विशाल जाधव सरांचे

VJTech Academy

Inspiring Your Success...

JAVA PROGRAMMING LANGUAGE IMPORTANT QUESTIONS LIST WITH ANSWERS

1. **Write a Java program to find out the even numbers from 1 to 100 using for loop.**

```
class FindEvenNo
{
	public static void main(String args[])
	{
		int i;
		System.out.println("Even Number between 1 to 100:");
		for(i=1;i<=100;i++)
		{
			if(i%2==0)
			{
				System.out.println(i);
			}
		}
	}
}
```

2. **Enlist the logical operators in Java.**

- Logical AND (&&)
- Logical OR (||)
- Logical NOT(!)

3. **Give the syntax and example for the following functions i) min () ii) Sqrt ()**

- Math.min(Variable1,Variable2) - find minimum value
- Math.sqrt(VariableName) - find square root of given number

```
class MathMethods
{
	public static void main(String args[])
	{
		int m=10,n=20;
		System.out.println("The minimum Value = "+Math.min(m,n));
		System.out.println("Square root of 9 = "+Math.sqrt(9));
	}
}
```

4. **Define the interface in Java**

- An interface is similar to the class.
- Interface is a collection of abstract methods and final static variables.
- Interface is used to achieve the multiple inheritance concepts in Java.
- All the abstract methods of the interface need to be defined in its sub class.
- An Interface does not contain any constructors.
- We cannot create the objects of an interface.
- All of the methods of the Interface are by default abstract.
- All of the variables of the Interface are by default static, final.

5. **Enlist any four inbuilt packages in Java**

- Java.io
- Java.applet
- Java.awt
- Java.util
- Java.lang

6. **Enlist any four compile time errors**

- Semicolon missing
- Mismatched brackets.
- Use of undeclared variables
- Syntax error
- Method is undefined.

7. **Explain any four features of Java.**

1) Compiled and Interpreted:

- Usually, computer language is either compiled or interpreted.
- But java combines both these approaches thats via Java is called as two stages compilation process programming language.
- Java compiler takes java source file(.java) as input and generates byte file(.byte).
- Byte file is not a machine code and this file not exists physically in your machine.
- Byte code generated virtually and process virtually.
- Java Interpreter generates machine code from byte code.

2) Platform Independent and Portable:

- Java program can be easily moved from one computer to another computer, anywhere and anytime.
- It means, if we develop Java code on Windows machine then you can easily run that code on other operating systems like Linux, Unix, etc.
- To move java code from one machine to another machine is known as portability.

3) Object Oriented:

- Java is true object-oriented language.
- Almost everything in java is an object.
- All program code and data reside within objects and classes.
- Java is a collection of rich set of predefined classes and packages, that we can use in our programs by inheritance.

4) Robust & Secure:

- Java is robust language.
- It provides many safeguards to ensure reliable code.
- It has strict compile time and runtime checking for data types.
- It supports garbage collection feature that would solve memory management problem.
- It supports exception handling which help us to captures many errors.
- Java is more secure programming language which is used for programming on internet.
- Java systems not only verify all memory access but also ensure that no viruses are communicated with an applet.

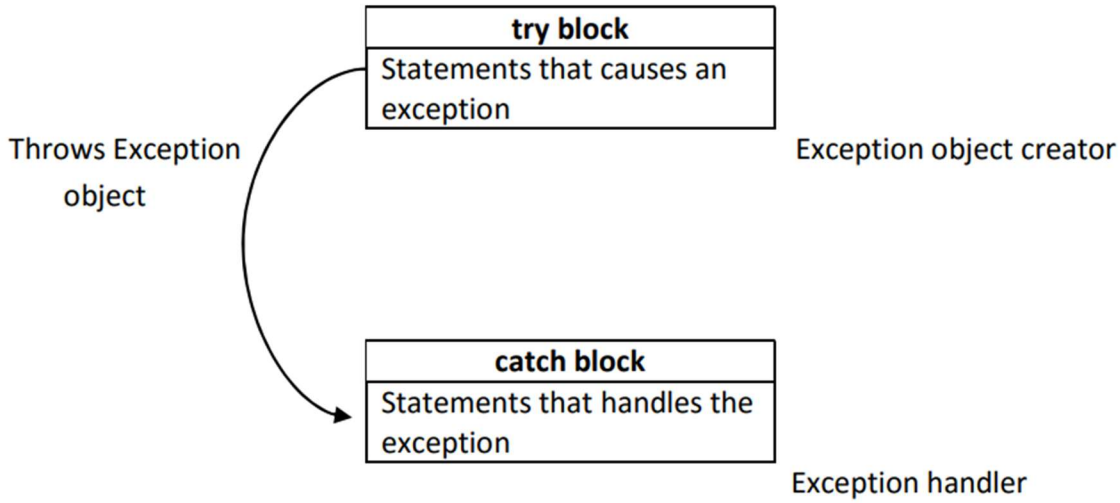
8. **Write the difference between vectors and arrays. (Any four points)**

Array	Vector
Array size cannot be changed	A Vector is a dynamic array, whose size can be increased
Array is not a class	A Vector is a class
The Array can store similar type of values	Vectors can store any type of objects
Array is not synchronized	Vector is synchronized
Methods are not provided for adding and removing the element from the array	Methods are provided for adding and removing the element from the Vector.
The size of array needs to be declared in advance.	No need to declare the size of the vector. You may give its size & you may not.
The Array can store primitive data types	Vector can store only object
Once declared array can't grow in size	Vector can always grow in size if you start adding more element to it than your declared size

9. **Explain exception handling mechanism. w.r.t. try, catch, throw and finally.**

- Exception may or may not be occur in program.
- An Exception is an unwanted event that interrupts the normal flow of the program.
- If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user.
- By using Exception handling mechanism, we handle an Exception and then print a user-friendly warning message to user.
- The basic idea of exception handling mechanism is to find the exception, Throws the exception, Catch the exception and handle the exception.
- We use five keywords for exception handling i.e. **try, catch, finally, throw, throws.**
    - **try** : try block contain those statements which may cause an exception.
    - **catch** : catch block contain statements which handle the exception.
    - **finally** : finally block always executed
    - **throw** : if we wants to throw the exception manually then we use throw keyword.
    - **throws** : we can list our multiple exceptions class which can occurs in method.

- Following diagram shows the exception handling mechanism:



- Program:

```
class ExceptionDemo
{
    public static void main(String args[])
    {
            int a=10,b=0,c;
            try
            {
                    c=a/b;
                    System.out.println("Value of c="+c);
            }
            catch(ArithmeticException e)
            {
                    System.out.println("Divide by zero exception occurred");
            }
    }
}
```

10. Explain any four visibility controls in Java

- Java provides four types of visibility control.

1) public
2) private
3) protected
4) Friendly Access

public:

- Any variables and methods declared as public, it can be accessible everywhere in the class, outside the class, outside the package, etc.

private:

- Those members are declared as private, it can accessible within the class in which they are declared.
- It cannot be inherited in its subclass.

protected:

- Those members are declared as protected, it can accessible in same class and its immediate sub-class.

Friendly Access:

- In the situation where no access modifier is specified then by default all members considered as friendly access level.
- There is basic difference between public and friendly access is that public members accessible anywhere but friendly access member available in same package not outside the package.

11. **Draw and explain life cycle of Thread.**

- A thread goes through various stages in its life cycle. The life cycle of the thread in java is controlled by JVM.
- The following diagram shows the complete life cycle of a thread.

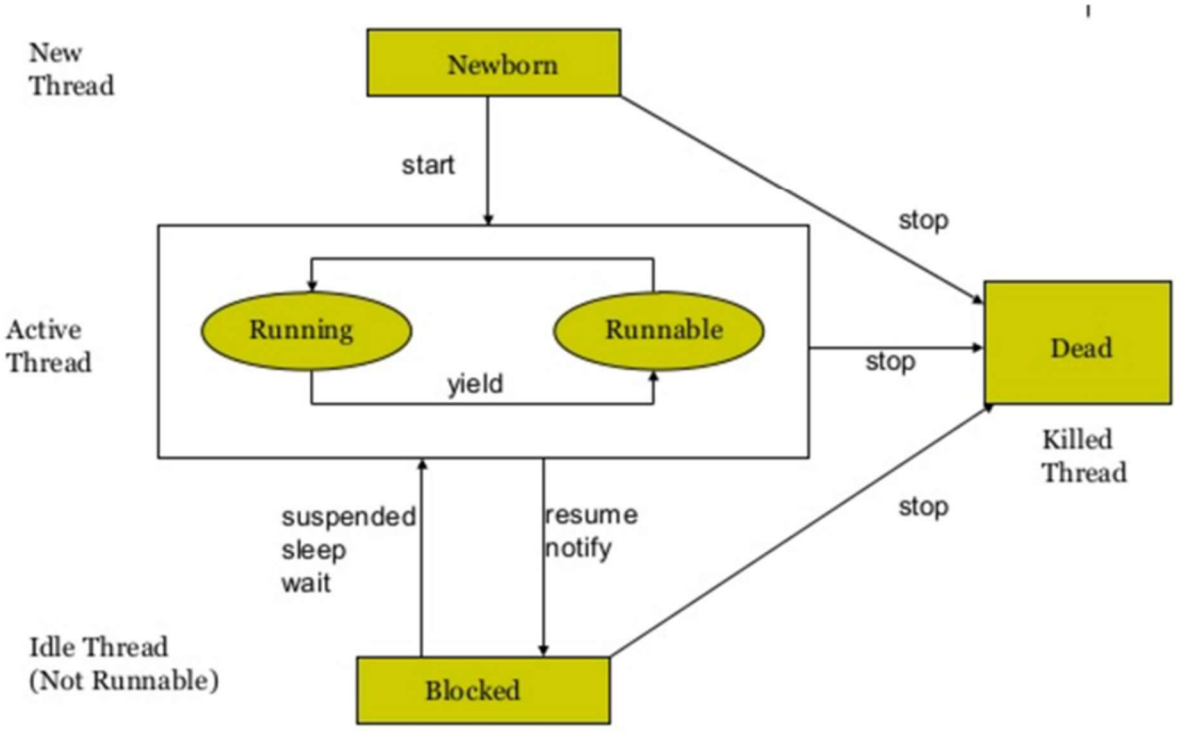


Fig. Life Cycle of Thread.

Following are the stages of the life cycle of a Thread.

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state

1. Newborn state:

- When a thread object is created, the thread is born then called as thread is in Newborn state.
- A new thread begins its life cycle in this state.
- When thread is in Newborn state then you can pass to the Running state by invoking start()
- method or kill it by using stop() method.

2. Runnable state:

- The thread is in runnable state after invocation of start() method.
- The Runnable state of thread means that the thread is ready for the execution and waiting for the availability of the processor.
- The threads which are ready for the execution are managed in the queue.
- The same priority threads are processed on the basis of First-come-First-Serve.

3. Running state:

- The Running state means that the processor has given it's time to thread for their execution
- When thread is executing, its state is changed to Running.
- A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

4. Blocked state:

- A thread can be temporarily suspended or blocked from entering into runnable or running state.
- Thread can be blocked due to waiting for some resources to available.
- Thread can be blocked by using following thread methods:

  I. suspended()

  II. wait()

  III. sleep()

- Following are the methods used to entering thread into Runnable state from Blocked state.

  I. The resume() method is invoked in case of suspended().

  II. The notify() method is invoked in case of wait().

  III. When the specified time is elapsed in the case of sleep().

5. Dead state:

- The thread will move to the dead state after completion of its execution. It means thread is in terminated or dead state when its run() method exits.
- Also Thread can be explicitly moved to the dead state by invoking stop() method.

12. **Explain how to create a package and how to import it.**

How to create user defined package in Java. Explain with a suitable example.

Package:

========

- Creating a package in java is quite easy.
- A package is always defined in a separate folder having the same name as a package name.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- The package must contain one or more classes or interfaces. This implies that a package cannot be empty.
- The classes or interfaces that are in the package must have their source files in the same directory structure as the name of the package.
- A package is always defined in a separate folder having the same name as a package name.
- Define all classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that its error free)

Steps for creation of package:

---

1) Write below line as first line of java source file

package packageName;

2) Write java class which you want to add inside the package and make it as public

3) Create directory whose name same as package name and stored java file inside it

and compile it.

4) Access created package in another program using below different ways

    import packageName.*;

    import packageName.className;

    fully qualified name

Example:

```
//Declare package msbte and make the class public
package msbte;
public class Sample
{
	public void display()
	{
		System.out.println("display method of Sample class");
	}
}
//Accessing msbte package
import msbte.*;
class AccessSamplePKG
{
	public static void main(String args[])
	{
		Sample s1=new Sample();
		s1.display();
	}
}
```

OUTPUT:

```
========
display method of Sample class
```

13. **Write a Java program in which thread A will display the even numbers between 1 to 50 and thread B will display the odd numbers between 1 to 50. After 3 iterations thread A should go to sleep for 500ms.**

```
class ThreadA extends Thread
{
	static int count=0;
	public void run()
	{
		for(int i=1;i<=50;i++)
		{
			if(i%2==0)
			{
				System.out.println("Even Number="+i);
			}
			count++;
			if(count==3)
			{
				try
				{
					sleep(500);
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
			}
		}

	}
}
class ThreadB extends Thread
{
	public void run()
	{
		for(int i=1;i<=50;i++)
		{
			if(i%2!=0)
			{
				System.out.println("Odd Number="+i);
			}
		}
```

```
    }
}
class MainThread
{
    public static void main(String args[])
    {
            ThreadA t1=new ThreadA();
            ThreadB t2=new ThreadB();
            t1.start();
            t2.start();
    }
}
```

14. **What is constructor? List types of constructors. Explain parameterized constructor with suitable example.**

- write constructor explanation

- list types of constructors

```
class Addition
{
    int a,b;
    Addition(int x,int y)
    {
            a=x;
            b=y;
    }
    void display()
    {
            System.out.println("Addition="+(a+b));
    }
    public static void main(String args[])
    {
            Addition a1=new Addition(100,200);
            a1.display();
    }
}
```

15. **Explain the difference between string class and string buffer class. Explain any four methods of string class.**

String	StringBuffer
The length of the String object is fixed.	The length of the StringBuffer can be increased.
String object is immutable.	StringBuffer object is mutable
String object is slower in performance	StringBuffer object is faster..
Consumes more memory.	Consumes less memory.
String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
String objects are stored in a constant pool	StringBuffer objects are stored in heap memory.
String objects provides less functionality to the strings as compared to the class StringBuffer	StringBuffer objects provide more functionality to the strings as compared to the class String.

16. **Write a java program to sort a 1-d array in ascending order using bubble-sort.**

```
import java.util.*;
class SortArray
{
    public static void main(String args[])
    {
            int a[]=new int[10];
            int n,i,j,temp;
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Size of Array:");
            n=sc.nextInt();
            System.out.println("Enter Array Elements:");
            for(i=0;i<n;i++)
            {
                    a[i]=sc.nextInt();
            }
            //sorting logic
            for(i=1;i<n;i++)
            {
                    for(j=0;j<n-i;j++)
                    {
                            if(a[j]>a[j+1])
                            {
                                    temp=a[j];
                                    a[j]=a[j+1];
                                    a[j+1]=temp;
                            }
                    }
            }
            System.out.println("Sorted Array elements:");
            for(i=0;i<n;i++)
            {
                    System.out.println(a[i]);
            }
    }
}
```

17. **Explain switch case and conditional operator in java with suitable example. 6 Marks**

Switch case:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.
- The break and default keywords are optional
- switch, case, break and default four keywords used.
- Syntax:

```
switch(expression/value)
{
        case value-1: //block of statements
                      break;
        case value-2: //block of statements
                      break;
        case value-N: //block of statements
                      break;
        default: //block of statements
}
```

- Example:

```
class SwitchCaseDemo
{
    Public static void main(String args[])
    {
        int no=2;
        switch(no)
        {
            case 1: System.out.println("Number is ONE");
                    break;
            case 2: System.out.println("Number is TWO");
                    break;
            case 3: System.out.println("Number is THREE");
                    break;
            default: System.out.println("Invalid input");
        }
    }

}
```

conditional operator

- Conditional operator is called as Ternary operator.
- The symbol (?:) is used for conditional operator.
- Syntax:

```
Condition? Expression1: Expression2;
```

- In above syntax, if condition is true then program controller executes Expression1.
- if condition is false then program controller executes Expression2.
- Program:

```
class ConditionalOPDemo
{
    public static void main(String args[])
    {
        int a,b,c;
        a=100;
        b=50;
        c=(a>b?a:b);
        System.out.println("Greatest Number=%d",c);
    }
}
```

18. **Explain single and multilevel inheritance with proper example 6 Marks**

1) Single Inheritance:

- This is one of the types of inheritance.

- To create new class from only one base class is known as single inheritance.

- Syntax:

```
class DerivedClassName extends BaseClassName
{
 //body of Derived Class
}
```

- Program:

```
//Single Inheritance
import java.util.*;
class Student
{
    int rollno;
    String name;
    void get_stud_info()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Student Roll No:");
            rollno=sc.nextInt();
            System.out.println("Enter Student Name:");
            name=sc.next();
    }
    void disp_stud_info()
    {
            System.out.println("Student ROll No:"+rollno);
            System.out.println("Student Name:"+name);
    }
}
class Test extends Student
{
            int marks1,marks2;
            void get_stud_marks()
            {
                    Scanner sc=new Scanner(System.in);
                    System.out.println("Enter Student Test-1 Marks:");
                    marks1=sc.nextInt();
```

```
                System.out.println("Enter Student Test-2 Marks:");
                marks2=sc.nextInt();
        }
        void disp_stud_marks()
        {
                System.out.println("Test-1 Marks:"+marks1);
                System.out.println("Test-2 Marks:"+marks2);
        }
}
class SingleInheritanceDemo
{
        public static void main(String args[])
        {
                Test t1=new Test();
                t1.get_stud_info();
                t1.get_stud_marks();
                System.out.println("*******STUDENT INFORMATION SYSTEM********");
                t1.disp_stud_info();
                t1.disp_stud_marks();
        }
}
```

2) Multi-level Inheritance:

- The mechanism of deriving the class from another derived class is known as multi-level inheritance.
- To create new class from another derived class is known as multi-level inheritance.
- It is one of the types of inheritance.
- Syntax:

```
class BaseClass1
{
    //body of BaseClass1 Class
}
class DerivedClass1 extends BaseClass1
{
     //body of DerivedClass1
}
 class DerivedClass2 extends DerivedClass1
{
     //body of DerivedClass2
}
```

-Example

```
//Multi-level Inheritance
import java.util.*;
class Student
```

```
{
    int rollno;
    String name;
    void get_stud_info()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Student Roll No:");
            rollno=sc.nextInt();
            System.out.println("Enter Student Name:");
            name=sc.next();
    }
    void disp_stud_info()
    {
            System.out.println("Student ROll No:"+rollno);
            System.out.println("Student Name:"+name);
    }
}
class Test extends Student
{
    int marks1,marks2;
    void get_stud_marks()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Student Test-1 Marks:");
            marks1=sc.nextInt();
            System.out.println("Enter Student Test-2 Marks:");
            marks2=sc.nextInt();
    }
    void disp_stud_marks()
    {
            System.out.println("Test-1 Marks:"+marks1);
            System.out.println("Test-2 Marks:"+marks2);
    }
}
class Result extends Test
{
    int total_marks;
    void get_total_marks()
    {
            total_marks=marks1+marks2;
    }
    void disp_total_marks()
```

```
    {
            System.out.println("Total Marks:"+total_marks);
    }
}
class MultilevelInheritanceDemo
{
    public static void main(String args[])
    {
            Result t1=new Result();
            t1.get_stud_info();
            t1.get_stud_marks();
             t1.get_total_marks();
             System.out.println("********STUDENT INFORMATION SYSTEM*********");
             t1.disp_stud_info();
            t1.disp_stud_marks();
             t1.disp_total_marks();
    }
}
```

19. **Define constructor. List its types**

- Constructor is a special member function of the class.
- It is used to initialize the data members of the objects.
- There is no any return type for the constructor.
- Constructor name and class name both are same.
- Constructor automatically called when object is created.
- Constructor is used for creation of an object.
- Suppose, in our class we have not defined constructor then system will supply the default constructor for creation of an objects.
- There are three different types of the constructor:

1) Default constructor

2) Parameterized constructor

3) Copy constructor

20. **Define class and object.**

1) Objects:

- Basic runtime entities known as object.
- They may represent person, table, bank account or any item that program may handle.
- Object is a collection of data and functions.
- Objects are created from class.

2) Classes:

- It is a collection of similar types of objects.
- Class is a collection of data and functions, functions operate on data.
- You may create objects from the class.
- When you define the class then memory will not be allocated for the members.
- Class shows data abstraction and data encapsulation features.
- Example: Fruit mango
- In above example, mango is an object which is created from class Fruit.

21. **List any four Java API packages.**

- java.io.*
- java.awt.*
- java.applet.*
- java.lang.*

22. **Define array. List its types.**

- Array is a collection of similar types of elements.
- Array index should be begin with 0 and end with SIZE-1.
- Array elements are stored in contiguous memory location.
- Three types of Array:

a) Single dimensional array

b) Two dimensional array

c) Multi-dimensional array

23. **List access specifiers in Java.**

- private
- public
- protected
- friendly (default)

24. **Define a class circle having data members Pi and radius. Initialize and display values of data members also calculate area of circle and display it.**

```
import java.util.*;
class Circle
{
    float radius;
    final static float PI=3.14f;
    void getdata()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter radius of circle:");
            radius=sc.nextFloat();
    }
    void putdata()
    {
            System.out.println("Radius of circle="+radius);
            System.out.println("Value of PI="+PI);
            System.out.println("Area of circle="+(PI*radius*radius));
    }
    public static void main(String args[])
    {
            Circle c1=new Circle();
            c1.getdata();
            c1.putdata();
    }
}
```

25. **Define exception. State built-in exceptions.**

- write exception theory

- give some list of predefined exception

```
class ExceptionDemo
{
    public static void main(String args[])
    {
            int a=10,b=0,c;
            try
            {
                    c=a/b;
                    System.out.println("Value of c="+c);
            }
            catch(ArithmeticException e)
            {
                    System.out.println("Divide by zero exception occurred");
            }
    }
}
```

26. **Differentiate between class and interfaces.**

Class	Interface
1) We can create objects from class.	We cannot create the objects from interface.
2) The members of a class can be private, public or protected.	The members of an interface are always public.
3) Class contains abstract or non-abstract methods.	Interface has only abstract methods.
4) Class can contain methods definitions.	Interface can contain only declaration of methods without body.
5) The class keyword is used to declare class.	The interface keyword is used to declare interface.
6) Class can implement any number of interfaces and can extend only one class.	An interface can extend multiple interfaces but cannot implement any class.
7) Constructor present in class	Constructor not present in class

8) Example:
class Abc
{
void draw();
}
	Example:
	interface Xyz
	{
	void draw();
	}

27. **Define type casting. Explain its types with syntax and example**

Type Casting:

- The conversion of one data type value to another data type is knows as Type Casting.

- There are two types of data type conversion/Type Casting.

I) Implicit Type Casting

- The type casting which is done by the system is known as implicit type casting.

- Example:

```
int a=10;
float b;
b=a; //implicit type casting
b=10.000000
```

II) Explicit Type Casting

- The type casting which is done by the programmer is known as explicit type casting.

- Example:

int a=10;

float b;

b=(float) a; //Explicit type casting

b=10.000000

28. **Write a program to create vector with five elements as (5, 15, 25, 35, 45). Insert new element at 2nd position. Remove 1st and 4th element from vector.**

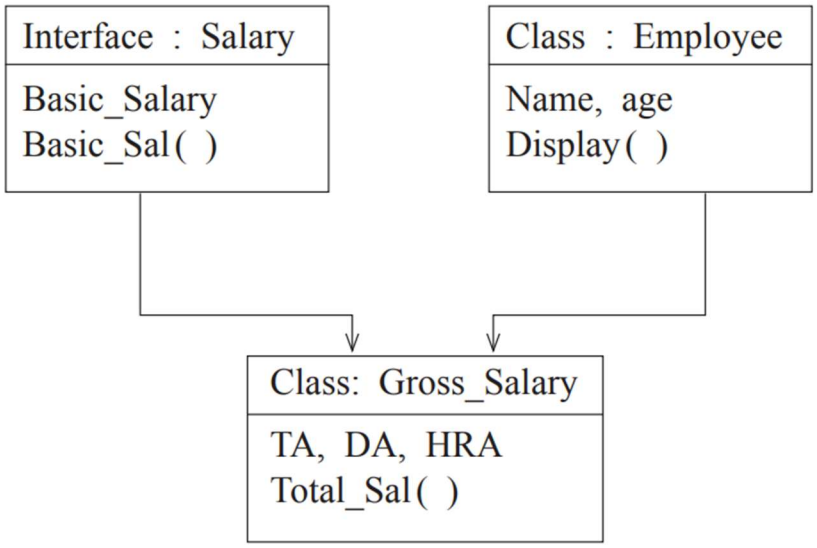
```
import java.util.*;
class VectorDemo
{
    public static void main(String args[])
    {
            Scanner sc=new Scanner(System.in);
            Vector v1=new Vector();
            v1.addElement(new Integer(5));
            v1.addElement(new Integer(15));
            v1.addElement(new Integer(25));
            v1.addElement(new Integer(35));
            v1.addElement(new Integer(45));
            System.out.println("Enter element for insertion:");
            int x=sc.nextInt();
            v1.insertElementAt(x,2);
            v1.removeElementAt(1);
            v1.removeElementAt(4);
            System.out.println("Vector Elements:"+v1);
    }
}
OUTPUT
Enter element for insertion:
120
Vector Elements:[5, 120, 25, 35]
```

29. **Write a program to create two threads one thread will print even no. between 1 to 50 and other will print odd number between 1 to 50**

```
class ThreadA extends Thread
{
	public void run()
	{
		for(int i=1;i<=50;i++)
		{
			if(i%2==0)
			{
				System.out.println("Even Number="+i);
			}
		}

	}
}
class ThreadB extends Thread
{
	public void run()
	{
		for(int i=1;i<=50;i++)
		{
			if(i%2!=0)
			{
				System.out.println("Odd Number="+i);
			}
		}
	}
}
class MainThread
{
	public static void main(String args[])
	{
		ThreadA t1=new ThreadA();
		ThreadB t2=new ThreadB();
		t1.start();
		t2.start();
	}
}
```

30. **Implement the following inheritance**




```
//Note: In this program, I assume 5%TA, 10% DA and 15% HRA of basic salary.

import java.util.*;
interface Salary
{
    int Basic_Salary=1000;
    void Basic_sal();
}
class Employee
{
    String name;
    int age;
    void getdata()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Employee Name:");
            name=sc.next();
            System.out.println("Enter Employee Age:");
            age=sc.nextInt();
    }
    void display()
    {
            System.out.println("Employee Name:"+name);
```

```
                System.out.println("Employee Age:"+age);
    }
}
class Gross_Salary extends Employee implements Salary
{
    int TA,DA,HRA;
    public void Basic_sal()
    {
                TA=(Basic_Salary*5)/100;
                DA=(Basic_Salary*10)/100;
                HRA=(Basic_Salary*15)/100;
    }
    void Total_Sal()
    {
                int total=(Basic_Salary+TA+DA+HRA);
                System.out.println("Basic Salary:"+Basic_Salary);
                System.out.println("TA:"+TA);
                System.out.println("DA:"+DA);
                System.out.println("HRA:"+HRA);
                System.out.println("Total Salary:"+total);
    }
    public static void main(String args[])
    {
                Gross_Salary g1=new Gross_Salary();
                g1.getdata();
                g1.Basic_sal();
                g1.display();
                g1.Total_Sal();
    }
}
```

OUTPUT

```
Enter Employee Name:
James
Enter Employee Age:
54
Employee Name:James
Employee Age:54
Basic Salary:1000
TA:50
DA:100
HRA:150
```

Total Salary:1300

31. **List any eight features of Java**

1) Compiled and Interpreted
2) Platform Independent and Portable
3) Object Oriented
4) Robust & Secure
5) Distributed
6) Simple, Small and Familiar
7) Multi-threaded and Interactive
8) High Performance
9) Dynamic & Extensible

32. **State use of finalize( ) method with its syntax**

- finalize() method: The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.
- Syntax:

```
protected void finalize()
{
        //body
}
```

- The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing.
- Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

33. **Name the wrapper class methods for the following: (i) To convert string objects to primitive int. (ii) To convert primitive int to string objects.**

=> int i=Integer.parseInt(str) Converts string to primitive integer

=> str=Integer.toString(i) Primitive Integer to String

34. **List the types of inheritances in Java**

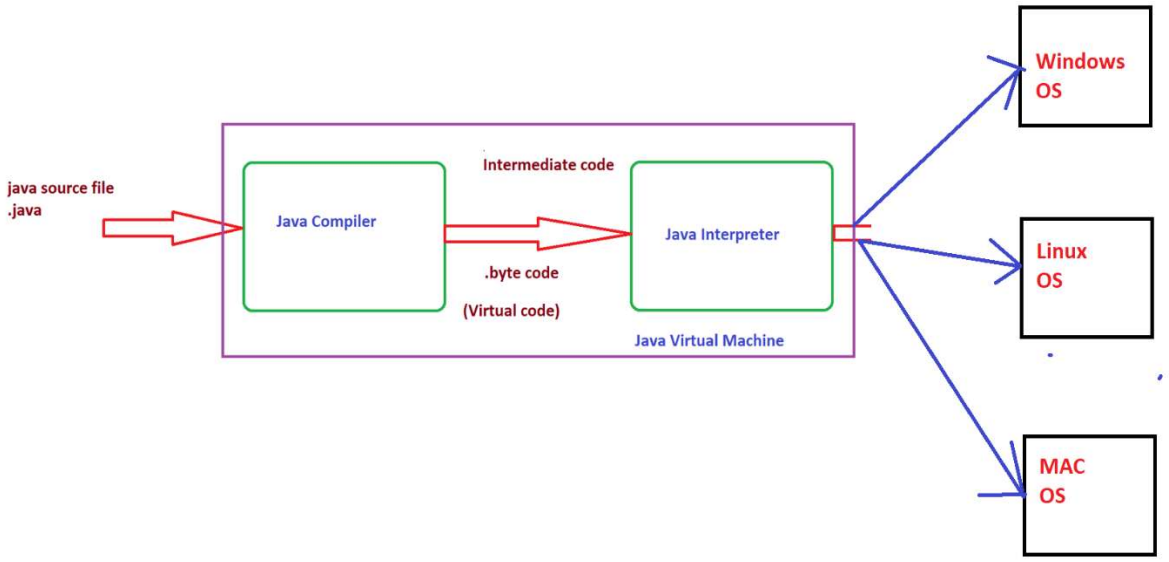
- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance
- Hybrid Inheritance
- Hierarchical Inheritance

35. **Write the syntax of try-catch-finally blocks.**

```
try
{
    //statements that may cause an exception
}
catch(ExceptionClassName object)
{
    //statements that handle the exception
}
finally
{
    //statements that always executed
}
```

36. **Explain the concept of platform independence and portability with respect to Java language.**

- Java program can be easily moved from one computer to another computer, anywhere and anytime.
- It means, if we develop Java code on Windows machine then you can easily run that code on other operating systems like Linux, Unix, etc.
- To move java code from one machine to another machine is known as portability.
- Following diagram will give more information about this feature.



37. Explain the types of constructors in Java with suitable example

- Constructor is a special member function of the class.
- It is used to initialize the data members of the objects.
- There is no any return type for the constructor.
- Constructor name and class name both are same.
- Constructor automatically called when object is created.
- Constructor is used for creation of an object.
- Suppose, in our class we have not defined constructor then system will supply the default constructor for creation of an objects.
- There are three different types of the constructor:

1) Default constructor
2) Parameterized constructor
3) Copy constructor

1) Default constructor:

- When constructor does not take any parameters then it is called as default constructor.
- Syntax:

```
class ClassName
{
    ClassName()
    {
        //body of constructor.
    }
}
```

2) Parameterized constructor:

- When constructor takes any parameters then it is called as Parameterized constructor.
- Syntax:

```
class ClassName
{
        ClassName(parameter_list)
        {
                //body of constructor.
        }
}
```

2) Copy constructor:

- To initialize data members of the object, we are passing another object as argument is called as copy constructor.
- When constructor takes reference of its class as parameter then it is called as copy constructor.
- Syntax:

```
class ClassName
{
        ClassName(ClassName ObjectName)
        {
                //body of constructor.
        }
}
```

- Example:

```
class Item
{
        int x;
        Item()
        {
                x=100;
        }
        Item(Item m)
        {
                x=m.x;
```

```
        }
        void display()
        {
                System.out.println("Value of X : "+x);
        }
        public static void main(String args[])
        {
                Item i1=new Item();
                Item i2=new Item(i1);
                i1.display();
                i2.display();
        }
}
```

38. **Explain the two ways of creating threads in Java.**

There are two different ways of creating the Thread in Java programming language.

By extending Thread class.

Following steps are required to create the thread in Java by using Thread class:

I. Declare the class by extending the Thread class.

```
class ThreadX extends Thread
{
    //body of ThreadX class
}
```

II. Implement the run() method.

```
public void run()
{
    //thread code
}
```

III. Create the object of a thread class.

```
ThreadX t1=new ThreadX();
```

IV. Invoke the start() method.

```
t1.start();
```

Example:

```
//*********************************************************************************************
class ThreadX extends Thread
{
    public void run()
    {
          for(int i=1;i<=5;i++)
          {
                    System.out.println("From ThreadX:i="+i);
          }
    }
}

//*********************************************************************************************
class ThreadY extends Thread
{
    public void run()
    {
          for(int j=1;j<=5;j++)
          {
                    System.out.println("From ThreadY:j="+j);
          }
    }
}

//*********************************************************************************************
class ThreadDemo
{
    public static void main(String args[])
    {
          ThreadX t1=new ThreadX();
          ThreadY t2=new ThreadY();
          t1.start();
          t2.start();
    }
}
```

By implementing Runnable interface.

We can create the thread by implementing the Runnable interface. Following steps are required to create the Thread in Java.

I. Declare the class by implementing the Runnable interface.

```
class RunnableX implements Runnable
{
    //body of ThreadX class
}
```

II. Implement the run() method.

```
public void run()
{
    //thread code
}
```

III. Create the object of a Thread class by passing object of class as argument which is implemented from the Runnable interface.

```
RunnableX r1=new RunnableX ();
Thread t1=new Thread(r1);
```

IV. Invoke the start() method.

```
t1.start();
```

Example:

```
//*********************************************************************************************
class RunnableX implements Runnable
{
    public void run()
    {
         for(int i=1;i<=5;i++)
         {
                  System.out.println("From RunnableX:i="+i);
         }
    }
}

//*********************************************************************************************
class RunnableY implements Runnable
{
    public void run()
    {
         for(int j=1;j<=5;j++)
         {
                  System.out.println("From RunnableY:j="+j);
         }
    }
}

//*********************************************************************************************
class RunnableDemo
{
    public static void main(String args[])
    {
         RunnableX r1=new RunnableX();
         RunnableY r2=new RunnableY();

         Thread t1=new Thread(r1);
         Thread t2=new Thread(r2);

         t1.start();
         t2.start();
    }
}
```

39. **Define a class student with int id and string name as data members and a method void SetData ( ). Accept and display the data for five students.**

```
import java.util.*;
class Student
{
    int id;
    String name;
    void SetData()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Student ID:");
            id=sc.nextInt();
            System.out.println("Enter Student Name:");
            name=sc.next();
    }
    void display()
    {
            System.out.println("Student ID:"+id);
            System.out.println("Student Name:"+name);
    }
    public static void main(String args[])
    {
            Student s[]=new Student[5];
            System.out.println("Enter five students information");
            for(int i=0;i<5;i++)
            {
                    s[i]=new Student();
                    s[i].SetData();
            }
            for(int i=0;i<5;i++)
            {
                    s[i].display();
            }
    }
}
```

40. **Write a program to create two threads. One thread will display the numbers from 1 to 50 (ascending order) and other thread will display numbers from 50 to 1 (descending order).**

```
class ThreadX extends Thread
{
	public void run()
	{
		for(int i=1;i<=50;i++)
		{
			System.out.println("ThreadX="+i);
		}
	}
}
class ThreadY extends Thread
{
	public void run()
	{
		for(int j=50;j>=1;j--)
		{
			System.out.println("ThreadY="+j);
		}
	}
}
class MainThreadDemo
{
	public static void main(String args[])
	{
		ThreadX t1=new ThreadX();
		ThreadY t2=new ThreadY();
		t1.start();
		t2.start();
	}
}
```

41. Explain the command line arguments with suitable example

- By using command line argument, we can pass input values to our java program.
- This is one of the way for giving inputs to our java program.
- While executing code we are passing list of values to our java program.
- Whatever the values we have passed that would be stored in command line argument i.e String args[].
- Initially string object args is empty but when we send list of arguments that all values should be stored in args of string array.
- This string array will grow automatically.
- Example: java filename java VJTech 100 200
- Program

```
class CommandLineArgsDemo
{
        public static void main(String args[])
        {
                int a,b,c;
                a=Integer.parseInt(args[0]);
                b=Integer.parseInt(args[1]);
                c=a+b;
                System.out.println("Addition="+c);
        }
}
```

42. **Write a program to input name and salary of employee and throw user defined exception if entered salary is negative.**

```
import java.util.*;
class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}
class Employee
{
    String name;
    int salary;
    void getdata()
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter employee name:");
        name=sc.next();
        System.out.println("Enter employee salary:");
        salary=sc.nextInt();
    }
    void putdata()
    {
        if(salary<0)
        {
            try
            {
                throw new MyException("Salary Negative Exception");
            }
            catch(MyException e)
            {
                System.out.println(e);
            }
        }
        else
        {
            System.out.println("Employee Name:"+name);
            System.out.println("Employee Salary:"+salary);
        }
    }
    public static void main(String args[])
    {
        Employee e1=new Employee();
        e1.getdata();
        e1.putdata();
    }
```

```
}
```

43. **Describe the use of any methods of vector class with their syntax**

- Vector is an extensible array.
- Vector is a collection of objects and it can be retrieved by using index number.
- Array is a collection of similar types of elements but its size is fixed.
- But vector is a collection of objects but its size is not fixed.
- Vector class provides array of variable size.
- The main difference between vector and array: Vector automatically grow when they run out of space.
- Vectors class provides extra method for adding and removing elements.
- The class is used to create dynamic array known as Vector that can holds objects of any type and any numbers.
- Vector is a predefined class which is present under java.util package.
- Vectors are created like array as follow:

1) Declaration of Vector without size.

Vector VectorName=new Vector():

2) Declaration of Vector with size.

Vector VectorName=new Vector(10):

Important Vector Methods:

Vector v1=new Vector();

1) v1.addElement(item) - Adds the item to the vector at the end.

2) v1.elementAt(10) - Gives the name of 10th object.

3) v1.size() - Gives the number of objects present.

4) v1.removeElement(item) - Removes the specified item from the vector.

5) v1.removeElementAt(n) - Removes the item stored in nth position.

6) v1.removeAllElements() - Removes all the elements in the list.

7) v1.copyInto(array) - Copies all items from vector to array.

8) v1.insertElementAt(item,n) - Insert the item at nth position.

Program:

```
import java.util.*;
class VectorDemo1
{
    public static void main(String args[])
    {
            Vector v1=new Vector();
            v1.addElement(new Integer(10));
            v1.addElement(new Integer(30));
            v1.addElement(new Integer(60));
            v1.addElement(new Integer(70));
            v1.addElement(new Integer(80));
            v1.addElement(new Integer(100));
            System.out.println("Initial Vector Elements = "+v1);
            v1.removeElementAt(3);
            v1.removeElementAt(4);
            v1.insertElementAt(new Integer(150),3);
            System.out.println("Final Vector Elements = "+v1);
    }
}
```

44. Describe instance Of and dot (.) operators in Java with suitable example

Instanceof Operator:

- This operator returns true if the object on the left side is an instance of the class given on the right side.
- Syntax:

```
if(object instanceof ClassName)
{
        //body
}
```

- Example:

```
import java.util.*;
class InstanceOfOpDemo
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        if(sc instanceof Scanner)
        {
            System.out.println("sc is an object of Scanner class");
        }
    }
}
```

Dot Operator:

- Object contain data members and member function.
- So, we can access them by using object name and dot operator.
- Syntax:

```
ObjectName.Variable_Name=value;
ObjectName.Method_Name(Parameter_list);
```

- Example:

```
a1.a=100;
a1.b=200;
a1.getdata();
a1.display();
```

45. **Differentiate between method overloading and method overriding.**

Method Overloading	Method Overriding
Method names are same but its arguments are different is knowns as method overloading.	Base class method and derived class method is same then base class method overridden by derived class.
Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
Method overloading is performed within class.	Method overriding occurs in two classes.
In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.

46. **WAP to accept a password from the user and throw "Authentication Failure" exception if the password is incorrect.**

```
import java.util.*;
class MyException extends Exception
{
	MyException(String msg)
	{
		super(msg);
	}
}
class AuthenticationDemo
{
	public static void main(String args[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter your password:");
		String psw=sc.next();
		if(psw.equals("vjtech"))
		{
			System.out.println("Authentication Sucessful");
		}
		else
		{
			try
			{
				throw new MyException("Authentication Failure");
			}
			catch(MyException e)
			{
				System.out.println(e);
			}
		}
	}
}
```

47. **Write a program to input name and balance of customer and thread a user defined exception if balance less than 1500.**

```
import java.util.*;
class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}
class UserdefinedExeDemo
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter customer name:");
        String name=sc.next();
        System.out.println("Enter customer balance:");
        int balance=sc.nextInt();

        if(balance<1500)
        {
            try
            {
                throw new MyException("Balance less than 1500");
            }
            catch(MyException e)
            {
                System.out.println(e);
            }
        }
        else
        {
            System.out.println("Balance greater than 1500");
        }
    }
}
```

48. **Define an exception called 'No match Exception' that is thrown when a string is not equal to "MSBTE". Write program.**

```
import java.util.*;
class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}
class UserdefinedExeDemo1
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter any string:");
        String str=sc.next();
        if(str.equals("MSBTE"))
        {
            System.out.println("Given String match with MSBTE");
        }
        else
        {
            try
            {
                throw new MyException("No match exception");
            }
            catch(MyException e)
            {
                System.out.println(e);
            }
        }
    }
}
```

49. **Write a program to create package Math_s having two classes as addition and subtraction. Use suitable methods in each class to perform basic operations.**

Create Addition class and added it under Math_s package

```
package Math_s;
public class Addition
{
    public void add(int x,int y)
    {
            System.out.println("Addition="+(x+y));
    }
}
```

Create Subtraction class and added it under Math_s package

```
package Math_s;
public class Subtraction
{
    public void sub(int x,int y)
    {
            System.out.println("Subtraction="+(x-y));
    }
}
```

Create class where we are accessing package Math_s.

```
import Math_s.*;
class AccessMathPkg
{
        public static void main(String args[])
        {
            Addition a1=new Addition();
            Subtraction s1=new Subtraction();
            a1.add(100,50);
            s1.sub(100,50);
        }
}
```

50. State three uses of final keyword?

- final is a predefined keyword.

- Following are the uses of final keyword:

1) To make constant variable:

- If we declare the variable using final keyword then that variable become constant in java.

- Constant variable means, once it is created then we cannot change its value later.

- Constant is a variable which cannot change its value during the execution of program.

Example:

```
class finalKeywordDemo
{
    public static void main(String args[])
    {
            final float PI=3.14f;                //constant variable
            int radius=2;
            float area;
            area=(PI*radius*radius);
            System.out.println("Area of Circle="+area);
    }
}
/*
Area of Circle=12.56
*/
```

2) To avoid method overriding:

- Suppose, base class method name and derived class method name are same then base class method overridden by derived class method.

- If we want to avoid method overriding then we can use final keyword before the base class method declaration.

- Example:

```
//To avoid method overriding
class Base
{
    final void display()
    {
            System.out.println("display method of base class");
    }
}
class Derived extends Base
{
    void display()
```

```
    {
        System.out.println("display method of derived class");
    }
}
class AvoidMethodOverriding
{
    public static void main(String args[])
    {
        Derived d1=new Derived();
        d1.display();
    }
}
/*
AvoidMethodOverriding.java:11: error: display() in Derived cannot override display() in Base
    void display()
        ^
  overridden method is final
1 error
*/
```

3) To avoid inheritance:

- To avoid inheritance then we can declare the base class using final keyword.

- If we declare base class using final keyword then we cannot create subclass from it.

- Example:

```
final class Base
{
    void display()
    {
        System.out.println("display method of base class");
    }
}
class Derived extends Base
{
    void show()
    {
        System.out.println("show method of derived class");
    }
}
class AvoidInheritance
{
    public static void main(String args[])
    {
        Derived d1=new Derived();
        d1.display();
        d1.show();
    }
```

```
}
/*
AvoidInheritance.java:9: error: cannot inherit from final Base
class Derived extends Base
                ^
1 error
*/
```

51. **Explain method overriding with suitable example.**

Method Overriding:

- Suppose, base class and derived class method names are same.
- When base class method derived in derived class then it got override.
- It means base class method overridden by derived class method.
- To call overridden method, we can use super keyword.
- We use syntax for calling hidden method: super.methodName();
- Program:

```
class Base
{
    void display()
    {
            System.out.println("display method of base class");
    }
}
class Derived extends Base
{
    void display()
    {
            super.display();
            System.out.println("display method of derived class");
    }
}
class MethodOverriding
{
    public static void main(String args[])
    {
            Derived d1=new Derived();
            d1.display();
    }
}
```

OUTPUT

```
display method of base class
display method of derived class
```

52. **What is the multiple inheritance? Write a java program to implement multiple inheritance**

OR

Explain how interface is used to achieve multiple Inheritances in Java.

- To inherit the properties of more than one base class into sub class is known as multiple inheritances.
- In multiple inheritance we can combine the features of more than one existing classes into new class.
- Below diagram shows the multiple inheritance concepts:

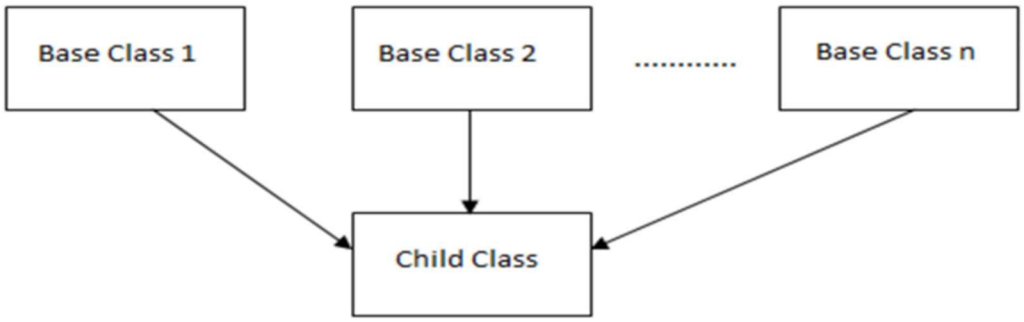


Fig: Multiple Inheritance

- Java classes cannot have more than one super class. But in most of the real time application multiple inheritances is required. So, java provides an alternative approach is known as interface.
- Interface is a collection of static final variables and abstract methods. It is used to achieve the multiple inheritance in Java.
- Below diagram shows, how to achieve the multiple inheritance in Java language:

Multiple Inheritance in Java

- In first diagram, class implementing more than one interface and in second diagram interface extending more than one interfaces to achieve the multiple inheritance in Java.

- Program:

```
interface Abc
{
        void display();
}

interface Xyz
{
        void show();
}

class Mnp implements Abc, Xyz
{
        public void display()
        {
                System.out.println("I am from Abc interface:");
        }
        public void show()
        {
                System.out.println("I am from Xyz interface:");
        }
        public static void main(String args[])
        {
                Mnp m1=new Mnp();
                m1.display();
                m1.show();
        }
}
```

53. **Define class Student with suitable data members create two objects using two different constructors of the class.**

```
class Student
{
    int rollno;
    String name;
    float marks;
    Student()
    {
            rollno=1010;
            name="Dennis";
            marks=98.78f;
```

```
    }
    Student(int r,String n,float m)
    {
            rollno=r;
            name=n;
            marks=m;
    }
    void display()
    {
            System.out.println("Student Roll No:"+rollno);
            System.out.println("Student Name:"+name);
            System.out.println("Student Marks:"+marks);
    }
    public static void main(String args[])
    {
            Student s1=new Student();
            Student s2=new Student(2020,"James",75.55f);
            s1.display();
            s2.display();
    }
}
```

54. **Write a program to define class Employee with members as id and salary. Accept data for five employees and display details of employees getting highest salary.**

```
import java.util.*;
class Employee
{
    int id;
    float salary;
    void get_emp_info()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Employee ID:");
            id=sc.nextInt();
            System.out.println("Enter Employee Salary:");
            salary=sc.nextFloat();
    }
    public static void main(String args[])
    {
```

```
            Employee e[]=new Employee[5];
            for(int i=0;i<5;i++)
            {
                  e[i]=new Employee();
                  e[i].get_emp_info();
            }
            int highest_id=e[0].id;
            float highest_salary=e[0].salary;
            for(int i=1;i<5;i++)
            {
                  if(e[i].salary>highest_salary)
                  {
                        highest_salary=e[i].salary;
                        highest_id=e[i].id;
                  }
            }
            System.out.println("***Highest Salary Employee Details***");
            System.out.println("Employee ID:"+highest_id);
            System.out.println("Employee Salary:"+highest_salary);
      }
}
```

55. **Define a class 'Book' with data members bookid, bookname and price. Accept data for seven objects using Array of objects and display it.**

```
import java.util.*;
class Book
{
      int book_id;
      String book_name;
      float book_price;
      void get_book_info()
      {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Book ID:");
            book_id=sc.nextInt();
            System.out.println("Enter Book Name:");
            book_name=sc.next();
            System.out.println("Enter Book price:");
            book_price=sc.nextFloat();
```

```
    }
    void disp_book_info()
    {
            System.out.println("Book ID:"+book_id);
            System.out.println("Book Name:"+book_name);
            System.out.println("Book Price:"+book_price);
    }
    public static void main(String args[])
    {
            Book b[]=new Book[7];
            for(int i=0;i<7;i++)
            {
                    b[i]=new Book();
                    b[i].get_book_info();
            }
            for(int i=1;i<5;i++)
            {
                    b[i].disp_book_info();
            }
    }
}
```

56. **What is garbage collection in Java? Explain finalize method in Java.**

- Garbage Collection in Java is a process by which the programs perform memory management automatically.
- The Garbage Collector(GC) finds the unused objects and deletes them to reclaim the memory.
- In Java, dynamic memory allocation of objects is achieved using the new operator that uses some memory and the memory remains allocated until there are references for the use of the object.
- When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed.
- There is no explicit need to destroy an object as Java handles the de-allocation automatically.
- Garbage collection in Java happens automatically during the lifetime of the program.
- We were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.
- How to delete an object in Java:

1) If you want to make your object eligible for Garbage Collection, assign its reference variable to null.

2) Primitive types are not objects. They cannot be assigned null.

- finalize() method: The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing.
- Syntax:

```
protected void finalize()
{
        //body
}
```

- The Garbage collector of JVM collects only those objects that are created by new keyword. So if you have created any object without new, you can use finalize method to perform cleanup processing.
- gc() method: The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.
- Syntax:

```
public static void gc()
{
        //body
}
```

- Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.
- Program:

```
class TestGarbage
{
    protected void finalize()
    {
            System.out.println("object is garbage collected");
    }
    public static void main(String args[])
    {
            TestGarbage s1=new TestGarbage();
            s1=null;
            System.gc();
    }
}
```

57. Describe types of variables in Java with their scope

- Scope of the variables is nothing but the life time of variables.
- Its scope is depended on where in the program that variables are declared.
- The area of the program where the variable is accessible is called as scope.
- There are three different types of variables present in java

1) Instance Variables:

- Instance variable is declared inside the class.
- Instance variables are created when the objects are instantiated.
- Instance variables allocate separate memory space when object is created.
- They take different values for each object.

2) Class Variables:

- Class variables are declared inside the class.
- They are the global to the class.
- It common between all objects.
- Only one memory location is created for each class variables.

3) Local Variables:

- Local Variables declared and used inside the functions.
- The variables which are declared inside the body of methods in known as local variables.
- They are not available outside the method.
- Local variables can be declared inside the body of methods which is starting from opening curly braces ({} and closing braces(}).

Example:

```
class Student
{
    int rollno; //instance variable
    String name; //instance variable
    float marks; //instance variable
    static int college_code=1010; //class variable
```

```
        void calc_marks()
        {
                int total; //local variable
        }
}
```

58. **Write all primitive data types available in Java with their storage sizes in bytes**

Data Type	Size
byte	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
boolean	1 bit
char	2 bytes

59. **Write a program to copy all elements of one array into another array.**

```
import java.util.*;
class CopyArray
{
	public static void main(String args[])
	{
		int a[]=new int[5];
		int b[]=new int[5];
		int i;
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter Arry Elements:");
		for(i=0;i<5;i++)
		{
			a[i]=sc.nextInt();
		}
		//copy one array into another array
		for(i=0;i<5;i++)
		{
			b[i]=a[i];
		}
		System.out.println("Copied Array Elements:");
		for(i=0;i<5;i++)
		{
			System.out.println(a[i]+"\t");
		}
	}
}
```

60. **Write a program to print even and odd number using two threads with delay of 1000ms after each number**

```
class ThreadX extends Thread
{
	public void run()
	{
		for(int i=1;i<=10;i++)
		{
			if(i%2==0)
			{
				System.out.println("Even Number:"+i);
				try
				{
					sleep(1000);
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
			}
		}
	}
}
class ThreadY extends Thread
{
	public void run()
	{
		for(int i=1;i<=10;i++)
		{
			if(i%2!=0)
			{
				System.out.println("Odd Number:"+i);
				try
				{
					sleep(1000);
				}
				catch(Exception e)
				{
					System.out.println(e);
				}
			}
		}
	}
}
class MainThread
{
	public static void main(String args[])
```

```
    {
            ThreadX t1=new ThreadX();
            ThreadY t2=new ThreadY();
            t1.start();
            t2.start();
    }
}
```

61. **Write a program to print all the Armstrong numbers from 0 to 999.**

```
class ArmstrongNumber
{
    public static void main(String args[])
    {
        int i,no,sum,rem;
        for(i=1;i<=999;i++)
        {
          no=i;
            sum=0;
            while(no>0)
            {
                rem=no%10;
                sum=sum+(rem*rem*rem);
                no=no/10;
            }
            if(i==sum)
            {
                System.out.println("Armstrong Number:"+i);
            }
        }
    }
}
```

62. **Develop and Interest Interface which contains Simple Interest and Compound Interest methods and static final field of rate 25%. Write a class to implement those methods**

-> Simple Interest formula: SI = P * R * T,

where P = Principal, R = Rate of Interest, T = Time period.

->I = (P*(1+(R/N))^(N*T))

Where I = Interest,P = Principle, the original amount,R = interest rate, N = number of times the interest is compounded in a year, T = number of years

```
import java.util.*;
interface Interest
{
    int R=25;
    void Simple_Interest();
    void Compound_Interest();
}
class InterestCalc implements Interest
{
    int P,T,N;
    public void Simple_Interest()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Principal Amount:");
            P=sc.nextInt();
            System.out.println("Enter no of years:");
            T=sc.nextInt();
            int SI=(P*R*T);
            System.out.println("Simple Interest="+SI);
    }
    public void Compound_Interest()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter Principal Amount:");
            P=sc.nextInt();
            System.out.println("Enter no of years:");
            T=sc.nextInt();
            System.out.println("No of times compounded interest calculated:");
            N=sc.nextInt();
            float CI=(P*(1+(R/N))^(N*T));
            System.out.println("Compound Interest="+CI);
    }
    public static void main(String args[])
    {
            InterestCalc c1=new InterestCalc();
            c1.Simple_Interest();
            c1.Compound_Interest();
```

```
    }
}
```

63. **Write a program to print the sum, difference and product of two complex numbers by creating a class named "Complex" with separate methods for each operation whose real and imaginary parts are entered by user.**

```
import java.util.*;
class Complex
{
    int real,imag;
    void getdata()
    {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter real number:");
            real=sc.nextInt();
            System.out.println("Enter Imaginary number:");
            imag=sc.nextInt();
    }
    void sum(Complex m)
    {
            System.out.println("Sum="+(real+m.real)+"+i"+(imag+m.imag));
    }
    void difference(Complex m)
    {
            System.out.println("Difference="+(real-m.real)+"+i"+(imag-m.imag));
    }
    void product(Complex m)
    {
            System.out.println("Product="+(real*m.real)+"+i"+(imag*m.imag));
    }
    public static void main(String args[])
    {
            Complex c1=new Complex();
            Complex c2=new Complex();
            c1.getdata();
            c2.getdata();
            c1.sum(c2);
            c1.difference(c2);
            c1.product(c2);
    }
}
```

64. **Write a program to display ASCII value of a number 9**

```
class DisplayASCII
{
    public static void main(String args[])
    {
            char a='9';
            int x=a;
            System.out.println("ASCII Value of 9="+x);
    }
}
```