RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Estd : 2001

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

OBJECT ORIENTED PROGRAMMING USING JAVA LABORATORY MANUAL

For Third Semester B.E - 2024 Batch [Autonomous, 2024 syllabus]

Subject Code – BCS306A

NAME ....

USN ....

SECTION ....
BATCH ....

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

VISION AND MISSION OF CSE DEPARTMENT

Vision

Preparing better computer professionals for a real world

Mission

The Department of Computer Science and Engineering will make every effort to promote an intellectual and an ethical environment in which the strengths and skills of Computer Professionals will flourish by

1. Imparting Solid foundations and Applied aspects in both Computer Science Theory and Programming practices
2. Providing Training and encouraging R&D and Consultancy Services in frontier areas of Computer Science with a Global outlook
3. Fostering the highest ideals of Ethics, Values and creating Awareness on the role of Computing in Global Environment
4. Educating and preparing the graduates, highly Sought-after, Productive, and Well-respected for their work culture
5. Supporting and inducing Lifelong Learning practice

PREFACE

We have developed this comprehensive laboratory manual on **Object Oriented Programming using Java** with the primary objectives: To make the students comfortable with the concepts of object-oriented programming and the Java programming language. The manual will help them to understand various principles of OOP such as classes, objects, inheritance, polymorphism, abstraction, and interfaces through hands-on experiments and case studies. Through this course, students will gain exposure to designing and implementing robust, reusable, and efficient programs using Java.

Our profound and sincere efforts will be fruitful only when students acquire extensive knowledge by reading this manual and apply the concepts learned beyond the requirements specified in **Object Oriented Programming using Java Laboratory** as prescribed by **VTU, Belagavi**.

Department of CSE

ACKNOWLEDGMENT

A material of this scope would not have been possible without the contribution of many people. We express our sincere gratitude to Mr. Satish R Shetty, Chairman and Mr. Karan R Shetty, CEO, RNS Group of Companies for providing magnanimous support in all our endeavors.

We are grateful to Dr. M K Venkatesha, Director and Dr. Ramesh Babu H S, Principal, Dr. Kavitha C, HOD, Dean, CSE, RNSIT for extending their constant encouragement and support.

Our heartfelt thanks to Dr. Sudhamani M.J, Dr. Hemanth H S, Prof. Pushpa G, Prof. Chetan Ghatage for their unparalleled contribution throughout the preparation of this comprehensive manual. We also acknowledge our colleagues for their timely suggestions and unconditional support.

Department of CSE

OBJECT ORIENTED PROGRAMMING USING JAVA LABORATORY MANUAL INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 25)			
TEST A	REGULAR EVALUATION B	RECORD C	TOTAL MARKS A+B+C

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No	Parameters	Good	Average	Needs improvement
a.	**Understanding of problem (3 marks)**	Clear understanding of problem statement while designing and implementing the program (3)	Problem statement is understood clearly but few mistakes while designing and implementing program (2)	Problem statement is not clearly understood while designing the program (1)
b.	**Writing program (4 marks)**	Program handles all possible conditions (4)	Average condition is defined and verified. (3)	Program does not handle possible conditions (1)
c.	**Result and documentation (3 marks)**	Meticulous documentation and all conditions are taken care (3)	Acceptable documentation shown (2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)				
Sl. No.	Parameter	Excellent	Good	Average
a.	**Conceptual understanding (10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	Parameters	Excellent	Good	Needs Improvement
a.	**Design, implementation and demonstration (5 marks)**	Program follows syntax and semantics of ARM7 assembly instructions and Embedded C programming. Demonstrates the complete knowledge of the program written (5)	Program has few logical errors, moderately demonstrates all possible concepts implemented in programs (3)	Syntax and semantics of programming is not clear (1)
b.	**Result and documentation (5 marks)**	All expected results are demonstrated successfully, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the program and Partial documentation (3)	Expected results are not demonstrated properly, unable to debug the errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 20)				
Sl. No.	Parameter	Excellent	Good	Average
a.	**Documentation (10 marks)**	Meticulous record writing including program, comments and expected output as per the guidelines mentioned (20)	Write up contains program and expected output, but comments are not included (18)	Write up contains only program (15)

DELIVERY PLAN WITH DETAILS

Week No.	**Topic/ Programs**	**Planned Week of Execution**	**COs covered**	**Remarks**
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 raises Salary (percent) increases the salary by the given percentage. Develop the Employee class and suitable main method for demonstration			
4	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.			
5	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.			
6	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			
7	**Internal-1**			
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	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).			
11	Implement a JAVA program to illustrate the use of different types of character extraction, string comparison, string search and string modification methods.			
12	Develop a Java application that checks whether a given string is a palindrome,designed using the SOLID principles of object-oriented programming.			
13	Develop a Java application that compresses characters in a given string by counting consecutive repeated characters, designed according to the SOLID principles of object-oriented programming. Input: AABBBCCCC Output: A3B3C4			
14	**Internal-2**			

SYALLABUS

SEMESTER – III OBJECT ORIENTED PROGRAMMING USING JAVA

LABORATORY MANUAL

(Effective from the academic year 2024-2025)

Course Code – BCS306A	**CIE Marks - 50**
Number of Contact Hours/Week -2:0:2:0	**SEE Marks - 50**
Total Number of Lab Contact Hours - 20	**Exam Hours - 03**

COURSE LEARNING OBJECTIVES**:** This course will enable students to
1. Understand the fundamentals of object-oriented programming concepts and features of Java.
2. Familiarize with classes, objects, methods, constructors, and arrays in Java.
3. Develop programs using the principles of inheritance, polymorphism, abstraction, and interfaces.
4. Understand and implement concepts of exception handling, multithreading, and file handling in Java.
5. Design and develop Java applications by applying OOP principles to solve real-world problems.

Programs List:

Sl. No	**Experiments** **IDE Used: Java version 23.0.2(JDK Version)**
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.	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.
5.	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.
6.	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.
7.	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.

8.	Develop a JAVA program to raise a custom exception (user defined exception) for Division By Zero using try, catch, throw and finally.
9.	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).
10.	Implement a JAVA program to illustrate the use of different types of character extraction, string comparison, string search and string modification methods.
11.	Develop a Java application that checks whether a given string is a palindrome, designed using the SOLID principles of object-oriented programming
12.	Develop a Java application that compresses characters in a given string by counting consecutive repeated characters, designed according to the SOLID principles of object-oriented programming. Input: AAABBBCCCC Output: A3B3C4

COs	COURSE OUTCOMES
BCS306A.1	Interpret Java classes with appropriate data members and methods to address specified scenarios.
BCS306A.2	Apply the concept of inheritance to model and solve real-world programming problems.
BCS306A.3	Demonstrate the effective use of multi-threading and exception handling to manage concurrency and runtime issues in Java programs.
BCS306A.4	Demonstrate the use of string handling methods to manipulate and process textual data in Java.
BCS306A.5	Develop Java programs integrating object-oriented concepts to address computational problems effectively.

Laboratory Outcomes: The student should be able to:

- Develop, compile, and execute programs using **Java Development Kit (JDK)**.
- Apply the principles of **object-oriented programming** to design and implement Java programs.
- Implement and test **classes, objects, constructors, arrays, inheritance, polymorphism, abstraction, and interfaces**.
- Develop programs using **exception handling, multithreading, and file handling** in Java.
- Design, test, and debug **Java applications** to solve real-world computational problems.

CIE for the practical component

CIE Practical (12 marks)

- What it covers: Weekly lab experiments and submitting lab records
- How it works: You get marks throughout the semester for:
  Doing experiments every week
  Submitting your lab record books
- Total possible: 12 marks

CIE Practical Test (8 marks)

- What it covers: One practical tests during the semester
- How it works:
  Each test is conducted for 100 marks (full marks)

You take 2 such tests covering all experiments
Average of both tests is calculated
This average is then scaled down to 8 marks

Total CIE Practical (20 marks)

Final calculation: 12 + 8 = 20 marks

- **What gets recorded: Your experiments, lab records, and test performance combined**
- **Maximum possible: 20 marks total**

CO-PO MATRIX

COURSE OUTCOMES	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	3	2	2	1	3	1	2	2	2	1	3	1	3	2	2	1
CO2	3	3	3	2	2	2	2	3	2	2	3	1	3	3	2	2
CO3	2	3	3	2	3	2	3	3	3	2	3	2	2	3	3	2
CO4	2	2	2	2	3	1	2	2	3	1	3	1	3	2	2	1
CO5	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3

TABLE OF CONTENTS

SL. NO.	CONTENTS	PAGE NO.
1	**INTRODUCTION**	
2	**A SIMPLE GUIDE ON Java version 23.0.2(JDK Version)**	
3	**SAMPLE PROGRAMS**	
4	**LABORATORY PROGRAMS**	
5	Develop a JAVA program to add TWO matrices of suitable order N (The value of N should be read from command line arguments).	
6	Develop a stack class to hold a maximum of 10 integers with suitable methods.Develop a JAVA main method to illustrate Stack operations.	
7	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	
8	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.	
9	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.	
10	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	
11	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.	
12	Develop a JAVA program to raise a custom exception (user defined exception) for DivisionByZero using try, catch, throw and finally.	
13	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).	
14	Implement a JAVA program to illustrate the use of different types of character extraction, string comparison, string search and string modification methods.	
15	Develop a Java application that checks whether a given string is a palindrome,designed using the SOLID principles of object-oriented programming.	
16	Develop a Java application that compresses characters in a given string by counting consecutive repeated characters, designed according to the SOLID principles of object-oriented programming. Input: AAABBBCCCC Output:	

	A3B3C4	
17	**VIVA QUESTIONS**	

Oops with JAVA LABORATORY INTRODUCTION

The **Object Oriented Programming using Java Laboratory** is designed for 3rd semester students to gain practical exposure to programming using the **Java programming language**. Java is one of the most widely used programming languages due to its platform independence, simplicity, security, and robust object-oriented features.

This laboratory course provides students with hands-on experience in implementing **fundamental programming constructs** such as data types, control structures, arrays, strings, and functions. Gradually, students are introduced to **object-oriented concepts** like classes, objects, inheritance, polymorphism, abstraction, and interfaces. The course also covers advanced features such as **exception handling, multithreading, collections, and file handling**, which are essential for building efficient and real-world applications.

Through this lab, students will:

- Learn to design, code, compile, debug, and execute Java programs.
- Apply object-oriented principles to develop modular and reusable software.
- Enhance problem-solving skills by implementing case studies and mini projects in Java.

This laboratory thus bridges theoretical concepts of **Object Oriented Programming** with practical implementation, preparing students to develop **robust applications** and strengthening their foundation for advanced courses in computer science and software development.

A SIMPLE GUIDE ON Java version 23.0.2(JDK Version)

1. What is JDK?

   JDK (Java Development Kit) is the software package required to develop, compile, and run Java programs.

   It includes:

   1.JRE (Java Runtime Environment) → To run Java applications.

   2.Compiler (javac) → To convert .java files into .class bytecode.

   3.JVM (Java Virtual Machine) → To execute the compiled bytecode.

---

2. Why JDK 23.0.2?

1. JDK 23 is the latest release with new features, performance improvements, and security updates.

2.Backward compatible → All programs written for older versions (Java 8/11/17) will still work.

3. Supports modern tools and libraries for industry-ready development.

---

3. Installing JDK 23.0.2

1.Download from Oracle Java Downloads or OpenJDK builds.

2.Install and set the PATH and JAVA_HOME environment variables:

- Add bin folder path (e.g., C:\Program Files\Java\jdk-23\bin) to PATH.
  - Set JAVA_HOME to C:\Program Files\Java\jdk-23.

3.Verify installation:

```
java -version
javac -version
```

Output should show:

```
java version "23.0.2"
javac 23.0.2
```

---

4. Using JDK in OOP Lab

1.Write code in any text editor (Notepad/VS Code/IntelliJ).

2.Save with .java extension (e.g., HelloWorld.java).

3.Compile the program:

```
javac HelloWorld.java
```

4.Run the program:

```
java HelloWorld
```

---

5. Example Program

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Welcome to OOP Lab with JDK 23.0.2!");
    }
}
```

Output:

```
Welcome to OOP Lab with JDK 23.0.2!
```

JAVA PROGRAMMING

Java Programming Language

Java is a **high-level, class-based, object-oriented programming language** designed to have as few implementation dependencies as possible. It follows the principle of *"Write Once, Run Anywhere (WORA)"*, meaning Java programs can run on any machine that has the Java Virtual Machine (JVM). Java is widely used in academic, research, and industry applications due to its simplicity, portability, and robustness. Programs written in Java are compiled into **bytecode** by the **Java compiler (javac)**, which is then interpreted or executed by the JVM.

The Reasons to write computer programs in Java:

1. **Platform Independence** – Programs run on multiple operating systems without modification.
2. **Object-Oriented Approach** – Easier to model real-world problems.
3. **Robust and Secure** – Built-in error handling and memory management.
4. **Rich Standard Library** – Provides APIs for data structures, networking, GUI, etc.
5. **Industry-Relevance** – Java is widely used in web, mobile, and enterprise applications.

Important Java Features in Lab

- **Encapsulation** → Using classes and access specifiers (public, private, protected).
- **Inheritance** → Reusing code by extending classes using extends.
- **Polymorphism** → Method overloading & overriding.
- **Abstraction** → Hiding implementation details using abstract classes & interface.
- **Exception Handling** → Using try-catch-finally.

Sample programs

Program 1: Hello World

Aim: Write a Java program to print "Hello, World!"

Program:

```
class Hello {
    public static void main(String[] args) {
     System.out.println("Hello, World!");
    }
}
```

Command to Execute (JDK):

```
javac Hello.java
java Hello
```

Output:

```
Hello, World!
```

Program 2: Sum of Two Numbers

Aim: Write a Java program to add two numbers.

Program:

```
class Sum {
    public static void main(String[] args) {
        int a = 10, b = 20;
        int sum = a + b;
        System.out.println("Sum = " + sum);
    }
}
```

Command to Execute:

```
javac Sum.java
java Sum
```

Output:

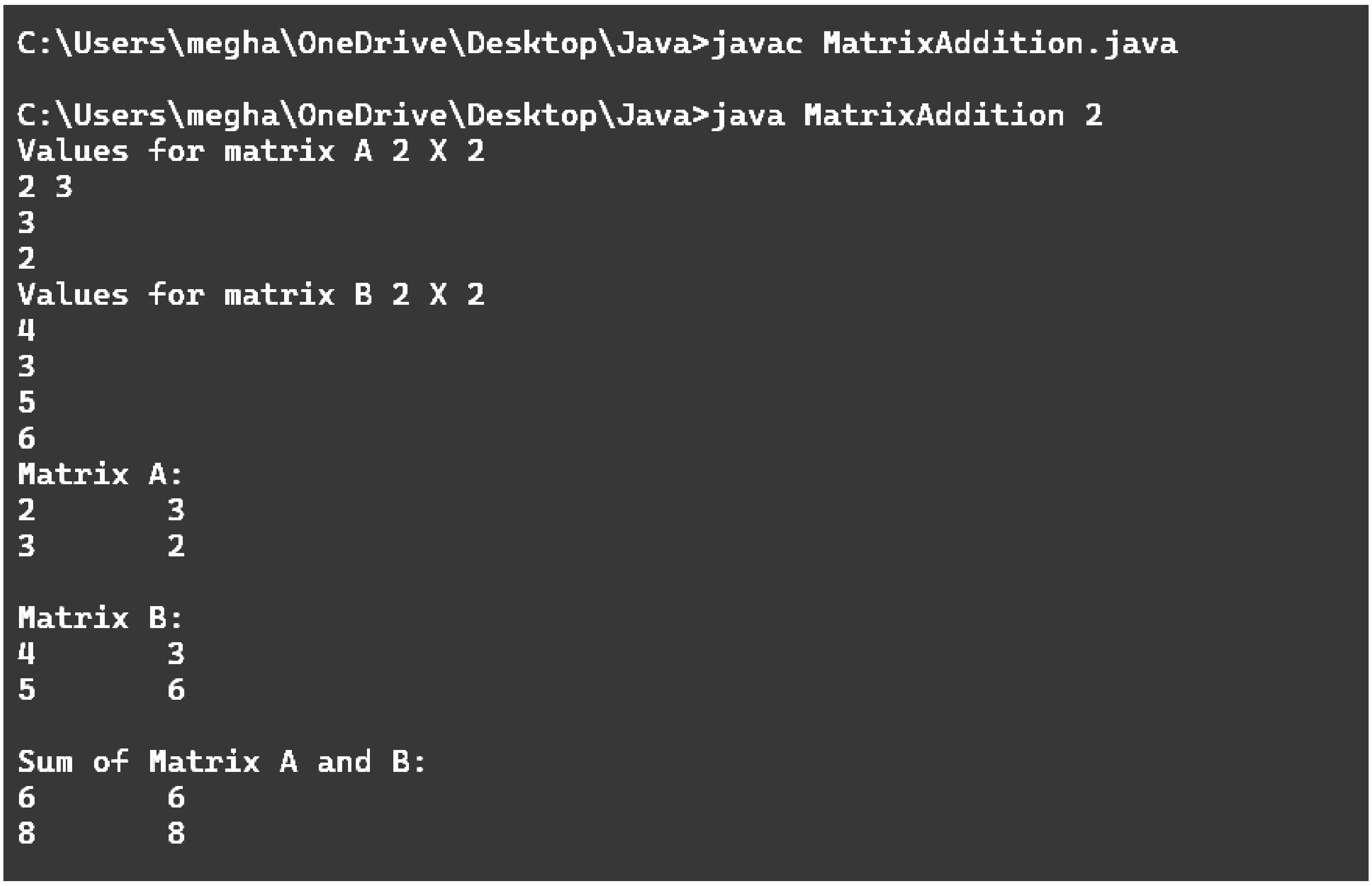
```
Sum = 30
```

LABORATORY PROGRAMS

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.*;
 public class MatrixAddition
 {
  public static void main(String[] args) {
    if (args.length < 1) {
       System.out.println("Please provide the matrix order N as a
                       command-line argument.");
      return;
   }

int N = Integer.parseInt(args[0]);
int i,j;
int[][] A = new int[N][N];
int[][] B = new int[N][N];
int[][] R = new int[N][N];

Scanner sc = new Scanner(System.in);

System.out.println("Values for matrix A "+N+" X "+N);
for(i=0;i<N;i++)
   for(j=0;j<N;j++)
      A[i][j] = sc.nextInt();

System.out.println("Values for matrix B "+N+" X "+N);
for(i=0;i<N;i++)
   for(j=0;j<N;j++)
      B[i][j] = sc.nextInt();

for(i=0; i<N;i++)
  for(j=0 ;j<N;j++)
      R[i][j] = A[i][j] + B[i][j];

System.out.println("Matrix A:");
printMatrix(A);
System.out.println("\nMatrix B:");
printMatrix(B);
System.out.println("\nSum of Matrix A and B:");
printMatrix(R);
}
public static void printMatrix(int[][] matrix)
        {
            for (int[] row : matrix)
            {
                for (int val : row)
                    System.out.print(val + "\t");
                System.out.println();
            }
        }
    }
```

OUTPUT:

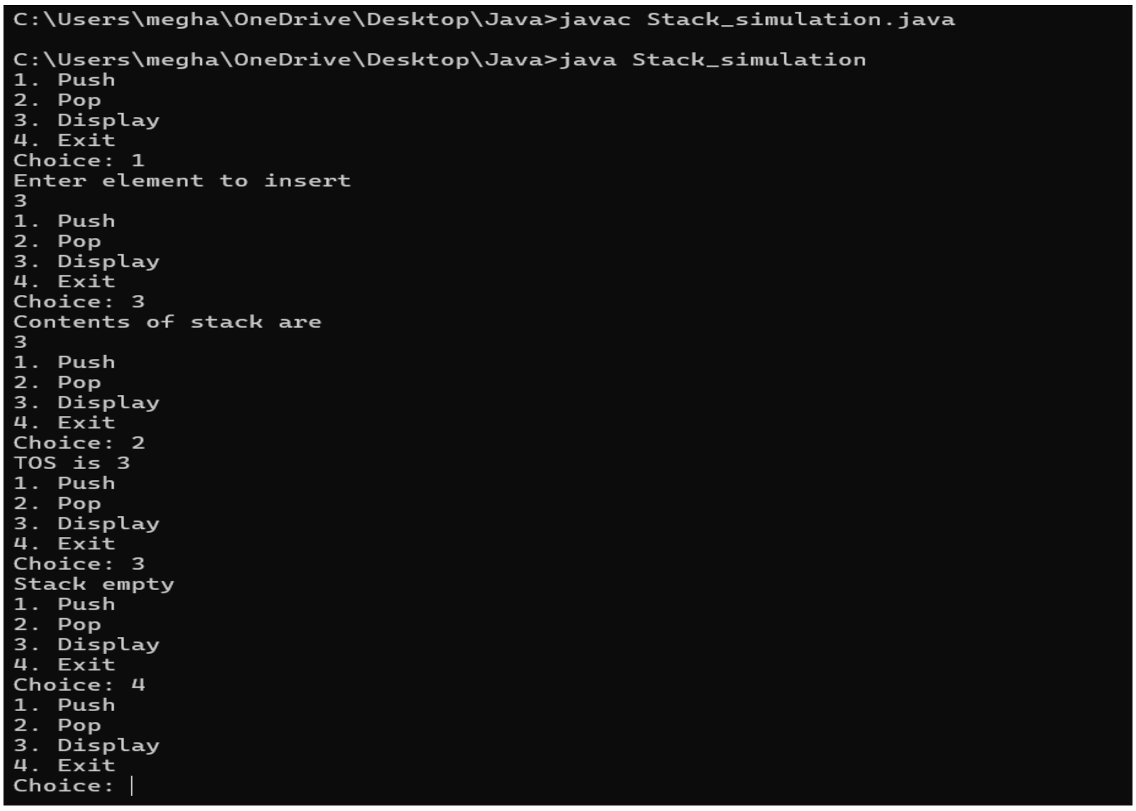


```
C:\Users\megha\OneDrive\Desktop\Java>javac MatrixAddition.java

C:\Users\megha\OneDrive\Desktop\Java>java MatrixAddition 2
Values for matrix A 2 X 2
2 3
3
2
Values for matrix B 2 X 2
4
3
5
6
Matrix A:
2       3
3       2

Matrix B:
4       3
5       6

Sum of Matrix A and B:
6       6
8       8
```

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.*;
  class Stack {
            private static final int MAX_SIZE = 3;
            private int[] stackArray;
            private int top;
            public Stack()
            {
              stackArray = new int[MAX_SIZE];
              top = -1;
            }

    public void push(int value)
    {
     if (top==(MAX_SIZE-1))
     {
       System.out.println("Stack full");
       return;
     }
     stackArray[++top] = value;
    }

    public void pop()
    {
     if (top==-1)
     {
       System.out.println("Stack empty");
       return;
     }
     System.out.println("TOS is "+ stackArray[top--]);
    }

    public void display()
    {
     if (top==-1)
     {
       System.out.println("Stack empty");
       return;
     }
     System.out.println("Contents of stack are");
     for(int i=top;i>=0;i--)
           System.out.println(stackArray[i]);
    }
  }
  public class Stack_simulation {
    public static void main(String[] args) {
      Stack st = new Stack();
      Scanner sc = new Scanner(System.in);
      for(;;)
      {
       System.out.println("1. Push\n2. Pop\n3. Display");
       System.out.print("4. Exit\nChoice: ");
       int ch=sc.nextInt();
       switch(ch)
       {
         case 1: System.out.println("Enter element to
                                  insert");
              ch=sc.nextInt();
                st.push(ch);
                break;
         case 2: st.pop(); break;
         case 3: st.display(); break;
       }
      }
    }
  }
```

OUTPUT:

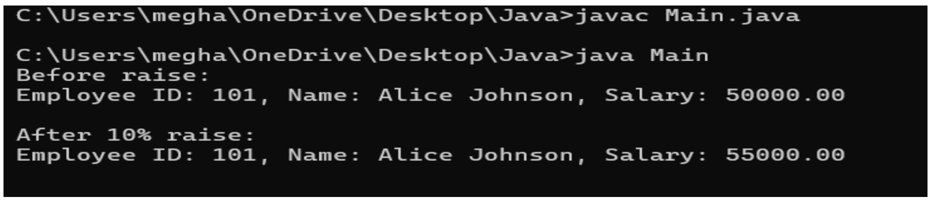


```
C:\Users\megha\OneDrive\Desktop\Java>javac Stack_simulation.java

C:\Users\megha\OneDrive\Desktop\Java>java Stack_simulation
1. Push
2. Pop
3. Display
4. Exit
Choice: 1
Enter element to insert
3
1. Push
2. Pop
3. Display
4. Exit
Choice: 3
Contents of stack are
3
1. Push
2. Pop
3. Display
4. Exit
Choice: 2
TOS is 3
1. Push
2. Pop
3. Display
4. Exit
Choice: 3
Stack empty
1. Push
2. Pop
3. Display
4. Exit
Choice: 4
1. Push
2. Pop
3. Display
4. Exit
Choice: |
```

3. 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.

```java
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)
       salary += salary * percent / 100;
  }

  @Override
  public String toString()
  {
   return "Employee ID: "+id+", Name: "+name+", "
                 +"Salary: "+String.format("%.2f", salary)+"\n";
  }
}
public class Main {
  public static void main(String[] args) {
   Employee emp = new Employee(101, "Alice Johnson", 50000);
   System.out.println("Before raise: \n" + emp);
   emp.raiseSalary(10);
   System.out.println("After 10% raise: \n" + emp);
  }
}
```

OUTPUT:



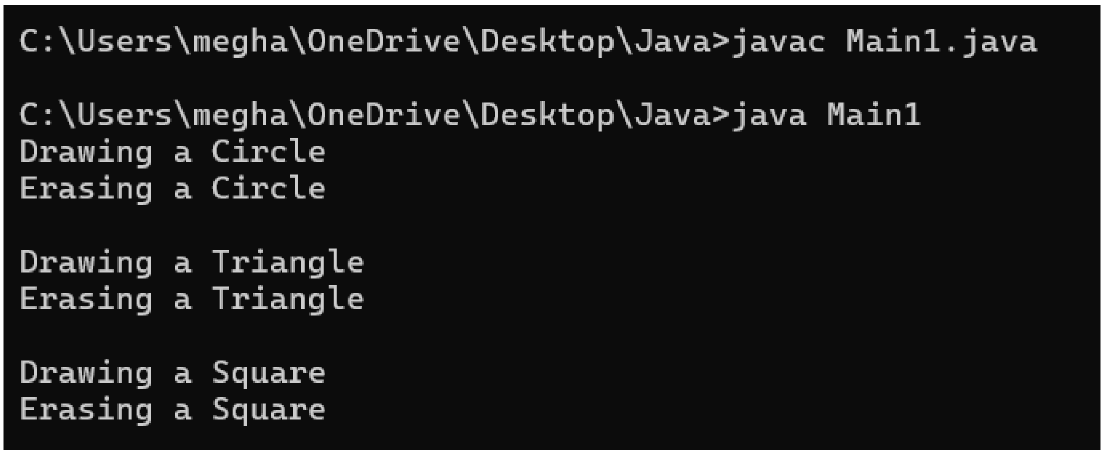
```
C:\Users\megha\OneDrive\Desktop\Java>javac Main.java

C:\Users\megha\OneDrive\Desktop\Java>java Main
Before raise:
Employee ID: 101, Name: Alice Johnson, Salary: 50000.00

After 10% raise:
Employee ID: 101, Name: Alice Johnson, Salary: 55000.00
```

4. 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.

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

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

class Circle extends Shape {
  @Override
  public void draw()
  { System.out.println("Drawing a Circle"); }

  @Override
  public void erase()
  { System.out.println("Erasing a Circle"); }
}


class Triangle extends Shape {
  @Override
  public void draw()
  {  System.out.println("Drawing a Triangle"); }

  @Override
  public void erase()
  {  System.out.println("Erasing a Triangle"); }
}


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)
```

```
    {
      Shape shape;
      shape = new Circle();
      shape.draw();    // Calls Circle's draw
      shape.erase();   // Calls Circle's erase
      System.out.println();
      shape = new Triangle();
      shape.draw();    // Calls Triangle's draw
      shape.erase();   // Calls Triangle's erase
      System.out.println();
      shape = new Square();
      shape.draw();    // Calls Square's draw
      shape.erase();   // Calls Square's erase
    }
  }
```

OUTPUT:

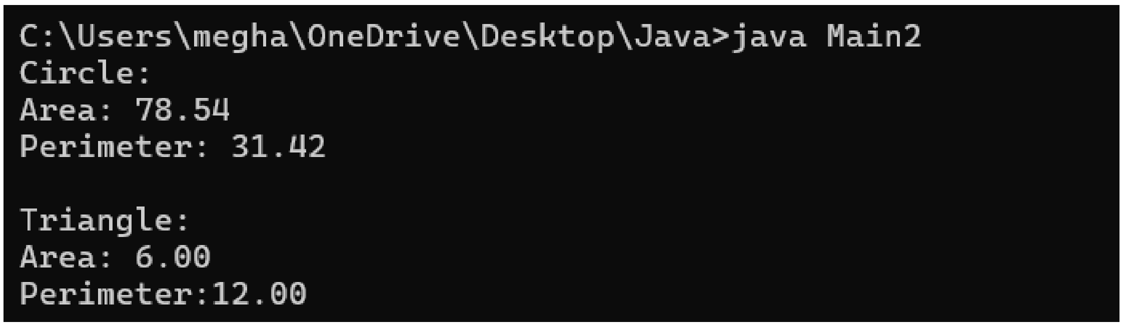


```
C:\Users\megha\OneDrive\Desktop\Java>javac Main1.java

C:\Users\megha\OneDrive\Desktop\Java>java Main1
Drawing a Circle
Erasing a Circle

Drawing a Triangle
Erasing a Triangle

Drawing a Square
Erasing a Square
```

5. 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 class Shape
{
  abstract double calculateArea();
  abstract double calculatePerimeter();
}

class Circle extends Shape
{
  private double radius;

  public Circle(double radius)
  { this.radius = radius; }
  @Override
  double calculateArea()
  { return Math.PI * radius * radius;  }

 @Override
  double calculatePerimeter()
  { return 2 * Math.PI * radius; }
}

class Triangle extends Shape
{
  private double sideA, sideB, sideC;

  public Triangle(double sideA, double sideB, double sideC)
  {this.sideA=sideA; this.sideB=sideB; this.sideC=sideC; }

  @Override
  double calculateArea()
  {  //Heron's Formula - Since all 3 sides are knwon
       double s = (sideA + sideB + sideC) / 2;
     return Math.sqrt(s*(s-sideA)*(s-sideB)*(s-sideC));
  }
  @Override
  double calculatePerimeter()
  {  return sideA + sideB + sideC; }
}

class Main
{
  public static void main(String[] args) {
    Shape circle = new Circle(5);
    System.out.println("Circle:");
    System.out.printf("Area:%.2f\n",circle.calculateArea());
    System.out.printf("Perimeter: %.2f\n",
                             circle.calculatePerimeter());
    System.out.println();

    Shape triangle = new Triangle(3, 4, 5);
    System.out.println("Triangle:");
    System.out.printf("Area: %.2f\n",
                               triangle.calculateArea());
    System.out.printf("Perimeter:%.2f\n",
                           triangle.calculatePerimeter());
  }
}
```

OUTPUT:



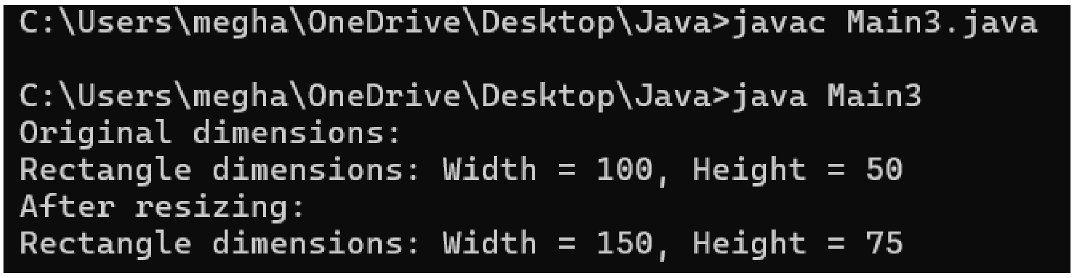
```
C:\Users\megha\OneDrive\Desktop\Java>java Main2
Circle:
Area: 78.54
Perimeter: 31.42

Triangle:
Area: 6.00
Perimeter:12.00
```

6. 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 {
  private int width,height;

  public Rectangle(int width, int height)
  {  this.width = width;  this.height = height; }

  @Override
  public void resizeWidth(int width)
  { this.width = width; }

  @Override
  public void resizeHeight(int height)
  { this.height = height; }

  public void displayDimensions()  {
   System.out.println("Rectangle dimensions: Width = "
                   +width+", Height = "+height);
  }
}

public class Main {
  public static void main(String[] args) {
    Rectangle rect = new Rectangle(100, 50);

    System.out.println("Original dimensions:");
```

```
        rect.displayDimensions();

        rect.resizeWidth(150);
        rect.resizeHeight(75);

        System.out.println("After resizing:");
        rect.displayDimensions();
      }  }
```

OUTPUT:

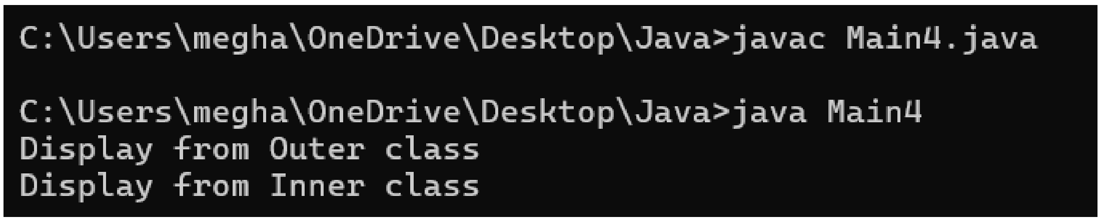


```
C:\Users\megha\OneDrive\Desktop\Java>javac Main3.java

C:\Users\megha\OneDrive\Desktop\Java>java Main3
Original dimensions:
Rectangle dimensions: Width = 100, Height = 50
After resizing:
Rectangle dimensions: Width = 150, Height = 75
```

7. 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("Display from Outer class"); }

  class Inner
  {
   void display()
   { System.out.println("Display from Inner class"); }
  }
}

public class Main
{
  public static void main(String[] args) {
    Outer outer = new Outer();
    outer.display();
    Outer.Inner inner = outer.new Inner();
    inner.display();
  }
}
```

OUTPUT:

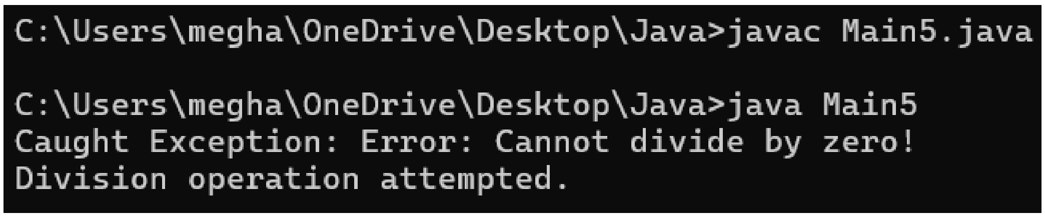


```
C:\Users\megha\OneDrive\Desktop\Java>javac Main4.java

C:\Users\megha\OneDrive\Desktop\Java>java Main4
Display from Outer class
Display from Inner class
```

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

```
class DivisionByZeroException extends Exception {
  public DivisionByZeroException(String message)
  { super(message); }
}

public class Main {
   public static int divide(int numerator, int denominator)
                              throws DivisionByZeroException
   {
     if (denominator == 0)
     throw new DivisionByZeroException("Error: Cannot
                                 divide by zero!");
        return numerator / denominator;
}
 public static void main(String[] args)
 {
    int a = 10;
    int b = 0;
//change the 'b' value other than 0 in the 2nd time execution

    try {
        int result = divide(a, b);
        System.out.println("Result: "+result);
    }
    catch (DivisionByZeroException e)
    {System.out.println("Caught Exception: "+e.getMessage());}
    finally
    {System.out.println("Division operation attempted."); }
 }
}
```

OUTPUT:

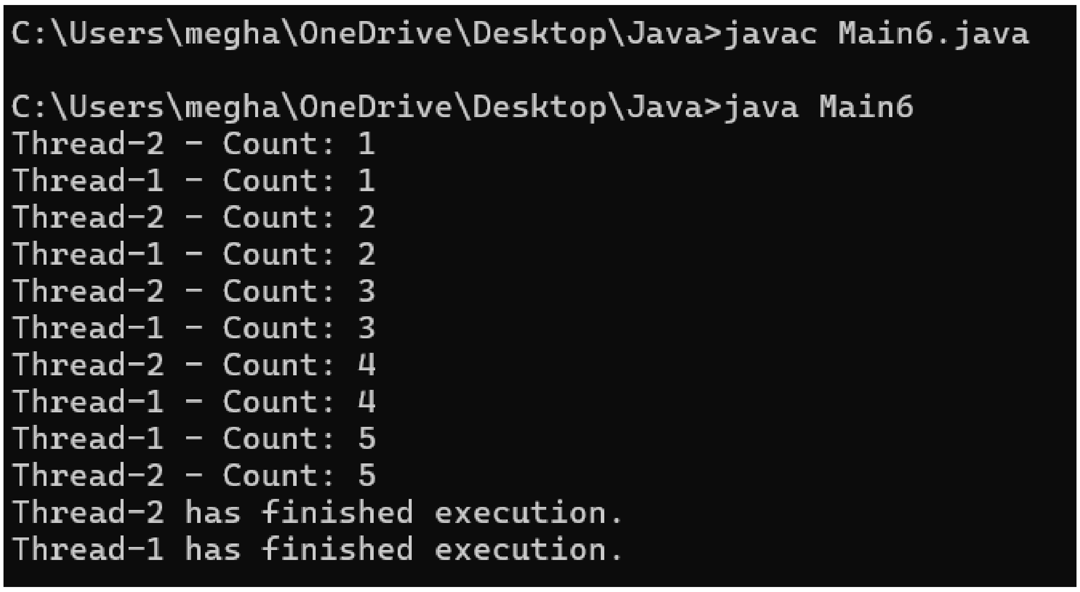


```
C:\Users\megha\OneDrive\Desktop\Java>javac Main5.java

C:\Users\megha\OneDrive\Desktop\Java>java Main5
Caught Exception: Error: Cannot divide by zero!
Division operation attempted.
```

9. Write a program to illustrate creation of threads using a 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 name)
  {  this.threadName = name; }

  public void run()
  {
   for(int i = 1; i <= 5; i++)
   {
     System.out.println(threadName + " - Count: " + i);
     try
     {
       Thread.sleep(500);
     }
     catch (InterruptedException e)
     {System.out.println(threadName+" was interrupted.");}
   }
   System.out.println(threadName+" has finished execution.");
 }
}

public class Main
{
  public static void main(String[] args)
  {
   MyRunnable task1 = new MyRunnable("Thread-1");
   MyRunnable task2 = new MyRunnable("Thread-2");

// Creating Thread objects and passing Runnable instances
    Thread t1 = new Thread(task1);
    Thread t2 = new Thread(task2);

// Starting the threads
    t1.start();
    t2.start();
  }
}
```

OUTPUT:



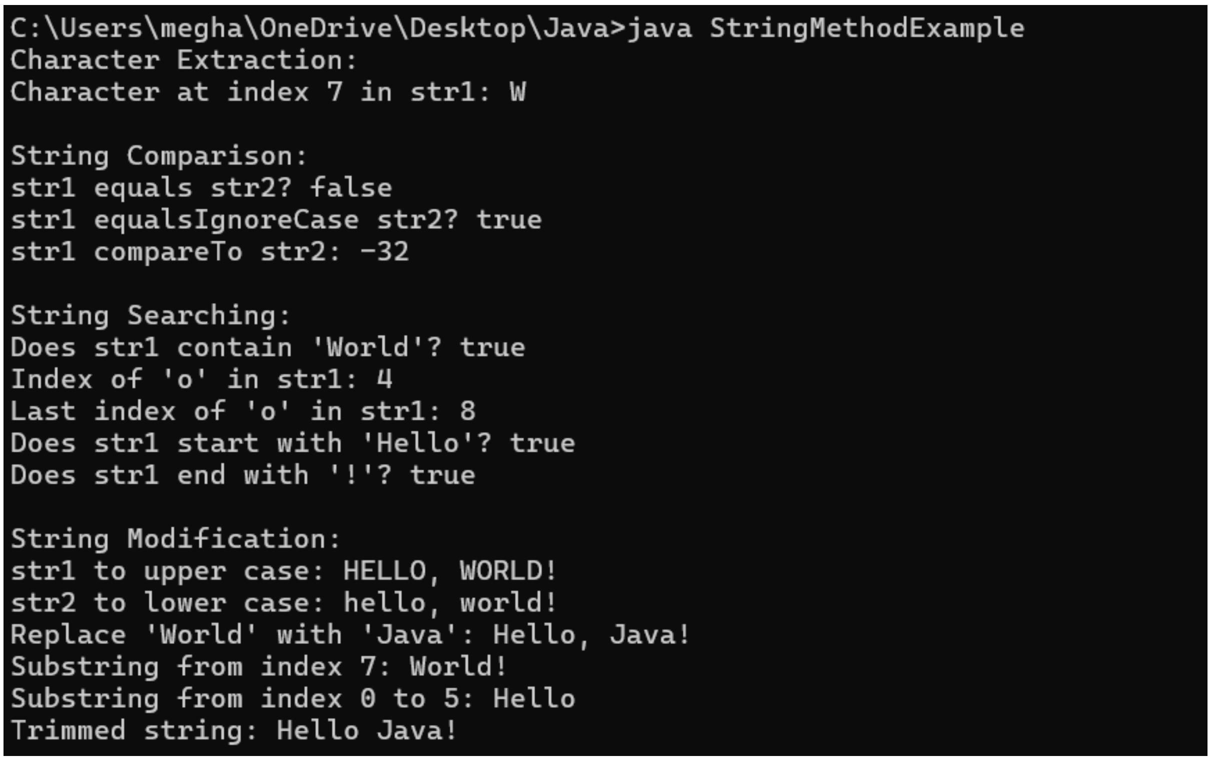
```
C:\Users\megha\OneDrive\Desktop\Java>javac Main6.java

C:\Users\megha\OneDrive\Desktop\Java>java Main6
Thread-2 - Count: 1
Thread-1 - Count: 1
Thread-2 - Count: 2
Thread-1 - Count: 2
Thread-2 - Count: 3
Thread-1 - Count: 3
Thread-2 - Count: 4
Thread-1 - Count: 4
Thread-1 - Count: 5
Thread-2 - Count: 5
Thread-2 has finished execution.
Thread-1 has finished execution.
```

10.Implement a JAVA program to illustrate the use of different types of character extraction, string comparison, string search and string modification methods.

```java
public class StringMethodExample
{
   public static void main(String[] args) {
       String str1 = "Hello, World!";
       String str2 = "hello, world!";

     // 1. Character Extraction
       System.out.println("Character Extraction:");
       System.out.println("Character at index 7 in str1: " +
                                         str1.charAt(7));
       System.out.println();

       // 2. String Comparison
       System.out.println("String Comparison:");
       System.out.println("str1 equals str2? "+
                                       str1.equals(str2));
       System.out.println("str1 equalsIgnoreCase str2? "+
                               str1.equalsIgnoreCase(str2));
       System.out.println("str1 compareTo str2: "+
                               str1.compareTo(str2));
       System.out.println();


       // 3. String Searching
       System.out.println("String Searching:");
       System.out.println("Does str1 contain 'World'? "+
                                   str1.contains("World"));
       System.out.println("Index of 'o' in str1: "+
                                   str1.indexOf('o'));
```

```
        System.out.println("Last index of 'o' in str1: "+
                                  str1.lastIndexOf('o'));
        System.out.println("Does str1 start with 'Hello'? "+
                                 str1.startsWith("Hello"));
        System.out.println("Does str1 end with '!'? "+
                                  str1.endsWith("!"));
        System.out.println();

        // 4. String Modification
        System.out.println("String Modification:");
        System.out.println("str1 to upper case: "+
                                  str1.toUpperCase());
        System.out.println("str2 to lower case: "+
                                  str2.toLowerCase());
        System.out.println("Replace 'World' with 'Java': "+
                              str1.replace("World", "Java"));
        System.out.println("Substring from index 7: "+
                                  str1.substring(7));
        System.out.println("Substring from index 0 to 5: "+
                                  str1.substring(0, 5));
        System.out.println("Trimmed string: " + (" Hello
                                   Java!   ".trim()));
    }
}
```

OUTPUT:



```
C:\Users\megha\OneDrive\Desktop\Java>java StringMethodExample
Character Extraction:
Character at index 7 in str1: W

String Comparison:
str1 equals str2? false
str1 equalsIgnoreCase str2? true
str1 compareTo str2: -32

String Searching:
Does str1 contain 'World'? true
Index of 'o' in str1: 4
Last index of 'o' in str1: 8
Does str1 start with 'Hello'? true
Does str1 end with '!'? true

String Modification:
str1 to upper case: HELLO, WORLD!
str2 to lower case: hello, world!
Replace 'World' with 'Java': Hello, Java!
Substring from index 7: World!
Substring from index 0 to 5: Hello
Trimmed string: Hello Java!
```

SOLID Principles

1. S - Single-responsibility Principle

2. O - Open-closed Principle
3. L - Liskov Substitution Principle
4. I - Interface Segregation Principle
5. D - Dependency Inversion Principle

1. Single-responsibility Principle (SRP)
SRP Principle states
“A class should have one and only one reason to change, meaning that a class should have only one job.”

2. Open-Closed Principle (OCP)
“Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification”

3. Liskov Substitution Principle (LSP)
"Derived or child classes must be substitutable for their base or parent classes".
This principle ensures that any class that is the child of a parent class should be usable in place of its parent without any unexpected behaviour.

4. Interface Segregation Principle (ISP)
"**do not force any client to implement an interface which is irrelevant to them**".
This principle is the first principle that applies to Interfaces instead of classes in SOLID and it is similar to the single responsibility principle.

5. Dependency Inversion Principle (DIP)
"High-level modules should not depend on low-level modules. Both should depend on abstractions".
Additionally, abstractions should not depend on details. Details should depend on abstractions.

DIP suggests that classes should rely on abstractions (e.g., interfaces or abstract classes) rather than concrete implementations, which allows for more flexible and decoupled code, making it easier to change implementations without affecting other parts of the codebase.

11. Develop a Java application that checks whether a given string is a palindrome, designed using the SOLID principles of object-oriented programming.

Package: input
UserInputProvider.java (SRP+DIP)

```
package input;
public interface UserInputProvider
{ String getInput(); }
```

ConsoleInputProvider.java

```
package input;
import java.util.Scanner;
public class ConsoleInputProvider implements UserInputProvider {
  @Override
  public String getInput()
    {
           Scanner sc = new Scanner(System.in);
           System.out.println("Enter a string to check: ");
           return sc.nextLine();
  }
}
```

OUTPUT:

```
project-folder/
 └── input/
       ├── UserInputProvider.java
       └── ConsoleInputProvider.java
```

Package: logic

PalindromeChecker.java (SRP + OCP)

```
package logic;
public interface PalindromeChecker
{ boolean isPalindrome(String input); }
```

```
project-folder/
 └── logic/
       ├── PalindromeChecker.java
```

DefaultPalindromeChecker.java

```
package logic;
public class DefaultPalindromeChecker implements PalindromeChecker
{
  @Override
  public boolean isPalindrome(String input)
    {
    if (input == null) return false;
      String rev =
         ((new StringBuilder(input)).reverse()).toString();

       if (input.equals(rev))
         return true;
       return false;
  }
}
```

Package: output

ResultPrinter.java (SRP + DIP)

```
package output;
public interface ResultPrinter
{ void printResult(String input, boolean isPalindrome); }
```

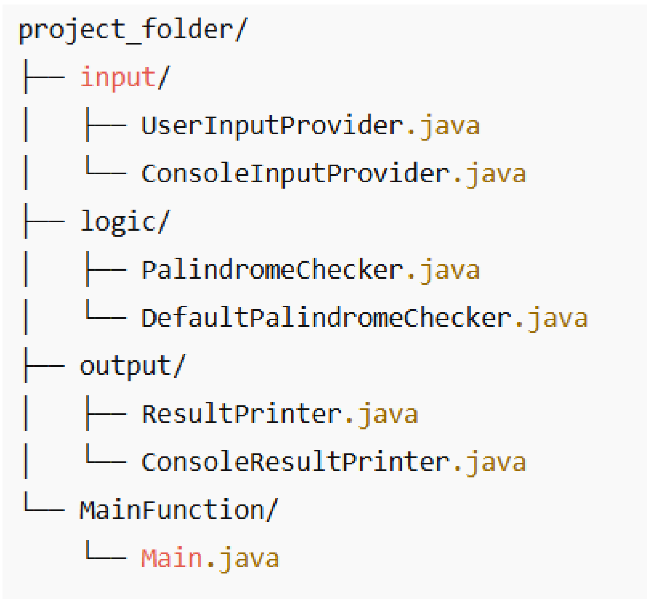
ConsoleResultPrinter.java

```
package output;

public class ConsoleResultPrinter implements ResultPrinter {
    @Override
    public void printResult(String input, boolean isPalindrome) {
        if (isPalindrome) {
            System.out.println(input + " is a palindrome.");
        } else {
            System.out.println(input + " is NOT a palindrome.");
        }
    }
}
```

Package: MainFunction (Dependency Injection+Composition Root)

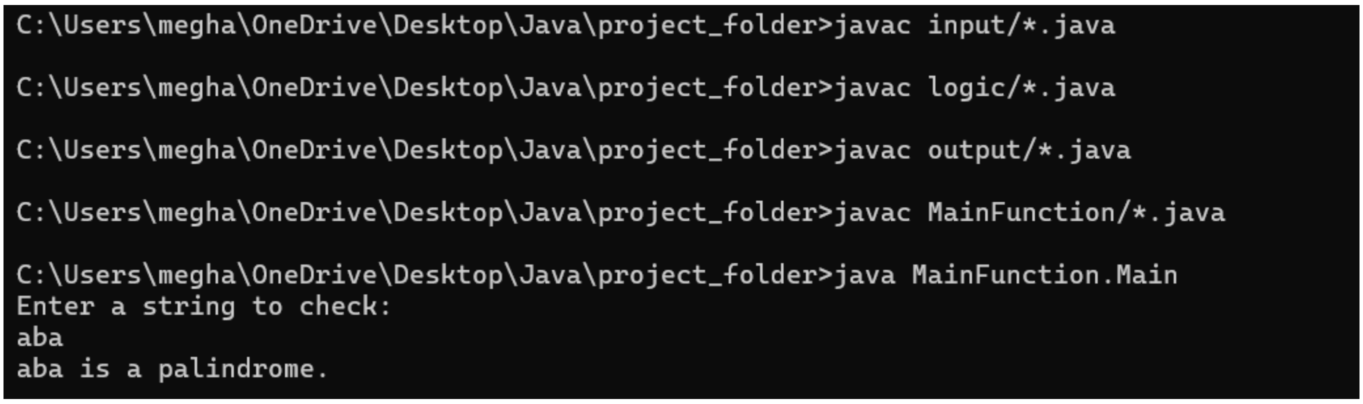
```
package MainFunction;
import input.ConsoleInputProvider;
import input.UserInputProvider;
import logic.DefaultPalindromeChecker;
import logic.PalindromeChecker;
import output.ConsoleResultPrinter;
import output.ResultPrinter;

public class Main
{
  public static void main(String[] args)
  {
 UserInputProvider inputProvider = new ConsoleInputProvider();
 PalindromeChecker checker = new DefaultPalindromeChecker();
 ResultPrinter printer = new ConsoleResultPrinter();
 String input = inputProvider.getInput();
 boolean isPalindrome = checker.isPalindrome(input);
 printer.printResult(input, isPalindrome);
  }
}
```

OUTPUT:



```
project_folder/
├── input/
│   ├── UserInputProvider.java
│   └── ConsoleInputProvider.java
├── logic/
│   ├── PalindromeChecker.java
│   └── DefaultPalindromeChecker.java
├── output/
│   ├── ResultPrinter.java
│   └── ConsoleResultPrinter.java
└── MainFunction/
    └── Main.java
```

OUTPUT:



```
C:\Users\megha\OneDrive\Desktop\Java\project_folder>javac input/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder>javac logic/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder>javac output/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder>javac MainFunction/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder>java MainFunction.Main
Enter a string to check:
aba
aba is a palindrome.
```

**12. Develop a Java application that compresses characters in a given string by counting consecutive repeated characters, designed according to the SOLID principles of object-oriented
programming. Input: AAABBBCCCC Output: A3B3C4**

Package: input

InputProvider.java

```
package input;
public interface InputProvider
{ String getInput(); }
```

ConsoleInputProvider.java

```
package input;
import java.util.Scanner;
public class ConsoleInputProvider implements InputProvider
{
   @Override
   public String getInput()
```

```
        {
            Scanner sc = new Scanner(System.in);
            System.out.println("Enter a string to compress: ");
            return sc.nextLine();
    }
}
```

Package: logic

Compressor.java

```
package logic;
public interface Compressor
{ String compress(String input); }
```

CharacterCompressor.java

```
package logic;
public class CharacterCompressor implements Compressor
{
    @Override
    public String compress(String input) {
       if (input == null || input.isEmpty())
          return "";

       StringBuilder result = new StringBuilder();
       int count = 1;
       char current = input.charAt(0);
       for (int i=1; i<input.length(); i++)
       {
          char next = input.charAt(i);
          if (next == current)
             count++;
          else
               {
             result.append(current).append(count);
             current = next;
             count = 1;
          }
       }
       // to append the last count
       result.append(current).append(count);
       return result.toString();
    }
}
```

Package: output

OutputPrinter.java

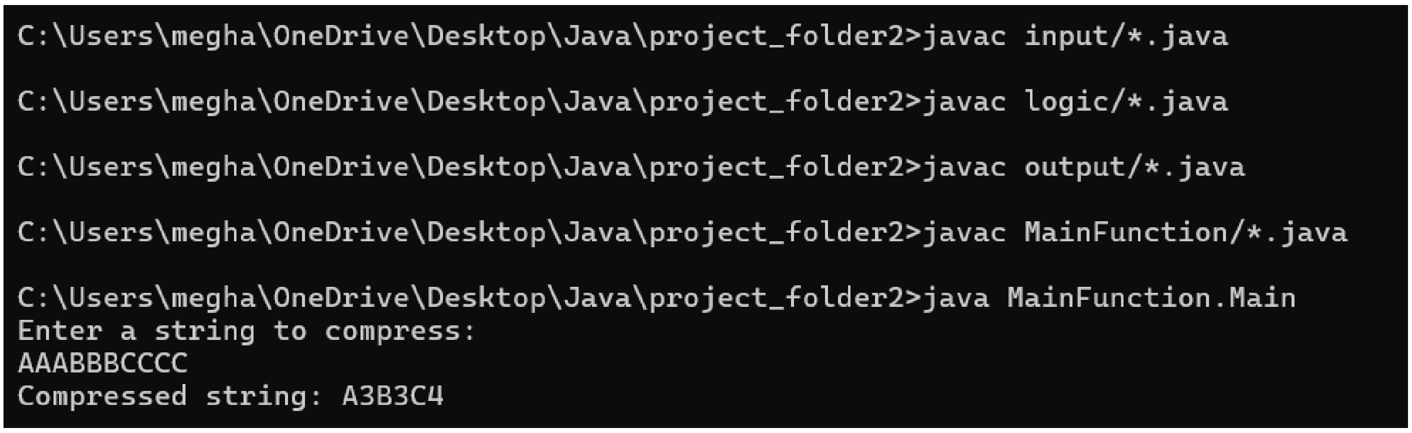
```
package output;
public interface OutputPrinter
{ void print(String output); }
```

ConsoleOutputPrinter.java

```
package output;
public class ConsoleOutputPrinter implements OutputPrinter
{
   @Override
   public void print(String output)
   { System.out.println("Compressed string: "+output); }
}
```

```
package MainFunction;

import input.InputProvider;
import input.ConsoleInputProvider;
import logic.Compressor;
import logic.CharacterCompressor;
import output.OutputPrinter;
import output.ConsoleOutputPrinter;

public class Main {
    public static void main(String[] args) {
        // Dependency Injection (Composition Root)
        InputProvider inputProvider = new ConsoleInputProvider();
        Compressor compressor = new CharacterCompressor();
        OutputPrinter printer = new ConsoleOutputPrinter();

        // Get input
        String input = inputProvider.getInput();

        // Compress input
        String compressed = compressor.compress(input);

        // Print output
        printer.print(compressed);
    }
}
```

OUTPUT:



```
C:\Users\megha\OneDrive\Desktop\Java\project_folder2>javac input/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder2>javac logic/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder2>javac output/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder2>javac MainFunction/*.java

C:\Users\megha\OneDrive\Desktop\Java\project_folder2>java MainFunction.Main
Enter a string to compress:
AAABBBCCCC
Compressed string: A3B3C4
```

VIVA QUESTIONS

1. What is Java?
2. Why is Java platform-independent?
3. What are the main features of Java?
4. What is JVM?
5. What is JIT compiler?
6. What is typecasting in Java?
7. How do you declare an array in Java?
8. What is the main method?
9. What are literals?
10. What is a constructor?
11. What is method overloading?
12. What is a package?
13. What is the difference between a class and an object?
14. What is inheritance?
15. What is polymorphism?
16. What is encapsulation?
17. What is abstraction?
18. What is the difference between an interface and an abstract class?
19. What is the difference between == and .equals()?
20. Why is String immutable?
21. What is the difference between String, StringBuilder, and StringBuffer?
22. What is the difference between a method and a constructor?
23. What is multithreading?
24. What are checked and unchecked exceptions?
25. What is Garbage Collection in Java?
26. What is the difference between ArrayList and LinkedList?
27. What is synchronization?
28. What happens if main method is not static?
29. What is the difference between a program and a process?