



||Jai Sri Gurudev||
BGSKH Education Trust
**BGS COLLEGE OF ENGINEERING AND
TECHNOLOGY**



MAHALAKSHMIPURAM, BENGALURU - 560086



Object Oriented Programming with JAVA Lab Manual

For third Semester BE [VTU/CBCS, 2022-23 Syllabus]

Subject Code:

B	C	S	3	0	6	A
---	---	---	---	---	---	---



Name:

Branch:

USN:

PREFACE

Welcome to the Object-Oriented Programming with JAVA Laboratory! As one of the most popular programming languages in use today. This book for practical “Object Oriented Programming with JAVA” is an introduction to object-oriented programming and their implementation with the JAVA language. It is planned for beginners who would like to learn the subject through programs.

The primary objective of this manual is to provide a comprehensive and hands-on experience in JAVA. Each experiment is meticulously outlined to guide you through the fundamental concepts and practical applications of object-oriented programming. The experiments cover a range of topics, from basic array to advanced concepts like inheritance, multithreading, exceptions etc.

BGS College of Engineering & Technology

Department of Artificial Intelligence and Data Science

Vision

To nurture innovative minds for academic excellence with emphasis on in-depth technical knowledge for creating a value based sustainable society.

Mission

M1: Developing practically trained skilled professionals to meet the demands of the corporate world.

M2: Creating an ecosystem of academic excellence through the best teaching-learning methods.

M3 Grooming professionals with high ethical values and ability to solve real-life problems.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

- 1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- 3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- 4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- 5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- 6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- 7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

- 8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- 11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

INDEX

Sl.No	Contents	Page No
	Lab assessment rubrics for CIE	iv
1	Develop a JAVA program to add TWO matrices of suitable order N (The value of N should be read from command line arguments).	1
2	Develop a stack class to hold a maximum of 10 integers with suitable methods. Develop a JAVA main method to illustrate Stack operations.	3
3	A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method raise Salary (percent) increases the salary by the given percentage. Develop the Employee class and suitable main method for demonstration.	9
4	<p>A class called MyPoint, which models a 2D point with x and y coordinates, is designed as follows:</p> <ul style="list-style-type: none"> • Two instance variables x (int) and y (int). • A default (or "no-arg") constructor that constructs a point at the default location of (0, 0). • An overloaded constructor that constructs a point with the given x and y coordinates. • A method setXY() to set both x and y. • A method getXY() which returns the x and y in a 2-element int array. • A toString() method that returns a string description of the instance in the format "(x, y)". • A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates • An overloaded distance(MyPoint another) that returns the distance from this point to the given MyPoint instance (called another) • Another overloaded distance() method that returns the distance from this point to the origin (0,0) <p>Develop the code for the class MyPoint. Also develop a JAVA program (called TestMyPoint) to test all the methods defined in the class.</p>	12
5	<p>Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase().</p> <p>Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.</p>	15
6	Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.	18

7	Develop a JAVA program to create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height) that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods	21
8	Develop a JAVA program to create an outer class with a function display. Create another class inside the outer class named inner with a function called display and call the two functions in the main class.	24
9	Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.	26
10	Develop a JAVA program to create a package named mypack and import & implement it in a suitable class.	28
11	Write a program to illustrate creation of threads using runnable class. (start method start each of the newly created thread. Inside the run method there is sleep() for suspend the thread for 500 milliseconds).	30
12	Develop a program to create a class MyThread in this class a constructor, call the base class constructor, using super and start the thread. The run method of the class starts after this. It can be observed that both main thread and created child thread are executed concurrently.	32
13	VIVA Questions	34

Laboratory Outcomes:

The student should be able to:

- Analyse primitive constructs of JAVA programming language.
- Analyse features of object-oriented programming with JAVA.
- Apply knowledge on packages, multithreaded programming and exceptions.

CO1	Demonstrate proficiency in writing simple programs involving branching and looping structures
CO2	Design a class involving data members and methods for the given scenario
CO3	Apply the concepts of inheritance and interfaces in solving real world problems.
CO4	Use the concept of packages and exception handling in solving complex problem
CO5	Apply concepts of multithreading, autoboxing and enumerations in program development

CO-PO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3	2	2	-	-	-	-	-	-	-	-	-
CO2	2	3	3	-	-	-	-	-	-	-	-	-
CO3	2	2	3	1	1	-	-	-	-	-	-	-
CO4	2	2	2	2	-	-	-	-	-	-	-	-
CO5	2	2	3	1	1	-	-	-	-	-	-	-

LABORATORY RUBRICS

Sl No	Description	Marks
1	<i>Continuous Evaluation</i>	<i>25 (Scaled down to 15 marks)</i>
	a. Record and Observation write up and punctuality	10
	b. Conduction of experiment and output and Viva Voice	15
2	<i>Laboratory Test</i>	<i>50 (Scaled down to 10 marks)</i>

Program 1:

Develop a JAVA program to add TWO matrices of suitable order N (The value of N should be read from command line arguments).

```
import java.util.Scanner;
public class matrix_add{
    public static void main (String args[])
    {
        Scanner in = new Scanner(System.in);
        int N = Integer.parseInt(args[0]);
        System.out.println("the value of N is "+N);
        int[][] A = new int[10][10];
        int[][] B = new int[10][10];
        int[][] C = new int[10][10];
        System.out.println("Enter the elements of the matrix A ");
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                A[i][j] = in.nextInt();
        System.out.println("Enter the elements of the matrix B ");
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                B[i][j] = in.nextInt();
        //addition of 2 matrix
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                C[i][j]= A[i][j] + B[i][j];
        // Display the elements of the matrix
        System.out.println("Elements of the matrix C are");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                System.out.print(C[i][j] + " ");
        }
    }
}
```

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

For compilation

C:\java3sem>javac matrix_add.java

For Execution

C:\java3sem>java matrix_add 3

//the value of N is 3

OUTPUT

Enter the elements of the matrix A

1 2 3

2 3 4

3 4 5

Enter the elements of the matrix B

9 7 6

3 2 1

3 7 8

Elements of the matrix C are

10 9 9

5 5 5

6 11 13

Program 2:

Develop a stack class to hold a maximum of 10 integers with suitable methods. Develop a JAVA main method to illustrate Stack operations.

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

```
//Develop a stack class to hold a maximum of 10 integers with suitable methods. Develop a JAVA main  
method to illustrate Stack operations.
```

```
class Stack{  
    //static int ch;  
    private int arr[];  
    private int top,size;  
    // Creating a stack  
    Stack() {  
        arr = new int[10];  
        top = -1;  
        size= 5;  
    }  
    // push elements to the top of stack  
    public void push(int x) {  
        if (top==size-1) {  
            System.out.println("Stack OverFlow");  
        }  
        else{  
            // insert element on top of stack  
            System.out.println("Inserting " + x);  
            arr[++top] = x;}  
    }  
    // pop elements from top of stack  
    public int pop() {  
        // if stack is empty  
        // no element to pop  
        if (top==-1) {  
            System.out.println("STACK EMPTY");  
            return 0;  
        }  
        else  
        {  
            // pop element from top of stack  
            return arr[top--];  
        }  
    }  
}
```

```

    }
}

// display elements of stack
public void printStack() {
    for (int i = 0; i <= top; i++) {
        System.out.print(arr[i] + ", ");
    }
}

public class Demo_stack
{
    public static void main(String[] args) {
        int ch;
        Stack s1 = new Stack();
        while (true) {
            System.out.println("\nEnter your choice\n1.PUSH\n2.POP\n3.Display \n4..EXIT");
            Scanner integer = new Scanner(System.in);
            ch = integer.nextInt();
            switch (ch) {
                case 1:
                    System.out.println("Enter Element");
                    int element = integer.nextInt();
                    s1.push(element);
                    break;
                case 2:
                    System.out.printf("Popped element is %d", s1.pop());
                    break;
                case 3:
                    s1.printStack();
                    break;
                case 4:
                    System.exit(0);
            }
        }
    }
}

```

default:

```
    System.out.println("Wrong option");  
}  
}  
}  
}
```

OUTPUT

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

11

Inserting 11

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

3

11,

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

22

Inserting 22

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

233

Wrong option

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

44

Inserting 44

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

55

Inserting 55

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

66

Inserting 66

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

2

Poped element is ,66

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

3

11, 22, 44, 55,

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

88

Inserting 88

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

1

Enter Element

88

Inserting 88

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

3

11, 22, 44, 55, 88, 88,

Enter your choice

1.PUSH

2.POP

3.Display

4..EXIT

Program 3:

A class called Employee, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method raiseSalary (percent) increases the salary by the given percentage. Develop the Employee class and suitable main method for demonstration.

```
import java.util.Scanner;
public class Employee {
    private int id;
    private String name;
    private double salary;

    public Employee(int id, String name, double salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
}
```

```

public void raiseSalary(double percent) {
    if (percent > 0 ) {
        double increaseAmount = salary * (percent / 100);
        salary += increaseAmount;
    }
    else
    {
        System.out.println("Invalid percentage. Salary remains unchanged.");
    }
}

public void display_Employee()
{
    System.out.println("Employee ID =" + id + ", Name=" + name + ",     Salary=" + salary );
}

public static void main(String[] args) {
    double p;
    Scanner in= new Scanner(System.in);
    // Create an Employee object
    Employee e1 = new Employee(12,"Sony",50000);
    System.out.println("Employee details before");
    e1.display_Employee();
    System.out.println("enter the percetage to be increased");
    p=in.nextDouble();
    e1.raiseSalary(p);
    // Display the updated details
    System.out.println("Employee Details after Raise:");
    e1.display_Employee();
}
}

```

C:\java3sem>java Employee

OUTPUT

C:\java24>java Employee

Employee details before

Employee ID =12, Name=Sony, Salary=50000.0

enter the percetage to be increased

30

Employee Details after Raise:

Employee ID =12, Name=Sony, Salary=65000.0

Program 4:

A class called MyPoint, which models a 2D point with x and y coordinates, is designed as follows:

- **Two instance variables x (int) and y (int).**
 - **A default (or "no-arg") constructor that construct a point at the default location of (0, 0).**
 - **A overloaded constructor that constructs a point with the given x and y coordinates.**
 - **A method setXY() to set both x and y.**
 - **A method getXY() which returns the x and y in a 2-element int array.**
 - **A toString() method that returns a string description of the instance in the format "(x, y)".**
 - **A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates**
 - **An overloaded distance(MyPoint another) that returns the distance from this point to the given MyPoint instance (called another)**
 - **Another overloaded distance() method that returns the distance from this point to the origin (0,0)**
- Develop the code for the class MyPoint. Also develop a JAVA program (called TestMyPoint) to test all the methods defined in the class.**

```
class MyPoint {  
    private int x;  
    private int y;  
  
    // Default constructor  
    public MyPoint() {
```

```

x = 0;
y = 0;
}

// Overloaded constructor

public MyPoint(int x, int y) {
    this.x = x;
    this.y = y;
}

// Setter method to set both x and y

public void setXY(int x, int y) {
    this.x = x;
    this.y = y;
}

// Getter method to return x and y in a 2-element int array

public int[] getXY() {
    int[] coordinates = {x, y};
    return coordinates;
}

// toString method to return a string description of the point

public String toString() {
    return "(" + x + ", " + y + ")";
}

// Method to calculate distance to another point (x, y)

public double distance(int x, int y) {
    int dx = this.x - x;
    int dy = this.y - y;
    return Math.sqrt(dx * dx + dy * dy);
}

// Overloaded method to calculate distance to another MyPoint instance

public double distance(MyPoint another) {
    int dx = this.x - another.x;

```

```

        int dy = this.y - another.y;
        return Math.sqrt(dx * dx + dy * dy);
    }

    // Method to calculate distance to the origin (0, 0)
    public double distance() {
        return Math.sqrt(x * x + y * y);
    }
}

public class TestMyPoint {
    public static void main(String[] args) {
        // Create two MyPoint instances
        MyPoint point1 = new MyPoint(1, 2);
        MyPoint point2 = new MyPoint(4, 6);

        // Test setXY and getXY methods
        point1.setXY(3, 4);
        int[] coordinates = point1.getXY();
        System.out.println("Coordinates of point1: (" + coordinates[0] + ", " + coordinates[1] + ")");

        // Test toString method
        System.out.println("Point1: " + point1.toString());
        System.out.println("Point2: " + point2.toString());
    }

    // Test distance methods
    System.out.println("Distance between point1 and point2: " + point1.distance(point2));
    System.out.println("Distance from point1 to (0, 0): " + point1.distance());
    System.out.println("Distance from point2 to (0, 0): " + point2.distance(0, 0));
}

```

C:\java3sem>javac TestMyPoint.java

C:\java3sem>java TestMyPoint

OUTPUT

Coordinates of point1: (3, 4)

Point1: (3, 4)

Point2: (4, 6)

Distance between point1 and point2: 2.23606797749979

Distance from point1 to (0, 0): 5.0

Distance from point2 to (0, 0): 7.211102550927978

Program 5

Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts by developing suitable methods, defining member data and main program.

```
// Base class
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
    public void erase() {
        System.out.println("Erasing a shape");
    }
}

// Subclass Circle
class Circle extends Shape {
    public void draw() {
        System.out.println("Drawing a circle");
    }
    public void erase() {
        System.out.println("Erasing a circle");
    }
}
```

```

}

// Subclass Triangle

class Triangle extends Shape {

    public void draw() {
        System.out.println("Drawing a triangle");
    }

    public void erase() {
        System.out.println("Erasing a triangle");
    }
}

// Subclass Square

class Square extends Shape {

    public void draw() {
        System.out.println("Drawing a square");
    }

    public void erase() {
        System.out.println("Erasing a square");
    }
}

public class PolymorphismDemo1 {

    public static void main(String[] args) {
        // Create objects of different shapes
        Shape s1 = new Shape();
        Circle c1 = new Circle();
        Triangle t1 = new Triangle();
        Square sq1 = new Square();

        // Demonstrate polymorphism through method calls
    }
}

```

```
s1.draw();
s1.erase();
c1.draw();
c1.erase();
t1.draw();
t1.erase();
sq1.draw();
sq1.erase();

}

}
```

```
C:\java3sem>javac PolymorphismDemo.java
C:\java3sem>java PolymorphismDemo
```

OUTPUT

Drawing a circle

Erasing a circle

Drawing a triangle

Erasing a triangle

Drawing a square

Erasing a square

Program 6:

Develop a JAVA program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.

```
// Abstract Shape class
abstract class Shape {
    // Abstract method to calculate area
    public abstract void calculateArea();

    // Abstract method to calculate perimeter
    public abstract void calculatePerimeter();
}

// Subclass Circle
class Circle extends Shape {
    private double r;
    // Constructor for Circle
    public Circle(double r) {
        this.r= r;
    }
    public void calculateArea() {
        double a= Math.PI * r * r;
        System.out.println("Area of circle = "+a);
    }
    public void calculatePerimeter() {
        double pc= 2 * Math.PI * r;
        System.out.println("Perimeter of a circle = "+pc);
    }
}
```

```

// Subclass Triangle
class Triangle extends Shape {
    private double a;
    private double b;
    private double c;

    // Constructor for Triangle
    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    public void calculateArea() {
        // Using Heron's formula to calculate the area of a triangle
        double s = (a + b + c) / 2;
        double at= Math.sqrt(s * (s - a) * (s -b) * (s - c));
        System.out.println("Area of triangle = "+at);
    }

    public void calculatePerimeter() {
        double pt= a + b + c;
        System.out.println("Perimeter of a triangle = "+pt);

    }
}

public class ShapeDemo {
    public static void main(String[] args) {
        // Create a Circle and calculate its area and perimeter
        Circle c1 = new Circle(5.0);
        c1.calculateArea();
    }
}

```

```

        c1.calculatePerimeter();

    // Create a Triangle and calculate its area and perimeter
    Triangle t1 = new Triangle(3.0, 4.0, 5.0);
    t1.calculateArea();
    t1.calculatePerimeter();
}

}

C:\java3sem>javac ShapeDemo.java
C:\java3sem>java ShapeDemo

```

OUTPUT

```

C:\java24>java ShapeDemo
Area of circle = 78.53981633974483
Perimeter of a circle = 31.41592653589793
Area of triangle = 6.0
Perimeter of a triangle = 12.0

```

Program 7

Develop a JAVA program to create an interface Resizable with methods resizeWidth(int width) and resizeHeight(int height) that allow an object to be resized. Create a class Rectangle that implements the Resizable interface and implements the resize methods

```

interface Resizable {
    void resizeWidth(int width);
    void resizeHeight(int height);
}

class Rectangle implements Resizable {
    int w;
    int h;

    Rectangle(int width, int height) {
        w = width;
        h = height;
    }
}

```

```

        System.out.println("original rectangle width=" + w + "height=" + h + "\n");
    }

    public void resizeWidth(int width) {
        w = width;
        System.out.println("Resized width to: " + w);
    }

    public void resizeHeight(int height) {
        h = height;
        System.out.println("Resized height to: " + h);
    }
}

class ResizeDemo {

    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle(10, 5);
        rectangle.resizeWidth(15);
        rectangle.resizeHeight(8);
    }
}

C:\java3sem>javac ResizableRectangleDemo.java
C:\java3sem>java ResizableRectangleDemo

```

OUTPUT

Initial Width: 10

Initial Height: 20

Resized Width: 15

Resized Height: 25

Program 8:

Develop a JAVA program to create an outer class with a function display. Create another class inside the outer class named inner with a function called display and call the two functions in the main class.

```
// Outer class
class Outer {
    // Display method in the outer class
    void display() {
        System.out.println("This is the display method in the Outer class.");
    }
}

// Inner class
class Inner {
    // Display method in the inner class
    void display() {
        System.out.println("This is the display method in the Inner class.");
    }
}

public class InnerClassDemo {
    public static void main(String[] args) {
        // Create an instance of the Outer class
        Outer o1 = new Outer();

        // Call the display method of the Outer class
        o1.display();

        // Create an instance of the Inner class
        Outer.ob inner =o1.new Inner();
    }
}
```

```

    // Call the display method of the Inner class
    ob.display();
}
}

```

```

C:\java3sem>javac InnerClassDemo.java
C:\java3sem>java InnerClassDemo

```

OUTPUT

This is the display method in the Outer class.

This is the display method in the Inner class.

Program 9:

Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.

```

import java.util.Scanner;
class validexception extends Exception {
    public validexception (String message) {
        super(message);
    }
}

public class demo_user{
    public static void main(String[] args) {
        int a,b;
        Scanner in = new Scanner(System.in);
        System.out.println("enter the values of a and b");
        a = in.nextInt();
        b = in.nextInt();
    }
}

```

```

try {

    if (b== 0) {
        throw new validexception("Division by zero is not allowed.");
    }

    int result = a / b ;
    System.out.println("Result: " + result);
} catch (validexception e) {
    System.err.println("Custom Exception Caught: " + e.getMessage());
} catch (ArithmeticeException e) {
    System.err.println("Arithmetice Exception Caught: Division by zero.");
} finally {
    System.out.println("Finally block executed.");
}
}

}

C:\java3sem>javac demo_user.java
C:\java3sem>java demo_user

```

OUTPUT

Custom Exception Caught: Division by zero is not allowed.

Finally block executed.

Program 10:

Develop a JAVA program to create a package named mypack and import & implement it in a suitable class.

Follow these steps

1.Create a directory structure to organize your code. You should have a directory named "mypack" to represent the package and a separate directory for your main class.

2. Inside the "mypack" directory, create a Java file with a class that you want to include in the package For example, let's create a class called MyClass in a file named MyClass.java

// File: mypack/MyClass.java

```
package mypack;
public class MyClass {
    public void display() {
        System.out.println("This is a method from the 'mypack' package.");
    }
}
```

Create File: MainClass.java

```
import mypack.MyClass;
public class MainClass {
    public static void main(String[] args) {
        MyClass myObj = new MyClass();
        myObj.display();
    }
}
```

Compilation

javac -d . MyClass.java

javac MainClass.java

Run the program

Java MainClass

Program 11:

Write a program to illustrate creation of threads using runnable class. (start method start each of the newly created thread. Inside the run method there is sleep() for suspend the thread for 500 milliseconds).

```
class MyRunnable implements Runnable {
```

```

private String name;

public MyRunnable(String name) {
    this.name = name;
}

@Override
public void run() {
    try {
        System.out.println("Thread " + name + " is running.");
        // Sleep for 500 milliseconds
        Thread.sleep(500);
        System.out.println("Thread " + name + " has completed.");
    } catch (InterruptedException e) {
        System.out.println("Thread " + name + " was interrupted.");
    }
}

public class RunnableThreadDemo {
    public static void main(String[] args) {
        // Create instances of MyRunnable
        MyRunnable r1 = new MyRunnable("Thread 1");
        MyRunnable r2 = new MyRunnable("Thread 2");

        // Create Thread objects and start them
        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);
        t1.start();
        t2.start();
    }
}

```

```
C:\java3sem>javac RunnableThreadDemo.java
```

```
C:\java3sem>java RunnableThreadDemo
```

OUTPUT

Thread Thread 2 is running.

Thread Thread 1 is running.

Thread Thread 1 has completed.

Thread Thread 2 has completed

Program 12:

Develop a program to create a class MyThread in this class a constructor, call the base class constructor, using super and start the thread. The run method of the class starts after this. It can be observed that both main thread and created child thread are executed concurrently.

```
class MyThread extends Thread {  
    public MyThread(String name) {  
        super(name); // Call the Thread class constructor with the thread name  
        start(); // Start the thread  
    }  
    @Override  
    public void run() {  
        for (int i = 1; i <= 5; i++) {  
            System.out.println("Child Thread: " + getName() + " - Count: " + i);  
            try {  
                Thread.sleep(1000); // Sleep for 1 second  
            } catch (InterruptedException e) {  
                System.out.println("Child Thread Interrupted.");  
            }  
        }  
    }  
}  
  
public class ThreadDemo {
```

```

public static void main(String[] args) {
    // Create an instance of MyThread
    MyThread childThread = new MyThread("ChildThread");

    // Main thread execution
    for (int i = 1; i <= 5; i++) {
        System.out.println("Main Thread - Count: " + i);
        try {
            Thread.sleep(1000); // Sleep for 1 second
        } catch (InterruptedException e) {
            System.out.println("Main Thread Interrupted.");
        }
    }
}

```

C:\java3sem>javac ThreadDemo.java
C:\java3sem>java ThreadDemo

OUTPUT:

Child Thread: ChildThread - Count: 1
Main Thread - Count: 1
Main Thread - Count: 2
Child Thread: ChildThread - Count: 2
Main Thread - Count: 3
Child Thread: ChildThread - Count: 3
Main Thread - Count: 4
Child Thread: ChildThread - Count: 4
Main Thread - Count: 5
Child Thread: ChildThread - Count: 5

VIVA QUESTIONS WITH ANSWERS

1. What is Object Oriented Programming?

Object-Oriented Programming(OOPs) is a type of programming that is based on objects rather than just functions and procedures. Individual objects are grouped into classes. OOPs implements real-world entities like inheritance, polymorphism, hiding, etc into programming. It also allows binding data and code together.

2. What are the main features of OOPs?

- Inheritance
- Encapsulation
- Polymorphism
- Data Abstraction

3. What is an object?

An object is a real-world entity which is the basic unit of OOPs for example chair, cat, dog, etc. Different objects have different states or attributes, and behaviors.

4. What is a class?

A class is a prototype that consists of objects in different states and with different behaviors. It has a number of methods that are common the objects present within that class.

5. What is the difference between a class and a structure?

Class: User-defined blueprint from which objects are created. It consists of methods or set of instructions that are to be performed on the objects.

Structure: A structure is basically a user-defined collection of variables which are of different data types.

6. What is inheritance?

Inheritance is a feature of OOPs which allows classes inherit common properties from other classes. For example, if there is a class such as ‘vehicle’, other classes like ‘car’, ‘bike’, etc can inherit common properties from the vehicle class. This property helps you get rid of redundant code thereby reducing the overall size of the code.

7. What are the different types of inheritance?

- Single inheritance
- Multiple inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Hybrid inheritance

8. What is polymorphism?

Polymorphism refers to the ability to exist in multiple forms. Multiple definitions can be given to a single interface. For example, if you have a class named Vehicle, it can have a method named speed but you cannot define it because different vehicles have different speed. This method will be defined in the subclasses with different definitions for different vehicles.

9. What is method overloading?

Method overloading is a feature of OOPs which makes it possible to give the same name to more than one methods within a class if the arguments passed differ.

10. What is method overriding?

Method overriding is a feature of OOPs by which the child class or the subclass can redefine methods present in the base class or parent class. Here, the method that is overridden has the same name as well as the signature meaning the arguments passed and the return type.

11. What is encapsulation?

Encapsulation refers to binding the data and the code that works on that together in a single unit. For example, a class. Encapsulation also allows data-hiding as the data specified in one class is hidden from other classes.

12. What are ‘access specifiers’?

Access specifiers or access modifiers are keywords that determine the accessibility of methods, classes, etc in OOPs. These access specifiers allow the implementation of encapsulation. The most common access specifiers are public, private and protected. However, there are a few more which are specific to the programming languages.

13. What is the difference between public, private and protected access modifiers?

Name	Accessibility from own class	Accessibility from derived class	Accessibility from world
Public	Yes	Yes	Yes
Private	Yes	No	No
Protected	Yes	Yes	No

14. What is data abstraction?

Data abstraction is a very important feature of OOPs that allows displaying only the important information and hiding the implementation details. For example, while riding a bike, you know that if you

raise the accelerator, the speed will increase, but you don't know how it actually happens. This is data abstraction as the implementation details are hidden from the rider.

15. How to achieve data abstraction?

Data abstraction can be achieved through:

- Abstract class
- Abstract method

16. What is an interface?

It is a concept of OOPs that allows you to declare methods without defining them. Interfaces, unlike classes, are not blueprints because they do not contain detailed instructions or actions to be performed. Any class that implements an interface defines the methods of the interface.

17. What are virtual functions?

Virtual functions are functions that are present in the parent class and are overridden by the subclass. These functions are used to achieve runtime polymorphism.

18. What are pure virtual functions?

Pure virtual functions or abstract functions are functions that are only declared in the base class. This means that they do not contain any definition in the base class and need to be redefined in the subclass.

19. What is a constructor?

A constructor is a special type of method that has the same name as the class and is used to initialize objects of that class.

20. What is a destructor?

A destructor is a method that is automatically invoked when an object is destroyed. The destructor also recovers the heap space that was allocated to the destroyed object, closes the files and database connections of the object, etc.

21. What is an exception?

An exception is a kind of notification that interrupts the normal execution of a program. Exceptions provide a pattern to the error and transfer the error to the exception handler to resolve it. The state of the program is saved as soon as an exception is raised.

22. What is exception handling?

Exception handling in Object-Oriented Programming is a very important concept that is used to manage errors. An exception handler allows errors to be thrown and caught and implements a centralized mechanism to resolve them.

23. What is the difference between an error and an exception?

Error	Exception
Errors are problems that should not be encountered by applications	Conditions that an application might try to catch

24. What is a try/ catch block?

A try/ catch block is used to handle exceptions. The try block defines a set of statements that may lead to an error. The catch block basically catches the exception.

25. What is a finally block?

A finally block consists of code that is used to execute important code such as closing a connection, etc. This block executes when the try block exits. It also makes sure that finally block executes even in case some unexpected exception is encountered.