CS405PC: JAVA PROGRAMMING

B.TECH II Year II Sem.

L	T	P	C
3	1	0	4

Course Objectives:

- To introduce the object oriented programming concepts.
- To understand object oriented programming concepts, and apply them in solving problems.
- To introduce the principles of inheritance and polymorphism; and demonstrate how they relate to the design of abstract classes
- To introduce the implementation of packages and interfaces
- To introduce the concepts of exception handling and multithreading.
- To introduce the design of Graphical User Interface using applets and swing controls.

Course Outcomes:

- Able to solve real world problems using OOP techniques.
- Able to understand the use of abstract classes.
- Able to solve problems using java collection framework and I/o classes.
- Able to develop multithreaded applications with synchronization.
- Able to develop applets for web applications.
- Able to design GUI based applications

UNIT - I

Object-Oriented Thinking- A way of viewing world – Agents and Communities, messages and methods, Responsibilities, Classes and Instances, Class Hierarchies- Inheritance, Method binding, Overriding and Exceptions, Summary of Object-Oriented concepts. Java buzzwords, An Overview of Java, Data types, Variables and Arrays, operators, expressions, control statements, Introducing classes, Methods and Classes, String handling.

Inheritance– Inheritance concept, Inheritance basics, Member access, Constructors, Creating Multilevel hierarchy, super uses, using final with inheritance, Polymorphism-ad hoc polymorphism, pure polymorphism, method overriding, abstract classes, Object class, forms of inheritance- specialization, specification, construction, extension, limitation, combination, benefits of inheritance, costs of inheritance.

UNIT - II

Packages- Defining a Package, CLASSPATH, Access protection, importing packages.

Interfaces- defining an interface, implementing interfaces, Nested interfaces, applying interfaces, variables in interfaces and extending interfaces.

Stream based I/O (java.io) – The Stream classes-Byte streams and Character streams, Reading console Input and Writing Console Output, File class, Reading and writing Files, Random access file operations, The Console class, Serialization, Enumerations, auto boxing, generics.

UNIT - III

Exception handling - Fundamentals of exception handling, Exception types, Termination or resumptive models, Uncaught exceptions, using try and catch, multiple catch clauses, nested try statements, throw, throws and finally, built- in exceptions, creating own exception sub classes.

Multithreading- Differences between thread-based multitasking and process-based multitasking, Java thread model, creating threads, thread priorities, synchronizing threads, inter thread communication.

UNIT - IV

The Collections Framework (java.util)- Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Accessing a Collection via an Iterator, Using an Iterator, The For-Each alternative, Map Interfaces and Classes, Comparators, Collection algorithms, Arrays, The Legacy Classes and Interfaces- Dictionary, Hashtable ,Properties, Stack, Vector

More Utility classes, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner

UNIT - V

GUI Programming with Swing – Introduction, limitations of AWT, MVC architecture, components, containers. Understanding Layout Managers, Flow Layout, Border Layout, Grid Layout, Card Layout, Grid Bag Layout.

Event Handling- The Delegation event model- Events, Event sources, Event Listeners, Event classes, Handling mouse and keyboard events, Adapter classes, Inner classes, Anonymous Inner classes.

A Simple Swing Application, Applets – Applets and HTML, Security Issues, Applets and Applications, passing parameters to applets. Creating a Swing Applet, Painting in Swing, A Paint example, Exploring Swing Controls- JLabel and Image Icon, JText Field, **The Swing Buttons**- JButton, JToggle Button, JCheck Box, JRadio Button, JTabbed Pane, JScroll Pane, JList, JCombo Box, Swing Menus, Dialogs.

TEXT BOOKS:

1. Java The complete reference, 9th edition, Herbert Schildt, McGraw Hill Education (India) Pvt. Ltd.
2. Understanding Object-Oriented Programming with Java, updated edition, T. Budd, Pearson Education.

REFERENCE BOOKS:

1. An Introduction to programming and OO design using Java, J. Nino and F.A. Hosch, John Wiley & sons.
2. Introduction to Java programming, Y. Daniel Liang, Pearson Education.
3. Object Oriented Programming through Java, P. Radha Krishna, University Press.
4. Programming in Java, S. Malhotra, S. Chudhary, 2nd edition, Oxford Univ. Press.
5. Java Programming and Object-oriented Application Development, R. A. Johnson, Cengage Learning.

UNIT-I

OBJECT ORIENTED THINKING

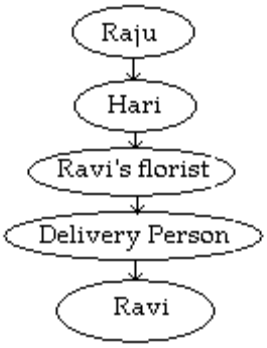
- When computers were first invented, programming was done manually by toggling in a binary machine instructions by use of front panel.
- As programs began to grow, high level languages were introduced that gives the programmer more tools to handle complexity.
- The first widespread high level language is FORTRAN. Which gave birth to structured programming in 1960's. The Main problem with the high level language was they have no specific structure and programs becomes larger, the problem of complexity also increases.
- So C became the popular structured oriented language to solve all the above problems.
- However in SOP, when project reaches certain size its complexity exceeds. So in 1980's a new way of programming was invented and it was OOP. OOP is a programming methodology that helps to organize complex programs through the use of inheritance, encapsulation & polymorphism.

A WAY OF VIEWING WORLD

- A way of viewing the world is an idea to illustrate the object-oriented programming concept with an example of a real-world situation.
- Eg: Raju wished to send some flowers to his friend for his birthday & he was living some miles away. He went to local florist and said the kind of flowers. He want to send to his friend's address. And florist assured those flowers will be sent automatically.

AGENTS AND COMMUNITIES

- An object-oriented program is structured as a community of interacting agents, called objects. Where each object provides a service (data and methods) that is used by other members of the community.
- Consider an eg, raju and ravi are good friends who live in two different cities far from each other. If Raju wants to send flowers to ravi, he can request his local florist 'hari' to send flowers to ravi by giving all the information along with the address.
- Hari works as an agent (or object) who performs the task of satisfying raju's request. Hari then performs a set of operations or methods to satisfy the request which is actually hidden from ravi, Hari forwards a message to ravi's local florist. Then local florist asks a delivery person to deliver those flowers to ravi. The objects or agents helping raju in solving the problem of sending flowers to his friend ravi can be shown in figure.



RESPONSIBILITIES

- A fundamental concept in OOP is to describe behavior in terms of responsibilities. A Request to perform an action denotes the desired result. An object can use any technique that helps in obtaining the desired result and this process will not have any interference from other object. The abstraction level will be increased when a problem is evaluated in terms of responsibilities. The objects will thus become independent from each other which will help in solving complex problems. An Object has a collection of responsibilities related with it which is termed as 'protocol'
- The Operation of a traditional program depends on how it acts on data structures. Where as an OOP operates by requesting data structure to perform a service.

MESSAGES & METHODS

In object-oriented programming, every action is initiated by passing a message to an agent (object), which is responsible for the action. The receiver is the object to whom the message was sent. In response to the message, the receiver performs some method to carry out the request. Every message may include any additional information as arguments.

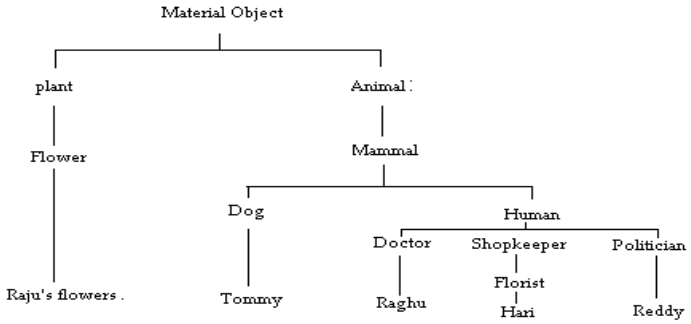
CLASSES & INSTANCES

- In object-oriented programming, all objects are instances of a class. The method invoked by an object in response to a message is decided by the class. All the objects of a class use the same method in response to a similar message.
- All objects are said to be the instances of a class.
- For e.g., If 'flower' is a class then Rose is its instance

CLASS HIERARCHIES (INHERITANCE)

A graphical representation is often used to illustrate the relationships among the classes (objects) of a community. This graphical representation shows classes listed in a hierarchical tree-like structure. In this more abstract class listed near the top of the tree, and more specific classes in the middle of the tree, and the individuals listed near the bottom.

In object-oriented programming, classes can be organized into a hierarchical inheritance structure. A child class inherits properties from the parent class that higher in the tree.



Class hierarchy for Different kinds of Material

Object Oriented Thinking

Let 'Hari' be a florist, but florist more specific form of shot keeper. Additionally, a shop keeper is a human and a human is definitely a mammal. But a mammal is an animal & animal is material object.

METHOD BINDING

When the method is super class have same name that of the method in sub class, then the subclass method overridden the super-class method. The program will find out a class to which the reference is actually pointing and that class method will be binded.

Example

```
class parent
{
        void print()
        {
                System.out.println("From Parent");
        }
}
class  child extends parent
{
        void print()
        {
                System.out.println("From Child");
        }
}
class  Bind
{
        public static void main(String arg[])
        {
                child ob=new child();
                ob.print();
        }
}
```

Output: From Child

The child's object 'ob' will point to child class print() method thus that method will be binded.

OVERRIDING

When the name and type of the method in a subclass is same as that of a method in its super class. Then it is said that the method present in subclass overrides the method present in super class. Calling an overridden method from a subclass will always point to the type of that method as defined by the subclass, where as the type of method defined by super class is hidden.

E.g. (above 'method binding' example)

EXCEPTIONS

Exception is a error condition that occurs in the program execution. There is an object called 'Exception' object that holds error information. This information includes the type and state of the program when the error occurred.

E.g. Stack overflow, Memory error etc

SUMMARY OF OOP CONCEPTS

- Everything is an object.
- Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending & receiving *messages*. A message is a request for an action bundled with whatever arguments may be necessary to complete the task.
- Each object has its own *memory*, which consists of other objects.
- Every Object is an *instance* of class. A class simply represents a grouping of similar objects, such as integers or lists.
- The class is the repository for *behavior* associated with an object. That is all objects that are instances of same class can perform the same actions.
- Classes are organized into a singly rooted tree structure, called *inheritance hierarchy.*

OOP Concepts in Java

OOP stands for Object-Oriented Programming. OOP is a programming paradigm in which every program is follows the concept of object. In other words, OOP is a way of writing programs based on the object concept.

The object-oriented programming paradigm has the following core concepts.

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

Class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

A class in java contains:

- Data Member
- Method
- Constructor
- Block
- Class and Interface

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.

Real life example of object and class

In real world many examples of object and class like dog, cat, and cow are belong to animal's class. Each object has state and behaviors. For example a dog has state:- color, name, height, age as well as behaviors:- barking, eating, and sleeping.

Vehicle class:

Car, bike, truck these all are belongs to vehicle class. These Objects have also different different states and behaviors. For Example car has state - color, name, model, speed, Mileage. as we;; as behaviors - distance travel

Encapsulation: **Encapsulation** is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in java language by class concept.

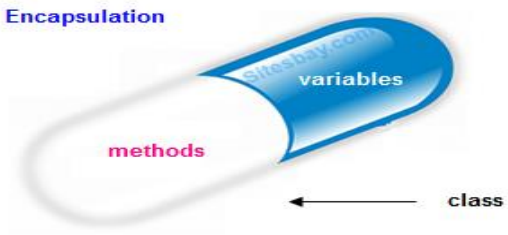
Combining of state and behavior in a single container is known as encapsulation. In java language encapsulation can be achieve using **class** keyword, state represents declaration of variables on attributes and behavior represents operations in terms of method.

Advantage of Encapsulation

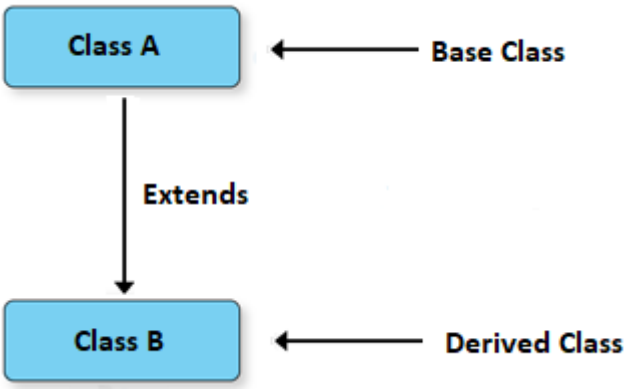
The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real life example of Encapsulation in Java

The common example of encapsulation is capsule. In capsule all medicine are encapsulated in side capsule.



Inheritance: Inheritance is one of the most important features of Object Oriented Programming. It creates new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called inheritance.



Ex : Inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

Here existing class is called base class. Base class is also known as parent class/super class. And new class is called derived class. Derived class is also known as child class/sub class.

Polymorphism: Polymorphism is derived from 2 greek words: **poly** and **morphs**. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Real life example of polymorphism in Java

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

Real life example of polymorphism in Java

You can see in the below Images, you can see, Man is only one, but he takes multiple roles like - he is a dad to his child, he is an employee, a salesperson and many more. This is known as Polymorphism.

Abstraction: There may be a lot of data. A class contains and the user does not need the entire data. The user requires only some part of data is available. In this case we can hide the unnecessary data from the user and expose only that data that is interest to the user. This is called Abstraction.

Ex :

```
class bank
{
        private int accno;
        private float  balance,profit,loan;
        vod display()
        {
                System.out.println("accno is :"+accno);
                System.out.println("balance is :"+balance);
         }
}
```

display() can access and display only the accno and balance. It cannot access profit and loan of the customer. This means the profit and loan data is hidden from the view of the bank.

Overview of Java

- Java is a computer programming language.
- Java was created based on C and C++.
- Java uses C syntax and many of the object-oriented features are taken from C++.
- Before Java was invented there were other languages like COBOL, FORTRAN, C, C++, Small Talk, etc. These languages had few disadvantages which were corrected in Java.
- Java also innovated many new features to solve the fundamental problems which the previous languages could not solve.
- Java was invented by a team of 13 employees of Sun Microsystems, Inc. which is lead by James Gosling, in 1991.
- The team includes persons like Patrick Naughton, Chris Warth, Ed Frank, and Mike Sheridan, etc.,
- Java was developed as a part of the Green project.
- Initially, it was called Oak, later it was changed to Java in 1995.

History of Java

- The C language developed in 1972 by Dennis Ritchie had taken a decade to become the most popular language.
- In 1979, Bjarne Stroustrup developed C++, an enhancement to the C language with included OOP fundamentals and features.
- A project named “Green” was initiated in December of 1990, whose aim was to create a programming tool that could render obsolete the C and C++ programming languages.
- Finally in the year of 1991 the Green Team was created a new Programming language named “OAK”.
- After some time they found that there is already a programming language with the name “OAK”.
- So, the green team had a meeting to choose a new name. After so many discussions they want to have a coffee. They went to a Coffee Shop which is just outside of the Gosling’s office and there they have decided name as “JAVA”.

JAVA BUZZ WORDS

- Java is the most popular object-oriented programming language.
- Java has many advanced features, a list of key features is known as Java Buzz Words.
- **The Following list of Buzz Words**
    - Simple
    - Secure
    - Portable
    - Object-oriented
    - Robust
    - Architecture-neutral (or) Platform Independent
    - Multi-threaded
    - Interpreted
    - High performance
    - Distributed
    - Dynamic

Simple

- Java programming language is very simple and easy to learn, understand, and code.
- Most of the syntaxes in java follow basic programming language C and object-oriented programming concepts are similar to C++.
- In a java programming language, many complicated features like pointers, operator overloading, structures, unions, etc. have been removed.
- One of the most useful features is the garbage collector it makes java more simple.

Secure

- Java is said to be more secure programming language because it does not have pointers concept.
- java provides a feature "applet" which can be embedded into a web application.
- The applet in java does not allow access to other parts of the computer, which keeps away from harmful programs like viruses and unauthorized access.

Portable

- Portability is one of the core features of java .
- If a program yields the same result on every machine, then that program is called portable.
- Java programs are portable
- This is the result of java System independence nature.

Object-oriented

- Java is an object oriented programming language.
- This means java programs use objects and classes.

Robust

Robust means strong. Java programs are strong and they don't crash easily like a C or C++ programs

There are two reasons

- Java has got excellent inbuilt exception handling features. An exception is an error that occurs at runtime. If an exception occurs, the program terminates suddenly giving rise to problems like loss of data. Overcoming such problem is called exception handling.
- Most of the C and C++ programs crash in the middle because of not allocating sufficient memory or forgetting the memory to be freed in a program. Such problems will not occur in java because the user need not allocate or deallocate the memory in java. Everything will be taken care of by JVM only.

Architecture-neutral (or) Platform Independent

- Java has invented to archive "write once; run anywhere, anytime, forever".
- The java provides JVM (Java Virtual Machine) to archive architectural-neutral or platform-independent.
- The JVM allows the java program created using one operating system can be executed on any other operating system.

Multi-threaded

- Java supports multi-threading programming.
- Which allows us to write programs that do multiple operations simultaneously.

Interpreted

- Java programs are compiled to generate byte code.
- This byte code can be downloaded and interpreted by the interpreter in JVM.
- If we take any other language, only an interpreter or a compiler is used to execute the program.
- But in java, we use both compiler and interpreter for the execution.

High performance

- The problem with interpreter inside the JVM is that it is slow.
- Because of Java programs used to run slow.
- To overcome this problem along with the interpreter.
- Java soft people have introduced JIT (Just in Time) compiler, to enhance the speed of execution.
- So now in JVM, both interpreter and JIT compiler work together to run the program.

Distributed

- Information is distributed on various computers on a network.
- Using Java, we can write programs, which capture information and distribute it to the client.
- This is possible because Java can handle the protocols like TCP/IP and UDP.

Dynamic

Java is said to be dynamic because the java byte code may be dynamically updated on a running system and it has a dynamic memory allocation and deallocation (objects and garbage collector).

JAVA KEYWORDS:

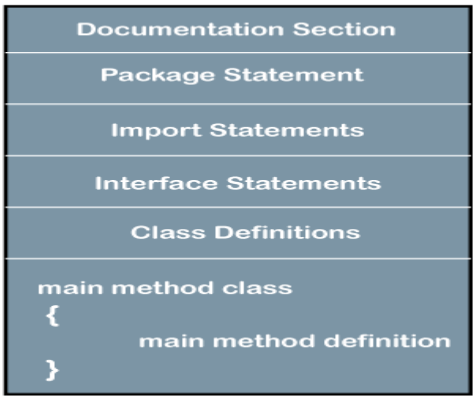
- Keywords are pre-defined words (or) reserved words.
- They can't be used as a variable name, object name, or any other identifier.
- All keywords must be written in lowercase.
- There are 51 reserved keywords in Java.

The following table shows the standard Java keywords.

abstract	assert	boolean	break	byte	case	catch	char	**class**
continue	default	do	double	else	enum	extends	Final	finally
float	for	if	implements	import	instanceof	Int	inteface	long
native	new	null	package	private	protected	public	return	short
Static	strictfp	super	Switch	synchronized	this	throw	throws	transient
try	void	voltaile	while	const	goto			

Structure of Java Program

Structure of a Java program contains the following elements:



Structure of Java Program

Documentation Section

The documentation section is an important section but optional for a Java program.

It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program. It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program. To write the statements in the documentation section, we use **comments**.

Comments there are three types

1. Single-line Comment: It starts with a pair of forwarding slash **(//)**.

Example : //First Java Program

2. Multi-line Comment: It starts with a **/*** and ends with ***/.** We write between these two symbols.

Example : /* It is an example of
multiline comment */

3. Documentation Comment: It starts with the delimiter **(/**)** and ends with ***/.**

Sample Java Program :

```
/* This is First Java Program */
class Sample
{
        public static void main(String arg[]);
        {
                System.out.println(" Welcome to CSE Department");
        }
}
```

Parameters used in First Java Program

What is the meaning of class, public, static, void, main, String[], System.out.println().

- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** or **String args[]** is used for command line argument.
- **System.out.println()** is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

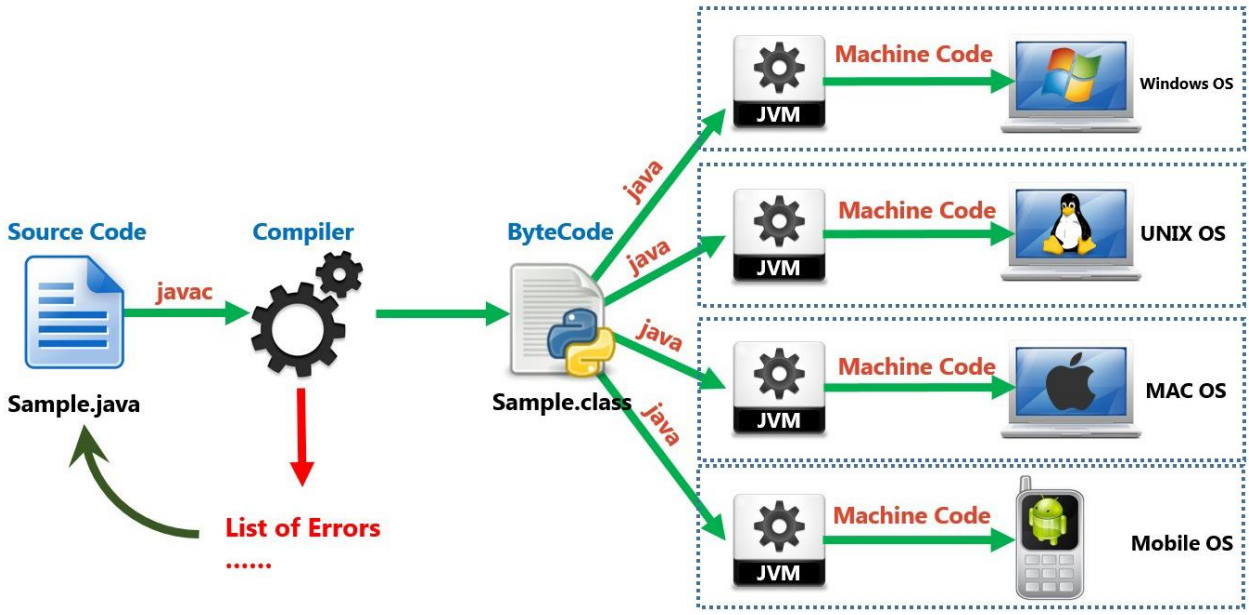
How to Compile and Run the Java Program

To Compile : javac Sample.java [program name]

To Run : java Sample

Output : Welcome to CSE Department

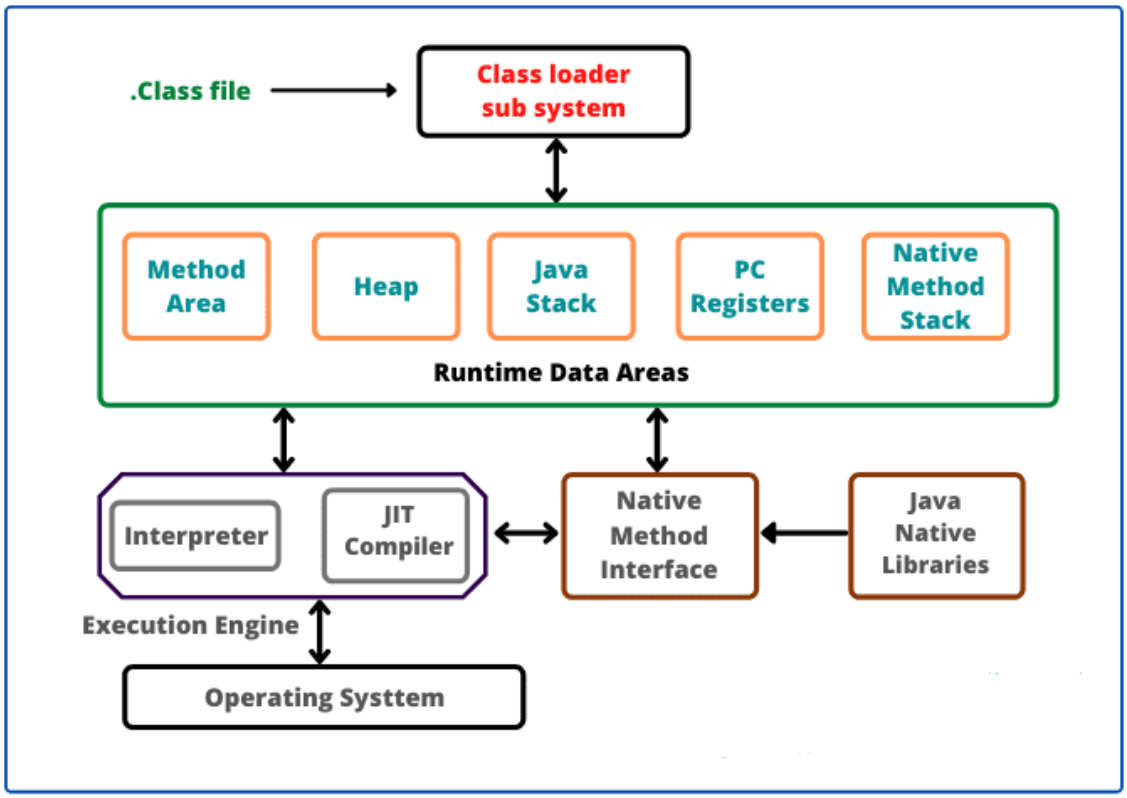
Execution Process of Java Program



What is JVM :

Java Virtual Machine is the heart of entire java program execution process. It is responsible for taking the .class file and converting each byte code instruction into the machine language instruction that can be executed by the microprocessor.

Architecture of Java Virtual Machine



Class Loader Subsystems:

First of all we compile the .java program is converted into a .class file. The .class file consisting of byte code instructions by the java compiler. In JVM there is a module called **class loader subsystem**. Which performs the following functions.

- It loads the .class file into memory.
- Then it verifies whether all byte code instructions are proper or not. If it finds any instruction suspicious, the execution is rejected immediately.
- If the byte code instructions are proper, then it allocates necessary memory to execute the program.

Runtime data areas: This memory is divided into 5 parts

1. **Method area:** Class (Method) area is a block of memory that stores the class code, code of variables, and methods of the Java program. Here methods mean functions declared in the class.

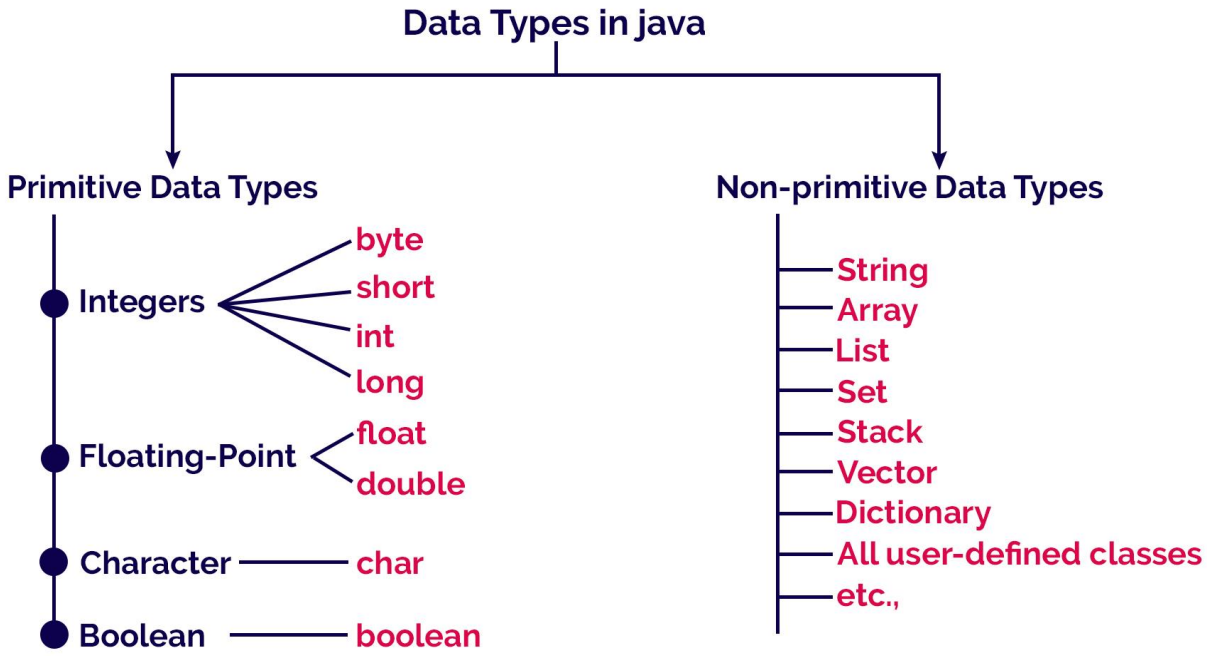
2. **Heap area :** Java objects reside in an area called the heap. The heap is created when the JVM starts up and may increase or decrease in size while the application runs. When the heap becomes full, garbage is collected. During the garbage collection objects that are no longer used are cleared, thus making space for new objects.
3. **Java stacks:**
    - Method code is stored in the Method area. But during the execution of a method, it requires some more memory to store the data and results. This memory is allocated on Java stacks.
    - Java stacks are those memory areas where Java methods are executed. In Java stacks, a separate frame is created where the method is executed.
    - Each time a method is called, a new frame is created into the stack. When method invocation is completed, a frame associated with it is destroyed.
    - JVM always creates a separate thread (or process) to execute each method.
4. **PC Registers:** The JVM supports multiple threads at the same time. Each thread has its own PC Registers to hold the address of the currently executing JVM instruction. Once the instruction is executed the PC register is updated with the next instruction.
5. **Native method stacks:** JVM contains stacks that support native methods. These methods are written in a language other than the java programming language , such as C and C++.

Execution Engine: Execution engine contains interpreter and JIT Compiler, which are responsible for converting the byte code instructions into machine code so that the processor will execute them. Most of the JVM implementations use both the interpreter and JIT compiler simultaneously to convert the byte code. This technique is also called adaptive optimizer.

JAVA DATA TYPES

Java programming language has a rich set of data types. The data type is a category of data stored in variables. In java, data types are classified into two types and they are as follows.

- Primitive Data Types
- Non-primitive Data Types



Primitive Data Types

The primitive data types are built-in data types and they specify the type of value stored in a variable and the memory size.

Integer Data Types: Integer Data Types represent integer numbers, i.e numbers without any fractional parts or decimal points.

Data Type	Memory Size	Minimum and Maximum values	Default Value
byte	1 byte	-128 to +128	0
short	2 bytes	-32768 to +32767	0
int	4 bytes	-2147483648 to +2147483647	0
long	8 bytes	-9223372036854775808 to +9223372036854775807	0L

Float Data Types: Float data types are represent numbers with decimal point.

Data Type	Memory Size	Minimum and Maximum values	Default Value
float	4 byte	-3.4e38 to -1.4e-45 and 1.4e-45 to 3.4e38	0.0f
double	8 bytes	-1.8e308 to -4.9e-324 and 4.9e-324 to 1.8e308	0.0d

Note: Float data type can represent up to 7 digits accurately after decimal point.

Double data type can represent up to 15 digits accurately after decimal point.

Character Data Type: Character data type are represents a single character like a, P, &, *,..etc.

Data Type	Memory Size	Minimum and Maximum values	Default Value
char	2 bytes	0 to 65538	\u0000

Boolean Data Types: Boolean data types represent any of the two values, true or false. JVM uses 1 bit to represent a Boolean value internally.

Data Type	Memory Size	Minimum and Maximum values	Default Value
boolean	1 byte	0 or 1	0 (false)

VARIABLES: Variable is a name given to a memory location where we can store different values of the same data type during the program execution.

The following are the rules to specify a variable name...

- A variable name may contain letters, digits and underscore symbol
- Variable name should not start with digit.
- Keywords should not be used as variable names.
- Variable name should not contain any special symbols except underscore(_).
- Variable name can be of any length but compiler considers only the first 31 characters of the variable name.

Declaration of Variable

Declaration of a variable tells to the compiler to allocate required amount of memory with specified variable name and allows only specified datatype values into that memory location.

Syntax: datatype variablename;

Example : int a;

Syntax : data_type variable_name_1, variable_name_2,...;

Example : int a, b;

Initialization of a variable:

Syntax: datatype variablename = value;

Example : int a = 10;

Syntax : data_type variable_name_1=value, variable_name_2 = value;

Example : int a = 10, b = 20;

Types of Variables

There are three types of variables in Java:

- local variable
- instance variable
- static variable

Local Variables:

- Variables declared inside the methods or constructors or blocks are called as local variables.
- The scope of local variables is within that particular method or constructor or block in which they have been declared.
- Local variables are allocated memory when the method or constructor or block in which they are declared is invoked and memory is released after that particular method or constructor or block is executed.
- Stack memory is allocated for storing local variables.
- JVM takes no responsibility for assigning default value to the local variables. It is the responsibility of the programmer to initialize the local variables explicitly before using them otherwise syntax error is raised.
- Access modifiers cannot be assigned to local variables.
- It can't be defined by a static keyword.
- Local variables can be accessed directly with their name.

Program :

```
public class LocalVariables
{
    public void show()
    {
        int a = 10;
        System.out.println("Inside show method, a = " + a);
    }
    public void display()
    {
        int b = 20;
        System.out.println("Inside display method, b = " + b);
        //System.out.println("Inside display method, a = " + a); // error
    }
    public static void main(String args[])
    {
        LocalVariables obj = new LocalVariables();
        obj.show();
        obj.display();
    }
}
```

Instance Variables:

- Variables declared outside the methods or constructors or blocks but inside the class are called as **instance variables**.
- The scope of instance variables is inside the class and therefore all methods, constructors and blocks can access them.
- Instance variables are allocated memory during object creation and memory is released during object destruction. If no object is created, then no memory is allocated.
- For each object, a separate copy of instance variable is created.
- Heap memory is allocated for storing instance variables.
- Access modifiers can be assigned to instance variables.
- It is the responsibility of the JVM to assign default value to the instance variables as per the type of Variable.
- Instance variables can be called directly inside the instance area.
- Instance variables cannot be called directly inside the static area and necessarily requires an object reference for calling them.

Program :

```
public class ClassVariables
{
        int x = 100;
        public void show()
        {
                System.out.println("Inside show method, x = " + x);
                x = x + 100;
        }
        public void display()
        {
                System.out.println("Inside display method, x = " + x);
        }
        public static void main(String[] args)
        {
                ClassVariables obj = new ClassVariables();
                obj.show();
                obj.display();
        }
}
```

Static variables:

- Static variables are also known as class variable.
- Static variables are declared with the keyword 'static ' .
- A static variable is a variable whose single copy in memory is shared by all the objects, any modification to it will also affect other objects.
- Static keyword in java is used for memory management, i.e it saves memory.
- Static variables gets memory only once in the class area at the time of class loading.
- Static variables can be invoked without the need for creating an instance of a class.
- Static variables contain values by default. For integers, the default value is 0. For Booleans, it is false. And for object references, it is null.
  - **Syntax:** static datatype variable name;
  - **Example:** static int x=100;
  - **Syntax:** classname.variablename;

Example1 : Without Static Variable

```
class Employee
{
	int empid=500;
	void emp1()
	{
		empid++;
		System.out.println("Employee id:"+empid);
	}
}
class Sample
{
	public static void main(String args[])
	{
		Employee e1=new Employee();
		e1.emp1();
		e1.emp1();
		e1.emp1();
		Employee e2=new Employee();
		e2.emp1();
		e2.emp1();
		e2.emp1();
	}
}
```

Output:

```
Employee id:501
Employee id:502
Employee id:503
Employee id:501
Employee id:502
Employee id:503
```

Example2 With Static variable

```
class Employee
{
   static int empid=500;
   static void emp1()
   {
                empid++;
                System.out.println("Employee id:"+empid);
   }
}
class Sample
{
        public static void main(String args[])
        {
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
                Employee.emp1();
        }
}
```

Output:

```
Employee id:501
Employee id:502
Employee id:503
```

Static method: A static method is declared by using the keyword 'static', it is known as static method.

Rules:

- A static method can be invoked without the need for creating an instance of a class.
- A static method can access only static data members.
- The static method cannot use non-static data member or call non-static method directly.
- this and super keyword cannot be used in static context.
- **These static variables can be accessed in two ways.**

  1. if a program consist of single class, the static variable can be accessed directly

  2. if a program consist of multiple class, the static variable can be accessed with the help of class name

  Syntax: classname.methodname;

Example 1:

```
class Employee
{
   static int empid;
   static String empname;
   static double empsal;
   static void empinfo()
   {
      System.out.println("Employee id:"+empid);
      System.out.println("Employee Name:"+empname);
      System.out.println("Employee Salary:"+empsal);
   }
   public static void main(String[] args)
   {
            empinfo();
   }
}
```

Example 2 :

```
class Employee
{
    static int empid;
    static String empname;
    static double empsal;
    static void empinfo()
    {
        System.out.println("Employee id:"+empid);
        System.out.println("Employee Name:"+empname);
        System.out.println("Employee Salary:"+empsal);
    }
}
class Sample
{
    public static void main(String[] args)
    {
        Employee.empinfo();
  }
}
```

ARRAYS

- An array is a collection of similar data values with a single name.
- An array can also be defined as, a special type of variable that holds multiple values of the same data type at a time.
- In java, arrays are objects and they are created dynamically using new operator.
- Every array in java is organized using index values.
- The index value of an array starts with '0' and ends with 'zise-1'.
- We use the index value to access individual elements of an array.

In java, there are two types of arrays and they are as follows.

- One Dimensional Array
- Multi Dimensional Array

One Dimensional Array

In the java programming language, an array must be created using new operator and with a specific size. The size must be an integer value but not a byte, short, or long. We use the following syntax to create an array.

Syntax

```
data_type array_name[ ] = new data_type[size];
                (or)
data_type[ ] array_name = new data_type[size];
```

Example :

```
class Onedarray
{
        public static void main(String args[])
        {
                int a[]=new int[5];
                a[0]=10;
                a[1]=20;
                a[2]=70;
                a[3]=40;
                a[4]=50;
                for(int i=0;i<5;i++)
                System.out.println(a[i]);
        }
}
```

- In java, an array can also be initialized at the time of its declaration.
- When an array is initialized at the time of its declaration, it need not specify the size of the array and use of the new operator.
- Here, the size is automatically decided based on the number of values that are initialized.

Example : int list[ ] = {10, 20, 30, 40, 50};

Multidimensional Array

- In java, we can create an array with multiple dimensions. We can create 2-dimensional, 3-dimensional, or any dimensional array.
- In Java, multidimensional arrays are arrays of arrays.
- To create a multidimensional array variable, specify each additional index using another set of square brackets.

Syntax

```
data_type array_name[ ][ ] = new data_type[rows][columns];
                    (or)
data_type[ ][ ] array_name = new data_type[rows][columns];
```

- When an array is initialized at the time of declaration, it need not specify the size of the array and
- use of the new operator.
- Here, the size is automatically decided based on the number of values that are initialized.

Example

```
class Twodarray
{
    public static void main(String args[])
    {
        int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
        for(int i=0;i<3;i++)
        {
            for(int j=0;j<3;j++)
            {
                System.out.print(arr[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

OPERATORS: An operator is a symbol that performs an operation. An operator acts on some variables called operands to get the desired result.

Example: a + b

Here a, b are operands and + is operator.

Types of Operators

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bit wise operators

1. Arithmetic Operators: Arithmetic Operators are used for mathematical calculations.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modular

```
class ArithmeticOperators
{
	public static void main(String[] args)
	{
		int a = 12, b = 5;
		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));
	}
}
```

2. Relational Operators: Relational operators are used to compare two values and return a true or false result based upon that comparison. Relational operators are of 6 types

Operator	Description
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to

Program: Java Program to implement Relational Operators

```
import java.io.*;
class  RelationalOperator
{
        public static void main(String[] args)
        {
                int a = 10;
                int b = 3;
                int c = 5;
                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 == c: " + (a == c));
                System.out.println("a != c: " + (a != c));
        }
}
```

3. Logical Operator: The Logical operators are used to combine two or more conditions .Logical operators are of three types

1. Logical AND (&&),
2. Logical OR (||)
3. Logician NOT (!)

1. Logical AND (&&) : Logical AND is denoted by double ampersand characters (&&).it is used to check the combinations of more than one conditions. if any one condition false the complete condition becomes false.

Truth table of Logical AND

Condition1	Condition2	Condition1 && Condition2
True	True	True
True	False	False
False	True	False
False	False	False

Example of Logical AND (&&) operator :

```
import java.io.*;
class  LogicalAND
{
        public static void main(String[] args)
        {
                int x=10;
                System.out.println(x==10 && x>=5));
                System.out.println(x>=5 && x<=50));
                System.out.println(x!=10 && x>=5));
                System.out.println(x>=20 && x<=50));
        }
}
```

2. Logical OR (||) : Logical OR is denoted by double pipe characters (||). it is used to check the combinations of more than one conditions. if any one condition true the complete condition becomes true.

Truth table of Logical OR

Condition1	Condition2	Condition1 && Condition2
True	True	True
True	False	True
False	True	True
False	False	False

Example of Logical OR (||) operator :

```
import java.io.*;
class  LogicalOR
{
        public static void main(String[] args)
        {
                int x=10;
                System.out.println(x==10 || x>=5));
                System.out.println(x>=5 || x<=50));
                System.out.println(x!=10 || x>=5));
                System.out.println(x>=20 || x<=50));
        }
}
```

3. Logician NOT (!): Logical NOT is denoted by exclamatory characters (!), it is used to check the opposite result of any given test condition. i.e, it makes a true condition false and false condition true.

Truth table of Logical NOT

Condition1	!Condition2
True	False
False	True

Example of Logical NOT (!) operator :

```
import java.io.*;
class  LogicalOR
{
        public static void main(String[] args)
        {
                int x=10;
                System.out.println ( ! ( x==10 ));
                System.out.println (!( x!=10));
                System.out.println (!( x>5));
                System.out.println ( !( x<5));
        }
}
```

4. Assignment Operator: Assignment operators are used to assign a value (or) an expression (or) a value of a variable to another variable.

Syntax : variable name=expression (or) value

Example : x=10;

y=20;

The following list of Assignment operators are.

Operator	Description	Example	Meaning
+=	Addition Assignment	x + = y	x= x + y
-=	Addition Assignment	x - = y	x= x - y
***=**	Addition Assignment	x * = y	x= x * y
/=	Addition Assignment	x / = y	x= x / y
%=	Addition Assignment	x % = y	x= x % y

Example of Assignment Operators

```
class AssignmentOperator
 {
        public static void main(String[] args)
        {
                int a = 4;
                int var;
                var = a;
                System.out.println("var using =: " + var);
                 var += a;
                System.out.println("var using +=: " + var);
                 var *= a;
                System.out.println("var using *=: " + var);
         }
}
```

5: Increment And Decrement Operators : The increment and decrement operators are very useful. ++ and == are called increment and decrement operators used to add or subtract. Both are unary operators.

The syntax of the operators is given below.

These operators in two forms : prefix (++x) and postfix(x++).

++<variable name> --<variable name>

<variable name>++ <variable name>--

Operator	Meaning
++x	Pre Increment
--x	Pre Decrement
x++	Post Increment
x--	Post Decrement

Where

1 : ++x : Pre increment, first increment and then do the operation.

2 : - -x : Pre decrement, first decrements and then do the operation.

3 : x++ : Post increment, first do the operation and then increment.

4 : x- - : Post decrement, first do the operation and then decrement.

Example :

```
class Increment
{
        public static void main(String[] args)
        {
                int var=5;
                System.out.println (var++);
                System.out.println (++var);
                System.out.println (var--);
                System.out.println (--var);
        }
}
```

6 : Conditional Operator: A conditional operator checks the condition and executes the statement depending on the condition. Conditional operator consists of two symbols.

1 : question mark (?).

2 : colon (:).

Syntax: condition ? exp1 : exp2;

It first evaluate the condition, if it is true (non-zero) then the "exp1" is evaluated, if the condition is false (zero) then the "exp2" is evaluated.

Example :

```
class  ConditionalOperator
{
        public static void main(String[] args)
        {
                int februaryDays = 29;
                String result;
                result = (februaryDays == 28) ? "Not a leap year" : "Leap year";
                System.out.println(result);
        }
}
```

7. Bitwise Operators:

- Bitwise operators are used for manipulating a data at the bit level, also called as bit level programming. Bit-level programming mainly consists of 0 and 1.
- They are used in numerical Computations to make the calculation process faster.
- The bitwise logical operators work on the data bit by bit.
- Starting from the least significant bit, i.e. LSB bit which is the rightmost bit, working towards the MSB (Most Significant Bit) which is the leftmost bit.

A list of Bitwise operators as follows...

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise Complement
<<	Left Shift
>>	Right Shift

1. Bitwise AND (&):

- Bitwise AND operator is represented by a single ampersand sign (&).
- Two integer expressions are written on each side of the (&) operator.
- if any one condition false (0) the complete condition becomes false (0).

Truth table of Bitwise AND

Condition1	Condition2	Condition1 & Condition2
0	0	0
0	1	0
1	0	0
1	1	1

Example :

```
int x = 10;
int y = 20;
x & y = ?
x = 0000  1010
y = 0000  1011
x & y = 0000 1010 = 10
```

2. Bitwise OR:

- Bitwise OR operator is represented by a single vertical bar sign (|).
- Two integer expressions are written on each side of the (|) operator.
- if any one condition true (1) the complete condition becomes true (1).

Truth table of Bitwise OR

Condition1	Condition2	Condition1 | Condition2
0	0	0
0	1	1
1	0	1
1	1	1

Example : int x = 10;

int y = 20;

x | y = ?

x = 0000 1010

y = 0000 1011

x | y = 0000 1011 = 11

3. Bitwise Exclusive OR :

- The XOR operator is denoted by a carrot (^) symbol.
- It takes two values and returns true if they are different; otherwise returns false.
- In binary, the true is represented by 1 and false is represented by 0.

Truth table of Bitwise XOR

Condition1	Condition2	Condition1 ^ Condition2
0	0	0
0	1	1
1	0	1
1	1	0

Example : int x = 10;

int y = 20;

x ^ y = ?

x = 0000 1010

y = 0000 1011

x ^ y = 0000 0001 = 1

4. Bitwise Complement (~):

- The bitwise complement operator is a unary operator.
- It is denoted by ~, which is pronounced as tilde.
- It changes binary digits **1** to **0** and **0** to **1**.
- bitwise complement of any integer **N** is equal to **- (N + 1)**.
- Consider an integer **35**. As per the rule, the bitwise complement of **35** should be **-(35 + 1)** = **-36**.

Example : int x = 10; find the ~x value.

x = 0000 1010

~x= 1111 0101

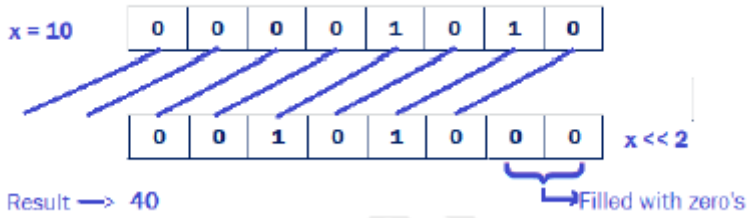
5. Bitwise Left Shift Operator (<<) :

- This Bitwise Left shift operator (<<) is a binary operator.
- It shifts the bits of a number towards left a specified no.of times.

Example:

int x = 10;

x << 2 = ?



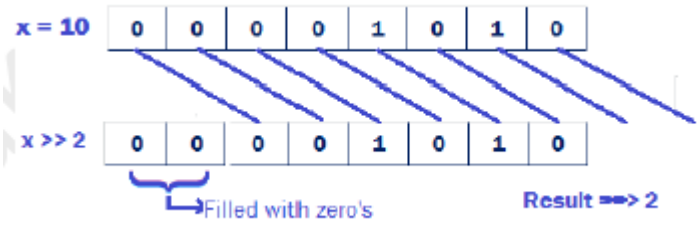
6. Bitwise Right Shift Operator (>>) :

- This Bitwise Right shift operator (>>) is a binary operator.
- It shifts the bits of a number towards right a specified no.of times.

Example:

int x = 10;

x >> 2 = ?



EXPRESSIONS

- In any programming language, if we want to perform any calculation or to frame any condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression. **In the java programming language, an expression is defined as follows..**
- An expression is a collection of operators and operands that represents a specific value.
- In the above definition, an operator is a symbol that performs tasks like arithmetic operations, logical operations, and conditional operations, etc.

Expression Types

In the java programming language, expressions are divided into THREE types. They are as follows.

- **Infix Expression**
- **Postfix Expression**
- **Prefix Expression**

The above classification is based on the operator position in the expression.

Infix Expression

The expression in which the operator is used between operands is called infix expression. The infix expression has the following general structure.

Example

a+b

Postfix Expression

The expression in which the operator is used after operands is called postfix expression. The postfix expression has the following general structure.

Example

ab+

Prefix Expression

The expression in which the operator is used before operands is called a prefix expression. The prefix expression has the following general structure.

Example

+ab

Control Statements

In java, the control statements are the statements which will tell us that in which order the instructions are getting executed. The control statements are used to control the order of execution according to our requirements.

In java, the control statements are classified as follows.

- Selection Control Statements (Decision Making Statements)
- Iterative Control Statements (Looping Statements)
- Jump Statements

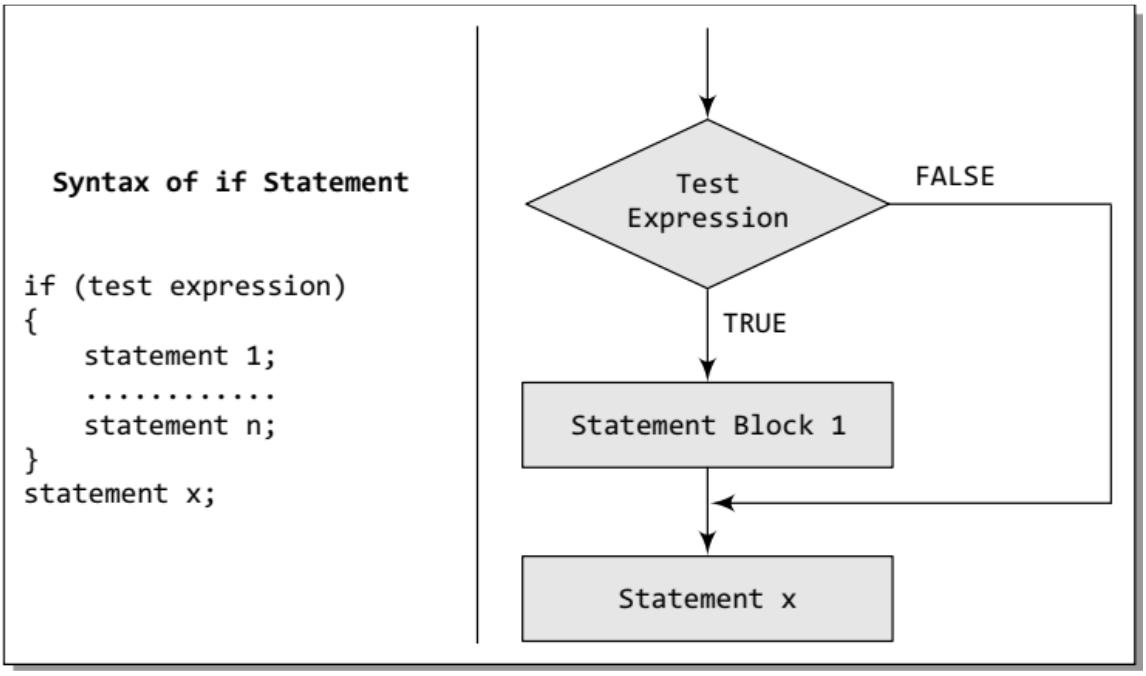
Selection Control Statements

In java, the selection statements are also known as decision making statements or branching statements. The selection statements are used to select a part of the program to be executed based on a condition. Java provides the following selection statements.

1. if statement
2. if-else statement
3. if-elseif statement
4. nested if statement
5. switch statement

1. if statement : if statement is the simplest decision control statement that is frequently used in decision making.

The general form of a simple if statement is



The if block may include 1 statement or *n* statements enclosed within curly brackets. First the test expression is evaluated. If the test expression is true, the statements of the if block are executed, otherwise these statements will be skipped and the execution will jump to statement x.

Note : In case the statement block contains only one statement, putting curly brackets becomes optional. If there are more than one statement in the statement block, putting curly brackets becomes mandatory.

Example :

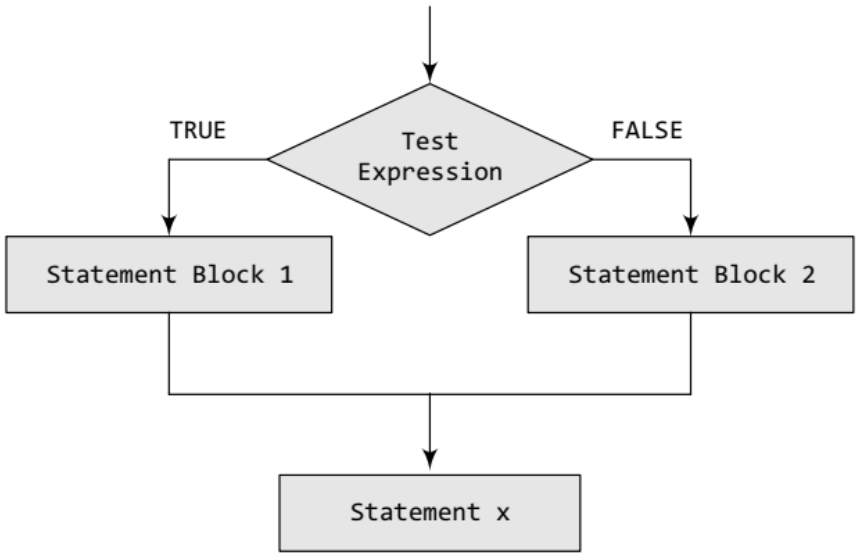
```
class Sample
{
        public static void main(String[] args)
        {
                int x=10;
                if(x>0)
                        x++;
                System.out.println("x value is:"+x);
        }
}
```

2. if–else statement : The if-else statement is an extension of the simple if statement. Its usage is very simple.

The general form of simple if–else statement is :

Syntax of if-else Statement

```
if (test expression)
{
    statement block 1;
}
else
{
    statement block 2;
}
statement x;
```



In the if–else statement, first the test expression is evaluated. If the expression is true, statement block 1 is executed and statement block 2 is skipped. Otherwise, if the expression is false, statement block 2 is

executed and statement block 1 is ignored. In any case after the statement block 1 or 2 gets executed, the control will pass to statement x. Therefore, statement x is executed in every case.

Example: Write a program to find whether a number is even or odd.

```
class Sample
{
        public static void main(String[] args)
        {
                int a=29;
                if(a % 2==0)
                        System.out.println("Even Number is :"+a);
                else
                        System.out.println("Odd Number is :"+a);
        }
}
```

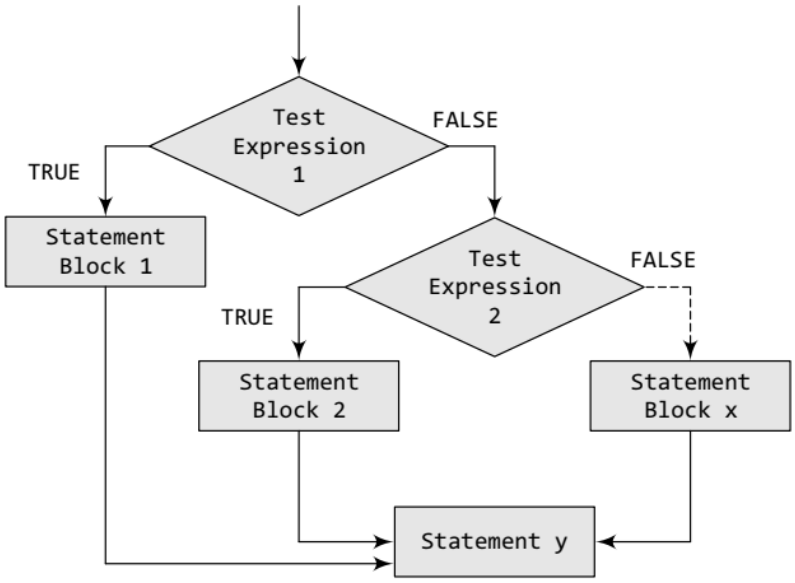
3. if–else–if statement : C language supports if–else–if statements to test additional conditions apart from the initial test expression. The if–else–if construct works in the same way as a normal if statement.

The general form of simple if–else–if statement is :

Syntax of if-else-if Statement

```
if (test expression 1)
{
        statement block 1;
}
else if (test expression 2)
{
        statement block 2;
}
.........................
else
{
        statement block x;
}

statement y;
```



```
class Test
{
	public static void main(String args[])
	{
		int x = 30;
		if( x == 10 )
		{
			System.out.print("Value of X is 10");
		}
		else if( x == 20 )
		{
			System.out.print("Value of X is 20");
		}
		else if( x == 30 )
		{
			System.out.print("Value of X is 30");
		}
		else
		{
			System.out.print("This is else statement");
		}
	}
}
```

4. Nested if-else statement : When a series of decision is required, nested if-else is used. Nesting means using one if-else construct within another one.

The general form of simple Nested if-else statement is :

```
if(condition-1)
{
        if (condition-2)
        {
                true statement-1;
        }
        else
        {
                false statement-2;
        }
}
else
{
        false statement-3;
}
statement-x;
```

If the condition-1 is false, the false statement-3 and statement-x will be executed. Otherwise it continues to perform the second test. If the condition-2 is true, the true statement-1 and statement-x will be executed otherwise the false statement-2 and statement-x will be executed.

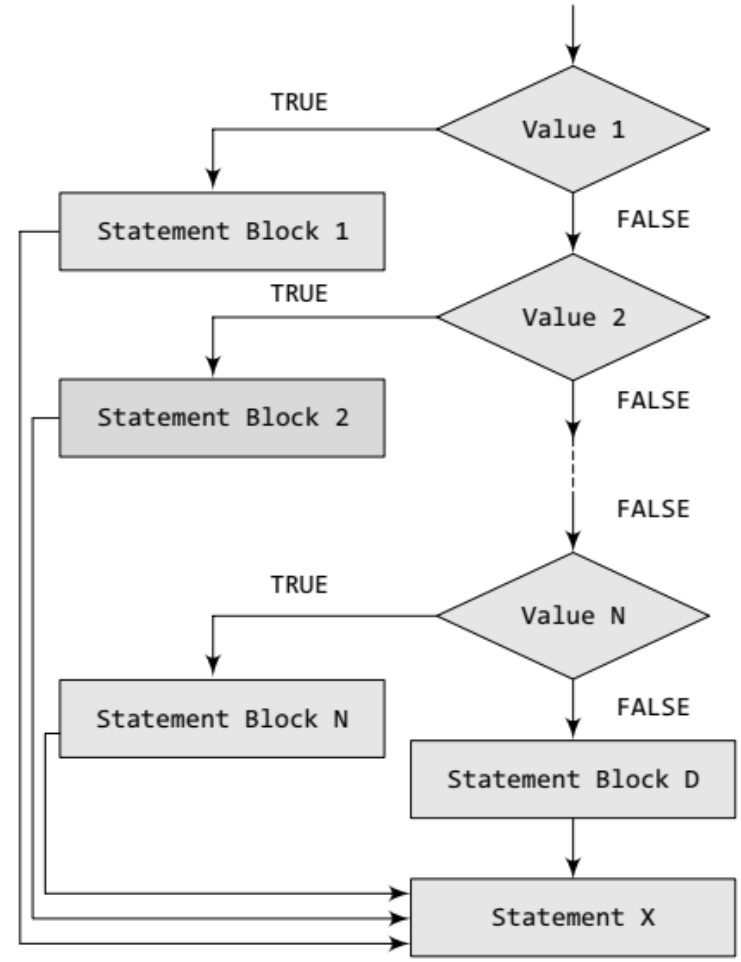
```
class Test
{
        public static void main(String args[])
        {
                int num=1;
                if(num<10)
                {
                        if(num==1)
                        {
                                System.out.print("The value is equal to 1);
                    }
                    else
                    {
                                System.out.print("The value is greater than 1");
                        }
                }
                else
                {
                        System.out.print("The value is greater than 10");
                }
                        System.out.print("Nested if - else statement ");
        }
}
```

5. switch–case statement: when there are several options and we have to choose only one option from the available ones, we can use switch statement. Depending on the selected option, a particular task can be performed. A task represents one or more statements.

The general form of switch–case statement is :

Syntax of Switch Statement

```
switch (variable)
{
        case value 1:
            statement block 1;
            break;
        case value 2:
            statement block 2;
            break;
        ......................
        case value N:
            statement block N;
            break;
        default:
            statement block D;
            break;
}
statement X;
```



The expression following the keyword **switch** in any 'C' expression that must yield an integer value. It must be an integer constants like 1,2,3 .

The keyword **case** is followed by an integer or a character constant, each constant in each must be different from all the other.

First the integer expression following the keyword **switch** is evaluated. The value it gives is searched against the constant values that follow the **case** statements. When a match is found, the program executes the statements following the case. If no match is found with any of the case statements, then the statements following the **default** are executed.

```
class SwitchStatementTest
{
	public static void main(String[] args)
	{
		char color ='g';
		switch( color )
		{
			case 'r':
				System.out.println("RED") ; break ;
			case 'g':
				System.out.println("GREEN") ; break ;
			case 'b':
				System.out.println("BLUE") ; break ;
			case 'w':
				System.out.println("WHITE") ; break ;
			default:
				System.out.println("No color") ;
		 }
	}
}
```

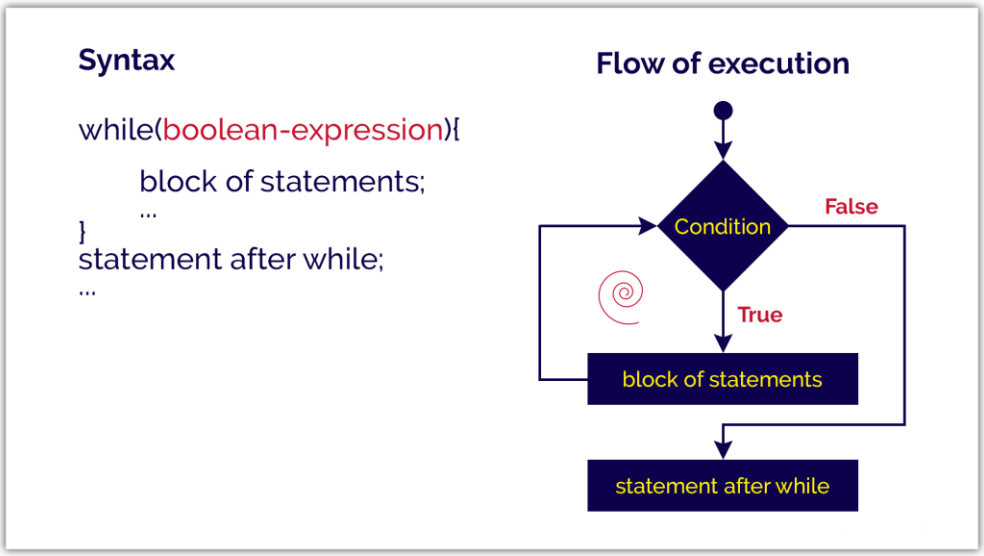
Iterative Statements

The java programming language provides a set of iterative statements that are used to execute a statement or a block of statements repeatedly as long as the given condition is true. The iterative statements are also known as looping statements or repetitive statements. Java provides the following iterative statements.

1. while statement
2. do-while statement
3. for statement

1.while statement : The while statement is used to execute a single statement or block of statements repeatedly as long as the given condition is TRUE. The while statement is also known as Entry control looping statement.

The syntax and execution flow of while statement is as follows.




```
class WhileTest
{
	public static void main(String[] args)
	{
		int num = 1;
		while(num <= 10)
		{
			System.out.println(num);
			num++;
		}
		System.out.println("Statement after while!");
	}
}
```

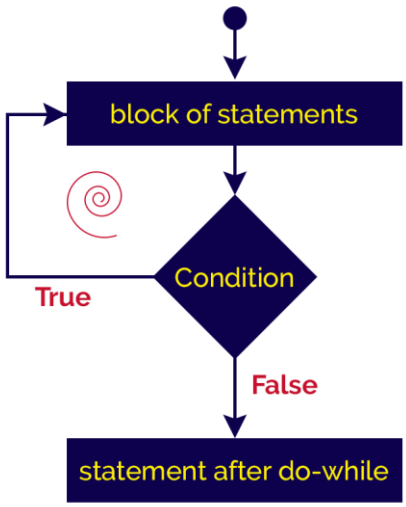
2.do-while statement : The do-while statement is used to execute a single statement or block of statements repeatedly as long as given the condition is TRUE. The do-while statement is also known as the **Exit control looping statement**.

The do-while statement has the following syntax.

Syntax

```
do{
      block of statements;
      ...
}while(boolean-expression);
statement after do-while;
...
```

Flow of execution

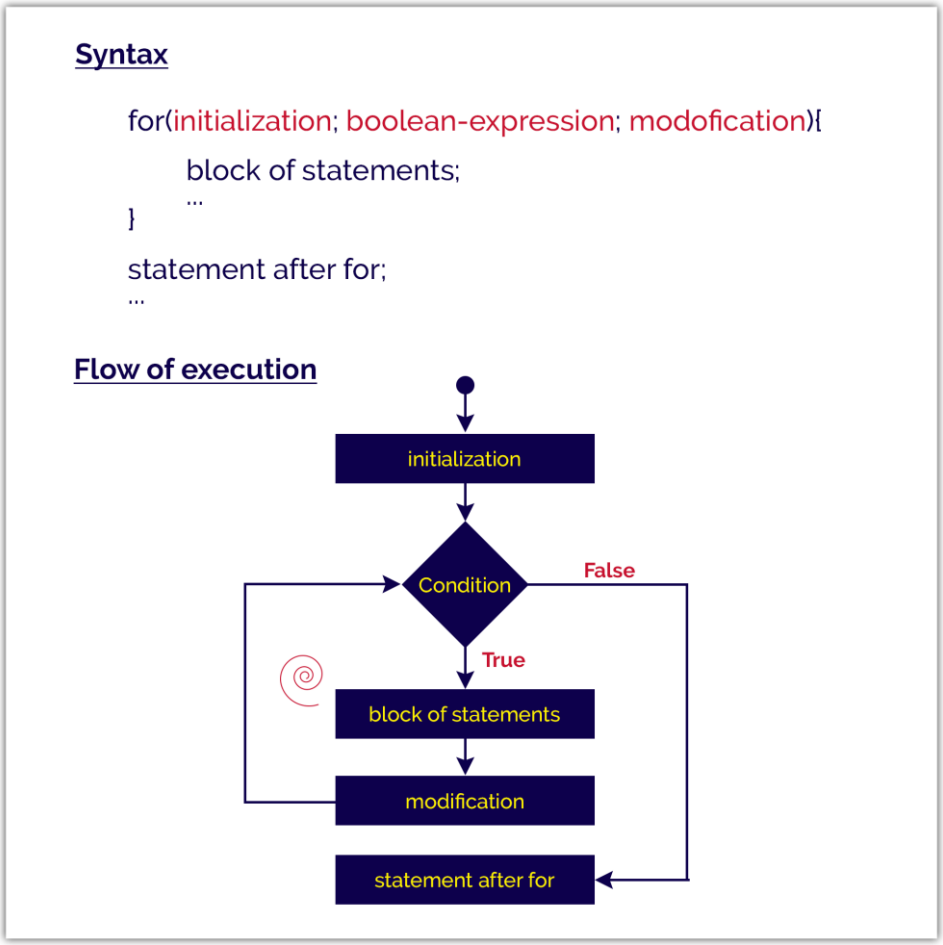



```
class DoWhileTest
{
        public static void main(String[] args)
        {
            int num = 1;
            do
            {
                System.out.println(num);
                num++;
            }while(num <= 10);
            System.out.println("Statement after do-while!");

        }
}
```

3.for loop statement : The for statement is used to execute a single statement or a block of statements repeatedly as long as the given condition is TRUE.

The for statement has the following syntax and execution flow diagram.



In for-statement, the execution begins with the initialization statement. After the initialization statement, it executes Condition. If the condition is evaluated to true, then the block of statements executed otherwise it terminates the for-statement. After the block of statements execution, the modification statement gets executed, followed by condition again.

```
class ForTest
{
        public static void main(String[] args)
        {
                for(int i = 0; i < 10; i++)
                {
                        System.out.println("i = " + i);
                }
                System.out.println("Statement after for!");
        }
}
```

3. Jump Statements

The java programming language supports jump statements that used to transfer execution control from one line to another line.

The java programming language provides the following jump statements.

- break statement
- continue statement

break

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Example

```
class BreakStatement
{
        public static void main(String args[] )
        {
                int i;
                i=1;
                while(true)
                {
                        if(i >10)
                        break;
                        System.out.print(i+" ");
                        i++;
                }
        }
}
```

Continue

This command skips the whole body of the loop and executes the loop with the next iteration. On finding continue command, control leaves the rest of the statements in the loop and goes back to the top of the loop to execute it with the next iteration (value).

Example

```
/* Print Number from 1 to 10 Except 5 */
class NumberExcept
{
        public static void main(String args[] )
        {
                int i;
                for(i=1;i<=10;i++)
                {
                        if(i==5)
                           continue;
                        System.out.print(i +" ");
                }
        }
}
```

INTRODUCING CLASSES AND METHODS

- classes usually consist of two things: instance variables and methods.
- Java is an object-oriented programming language, so everything in java program must be based on the object concept.
- In a java programming language, the class concept defines the skeleton of an object.
- The java class is a template of an object.
- The class defines the blueprint of an object.
- Every class in java forms a new data type.
- Once a class got created, we can generate as many objects as we want.
- Every class defines the properties and behaviors of an object.
- All the objects of a class have the same properties and behaviors that were defined in the class.

Every class of java programming language has the following characteristics.

- **Identity -** It is the name given to the class.
- **State -** Represents data values that are associated with an object.
- **Behavior -** Represents actions can be performed by an object.

Creating a Class

In java, we use the keyword class to create a class. A class in java contains properties as variables and behaviors as methods.

Syntax

```
class ClassName
{
        Variables/Data members declaration;
        Methods defination;
}
```

- The ClassName must begin with an alphabet, and the Upper-case letter is preferred.
- The ClassName must follow all naming rules.

Example

Here is a class called Box that defines three instance variables: width, height, and depth.

```
class Box
 {
        double width;
        double height;
        double depth;
        void volume()
        {
                ……..……..
        }
}
```

Creating an Object

- In java, an object is an instance of a class. When an object of a class is created, the class is said to be instantiated.
- All the objects that are created using a single class have the same properties and methods.
- But the value of properties is different for every object.

Syntax

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

- The objectName must begin with an alphabet, and a Lower-case letter is preferred.
- The objectName must follow all naming rules.

Example

To actually create a Box object, you will use a statement like the following:

```
Box  mybox = new Box();
```

The new operator dynamically allocates memory for an object.

Example

```
class Box
{
        double width;
        double height;
        double depth;
}
class BoxDemo
{
        public static void main(String args[])
        {
                Box mybox = new Box();
                double vol;
                mybox.width = 10;
                mybox.height = 20;
                mybox.depth = 15;
                vol = mybox.width * mybox.height * mybox.depth;
                System.out.println("Volume is " + vol);
        }
}
```

METHODS

- A method is a block of statements under a name that gets executes only when it is called.
- Every method is used to perform a specific task. The major advantage of methods is code re-usability (define the code once, and use it many times).
- In a java programming language, a method defined as a behavior of an object. That means, every method in java must belong to a class.
- Every method in java must be declared inside a class.

Every method declaration has the following characteristics.

- **returnType** - Specifies the data type of a return value.
- **name** - Specifies a unique name to identify it.
- **parameters** - The data values it may accept or recieve.
- **{ }** - Defienes the block belongs to the method.

Creating a method

A method is created inside the class

Syntax

```
class ClassName
{
        returnType methodName( parameters )
        {
            // body of method
        }
}
```

Calling a method

- In java, a method call precedes with the object name of the class to which it belongs and a dot operator.
- It may call directly if the method defined with the static modifier.
- Every method call must be made, as to the method name with parentheses (), and it must terminate with a semicolon.

Syntax

```
objectName.methodName(actualArguments );
```

Example

//Adding a Method to the Box Class

```
Class Box
{
        double width, height, depth;
        void volume()
        {
                System.out.print("Volume is ");
                System.out.println(width * height * depth);
        }
}
class BoxDemo3
{
        public static void main(String args[])
        {
                Box mybox1 = new Box();
                Box mybox2 = new Box();
                mybox1.width = 10;
                mybox1.height = 20;
                mybox1.depth = 15;
                mybox2.width = 3;
                mybox2.height = 6;
                mybox2.depth = 9;
                mybox1.volume();
                mybox2.volume();
        }
  }
```

STRING HANDLING:

The String class defined in the package java.lang package. A string is a sequence of characters are enclosed by double quotations. In java, objects of String are immutable which means a constant and cannot be changed once created.

How to Create a String class Object?

String class object can be created using two ways:

1. Using **String Literal**.
2. Using **new keyword**.

1. String literal: Java String literal is created by using double quotes.

Example: String s="Welcome";

2. new keyword : Java String is created by using a keyword "new".

Example: String s=new String("Welcome");

In such a case, JVM will create a new string object in normal (non-pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in the heap (non-pool).

Example :

```
class StringStorage
{
    public static void main(String args[])
    {
            String s1 = "Personal";
            String s2 = new String("Computer");
            System.out.println(s1);
            System.out.println(s2);
    }
}
```

Methods of String class

1. length(): This method is used to get the number of character of any string.

Example:

```
class StringHandling
{
    public static void main(String arg[])
    {
        int l;
        String s=new String("Java");
        l=s.length();
        System.out.println("Length: "+l);
    }
}
```

OUTPUT : 4

2. charAt(index) : charAt() method is used to get the character at a given index value.

Example:

```
class StringHandling
{
    public static void main(String arg[])
    {
        char c;
        String s=new String("Java");
        c=s.charAt(2);
        System.out.println("Character: "+c);
    }
}
```

OUTPUT : v

3. toUpperCase() : toUpperCase() method is use to convert lower case string into upper case.

Example:

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="Java";
        System.out.println("String: "+s.toUpperCase());
    }
}
```

OUTPUT : JAVA

4. toLowerCase() : toLowerCase() method is use to convert Upper case string into Lower case.

Example:

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="JAVA";
        System.out.println("String: "+s.toLowerCase());
    }
}
```

OUTPUT : java

5. concat() : concat() method is used to combined two string.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s1="JBREC";
                String s2="CSE AI&ML";
                System.out.println("Combined String: "+s1.concat(s2));
        }
}
```

6. Equals (): equals() method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s1="Java";
                String s2="Programming";
                String s3="Java";
                System.out.println("Compare String: "+s1.equals(s2));
                System.out.println("Compare String: "+s1.equals(s3));
        }
}
```

OUTPUT :

```
Compare String: False
Compare String: True
```

7. equalsIgnoreCase(): equalsIgnoreCase() method is case insensitive method, It return true if the contents of both strings are same otherwise false.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s1="Java";
                String s2="jAva";
                String s3="JAVA";
                System.out.println("Compare String: "+s1. equalsIgnoreCase(s2));
                System.out.println("Compare String: "+s1. equalsIgnoreCase(s3));
        }
}
```

OUTPUT :

```
Compare String: True
Compare String: True
```

8. compareTo() : compareTo() method is used to compare two strings by taking unicode values,

It return 0 if the string are same (string1 = = string2)

It return positive number, if string1 > string2

It return negative number, if string1 < string2

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s1="Java";
        String s2="Programming";
        int i;
        i=s1.compareTo(s2);
        if(i==0)
        {
            System.out.println("Strings are same");
        }
        else
        {
            System.out.println("Strings are not same");
        }
    }
}
```

OUTPUT: Strings are not same

9. compareToIgnoreCase() : compareToIgnoreCase() method is case insensitive method, which is used to compare two strings similar to compareTo().

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s1="Java";
        String s2="JAVA";
        int i;
        i=s1. compareToIgnoreCase(s2);
        if(i==0)
        {
            System.out.println("Strings are same");
        }
        else
        {
            System.out.println("Strings are not same");
        }
    }
}
```

OUTPUT: Strings are same

10. startsWith() : startsWith() method return true if string is start with given another string, otherwise it returns false.

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="Java is programming language";
        System.out.println(s.startsWith("Java"));
    }
}
```

OUTPUT : True

11.endsWith() : endsWith() method return true if string is end with given another string, otherwise it returns false.

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="Java is programming language";
        System.out.println(s.startsWith("language"));
    }
}
```

OUTPUT : True

12. subString() : subString() method is used to get the part of given string.

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="Java is programming language";
        System.out.println(s.substring(8)); // 8 is starting index
    }
}
```

OUTPUT:

programming language

13. indexOf() : indexOf() method is used find the index value of given string. It always gives starting index value of first occurrence of string.

Example :

```
class StringHandling
{
    public static void main(String arg[])
    {
        String s="Java is programming language";
        System.out.println(s.indexOf("programming"));
    }
}
```

OUTPUT : 8

14. lastIndexOf() : lastIndexOf() method used to return the starting index value of last occurence of the given string.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s1="Java is programming language";
                String s2="Java is good programming language";
                System.out.println(s1.lastIndexOf("programming"));
                System.out.println(s2.lastIndexOf("programming"));
        }
}
```

OUTPUT

8

13

15. trim() : trim() method remove space which are available before starting of string and after ending of string.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s="    Java is programming language    ";
                System.out.println(s.trim());
        }
}
```

16. split() : **split()** method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s="contact@tutorial4us.com";
                String[] s1=s.split("@");  // divide string based on @
                for(String c:s1) //  foreach loop
                {
                        System.out.println(c);
                }
        }
}
```

17. replace(): replace() method is used to return a duplicate string by replacing old character with new character.

Example :

```
class StringHandling
{
        public static void main(String arg[])
        {
                String s1="java";
                String s2=s1.replace('j', 'k');
                System.out.println(s2);
        }
}
```

Constructors: A Constructor is similar to a method that is used to initialize the instance variables. The purpose of a constructor is to initialize the instance variables. A constructor has the following characteristics.

- The constructor's name and class name should be same. And the constructor's name should end with a pair of simple braces.

  Example : In Person class, we can write a constructor as

```
Person()
{
}
```

- A constructor may have or may not have parameters. Parameters are variables to receive data from outside into the constructor.
- If a constructor does not have any parameters, it is called default constructor.

  Example :

```
Person()
{
}
```

- If a constructor has one or more parameters, it is called parameterized constructor.

```
Person( String s, int i )
{
}
```

- A constructor does not return any value, not even void.
- A constructor is automatically called and executed at the time of creating an object. While creating an object, if nothing is passed to the object, the default constructor is called and executed.

  Person p = new Person(); **// here default constructor is called.**

  If some values are passed to the object, then the parameterized constructor is called and executed.

  Person p = new Person("JBREC", 10); **// here parameterized constructor is called.**
- A Constructor is called and executed only once per object. This means when we create an object, the constructor is called. When we create second object, again the constructor is called second time.

Types of Constructors:

1 : Default or no argument Constructor

2 : Parameterized constructor.

3 : Constructor Overloading

1: Default or no argument Constructor: If a constructor does not have any parameters, it is called default constructor.

Syntax:

```
class ClassName
{
        .....
        ClassName( ) //Default constructor
        {
                .....
        }
}
```

```
class Person
{
        String name;
        int age;
        Person()
        {
                name="JBREC";
                age=25;
        }
        void display()
        {
                System.out.println("Name is ="+name);
                System.out.println("Age is ="+age);
        }
}
class Demo
{
        public static void main(String args[])
        {
                Person p = new Person();
                p.display();
                Person p1 = new Person();
                p1.display();

        }
}
```

2: Parameterized Constructor: If a constructor has one or more parameters, it is called parameterized constructor.

Syntax:

```
class ClassName
{
        .....
        ClassName(list of parameters) //parameterized constructor
        {
                .....
        }
}
```

```
class Person
{
        String name;
        int age;
        Person()
        {
                name="JBREC";
                age=25;
        }
        Person(String s, int i)
        {
                name=s;
                age=i;
        }
        void display()
        {
                System.out.println("Name is ="+name);
                System.out.println("Age is ="+age);
        }
}
class Demo
{
        public static void main(String args[])
        {
                Person p = new Person();
                p.display();
                Person p1 = new Person("JBIET",30);
                p1.display();

        }
}
```

3. Constructor Overloading: Writing two or more constructors with different parameters. This is called Constructor Overloading.

Syntax :

```
class  ClassName
{
        ClassName()
        { ..... }
        ClassName(datatype1 value1)
        { ..... }
        ClassName(datatype1 value1, datatype2 value2)
        {..... }
}
```

```
class Person
{
        String name;
        int age;
        Person()
        {
                name=”JBREC”;
                age=25;
                System.out.println(“Name is =”+name);
                System.out.println(“Age is =”+age);
        }
        Person(String s1)
        {
                name=”JBREC”;
                System.out.println(“Name is =”+name);
        }
        Person(String s, int i)
        {
                name=s;
                age=i;
                System.out.println(“Name is =”+name);
                System.out.println(“Age is =”+age);
        }
}
class Demo
{
        public static void main(String args[])
        {
                Person p = new Person();
                Person p1 = new Person(“BEC”);
                Person p2 = new Person(“JBIET”,30);
        }
}
```

What is the difference between Default constructor and Parameterized constructor.

Default Constructor	Parameterized Constructor
1 : Default Constructor is useful to initialize all objects with same data	1 : Parameterized Constructor is useful to initialize each object with different data.
2 : Default Constructor does not have any parameters	2 : Parameterized Constructor Will have one or more paramentets
3 : when data is not passed at the time of creating an object. Default constructor is called.	3 : when data passed at the time of creating an object. Parameterized constructor is called.

What is the difference between constructor and Method.

Constructors	Methods
1: A constructor is used to initialize the instance variables of a class.	1: A method is used for any general purpose processing or calculations.
2: A constructor's name and class name should be same.	2: A method name and class name can be same or different.
3: A constructor is called at the time of creating **the** object.	3: A method can be called after creating the object.
4 : A constructor is called only once per object	4: A method can be called several times on the object.
5: A constructor is automatically called and executed.	5: A method is executed only when we call it.

this Keyword :

- this is a keyword that refers to the current instance variables of a class.
- The main purpose of using this keyword is to differentiate the formal parameters and instance variables of the class are similar then JVM get ambiguity.
- To differentiate between formal parameters and instance variables of the class, the instance variable of the class must be preceded by 'this'.

Usage of this keyword

- It can be used to refer current class instance variable.
- It can be used to invoke current class constructor.
- It can be used to invoke current class method.

Example 1:

```
class Student
{
        int x=10;
        void display()
        {
                int x=20;
                System.out.println("Instance Variable x is="+this.x);
                System.out.println("Local Variable x is="+x);
        }
        public static void main(String args[])
        {
                Student s=new Student();
                s.display();
        }
}
```

OUTPUT :

```
javac Student.java
java Student
Instance Variable x is=10
Local Variable x is=20
```

Example 2:

```
class Demo
{
        int x,y;
        void display(int x, int y)
        {
                this.x=x;
                this.y=y;
        }
        void show(int x)
        {
                x=x;
                y=y;
        }
        public static void main(String args[])
        {
                Demo d=new Demo();
                d.display(20,30);
                System.out.println(d.x);
                System.out.println(d.y);

                Demo d1=new Demo();
                d1.show(40,60);
                System.out.println(d1.x);
                System.out.println(d1.y);
        }
}
```

OUTPUT :

```
javac Demo.java
java Demo
20
30
0
0
```

this() : this() method is used to call one constructor from another within the same class. Depending on the parameters we pass, appropriate this() method is called.

Restrictions of using this()

- If exist, it must be the first statement in the constructor.
- We cannot use two this() methods within the same constructor as both cannot be the first.

Example:

```
class Test
{
        Test()
        {
                this(10,20);
                System.out.println("Default Constructor");
        }
        Test(int x)
        {
                this();
                System.out.println(x);
        }
        Test(int x,int y)
        {
                System.out.println(x*y);
        }
        public static void main(String args[])
        {
                Test t1=new Test(10);
        }
}
```

OUTPUT :

```
javac Test.java
java Test
200
Default Constructor
10
```

Inheritance: Inheritance is one of the most important features of Object Oriented Programming. It creates new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called inheritance.

The existing class is referred to as the base class and the new class is referred to as the derived class.

Sub Class: The class that inherits properties from another class is called Sub class or Derived Class.

Super Class: The class whose properties are inherited by sub class is called Base Class or Super class.

Advantages of Inheritance:

- In inheritance, a programmer reuses the super class code without rewriting it. In creation of subclasses. So, developing the classes becomes very easy. Hence, the programmer's productivity is increased.(Redundancy of the code)
- Application development time is less.
- Application takes less memory.
- Application execution time is less.
- Application performance is enhance (improved).

How to define a Derived Class:

A derived class can be defined by specifying its relationship with the base class in addition to its own details.

Syntax : classname derived_classname extends base_classname

{

}

The **extends** keyword indicates that you are making a new class that derives from an existing class. The "extends" keyword is to increase the functionality.

Types of Inheritance:

Java Supports Three Types of Inheritance

1: Single Inheritance.

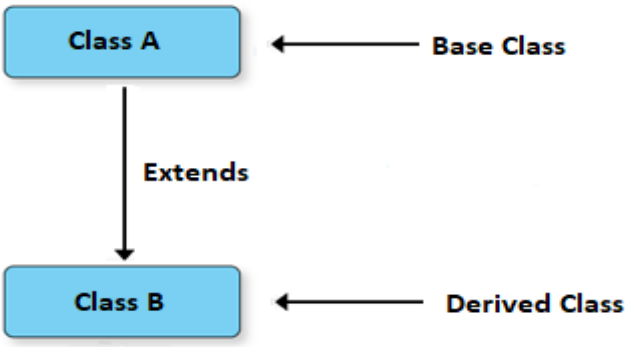
2: Multilevel Inheritance.

3: Hierarchical Inheritance.

4: Multiple Inheritances.

5: Hybrid Inheritance.

1. Single Inheritance: In Single Inheritance there is only one base class and only one derived class.



Syntax :

```
class A
{
        ---------
}
class B extends A
{
        -----------
}
```

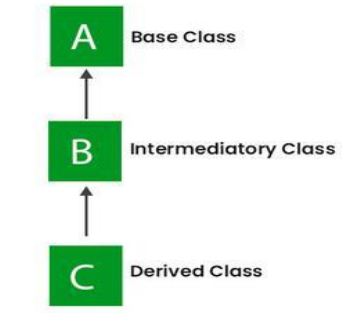
Single Inheritance Program

```
class A
{
        int x;
        void getx()
        {
                x=60;
                System.out.println("x value is="+x);
        }
}
class B extends A
{
        int y,total;
        void gety()
        {
                y=80;
                System.out.println("y value is="+y);
        }
        void sum()
        {
                total=x+y;
                System.out.println("Total value is="+total);
        }
}
```

```
class SingleInheritence
{
        public static void main(String args[])
        {
                B b=new B();
                b.getx();
                b.gety();
                b.sum();
        }
}
```

2. Multilevel Inheritance:

- When there is a chain of inheritance, it is known as multilevel inheritance.
- In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.



Multilevel Inheritance

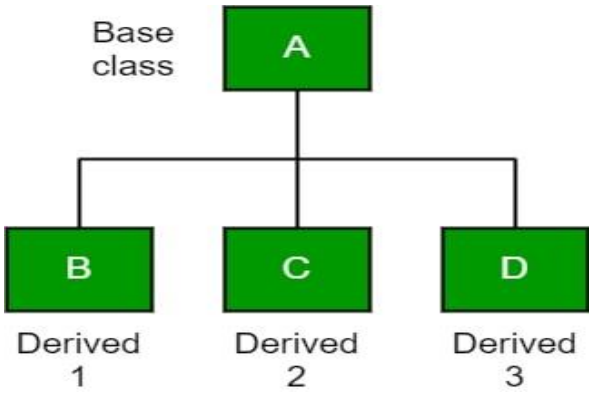
Syntax:

```
class A
{
        ---------
}
class B extends A
{
        ----------
}
class C extends B
{
        ----------
}
```

Multilevel Inheritance Program:

```
class A
{
        int x;
        void getx()
        {
                x=60;
                System.out.println("x value is="+x);
        }
}
class B extends A
{
        int y,total;
        void gety()
        {
                y=80;
                System.out.println("y value is="+y);
         }
}
class C extends B
{
        int total;
        void add()
        {
                total=x+y;
                System.out.println("Total value is="+total);
        }
}
class MultilevelInheritence
{
        public static void main(String args[])
        {
                C c=new C();
                c.getx();
                c.gety();
                c.add();
        }
}
```

3 : Hierarchical Inheritance : In Hierarchical Inheritance, one class serves as a super class (base class) for more than one subclass. In the below image, class A serves as a base class for the derived class B, C and D.



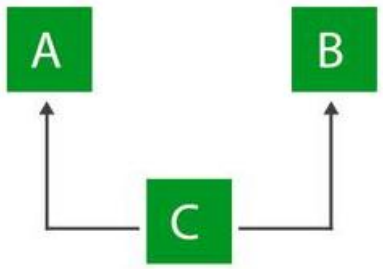

```
class Super
{
        int x,y;
        void getxy()
        {
                x=60;
                y=40;
        }
}
class Sub1 extends Super
{
        int sum;
        void add()
        {
                sum=x+y;
                System.out.println("Addition is="+sum);
        }
}
class Sub2 extends Super
{
        int sub;
        void minus()
        {
                sub=x-y;
                System.out.println("Subtaction is="+sub);
        }
}
```

```
class HierarchicalInheritence
{
	public static void main(String args[])
	{
		Sub1 s1=new Sub1();
		Sub2 s2=new Sub2();
		s1.getxy();
		s1.add();
		s2.getxy();
		s2.minus();

	}
}
```

4. Multiple inheritance

In multiple inheritance there exist multiple classes and singel derived class.

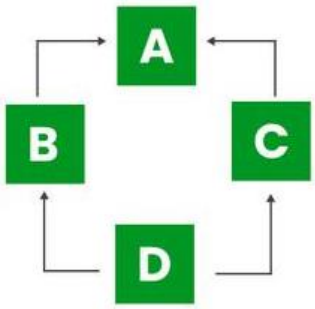


Multiple Inheritance

The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

5. Hybrid inheritance

It is a mix of two or more of the above types of inheritance. Since Java doesn't support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In Java, we can achieve hybrid inheritance only through Interfaces.



Hybrid Inheritance

ACCESS CONTROL(MEMBER ACCESS)

In Java, Access modifiers help to restrict the scope of a class, constructor, variable, method, or data member. It provides security, accessibility, etc to the user depending upon the access modifier used with the element.

Types of Access Modifiers in Java

There are four types of access modifiers available in Java:

- Default – No keyword required
- Private
- Protected
- Public

1. Default Access Modifier

- When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default.
- The default modifier is accessible only within package.
- It cannot be accessed from outside the package.
- It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//save by A.java

```
package pack;
class A
{
        void msg()
        {
                System.out.println("Hello");
        }
}
```

//save by B.java

```
package mypack;
import pack.*;
class B
{
    public static void main(String args[])
    {
        A obj = new A();    //Compile Time Error
        obj.msg();          //Compile Time Error
    }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

2. private

- The private access modifier is accessible only within the class.
- The private access modifier is specified using the keyword **private**.
- The methods or data members declared as private are accessible only **within the class** in which they are declared.
- Any other **class of** the **same package will not be able to access** these members.
- Top-level classes or interfaces can not be declared as private because private means “only visible within the enclosing class”.

Example

- In this example, we have created two classes A and Simple.
- A class contains private data member and private method.
- We are accessing these private members from outside the class, so there is a compile-time error.

```
class A
{
        private int data=40;
        private void msg()
        {
                System.out.println("Hello java");
        }
}
 public class Simple
  {
        public static void main(String args[])
        {
                A obj=new A();
                System.out.println(obj.data);        //Compile Time Error
                obj.msg();                           //Compile Time Error
        }
}
```

3. protected

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access modifier is specified using the keyword **protected**.

Example

- In this example, we have created the two packages pack and mypack.
- The A class of pack package is public, so can be accessed from outside the package.
- But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

//save by A.java

```
package pack;
public class A
{
        protected void msg()
        {
                System.out.println("Hello");
        }
}
```

//save by B.java

```
package mypack;
import pack.*;
class B extends A
{
        public static void main(String args[])
        {
                B obj = new B();
                obj.msg();
        }
}
```

4. public

- The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.
- The public access modifier is specified using the keyword **public**.

Example

//save by A.java

```
package pack;
public class A
{
        public void msg()
        {
                System.out.println("Hello");
        }
}
```

//save by B.java

```
package mypack;
import pack.*;
class B
{
        public static void main(String args[])
        {
                A obj = new A();
                 obj.msg();
        }
}
```

Table: class member access

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	YES	NO	NO	NO
Default	YES	YES	NO	NO
Protected	YES	YES	YES	NO
Public	YES	YES	YES	YES

Super Keyword

Super keyword in java is a reference variable that is used to refer parent class features.

Usage of Java super Keyword

- Super keyword At Variable Level
- Super keyword At Method Level
- Super keyword At Constructor Level

Super keyword at Variable Level:

- Whenever the derived class inherits base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.
- In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

Syntax : super.baseclass datamembername;

Example:

```
class Test
{
        int x=10;
}
class Demo extends Test
{
        int x=20;
        void display()
        {
                System.out.println(x);
                System.out.println(super.x);
                System.out.println(x+super.x);
        }
}
class Sample
{
        public static void main(String args[])
        {
                Demo d=new Demo();
                d.display();
        }
}
```

OUTPUT :

```
20
10
30
```

2. Super keyword At Method Level

- The super keyword can also be used to call parent class method.
- It should be use in case of method overriding.
- Super keyword use when base class method name and derived class method name have same name.

Example:

```
class One
{
        int i=10;
        void show()
        {
                System.out.println("Super class method : i="+i);
        }
}
class Two extends One
{
        int i=20;
        void show()
        {
                System.out.println("Sub class method : i="+i);
                super.show();
        }
}
class Demo
{
        public static void  main(String args[])
        {
                Two t = new Two();
                t.show();
        }
}
```

OUTPUT:

```
javac One.java
java Demo
Sub class method: i=20
Super class method: i=10
```

3. Super keyword At Constructor Level :

- The super keyword can also be used to call parent class constructor.
- Constructor are calling from bottom to top and executing from top to bottom.
- Super() method is used to call superclass constructor from subclass constructor. Depending on the parameters we pass, appropriate super() method is called.

Example:

```
class One
{
        int i;
        One(int i)
        {
                this.i=i;
        }
}
class Two extends One
{
        int i;
        Two(int a,int b)
        {
                super(a);
                i=b;
        }
        void show()
        {
                System.out.println("Sub class i="+i);
                System.out.println("Super class i="+super.i);
        }
}
class Demo
{
        public static void main(String args[])
        {
                Two t = new Two (20, 30);
                t.show();
        }
}
```

OUTPUT:

```
javac One.java
java Demo
Sub class method: i=30
Super class method: i=20
```

Final keyword in java: Final is a keyword it is used to make a variable as a constant, Restrict method overriding, Restrict inheritance.

In java language final keyword can be used in following way.

- Final Keyword at Variable Level
- Final Keyword at Method Level
- Final Keyword at Class Level

1. Final Keyword at Variable Level:

- Final keyword is used to make a variable as a constant. This is similar to const in other language.
- A variable declared with the final keyword cannot be modified by the program after initialization.
- This is useful to universal constants, such as "PI".

Example:

```
class FinalVariable
{
        public static void main(String args[])
        {
                int x=10;
                System.out.println(x);
                x=20;
                System.out.println(x);

                final int y=100;
                System.out.println(y);
                //y=200;   y cannot change
        }
}
```

OUTPUT:

```
javac FinalVariable.java
java FinalVariable
10
20
100
```

2. Final Keyword at Method Level:

- Methods which are declared as 'final' are called final methods.
- Final methods cannot be overridden, because they are not available to the sub classes.
- Therefore, only method overloading is possible with final methods.

Example:

```
class Test
{
        void display()
        {
                System.out.println("This is Non-Final Method");
        }
        final void show()
        {
                System.out.println("This is Final Method");
        }
}
class FinalMethod extends Test
{
        void display()
        {
                System.out.println("This is Non-Final Method overridden");
        }
        public static void main(String args[])
        {
                FinalMethod fm=new FinalMethod();
                fm.display();
                fm.show();
        }
}
```

OUTPUT:

```
javac Test.java
java FinalMethod

This is Non-Final Method overridden
This is Final Method
```

3. Final Keyword at Class Level:

- A final class is a class which is declared as 'final'.
- Final keyword before a class prevents inheritance. This means subclasses cannot be created to a final class.

Example:

```
final class Test
{
        void show()
        {
                System.out.println("This is Show Method");
        }
}
class Demo
{
        public static void main(String args[])
        {
                Test t=new Test();
                t.show();
        }
}
```

OUTPUT:

```
javac Test.java
java Demo

This is Show Method
```

Polymorphism: Polymorphism is derived from 2 greek words: **poly** and **morphs**. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Real life example of polymorphism in Java

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

Polymorphism is classified into two types

1. Compile Time Polymorphism
2. Run Time Polymorphism

1. Compile Time Polymorphism :

- Compile time Polymorphism is also known as Static Polymorphism or Early Binding.
- If the Compile time polymorphism nature of an entity is decided by the compiler during the compilation time then it is called Compile time polymorphism.
- Compile Time polymorphism in Java is achieved by method overloading.

Ad Hoc Polymorphism (Method Overloading): Writing two or more methods in the same class in such a way that each method has same name but with different parameters is called method overloading.

Example:

```
class OverloadDemo
{
        void add(int a)
        {
                System.out.println("a value is="+a);
        }
        void add(double b)
        {
                System.out.println("b value is="+b);
        }
        void add(int x,int y)
        {
                System.out.println("x + y value is="+(x+y));
        }
        void add(int x,int y,int z)
        {
                System.out.println("x + y +z value is="+(x+y+z));
        }
```

```
	public static void main(String args[])
	{
		OverloadDemo d=new OverloadDemo();
		d.add(10);
		d.add(10.5);
		d.add(20,30);
		d.add(20,35,45);
	}
}
```

OUTPUT:

```
10
10.5
50
100
```

2. Run Time Polymorphism :

- Run time Polymorphism is also known as Dynamic Polymorphism or Late Binding.
- If the Run time polymorphism nature of an entity is decided by the JVM during the run time then it is called Run time polymorphism.
- Run Time polymorphism in Java is achieved by method overriding.

Pure Polymorphism (Method Overriding): Writing two or more methods in super class and sub classes such that the methods have same name same parameters is called method overriding.

```
class Test
{
	void display()
	{
		System.out.println("Super class method");
	}
}
class Demo extends Test
{
	void display()
	{
		System.out.println("Sub class method");
	}
	public static void main(String args[])
	{
		Demo d=new Demo();
		d.display();
	}
}
```

OUTPUT:

```
Sub class method
```

Difference between Method Overloading and Method Overriding.

Method Overloading	Method Overriding
Writing two or more methods with the same name but with different parameters is called method overloading.	Writing two or more methods with the same name and same parameters is called method overriding.
Method overloading is done in the same class.	Method overriding is done in super and sub class.
In Method Overloading, Method return type can be same or different.	In Method Overriding, Method return types should also be same.
JVM decides which method is called depending on the difference in the method **parameter**.	JVM decides which method is called depending on the Data Type (Class) Of The Object Used To Call The Method.
Method overloading is done when the programmer wants to extend the already available feature.	Method overriding is done when the programmer wants to provide a different implementation for the same feature.
Method overloading **is code refinement. Same method is refined to perform a different task.**	Method overriding is code replacement. The subclass method overrides the super class method.

Object Class:

- There is a class with the name 'Object' in java.lang package which is the super class of all classes in java.
- Every class in java is a direct or indirect sub class of the Object class.
- The Object class defines the methods to compare objects, to convert an object into a string, to notify threads regarding the availability of an object.

The methods of Object Class are given

Method	Description
equals()	This method compares the references of two objects and if they are equal, it returns true, otherwise false. It compares the objects is dependent on the objects.
toString()	This method returns a string representation of an object. **Example**, the string representation of an integer object is an integer number displayed as string.
getClass()	This method gives an object that contains the name of a class to which an object belongs.
hashCode()	This method returns hash code number of an object.
Notify()	This method sends a notification to a thread which is waiting for an object.
notifyAll()	This method sends a notification for all waiting threads for the object.
Wait()	This method causes a thread to wait till a notification is received from a notify() or notifyAll() methods.
Clone()	This method creates a bit wise exact copy of an existing object.
Finalize()	This method is called by the garbage collector when an object is removed from memory.

FORMS OF INHERITANCE

- The inheritance concept used for the number of purposes in the java programming language. One of the main purposes is substitutability.
- The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object.
- **For example,** if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.
- The substitutability can achieve using inheritance, whether using extends or implements keywords.

The following are the differnt forms of inheritance in java.

- Specialization
- Specification
- Construction
- Extension
- Limitation
- Combination

Specialization

It is the most ideal form of inheritance. The subclass is a special case of the parent class. It holds the principle of substitutability.

Specification

This is another commonly used form of inheritance. In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them. The java provides concepts like abstract and interfaces to support this form of inheritance. It holds the principle of substitutability.

Construction

This is another form of inheritance where the child class may change the behavior defined by the parent class (overriding). It does not hold the principle of substitutability.

Eextension

This is another form of inheritance where the child class may add its new properties. It holds the principle of substitutability.

Limitation

This is another form of inheritance where the subclass restricts the inherited behavior. It does not hold the principle of substitutability.

Combination

This is another form of inheritance where the subclass inherits properties from multiple parent classes. Java does not support multiple inheritance type.

BENEFITS OF INHERITANCE

- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- Inheritance provides a clear model structure which is easy to understand.
- An inheritance leads to less development and maintenance costs.
- With inheritance, we will be able to override the methods of the base class so that the meaningful
- implementation of the base class method can be designed in the derived class. An inheritance leads to less development and maintenance costs.
- In inheritance base class can decide to keep some data private so that it cannot be altered by the derived class.

THE COSTS OF INHERITANCE

- Inheritance decreases the execution speed due to the increased time and effort it takes, the program to jump through all the levels of overloaded classes.
- Inheritance makes the two classes (base and inherited class) get tightly coupled. This means one cannot be used independently of each other.
- The changes made in the parent class will affect the behavior of child class too.
- The overuse of inheritance makes the program more complex.