VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELGAVI-590018, KARNATAKA

Semester-III

OBJECT ORIENTED PROGRAMMING WITH JAVA

LAB MANUAL (BCS306A)

(As per CBCS Scheme 2022)

AcademicYear:2023-2024

Prepared By,
Prof. ATEEQ AHMED
Assistant Professor, Dept. of ISE

HKBK College of Engineering

No.22/1, Opp., Manyata Tech Park Rd, Nagawara,
Bengaluru, Karnataka 560045.
(Approved by AICTE and Affiliated to VTU)

Department of Information Science & Engineering

Object Oriented Programming with JAVA		Semester	3
Course Code	BCS306A	CIE Marks	50
Teaching Hours/Week (L: T:P: S)	2:0:2	SEE Marks	50
Total Hours of Pedagogy	28 Hours of Theory + 20 Hours of Practical	Total Marks	100
Credits	03	Exam Hours	03
Examination type (SEE)	Theory		

Note - Students who have undergone " Basics of Java Programming- BPLCK105C/205C" in first year are not eligible to opt this course

Course objectives:

- To learn primitive constructs JAVA programming language.
- To understand Object Oriented Programming Features of JAVA.
- To gain knowledge on: packages, multithreaded programing and exceptions.

Teaching-Learning Process (General Instructions)

These are sample Strategies, which teachers can use to accelerate the attainment of the various course outcomes and make Teaching –Learning more effective

1. Use Online Java Compiler IDE: https://www.jdoodle.com/online-java-compiler/ or any other.
2. Demonstration of programing examples.
3. Chalk and board, power point presentations
4. Online material (Tutorials) and video lectures.

Module-1

An Overview of Java: Object-Oriented Programming (Two Paradigms, Abstraction, The Three OOP Principles), Using Blocks of Code, Lexical Issues (Whitespace, Identifiers, Literals, Comments, Separators, The Java Keywords).

Data Types, Variables, and Arrays: The Primitive Types (Integers, Floating-Point Types, Characters, Booleans), Variables, Type Conversion and Casting, Automatic Type Promotion in Expressions, Arrays, Introducing Type Inference with Local Variables.

Operators: Arithmetic Operators, Relational Operators, Boolean Logical Operators, The Assignment Operator, The ? Operator, Operator Precedence, Using Parentheses.

Control Statements: Java's Selection Statements (if, The Traditional switch), Iteration Statements (while, do-while, for, The For-Each Version of the for Loop, Local Variable Type Inference in a for Loop, Nested Loops), Jump Statements (Using break, Using continue, return).

Chapter 2, 3, 4, 5

Module-2

Introducing Classes: Class Fundamentals**,** Declaring Objects**,** Assigning Object Reference Variables**,** Introducing Methods**,** Constructors**,** The this Keyword**,** Garbage Collection.

Methods and Classes: Overloading Methods, Objects as Parameters, Argument Passing, Returning Objects, Recursion, Access Control, Understanding static, Introducing final, Introducing Nested and Inner Classes.

Chapter 6, 7

Module-3

Inheritance: Inheritance Basics, Using super, Creating a Multilevel Hierarchy, When Constructors Are Executed, Method Overriding, Dynamic Method Dispatch, Using Abstract Classes, Using final with Inheritance, Local Variable Type Inference and Inheritance, The Object Class.

Interfaces: Interfaces, Default Interface Methods, Use static Methods in an Interface, Private Interface Methods.

Chapter 8, 9

Module-4:

Packages: Packages, Packages and Member Access, Importing Packages.

Exceptions: Exception-Handling Fundamentals, Exception Types, Uncaught Exceptions, Using try and catch, Multiple catch Clauses, Nested try Statements, throw, throws, finally, Java's Built-in Exceptions, Creating Your Own Exception Subclasses, Chained Exceptions.

Chapter 9, 10

Module-5:

Multithreaded Programming: The Java Thread Model, The Main Thread, Creating a Thread, Creating Multiple Threads, Using isAlive() and join(), Thread Priorities, Synchronization, Interthread Communication, Suspending, Resuming, and Stopping Threads, Obtaining a Thread's State.

Enumerations, Type Wrappers and Autoboxing: Enumerations (Enumeration Fundamentals, The values() and valueOf() Methods), Type Wrappers (Character, Boolean, The Numeric Type Wrappers), Autoboxing (Autoboxing and Methods, Autoboxing/Unboxing Occurs in Expressions, Autoboxing/Unboxing Boolean and Character Values).

Chapter 11, 12

Course outcome (Course Skill Set)

At the end of the course, the student will be able to:

1. Demonstrate proficiency in writing simple programs involving branching and looping structures.
2. Design a class involving data members and methods for the given scenario.
3. Apply the concepts of inheritance and interfaces in solving real world problems.
4. Use the concept of packages and exception handling in solving complex problem

Apply concepts of multithreading, autoboxing and enumerations in program development

Programming Experiments (Suggested and are not limited to)

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

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.**
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**
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.**
9. **Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.**
10. **Develop a JAVA program to create a package named mypack and import & implement it in a suitable class.**
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).**
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.**

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.

CIE for the theory component of the IPCC (maximum marks 50)

- **IPCC means practical portion integrated with the theory of the course.**
- **CIE marks for the theory component are 25 marks and that for the practical component is 25 marks.**
- **25 marks for the theory component are split into 15 marks for two Internal Assessment Tests (Two Tests, each of 15 Marks with 01-hour duration, are to be conducted) and 10 marks for other assessment methods mentioned in 22OB4.2. The first test at the end of 40-50% coverage of the syllabus and the second test after covering 85-90% of the syllabus.**
- **Scaled-down marks of the sum of two tests and other assessment methods will be CIE marks for the theory component of IPCC (that is for 25 marks).**
- **The student has to secure 40% of 25 marks to qualify in the CIE of the theory component of IPCC.**

CIE for the practical component of the IPCC

- **15 marks for the conduction of the experiment and preparation of laboratory record, and 10 marks for the test to be conducted after the completion of all the laboratory sessions.**
- **On completion of every experiment/program in the laboratory, the students shall be evaluated including viva-voce and marks shall be awarded on the same day.**
- **The CIE marks awarded in the case of the Practical component shall be based on the continuous evaluation of the laboratory report. Each experiment report can be evaluated for 10 marks. Marks of all experiments' write-ups are added and scaled down to 15 marks.**
- **The laboratory test (duration 02/03 hours) after completion of all the experiments shall be conducted for 50 marks and scaled down to 10 marks.**
- **Scaled-down marks of write-up evaluations and tests added will be CIE marks for the laboratory component of IPCC for 25 marks.**
- **The student has to secure 40% of 25 marks to qualify in the CIE of the practical component of the IPCC.**

SEE for IPCC

Theory SEE will be conducted by University as per the scheduled timetable, with common question papers for the course (duration 03 hours)

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 AddMatrices {

    public static void main(String[] args) {

        if (args.length != 1) {

            System.out.println("Please provide the order of the matrix as a command-line argument.");

            return;

        }

        int N = 0;

        try {

            N = Integer.parseInt(args[0]);

        } catch (NumberFormatException e) {

            System.out.println("Please provide a valid integer for the order of the matrix.");

            return;

        }

        if (N <= 0) {

            System.out.println("Please provide a positive value for the order of the matrix.");

            return;

        }

        int[][] matrixA = new int[N][N];

        int[][] matrixB = new int[N][N];

        int[][] sumMatrix = new int[N][N];

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the elements of the first matrix:");

        enterMatrixElements(matrixA, scanner);

        System.out.println("Enter the elements of the second matrix:");

        enterMatrixElements(matrixB, scanner);

        addMatrices(matrixA, matrixB, sumMatrix, N);
```

```
        System.out.println("The sum of the matrices is:");
        displayMatrix(sumMatrix);
    }
    public static void enterMatrixElements(int[][] matrix, Scanner scanner) {
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                System.out.print("Enter element [" + i + "][" + j + "]: ");
                matrix[i][j] = scanner.nextInt();
            }
        }
    }
    public static void addMatrices(int[][] matrixA, int[][] matrixB, int[][] sumMatrix, int N) {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                sumMatrix[i][j] = matrixA[i][j] + matrixB[i][j];
            }
        }
    }
    public static void displayMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int element : row) {
                System.out.print(element + " ");
            }
            System.out.println();
        }
    }
}
```

2. 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 {
  int stack[] = new int[10];
  int top;

  // Initialize top-of-stack
  STACK() {
    top = -1;
  }

  // Push an item onto the stack
  void push(int item) {
    if(top==9)
      System.out.println("Stack is Overflow.");
    else
      stack[++top] = item;
  }

  // Pop an item from the stack
  int pop() {
    if(top < 0) {
      System.out.println("Stack is Underflow.");
      return 0;
    }
    else
      return stack[top--];
  }

   void displayStack()
{
if(top < 0)
      System.out.println("Stack is  Empty");
else
for(int i=0; i<=top; i++)
System.out.println(stack[i]);

}
}
class TestStack {
  public static void main(String args[]) {
    STACK mystack = new STACK();
Scanner sc =new Scanner(System.in);
 While(true)
{
System.out.println("Enter 1.Push()\n 2.Pop()\n 3.Display()\n 4.Exit()");
 int choice=sc.nextInt();
 switch(choice)
{
    // push some numbers onto the stack
```

```
  case 1:  for(int i=0; i<10; i++) mystack.push(i);
           break;

    // pop those numbers off the stack
   case 2: System.out.println("Stack in mystack:");
    for(int i=0; i<10; i++)
       System.out.println(mystack.pop());
break;
case 3: displaystack();
            break;
case 4:return;
 default: System.out.println("Invalid Choice");


  }
}
```

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.

```
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 raise = salary * (percent / 100);

            salary += raise;

            System.out.println(name + "'s salary raised by " + percent + "%. New salary: " + salary);

        } else {

            System.out.println("Please provide a positive percentage for salary raise.");
```

```
        }
    }
    public void displayInfo() {
        System.out.println("Employee ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
    public static void main(String[] args) {
        // Creating an employee object
        Employee emp = new Employee(1001, "John Doe", 50000);

        // Displaying initial information
        System.out.println("Initial Information:");
        emp.displayInfo();
        // Raising salary by a given percentage
        double raisePercentage = 10; // Example: 10% raise
        emp.raiseSalary(raisePercentage);
        // Displaying updated information after the raise
        System.out.println("\nInformation after salary raise:");
        emp.displayInfo();
    }
}
```

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;
   public MyPoint() {
      this(0, 0); // Default constructor setting coordinates to (0, 0)
   }
   public MyPoint(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public void setXY(int x, int y) {
      this.x = x;
```

```
        this.y = y;
    }
    public int[] getXY() {
        return new int[]{x, y};
    }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public double distance(int x, int y) {
        int xDiff = this.x - x;
        int yDiff = this.y - y;
        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
    }
      public double distance(MyPoint another) {
        return distance(another.x, another.y);
    }
    public double distance() {
        return distance(0, 0);
    }
}

public class TestMyPoint {
    public static void main(String[] args) {
        MyPoint point1 = new MyPoint();
        MyPoint point2 = new MyPoint(3, 4);
        // Testing setXY() method
        point1.setXY(5, 6);
        // Testing getXY() method
        int[] coordinates = point1.getXY();
        System.out.println("Point 1 coordinates: (" + coordinates[0] + ", " + coordinates[1] + ")");
        // Testing toString() method
        System.out.println("Point 2: " + point2);
        // Testing distance() methods
        System.out.println("Distance between Point 1 and Point 2: " + point1.distance(point2));
        System.out.println("Distance from Point 1 to origin: " + point1.distance());
    }
}
```

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.

```
// Shape superclass
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }
    public void erase() {
        System.out.println("Erasing a shape");
    }
}
// Circle subclass
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
    @Override
    public void erase() {
        System.out.println("Erasing a circle");
    }
}
// Triangle subclass
class Triangle extends Shape {
```

```
    @Override
    public void draw() {
        System.out.println("Drawing a triangle");
    }
    @Override
    public void erase() {
        System.out.println("Erasing a triangle");
    }
}
// Square subclass
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }
    @Override
    public void erase() {
        System.out.println("Erasing a square");
    }
}

public class Main {
    public static void main(String[] args) {
        // Creating instances of different shapes
        Shape circle = new Circle();
        Shape triangle = new Triangle();
        Shape square = new Square();
        // Demonstrating polymorphism by calling draw and erase methods
        circle.draw();
        circle.erase();
        triangle.draw();
        triangle.erase();
        square.draw();
        square.erase();
    }
}
```

6.Develop a JAVA program to create an abstract class Shape with abstract methods calculate Area() and calculate Perimeter(). 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 class Shape {
    // Abstract methods to be implemented by subclasses
    public abstract double calculateArea();
    public abstract double calculatePerimeter();
}
class Circle extends Shape {
    private double radius;
    // Constructor for Circle
    public Circle(double radius) {
        this.radius = radius;
    }
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
```

```
    }
}
class Triangle extends Shape {
    private double side1;
    private double side2;
    private double side3;
    // Constructor for Triangle
    public Triangle(double side1, double side2, double side3) {
        this.side1 = side1;
        this.side2 = side2;
        this.side3 = side3;
    }
    @Override
    public double calculateArea() {
        // Using Heron's formula to calculate area of a triangle
        double s = (side1 + side2 + side3) / 2;
        return Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
    }
    @Override
    public double calculatePerimeter() {
        return side1 + side2 + side3;
    }
}
public class Main {
    public static void main(String[] args) {
        // Creating instances of Circle and Triangle
        Circle circle = new Circle(5);
        Triangle triangle = new Triangle(3, 4, 5);
        // Calculating and displaying area and perimeter for Circle
```

```
        System.out.println("Circle - Area: " + circle.calculateArea());
        System.out.println("Circle - Perimeter: " + circle.calculatePerimeter());
        // Calculating and displaying area and perimeter for Triangle
        System.out.println("Triangle - Area: " + triangle.calculateArea());
        System.out.println("Triangle - Perimeter: " + triangle.calculatePerimeter());
    }
}
```

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

```
// Resizable interface
interface Resizable {
    void resizeWidth(int width);
    void resizeHeight(int height);
}
// Rectangle class implementing Resizable interface
class Rectangle implements Resizable {
    private int width;
    private int height;
    // Constructor for Rectangle
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    // Implementing resizeWidth method from Resizable interface
    @Override
    public void resizeWidth(int width) {
```

```
        this.width = width;
    }
    // Implementing resizeHeight method from Resizable interface
    @Override
    public void resizeHeight(int height) {
        this.height = height;
    }



    // Method to display current width and height of the rectangle
    public void displaySize() {
        System.out.println("Rectangle Width: " + width);
        System.out.println("Rectangle Height: " + height);
    }
}
public class Main {
    public static void main(String[] args) {
        // Creating an instance of Rectangle
        Rectangle rectangle = new Rectangle(10, 20);
        // Displaying initial size of the rectangle
        System.out.println("Initial Size:");
        rectangle.displaySize();
        // Resizing width and height of the rectangle
        rectangle.resizeWidth(15);
        rectangle.resizeHeight(25);
        // Displaying resized size of the rectangle
        System.out.println("\nResized Size:");
        rectangle.displaySize();
    }
```

```
}
```

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.

```
class Outer {
    void display() {
        System.out.println("This is the display() method of the outer class.");
    }
    class Inner {
        void display() {
            System.out.println("This is the display() method of the inner class.");
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Outer outer = new Outer();
        // Calling the display() method of the outer class
        outer.display();
        // Creating an instance of the inner class and calling its display() method
        Outer.Inner inner = outer.new Inner();
        inner.display();
```

```
    }
}
```

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

```
// Custom exception class for DivisionByZero
class DivisionByZeroException extends Exception {
    public DivisionByZeroException(String message) {
        super(message);
    }
}
public class Main {
    public static void main(String[] args) {
        try {
            int numerator = 10;
            int denominator = 0;

            // Perform division and throw exception if denominator is zero
            if (denominator == 0) {m
                throw new DivisionByZeroException("Division by zero error!");
            }
            int result = numerator / denominator;
            System.out.println("Result of division: " + result);
        } catch (DivisionByZeroException e) {
```

```
            System.out.println("Caught DivisionByZeroException: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Caught ArithmeticException: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

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

class.

1. 1. **Creating the package:**

Create a directory named **mypack** and within that directory, create a Java file named **MyPackageClass.java** with the following content:

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

2. 2. **Using the package in another Java class:**

Create a Java class (let's name it **MainClass.java**) in a separate directory (not inside **mypack**) and import and use the **MyPackageClass** from the **mypack** package.

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

}

3. **Directory structure:**
   - RootDirectory
   - mypack
   - MyPackageClass.java
   - MainClass.java

4. **Compilation and Execution:**

Compile both **MyPackageClass.java** and **MainClass.java** files:

```
javac mypack/MyPackageClass.java
javac MainClass.java
```

5. **Run the MainClass using:**

```
java MainClass
```

This will execute the **MainClass** and import the **MyPackageClass** from the **mypack** package. The **display()** method from **MyPackageClass** will be invoked, printing the specified message.

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 threadName;
    public MyRunnable(String threadName) {
    this.threadName = threadName;
    }
    @Override
    public void run() {
        System.out.println("Thread " + threadName + " is running.");
        try {
            // Suspend the thread for 500 milliseconds
            Thread.sleep(500);
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + " interrupted.");
        }
        System.out.println("Thread " + threadName + " is finished.");
    }
}
public class Main {
    public static void main(String[] args) {
```

```
        System.out.println("Creating threads using Runnable interface...");
        // Creating threads using Runnable interface
        Thread thread1 = new Thread(new MyRunnable("Thread 1"));
        Thread thread2 = new Thread(new MyRunnable("Thread 2"));
        Thread thread3 = new Thread(new MyRunnable("Thread 3"));
        // Starting threads using start() method
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

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 threadName) {
        super(threadName); // Calling base class (Thread) constructor
        start(); // Starting the thread immediately after initialization
    }
    public void run() {
        System.out.println("Inside run method of thread: " + Thread.currentThread().getName());
        try {
            Thread.sleep(1000); // Simulating some task for the thread
        } catch (InterruptedException e) {
            System.out.println("Thread " + Thread.currentThread().getName() + " interrupted.");
        }
        System.out.println("Thread " + Thread.currentThread().getName() + " is finished.");
    }
}
public class Main {
    public static void main(String[] args) {
        System.out.println("Main thread is running.");
        // Creating an instance of MyThread and observing concurrent execution
        MyThread myThread = new MyThread("Child Thread");
```

```
        // Continuing execution in the main thread
        for (int i = 0; i < 5; i++) {
            System.out.println("Inside main thread: " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("Main thread interrupted.");
            }
        }
        System.out.println("Main thread is finished.");
    }
}
```