a single character, such as a letter, digit, or symbol. The char data type is a single 16-bit(or 2 bytes) Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).

**boolean datatype:**

In Java, the boolean data type is a primitive data type that represents a logical value, which can be either: true or false. It uses 1-bit of storage(although technically it may use more memory which is JVM specific)

**Non-primitive data types**, also known as reference data types, are more complex than primitive data types. They are defined by the user or provided by the Java standard library.

Note:

Java thinks any value with a decimal point is a double and allocates 8 bytes. To allocate 4 bytes, attach f or F to the value as shown below

```
float f=32.5f;
```

**Variables:**

A variable is a named location in memory that may be used to store a data value The data represented by a variable can change during the execution of the program. The variable is associated with a particular data type.

Variable names:

Variable names are identifiers used to name variables. A variable name must be chosen in a meaningful way so as to reflect its nature in the program. The following rules must be followed while naming variables.

1. Variable names may be formed with alphabets, digits, dollar or a underscore character.

2. Variable names must noy begin with a digit.

3. Keywords are not used as variable names.

4. white space is not allowed

5. Uppercase and lowercase are significant i.e., the variable Total is not the same as total or TOTAL.

Variable Declaration:

A variable must be declared before it is used in the program. This declaration tells the compiler what the variable name is and what type of data it can hold. A variable can be used to store a value of any data type. The syntax for declaring a variable is

datatype var1,var2,.....,varn ;

Where datatype is any valid data type and var1, var2,..., varn are variable names. A declaration statement must end with a semicolon (;).

Ex:

```
byte x = 10;

short y = 20;

int z = 30;

long w = 40L;

float x = 3.14f;

double y = 3.14;

char c = 'A';

char d = 65; // equivalent to 'A'

boolean isAdmin = true;

boolean isStudent = false;
```

Type Casting:

The process of converting the value of one data type (int, float, double, etc.) to another data type is known as typecasting.

Conversion types:

a) Widening type conversion(or) Implicit type casting

The process of converting data from lower data type to higher data type is called as Implicit Type Casting as shown below.

- byte -> short, int, long, float, double
- short -> int, long, float, double
- char -> int, long, float, double
- int -> long, float, double
- long -> float, double
- float -> double

This type of casting is done automatically by the Java compiler and does not require any special syntax.

b) Narrowing type conversion (or) explicit type casting:

Explicit casting, also known as narrowing conversion, occurs when a larger data type is converted to a smaller data type as shown below.

- byte -> char
- short -> byte, char
- char -> byte, short
- int -> byte, short, char
- long -> byte, short, char, int
- float -> byte, short, char, int, long
- double -> byte, short, char, int, long, float

This type of casting must be done manually by the programmer because there is a possible of losing data as given below.

(type) expression;

Ex:

```
double num=10.99;

int data=(int) num;   //explicit casting
```

**Scope of variables:**

In java, there are 3 types of variables

1. **Instance Variables**
2. **Class Variables (Static Variables)**
3. **Local Variables**

**1. Instance Variables**

Instance variables are non-static variables declared in a class outside any method, constructor, or block. They are associated with individual objects of a class and hold data specific to each object. Each object of a class has its own copy of instance variables.

They are stored in the heap memory, which is the area of memory used for dynamic memory allocation.

**Scope**: The scope of an instance variable is the entire class. They are accessible from all non-static methods of the class.

**Lifetime**: Instance variables are created when an object is instantiated using the new keyword and destroyed when the object is destroyed.

**Example**:

```
public class Employee {
    // Instance variable
    private String name;
    // Constructor
    public Employee(String name) {
        this.name = name;
    }
    // Getter method
    public String getName() {
        return name;
    }
}
```

Note:

Accessing an instance variable from a static context is not allowed in Java and will result in a compilation error.

**2. Class Variables (Static Variables)**

Class variables are variables declared with the static keyword inside a class, but outside any method, constructor, or block. static variables belong to the class itself rather than to specific objects of the class.

There's only one copy of a class variable shared by all instances of the class. class variable can be accessed directly using the class name without creating an object.

They are stored in the method area (part of the heap memory in some JVM implementations)

**Scope**: The scope of a class variable is the entire class. They can be accessed from any static or non-static method of the class.

**Lifetime**: Class variables are created when the class is loaded first in to the memory and destroyed when the class is unloaded.

**Example**:

```
public class Employee {
    // Static variable
    private static int employeeCount;
    // Constructor
    public Employee() {
        employeeCount++;
    }
    // Static method
    public static int getEmployeeCount() {
        return employeeCount;
    }
}
```

### 3. Local Variables

Local variables are variables declared inside a method, constructor, or block.

They are stored in the stack memory, which is the area of memory used for method execution and local variable storage.

**Scope**: The scope of a local variable is limited to the method, constructor, or block in which it is declared.

**Lifetime**: Local variables are created when the method, constructor, or block is entered and destroyed when it is exited.

**Example**:

```
public class Example {
    public void display() {
        // Local variable
        int number = 10;
        System.out.println("Number: " + number);
    }
}
```

**Note:**

Unlike instance variables, local variables don't have default values. They must be explicitly initialized before use.

Symbolic constants:

In Java, a symbolic constant is a named constant value that remains unchanged throughout the program.

A symbolic constant is a final variable that is typically defined with the static keyword and often follows naming conventions such as all uppercase letters with underscores. Symbolic constants are used to represent fixed values that are shared across all instances of a class.

Syntax:

static final datatype CONSTANT_NAME = value;

Ex:

a) public static final double PI = 3.14159;
b) public static final int PASS_MARK=50;

symbolic constants provide a way to define fixed values that can be used consistently throughout your Java code, leading to more readable, maintainable, and error-free programs.

**Final variables:**

A variable declared with the final keyword, whose value cannot be changed after initialization is called a final variable.

Final variables can be instance variables, local variables, or static variables.

A final variable must be initialized when it is declared or in the constructor (if it's an instance variable).

Final variables Can be of any data type (primitive or reference).

```
public class FinalPrimitiveExample {
    public static void main(String[] args) {
        final int MAX_HEIGHT = 100;
        // MAX_HEIGHT = 200; // This will cause a compile-time error
        System.out.println("MAX_HEIGHT: " + MAX_HEIGHT);
    }
}
```

| Final Variables | Symbolic constants |
|---|---|
| Final Variables Can be instance variables, local variables, or static variables. They can be used for values that need to be assigned only once but might differ between instances. | Symbolic Constants are typically static final variables, representing fixed values that are the same across all instances of a class. |
| Final Variables Can be initialized at the point of declaration or in the constructor (if it's an instance variable). | Symbolic Constants are Usually initialized at the point of declaration. |
| Final Variable Naming follows standard variable naming conventions | Symbolic Constants typically named using uppercase letters with underscores. |

**Ex:**

```
class Example {
    final int instanceVariable; // Must be initialized in constructor
    public static final int CONSTANT = 100; // Symbolic constant

    Example(int value) {
        this.instanceVariable = value; // Initialized in constructor
    }

    void someMethod() {
        final int localVariable = 10; // Final local variable
    }
}
```

**Formatted Output with printf() Method:**

The printf() method in Java is used for formatted output. It allows you to control the appearance of your output by using format specifiers i.e., it allows you to format strings, numbers, and other data types in a way that is similar to the printf function in C/C++.

Syntax:

```
System.out.printf(formatstring, arguments);
```

Where formatstring is a string containing format specifiers and text and arguments is a list of arguments to be formatted and inserted into the format string.

format specifiers:

Format specifiers start with a percentage sign (%) followed by a character that specifies the data type and define how the arguments are formatted and displayed. Here are some common format specifiers:

| Format Specifier | Data Type | Description |
|---|---|---|
| %d | int | Integer |
| %f | float, double | Floating-point number |
| %s | String | String |
| %c | char | Character |
| %b | boolean | Boolean |
| %n | - | Newline |

Ex1:

```
int age = 25;
double price = 99.99;
String name = "Pavan";

System.out.printf("Name: %s, Age: %d, Price: %.2f%n", name, age, price);
```

Output:

```
Name: Pavan, Age: 25, Price: 99.99
```

Ex2:

```
double pi = 3.14159265;

System.out.printf("Pi with 6 decimal places: %.6f%n", pi);
System.out.printf("Pi with 8 characters width: %8.2f%n", pi);
```

Output:

```
Pi with 6 decimal places: 3.141593
Pi with 8 characters width:     3.14
```

**static keyword**:

The static keyword in Java signifies that a member is a part of the class definition, and is shared by all instances, rather than being unique to each instance

static keyword can be applied to variables, methods, and blocks.

static methods:

Static methods are methods that are declared with the static keyword.

**Characteristics:**

- Can be called without creating an instance of the class.
- Can only access static members (variables and methods) directly.
- Can't access instance variables or methods (since there's no specific object associated with it).
- Often used for operations that don't require any data from instances of the class.

```
public class MathUtil {
    // Static method
    public static int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int result = MathUtil.add(5, 3);
        System.out.println("Sum: " + result); // Output: 8
    }
}
```

Output:

Sum: 8

Static block:

static blocks are blocks of code that are executed when the class is loaded into memory. They are used for static initialization of a class. **Static blocks are executed only once, when the class is loaded.**

Static blocks are typically used to initialize static variables or perform operations that need to be executed once at the time of class loading.

Note1:

A static block cannot directly access non-static members of the same class. To access non-static members within a static block, you must first create an instance of the class and then use the resulting object reference to access the non-static members.

Note2:

'this' keyword cannot be used within a static block, as it refers to an instance of the class, which is not applicable in a static context."

```
class A
{
   int i=10;
   static int j=20;
   static
   {
      System.out.println("SB-A");
      //System.out.println(i);//-------->Error
      A a = new A();
      System.out.println(a.i);
      System.out.println(j);
      //System.out.println(this.j);-------->Error
   }
}
 public class Test
 {
   public static void main(String[] args) {
   A a = new A();
 }
}
```

Output:

```
SB-A
10
20
```

Operators:

An operator is a symbol that tells the computer to perform certain mathematical (or) logical manipulations. Operators are used to manipulate data and variables. The variables and constants can be combined together by various operators to form expressions. The data items that operators act upon are called operands (variables and constants).

Java Provides rich set of operators to perform different operations and are classified into several categories. They are

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment & Decrement Operators
- Conditional Operator
- Bitwise Operators
- Special Operators

Arithmetic Operators:

The Arithmetic operators are used to perform arithmetic operations like addition, subtraction, multiplication and division over numeric values by constructing arithmetic expressions.

The Arithmetic operators in 'Java' language are given below

| Operator | Meaning |
|---|---|
| + | **Addition (or) Unary plus** |
| - | **Subtraction (or) Unary Minus** |
| * | **Multiplication** |
| / | **Division** |
| % | **Modulus operator** |

**Division Operator(/)**

The division operator / is used to divide one number by another. It can be used with both integer and floating-point types, and the result varies depending on the types of operands

**Modulus Operator (%)**

The modulus operator % is used to find the remainder of the division of one number by another. It works with both integer and floating-point types, but the result is an integer remainder when both operands are integers, or a floating-point remainder when at least one operand is a floating-point number.

Ex:

```
import java.util.Scanner;
public class Arithmetic
{
	public static void main(String[] args) {
		int a;
		double b;
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter two numbers:");
		a = sc.nextInt();
		b = sc.nextDouble();
		System.out.println("Results are:");
		System.out.println(a+b);
		System.out.println(a-b);
		System.out.println(a*b);
		System.out.println(a/b);
		System.out.println(a%b);
	}
}
```

Output:

```
Enter two numbers:
8 5.0
Results are:
13.0
3.0
40.0
1.6
3.0
```

Relational Operators:

The relational operators are used to compare two quantities. The comparison is made to take a decision depending on the relationship between the two quantities.

A relational operator accepts two operands and returns a truth value giving information about the relationship between the operands.

The relational operators in 'Java' language are given below

| Operator | Meaning |
|---|---|
| < | is less than |
| <= | is less than or equal to |
| > | is greater than |
| >= | is greater than or equal to |
| = = | is equal to |
| != | is not equal to |

```
import java.util.Scanner;
public class Relation
{
	public static void main(String[] args) {
		int a,b;
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter two numbers:");
		a = scan.nextInt();
		b = scan.nextInt();
		System.out.println("a > b -> "+ (a>b));
		System.out.println("a >= b -> "+ (a>=b));
		System.out.println("a < b -> "+ (a<b));
		System.out.println("a <= b -> "+ (a<=b));
		System.out.println("a == b -> "+ (a==b));
		System.out.println("a != b -> "+ (a!=b));
	}
}
```

Output:

```
Enter two numbers:
8 5
```

```
a > b -> true
a >= b -> true
a < b -> false
a <= b -> false
a == b -> false
a != b -> true
```

Logical Operators:

The logical operators are used when we want to test more than one condition and make decisions. An example

a > b && x == 10

java Language has the following three logical operators

| Operator | Meaning |
|---|---|
| && | logical AND |
| \|\| | logical OR |
| ! | logical NOT |

**Logical AND (&&)**

The logical AND operator && returns true if and only if both of its operands are true. If either or both operands are false, it returns false.

**Logical OR (||)**

The logical OR operator || returns true if at least one of its operands is true. If both operands are false, it returns false.

**Logical NOT (!)**

The logical NOT operator ! is a unary operator that inverts the boolean value of its operand. It returns true if the operand is false and false if the operand is true.

```
import java.util.Scanner;
public class Logical
{
        public static void main(String[] args) {
        int m=40, n=20, o=20, p=30;
```

```
        if ( m > n && m != 0 )
        {
                System.out.println("&& Operator : Both conditions are true");
        }
        if ( o > p || p != 20 )
        {
                System.out.println("|| Operator : Only one condition is true");
        }
        if( ! ( m > n && m != 0 ) )
        {
                System.out.println("! Operator : Both conditions are false\n");
        }
        else
        {
                System.out.println("! Operator : Both conditions are true. But,
status is inverted as false");
        }
        }
}
```

Output:

```
&& Operator : Both conditions are true
|| Operator : Only one condition is true
! Operator : Both conditions are true. But, status is inverted as false
```

Assignment Operators:

Assignment operator (=) are used to assign the result of an expression to a variable.

For ex, consider the statement x=3;

This statement assigns the value 3 to variable x. In addition, Java has a set of shorthand assignment operators of the form

v op=exp;

Where v is a variable, exp is an expression and op is a binary arithmetic operator. The operator op= is known as the shorthand assignment operator.

Ex: x+=3; is same as the statement x=x+3;

| Statement with shorthand operator | Statement with simple assignment operator |
|---|---|
| a +=1 | a = a + 1 |
| a -=1 | a = a – 1 |
| a *= n+1 | a = a * (n+1) |
| a /= n+1 | a = a / (n+1) |
| a %=b | a = a % b |

Increment & Decrement Operators:

Increment operator(++) increases value of the operand by 1 where as decrement operator(--) decreases the value of the operand by 1.

When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one.

Consider the following:

**m = 5;**

**y = ++ m;**

In this case, the value of y and m would be 6.

When prefix ++ (or --) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.

Consider the following:

**m = 5;**

**y = m ++;**

Then, the value of y would be 5 and m would be 6.

Conditional Operator:

Conditional operator (?:) is a ternary operator and the syntax of the conditional operator is

**exp1? exp2: exp3**

Where exp1, exp2 and exp3 are expressions.

**It works as follows:**

- exp1 is evaluated first .
- If it is nonzero(true) then *exp2* is evaluated and becomes the result of the expression.
- If it is zero(false) then *exp3* is evaluated and becomes the result of the expression.

BITWISE OPERATORS:

Bitwise operators are used for manipulation of data at bit level. These operators will work on only integers. The Bitwise operators in java language are given below

| Operator | Meaning |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise Exclusive OR |
| ~ | One"s Complement |
| << | Shift left |
| >> | Shift right |

**Bitwise AND operator (&):**

The bitwise AND operator is a binary operator that requires two integral operands. It does a bit-by-bit comparison between the two operands. When both bits in two operands are 1 the result bit is set to 1 otherwise the result bit is set to 0. The result of bit-by-bit comparison is shown in the following table

| First operand bit (p) | Second operand bit (q) | Result bit (p & q) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Example:

Consider int x=4, y=5

Binary equivalent of x=4: 0000 0000 0000 0100

Binary equivalent of y=5: 0000 0000 0000 0101

---

x&y : 0000 0000 0000 0100

---

**Bitwise OR operator (|):**

The bitwise OR operator is a binary operator that requires two integral operands. It does a bit-by-bit comparison between the two operands. When both bits in two operands are 0 the result bit is set to 0 otherwise the result bit is set to 1. The result of bit-by-bit comparison is shown in the following table

| First operand bit (p) | Second operand bit (q) | Result bit (p \| q) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Example:

Consider int x=4, y=5

Binary equivalent of x=4: 0000 0000 0000 0100

Binary equivalent of y=5: 0000 0000 0000 0101

---

x|y : 0000 0000 0000 0101

---

**Bitwise Exclusive OR operator (^):**

The bitwise Exclusive OR operator is a binary operator that requires two integral operands. It does a bit-by-bit comparison between the two operands. When both bits in two operands are same the result bit is set to 0 otherwise the result bit is set to 1. The result of bit-by-bit comparison is shown in the following table

| First operand bit (p) | Second operand bit (q) | Result bit (p \| q) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Example:

Consider int x=4, y=5

```
Binary equivalent of x=4:   0000 0000 0000 0100
Binary equivalent of y=5:   0000 0000 0000 0101
                            ------------------------------
          x^y           :   0000 0000 0000 0001
                            ------------------------------
```

**One's complement operator(~):**

The one's complement operator is a unary operator applied to an integral operand.

It complements the bits in the operand i.e., it reverses the bit value. The result bit is 1 when the operand bit is 0 and the result bit is 0 when the operand bit is 1.it is shown in the following table.

| Operand bit (p) | Result bit (~p) |
|---|---|
| 0 | 1 |
| 1 | 0 |

Example:

Consider int x=4

Binary equivalent of x=4: 0000 0000 0000 0100

~x: 1111 1111 1111 1011

**Shift Left operator(<<):**

Shift left operator move bits towards left. The shift left operator is a binary operator that requires two integral operands. The first operand is the value to be shifted. The second operand specifies the number of bits to be shifted.

Consider unsigned int x=4

Binary equivalent of x=4: 0000 0000 0000 0100

x<<2 : 0000 0000 0001 0000

**Shift Right operator(>>):**

Shift right operator move bits towards right. The shift right operator is a binary operator that requires two integral operands. The first operand is the value to be shifted. The second operand specifies the number of bits to be shifted.

Consider unsigned int x=4

Binary equivalent of x=4: 0000 0000 0000 0100

x>>2 : 0000 0000 0000 0001

```
public class BitWise
{
   public static void main(String[] args)
   {
      int x=4,y=5;
      System.out.println("x & y = "+ (x&y));
      System.out.println("x | y = "+ (x|y));
      System.out.println("x ^ y = "+ (x^y));
      System.out.println("~x = "+ (~x));
```

```
        System.out.println("x<<2 = "+ (x<<2));
        System.out.println("x>>2 = "+ (x>>2));
    }
}
```

Output:

```
x & y = 4
x | y = 5
x ^ y = 1
~x = -5
x<<2 = 16
x>>2 = 1
```

Special Operators:

instanceof operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as

( Object reference variable ) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true

```
public class InstanceOfExample {
    public static void main(String[] args) {
        String str = "Hello, World!";
        Integer num = 42;

        // Check if str is an instance of String
        boolean isString = str instanceof String; // true

        // Check if num is an instance of Integer
        boolean isInteger = num instanceof Integer; // true

        // Check if str is an instance of Object (which it is, because all classes in
Java inherit from Object)
        boolean isObject = str instanceof Object; // true

        // Check if num is an instance of Number
        boolean isNumber = num instanceof Number; // true
```

```
        System.out.println("Is str a String? " + isString);
        System.out.println("Is num an Integer? " + isInteger);
        System.out.println("Is str an Object? " + isObject);
        System.out.println("Is num a Number? " + isNumber);

    }
}
```

Output:

```
Is str a String? true
Is num an Integer? true
Is str an Object? true
Is num a Number? true
```

**new Operator:**

The new operator is used to create new instances of classes or arrays. It allocates memory for the new object or array and initializes it using a constructor.

**Syntax**

```
ClassName objectName = new ClassName();
```

Here ClassName is the name of the class you want to instantiate and objectName is the name of the variable that will reference the newly created object.

```
public class NewOperatorExample {
    public static void main(String[] args) {
        // Create a new instance of the String class
        String str = new String("Hello, World!");

        // Create a new instance of the Integer class
        Integer num = new Integer(42);

        // Create a new instance of an array of integers
        int[] numbers = new int[5]; // Array of 5 integers

        // Initialize the array elements
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = i + 1; // Assign values 1, 2, 3, 4, 5
        }
```

```
        // Print the array elements
        System.out.println("String: " + str);
        System.out.println("Integer: " + num);
        System.out.print("Array: ");
        for (int number : numbers) {
            System.out.print(number + " ");
        }
    }
}
```

Output:

```
String: Hello, World!
Integer: 42
Array: 1 2 3 4 5
```

**Operator Precedence and Associativity**

- The Precedence is used to determine how an expression evaluated when it involves more than one operator.
- There are distinct levels of precedence and an operator may belong to one of these levels.
- The operators with highest precedence are evaluated first.
- The operators with same precedence level are evaluated either from "Left to Right" Or "Right to Left", this is known as associativity property of an operator.
- The following table provides a complete list of operators, their precedence level, and their rules of association.

| Level | Operator | Description | Associativity |
|---|---|---|---|
| 16 | []<br>.<br>() | access array element<br>access object member<br>parentheses | left to right |
| 15 | ++<br>-- | unary post-increment<br>unary post-decrement | not associative |
| 14 | ++<br>--<br>+<br>-<br>!<br>~ | unary pre-increment<br>unary pre-decrement<br>unary plus<br>unary minus<br>unary logical NOT<br>unary bitwise NOT | right to left |
| 13 | ()<br>new | cast<br>object creation | right to left |
| 12 | * / % | multiplicative | left to right |
| 11 | + -<br>+ | additive<br>string concatenation | left to right |
| 10 | << >><br>>>> | shift | left to right |

| Level | Operator | Description | Associativity |
|---|---|---|---|
| 9 | < <=<br>> >=<br>instanceof | relational | not associative |
| 8 | ==<br>!= | equality | left to right |
| 7 | & | bitwise AND | left to right |
| 6 | ^ | bitwise XOR | left to right |
| 5 | \| | bitwise OR | left to right |
| 4 | && | logical AND | left to right |
| 3 | \|\| | logical OR | left to right |
| 2 | ?: | ternary | right to left |
| 1 | = += -=<br>*= /= %=<br>&= ^= \|=<br><<= >>= >>>= | assignment | right to left |

Expressions:

An expression is a combination of variables, constants, and operators arranged as per the syntax of the language

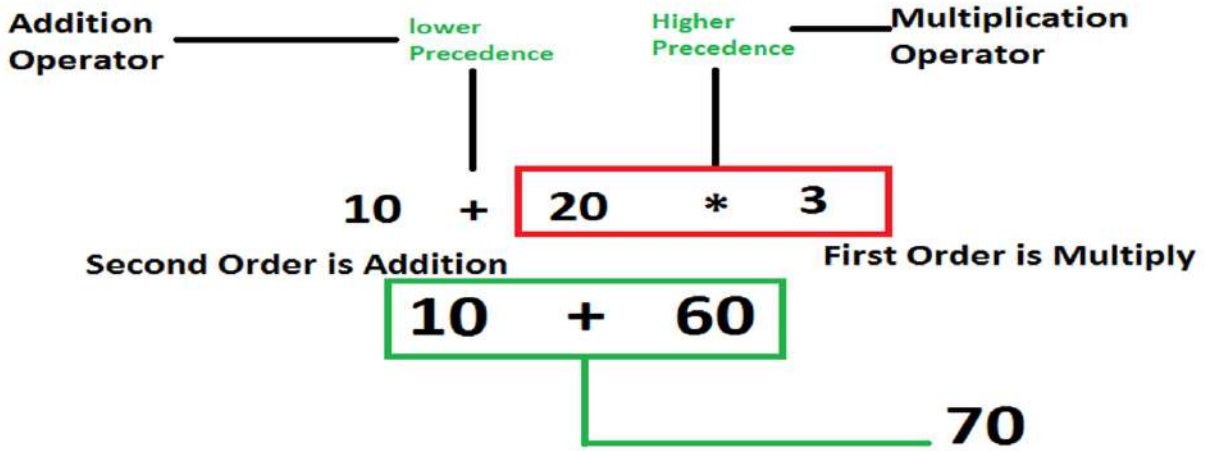
| *Algebraic expression* | *C expression* |
|---|---|
| a x b - c | a * b - c |
| (m+n) (x+y) | (m+n) * (x+y) |
| $\left(\frac{ab}{c}\right)$ | a * b/c |
| $3x^2 + 2x + 1$ | 3 * x * x 2 * x + 1 |
| $\left(\frac{x}{y}\right) + c$ | x/y+c |

Rules for Evaluating the Expressions:

- When parentheses are used, the expressions within parentheses assume highest priority.
- First parenthesized sub expressions from left to right are evaluated

- If parentheses are nested, the evaluation begins with the innermost sub-expression i.e. the innermost parenthesis will be solved first, followed by the second and so on.
- The precedence rule is applied for evaluating sub-expressions.
- The associativity rule is applied when two or more operators of same precedence level appear in sub-expression. i.e. when two operators of the same priority are found in the expression, precedence is given to the extreme left operator.

Ex:




## Naming Conventions:

- A package represents a sub directory that contains a group of classes and interfaces. Names of packages in Java are written in small letters
- Each word of class names and interface names start with a capital letter (String, DataInputStream, ActionListener etc.)
- The first word of a variable and method name is in small letters; then from second word onwards, each new word starts with a capital letter (readLine( ), nextInt( ) )
- Constants represent fixed values that cannot be altered. Such constants should be written by using all capital letters
- All keywords should be written by using all small letters

**Command line arguments:**

Command line arguments are parameters passed to the main method of a Java application from the command line.

They are used to provide input to the program and can be used to customize its behaviour. For example, you can use them to specify input files, output files, or other options.

In Java, command line arguments are stored in an array of strings called args in the main method. we can access the array elements and use them in the program as we wish.

Ex: consider the following program named Test.java

Test.java

```
public class Test{
    public static void main(String[] args) {
        for (String arg : args) {
            System.out.println(arg);
        }
    }
}
```

To pass command line arguments to the Java program, you use the java command followed by the name of the class and the arguments.

java Test BASIC FORTRAN PYTHON

This command line contains three arguments. They are assigned to args as follows.

BASIC → args[0]

FORTRAN → args[1]

PYTHON → args[2]

Ex2: Finding the sum of two integers using command line arguments

```
public class Sum {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Please provide two numbers.");
            return;
        }

        int num1 = Integer.parseInt(args[0]);
        int num2 = Integer.parseInt(args[1]);
        int sum = num1 + num2;

        System.out.println("Sum: " + sum);
    }
}
```

If we pass the command line arguments to the above java program as

java Sum 25 45

then we get the output as Sum: 70

## User Input to programs:

In Java, user inputs can be provided to programs in several ways. One of the most common methods include using the Scanner class of java.util package.

It is used to read input data from different sources like input streams, users, files, etc. we need to import the Scanner class from java.util package before we can use the Scanner class as

import java.util.Scanner;

After importing the package, create Scanner class objects.

Scanner sc = new Scanner(System.in);

The System.in parameter is used to take input from the standard input device i.e., from the keyboard. **Then use Scanner class methods to read input.**

**Some of the Scanner class methods are**

| Method | Description |
|---|---|
| nextInt() | reads an int value from the user |
| nextFloat() | reads a float value form the user |
| nextBoolean() | reads a boolean value from the user |
| nextLine() | reads a line of text from the user |
| next() | reads a word from the user |
| nextByte() | reads a byte value from the user |
| nextDouble() | reads a double value from the user |
| nextShort() | reads a short value from the user |
| nextLong() | reads a long value from the user |

Ex:

```
import java.util.Scanner;
public class Arithmetic
{
        public static void main(String[] args)
        {
                int a,b;
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter two numbers:");
                a = sc.nextInt();
                b = sc.nextInt();
                System.out.println(a+b);
                System.out.println(a-b);
                System.out.println(a*b);
                System.out.println(a/b);
                System.out.println(a%b);
        }
}
```

Escape Sequences(or) Back slash characters:

Escape sequences in Java are used to represent special characters within string literals and character literals. They are prefixed with a backslash (\) followed by a character that specifies the special character to be included in the string.

| *Constant* | *Meaning* |
|---|---|
| '\b' | back space |
| '\f' | form feed |
| '\n' | new line |
| '\r' | carriage return |
| '\t' | horizontal tab |
| '\'' | single quote |
| '\"' | double quote |
| '\\' | backslash |

Ex:

a) System.out.println("Hello\nWorld");

- prints "Hello" and "World" on separate lines

b)System.out.println("Hello\tWorld");

- prints "Hello" and "World" separated by a tab

c) System.out.println("C:\\Windows");

- prints "C:\Windows" (the backslash is escaped)

**Control statements:**

- Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear.
- But sometimes, we have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are satisfied.
- For this, Java provides Control statements (or decision making statements) which alter the flow of execution and provide better control to the programmer on the flow of execution.
- They are two types.

1. Conditional statements
2. Iterative statements

**Conditional Statements:**

Conditional statements are used to execute a statement or a group of statements based on certain conditions.

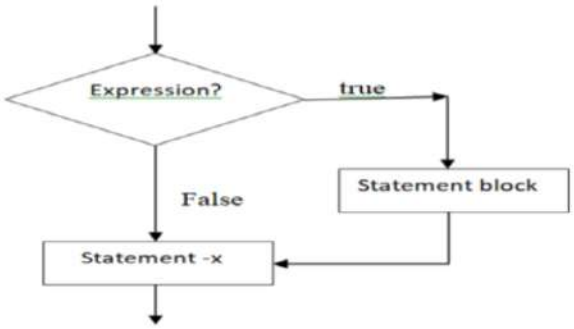
The various conditional statements are

1. Simple if statement
2. if…else statement
3. nested if…else statement
4. else if ladder statement
5. switch statement

**a) simple if statement:**

The general form of simple if statement is

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




Here, the expression can be any valid expression. Statement block may contain one statement (or) more statements. First expression will be evaluated.

**If the expression is true** then the statements in the statement-block gets executed and then control transfers to the next immediate statement of simple if i.e., statement-x.

**If the expression is false** then control transfers to execute the next immediate statement of simple if i.e., statement-x by skipping the statements in the statement block.

```
import java.util.Scanner;
public class SimpleIf
{
        public static void main(String args[])
        {
                int num;
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter any number:");
                num = scan.nextInt();
                //simple if statement
                if(num < 0)
                        num = -num;
                System.out.println("Absolute Number is :"+num);
        }
}
```

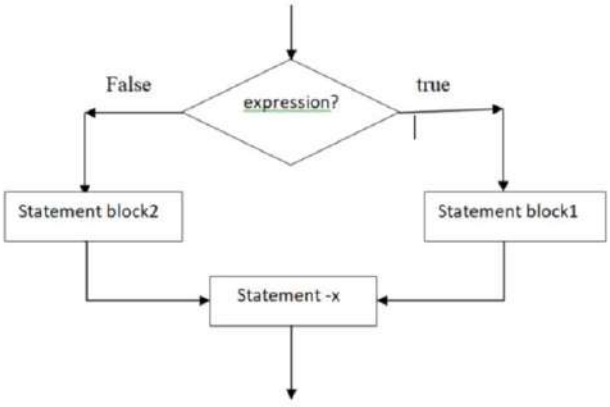
Output:

```
Enter any number:
-78
Absolute Number is :78
```

**if….else statement:**

The general form of if-else statement is

```
if (expression)
{
    Statement block1;
}
else
{
    Statement block2;
 }
 Statement-x;
```




Here, the expression can be any valid expression. Statement block1 and statement block2 may contain one statement (or) more statements. First expression will be evaluated.

**If the expression is true** then the statements in the statement block1 gets executed and then control transfers to the next immediate statement of if..else statement i.e., statement-x by skipping the statements in the statement block2.

**If the expression is false** then the statements in the statement block2 gets executed and then control transfers to the next immediate statement of if..else statement i.e., statement-x by skipping the statements in the statement block1.

```
/*program to find  whether an integer is a even number or not using if..else statement*/
import java.util.Scanner;
public class IfElse
{
	public static void main(String args[])
	{
		int num;
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter any number:");
		num = scan.nextInt();
		//if-else statement
		if(num % 2 == 0)
			System.out.println(num + " is even");
		else
			System.out.println(num + " is odd");
	}
}
```

```
Enter any number:
56
56 is even
```

**Nested if…else statement:**

When an if..else statement is included within another if..else statement, it is known as nested if ..else statement.

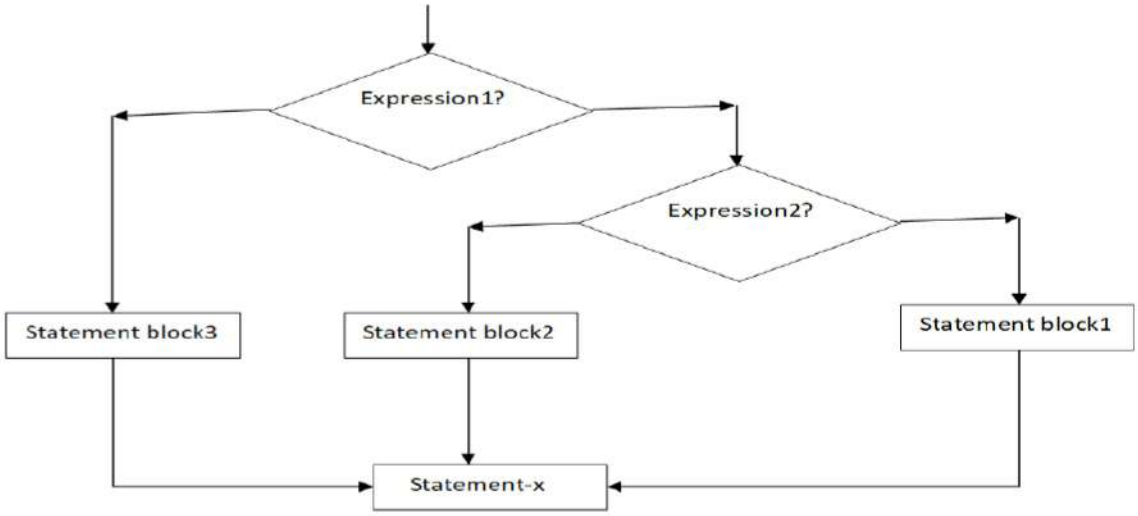
```
if (expression1)
{
		if (expression2)
		{
			Statement block1;
		}
		else
		{
			Statement block2;
		}
}
else
{
		Statement block3;
}
Statement-x;
```

Here, the expression1 and expression2 can be any valid expression.Statement block1, statement block2 and statement block3 may contain one statement (or) more statements.

First expression1 will be evaluated. **If the expression1 is true** then the expression2 will be evaluated. **If expression2 is true** then statements in the statement block1 gets executed and then control transfers to statement-x. **If expression2 is false** then statements in the statement block2 gets executed and then control transfers to statement-x.

**If the expression1 is false** then the statements in the statement block3 gets executed and then control transfers to the statement-x. This is illustrated in the following flowchart.

It is illustrated in the following flowchart




```
/*program to find  biggest of three numbers using nested if..else statement*/
import java.util.Scanner;
public class NestedIfElse
{
        public static void main(String args[])
        {
                int a,b,c;
                Scanner scan = new Scanner(System.in);
                System.out.println("Enter any three numbers:");
                a = scan.nextInt();
                b = scan.nextInt();
                c = scan.nextInt();
```

```
                //nested-if-else statement
                if(a>b)
                {
                        if(a>c)
                                System.out.println(a + " is biggest");
                        else
                                System.out.println(c + " is biggest");
                }
                else
                {

                        if(b>c)
                                System.out.println(b + " is biggest");
                        else
                                System.out.println(c + " is biggest");
                }
        }
}
```

```
Enter any three numbers:
56
67
89
89 is biggest
```

**else ….if ladder statement:**

The general form of else…if statement is

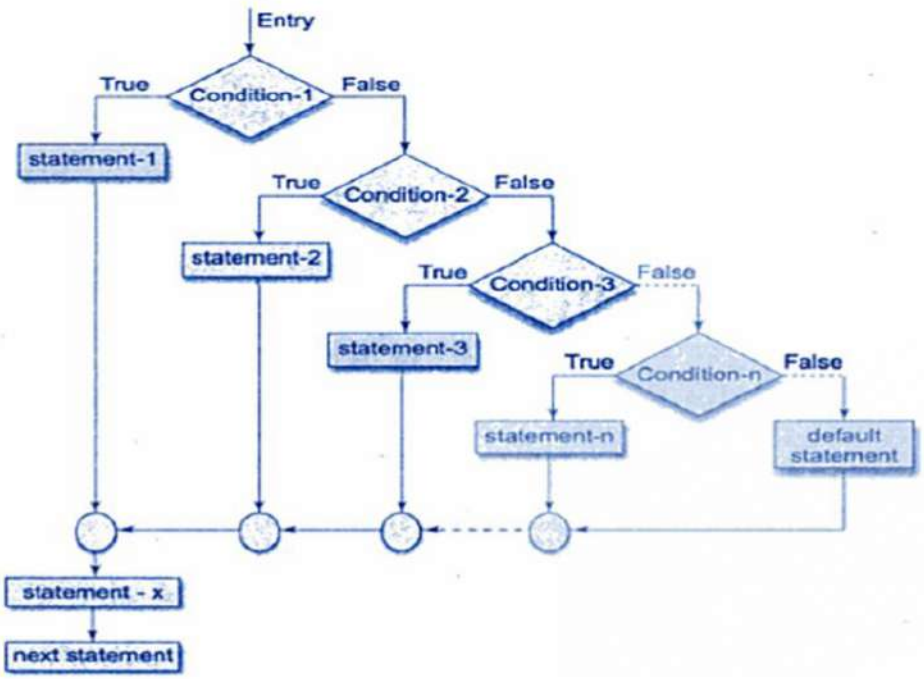
```
if(condition 1)
    statement-1
  else if(condition 2)
        statement-2
     else if(condition 3)
           statement-3
        else if(condition n)
              statement-n
           else
              default-statement
 statement-x
```

This construction is known as else if ladder.The conditions are evaluated from the top (of the ladder), downwards.

As soon as true condition is found, the statements associated with it is executed and the control is transferred to the statement –x (skipping the rest of the ladder).

When all the n conditions become false, the fine else containing the default-statement will be executed

It is illustrated in the following figure.




```
/*program to find grade of a student */
import java.util.Scanner;
public class ElseIfLadder
{
	public static void main(String args[])
	{
		int sub1,sub2,sub3;
		float sum=0.0f;
		float per;
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter the marks of Subject1:");
		sub1 = scan.nextInt();
		System.out.println("Enter the marks of Subject2:");
		sub2 = scan.nextInt();
```

```
                System.out.println("Enter the marks of Subject3:");
                sub3 = scan.nextInt();
                sum = sub1+sub2+sub3;
                per = sum/3;
                if(per >= 75)
                        System.out.println("Grade is A");
                else if(per >= 60)
                        System.out.println("Grade is B");
                else if(per >= 50)
                        System.out.println("Grade is C");
                else if(per >= 40)
                        System.out.println("Grade is D");
                else
                        System.out.println("Grade is E");
        }
}
```

Output:

```
Enter the marks of Subject1:
67
Enter the marks of Subject2:
67
Enter the marks of Subject3:
67
Grade is B
```

**switch statement**:

The switch statement in Java is a control flow statement that allows you to execute one code block from multiple possible blocks based on the value of a given expression. It's a more readable and organized alternative to using multiple if-else statements when dealing with a single variable or expression.

The general form of switch statement is

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

The expression (or variable) inside the parentheses of the switch statement is evaluated once, and its value is compared with the values of each case label.

Each case label specifies a possible value for the expression. If the expression matches the value of a case label, the corresponding code block is executed.

The break statement is used to exit the switch statement. If break is omitted, execution will continue to the next case (known as "fall-through").

The default block is optional and is executed if none of the case labels match the expression.

Note:

a) The expression in a switch statement can be of the following types: byte, short, char, int, String, or an enum.
b) Each of the case label values should be unique within a switch statement

```
public class SwitchExample {
    public static void main(String[] args) {
        int dayOfWeek = 3;

        switch (dayOfWeek) {
            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 day of the week");
        }
    }
}
```

Output:

```
Wednesday
```

**Iterative statements:**

Iterative statements in Java, also known as loops, are used to repeatedly execute a block of code as long as a specified condition is true.

Java provides several types of loops to handle different scenarios. Here's a detailed explanation of each type of loop in Java:

**Types of Iterative Statements in Java**

1. **while Loop**
2. **do-while Loop**
3. **for Loop**
4. **Enhanced for Loop** (also known as the "for-each" loop)

**1. while Loop**

The while loop evaluates a condition before executing the block of code. If the condition is true, the code block is executed. This process repeats until the condition becomes false.

**Syntax**:

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

**Example**:

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

In this example, the loop will print numbers from 0 to 4.

**2. do-while Loop**

The do-while loop is similar to the while loop, but it guarantees that the code block is executed at least once because the condition is evaluated after the code block.

**Syntax**:

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

**Example**:

```
int i = 0;
do {
   System.out.println(i);
   i++;
} while (i < 5);
```

In this example, the loop will also print numbers from 0 to 4, but the code inside the loop is executed at least once, even if the condition is false initially.

**3. for Loop**

The for loop is used to execute a block of code a specific number of times. It is typically used when the number of iterations is known beforehand.

**Syntax**:

```
for (initialization; condition; update) {
   // Code to be executed
}
```

**Example**:

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

In this example, the loop will print numbers from 0 to 4.

### 4. Enhanced for Loop (For-Each Loop)

The enhanced for loop is used to iterate over elements in an array or a collection. It is simpler and more readable than a traditional for loop for this purpose.

**Syntax**:

```
for (type variable : array) {
    // Code to be executed
}
```

**Example**:

```
int[] numbers = {1, 2, 3, 4, 5};
for (int number : numbers) {
    System.out.println(number);
}
```

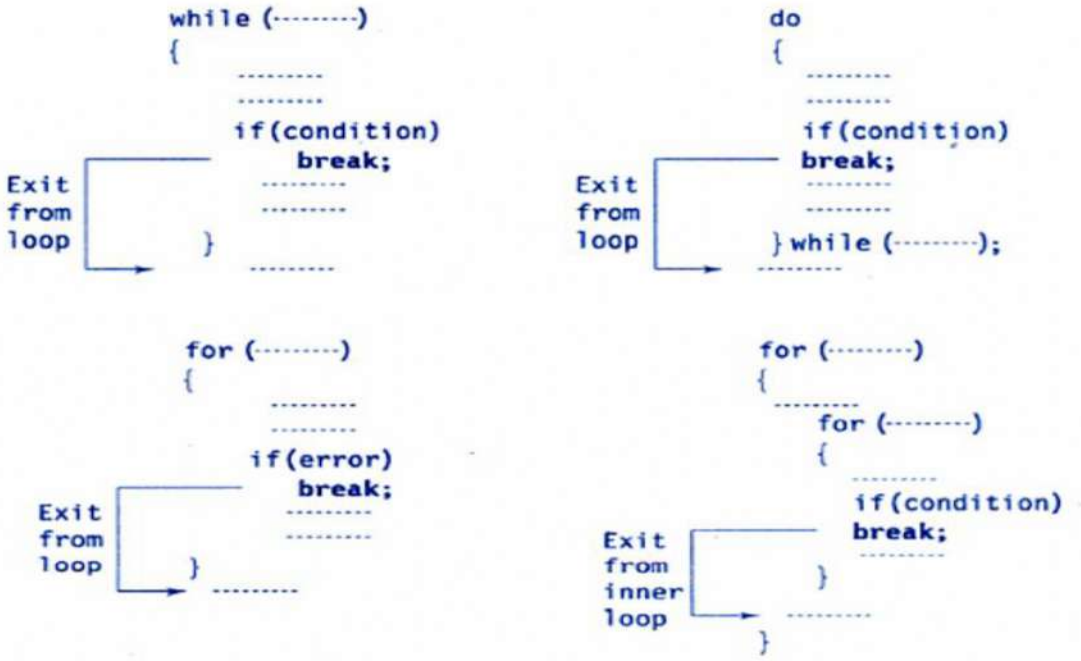
In this example, the loop will print each element in the numbers array.

**break Statement**:

The break statement is a jump instruction and can be used inside a switch (or) in loops. When executed, the control transfers out of switch (or) from the loop which contains the break statement and program execution continues with the next immediate statement of the switch (or) loop. The general form of break statement is

**break;**

It is illustrated as below.




```
//Java Program to demonstrate the use of break statement
//inside the for loop.
public class ContinueExample
{
   public static void main(String[] args)
   {

      for(int i=1;i<=10;i++)
      {
         if(i==5)
         {
            break;
         }
         System.out.print(i+"\t");
      }
   }
}
```

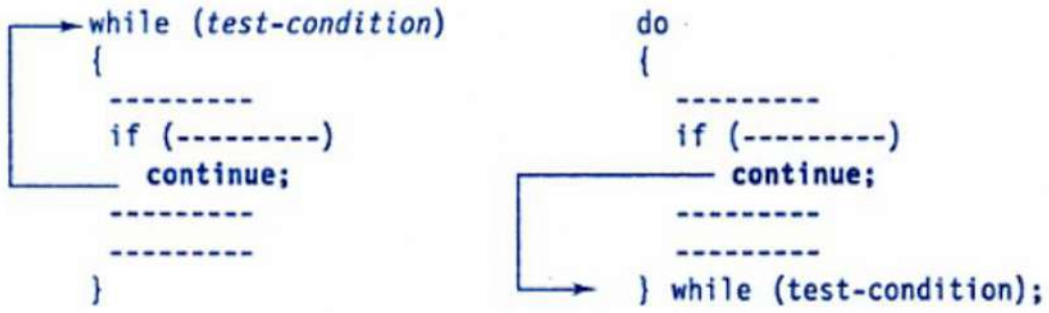
```
Output:

1    2    3    4
```

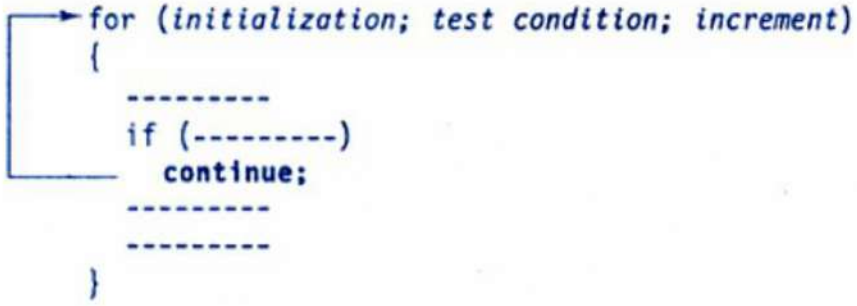
**continue Statement**:

continue statement is a jump statement. The continue statement can be used only inside for loop, while loop and do-while loop. The general form of continue is

continue;

Execution of continue statement does not cause an exit from the loop but it suspend the execution of the loop for that iteration and transfer control back to the loop for the next iteration.




```
//Java Program to demonstrate the use of continue statement
//inside the for loop.
public class ContinueExample
{
   public static void main(String[] args)
   {

      for(int i=1;i<=10;i++)
      {
```

```
            if(i==5)
            {
               continue;//it will skip the rest statement
            }
            System.out.print(i+"\t");
        }
    }
}
```

Output:

1 2 3 4 6 7 8 9 10