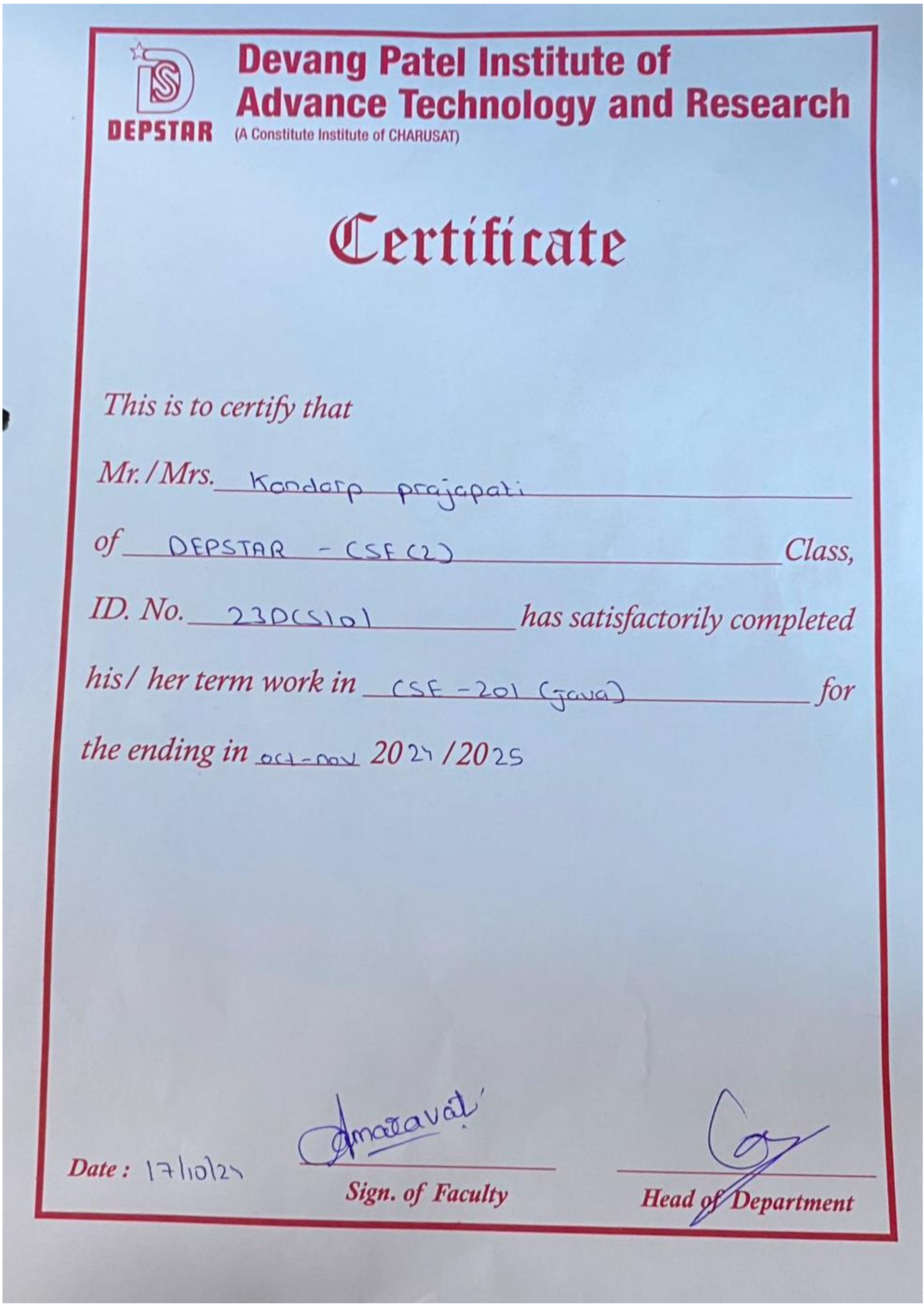


Devang Patel Institute of Advance Technology and Research

DEPSTAR

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr. / Mrs. Kandarp prajapati

of DEPSTAR - CSE (2) *Class,*

ID. No. 23DCS101 *has satisfactorily completed*

his/ her term work in CSE-201 (Java) *for*

the ending in oct-nov 2024 /2025

Date : 17/10/24

Amaravat

Sign. of Faculty

Head of Department

sav

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: JAVA PROGRAMMING

Semester: 3

Subject Code: CSE201

Academic year: 2024-25

PART-1

NO.	AIM OF THE PRACTICAL
1.	Demonstration of installation steps of Java,Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.

PROGRAM:

```
import java.util.*;

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

OUTPUT:

```
Hello world
```

CONCLUSION:

Introduction to Object-Oriented Concepts
Classes and Objects: A class is a blueprint for objects. An object is an instance of a class.
Encapsulation: Bundling data (variables) and methods that operate on the data into a single unit or class.
Inheritance: Mechanism where one class acquires the properties (fields and methods) of another.

Polymorphism: Ability to present the same interface for different underlying forms (data types).
Abstraction: Hiding complex implementation details and showing only the essential features.

Introduction to JDK, JRE, JVM, Javadoc, Command Line Argument
JDK (Java Development Kit): Includes the JRE and development tools (compilers, debuggers).
JRE (Java Runtime Environment): Includes the JVM and libraries necessary to run Java applications.
JVM (Java Virtual Machine): Executes Java bytecode and provides platform independence.
Javadoc: Tool for generating API documentation from Java source code.
Command Line Argument: Parameters passed to the program when it is executed from the command line. Accessible via args parameter in the main method.

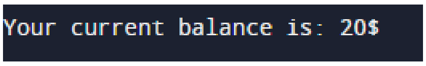
Comparison of Java with Other Object-Oriented Languages
Java vs. C++:
Java is platform-independent due to the JVM, while C++ is platform-specific.
Java has automatic garbage collection; C++ requires manual memory management.
Java does not support pointers directly, while C++ does.
Java has a simpler syntax and is generally easier to learn than C++.

NO.	AIM OF THE PRACTICAL
2.	Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20. Write a java program to store this balance in a variable and then display it to the user.

PROGRAM:

```
import java.util.*;

public class Input {
   public static void main(String[] args) {
      int a=20;
      System.out.println("Your current balance is: "+ a +"$");
   }
}
```

OUTPUT:



```
Your current balance is: 20$
```

CONCLUSION:

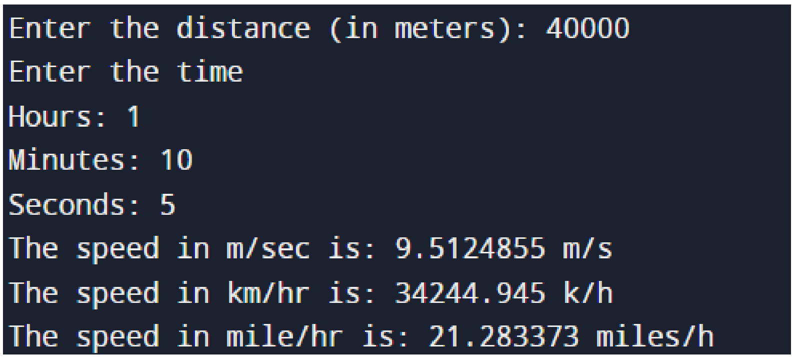
This program demonstrates how to store a user's account balance in a variable and display it. The variable currentBalance is initialized with a value of $20.00, and the balance is printed to the console using System.out.println(). This basic structure can be expanded upon for more complex banking functionalities in the future.

NO.	AIM OF THE PRACTICAL
3.	Write a program to take the user for a distance (in meters) and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint:1 mile = 1609 meters).

PROGRAM:

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {

    Scanner obj = new Scanner(System.in);
    System.out.print("Enter the distance (in meters): ");
    float dist = obj.nextInt();

    System.out.println("Enter the time ");
    System.out.print("Hours: ");
    float hr = obj.nextFloat();
    System.out.print("Minutes: ");
    float min = obj.nextFloat();
    System.out.print("Seconds: ");
    float sec = obj.nextFloat();

    float ms = hr*3600 + min*60 + sec;
    float kh = hr + (min/60) + (sec/3600);
    float mh = dist/1609;

    System.out.println("The speed in m/sec is: "+dist/ms+" m/s");
    System.out.println("The speed in km/hr is: "+dist/kh+" k/h");
    System.out.print("The speed in mile/hr is: "+mh/kh+" miles/h");
  }
}
```

OUTPUT:



```
Enter the distance (in meters): 40000
Enter the time
Hours: 1
Minutes: 10
Seconds: 5
The speed in m/sec is: 9.5124855 m/s
The speed in km/hr is: 34244.945 k/h
The speed in mile/hr is: 21.283373 miles/h
```

CONCLUSION:

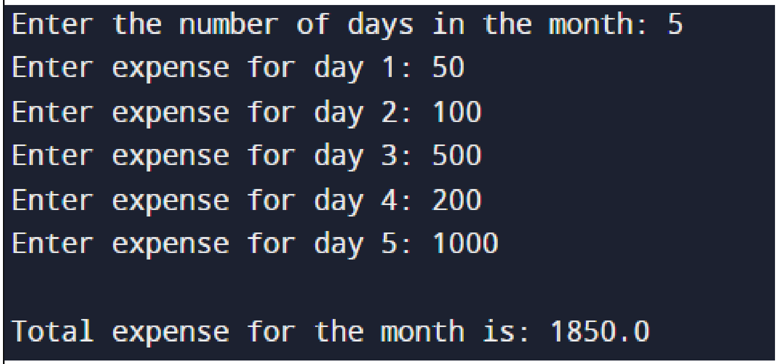
This program calculates the speed in different units based on the distance travelled and the time taken. It prompts the user to input the distance in meters and the time in hours, minutes, and seconds. Using these inputs, it calculates and displays the speed in meters per second, kilometers per hour, and miles per hour. This program showcases basic user input handling, arithmetic operations, and formatted output in Java.

NO.	AIM OF THE PRACTICAL
4.	Imagine you are developing a budget tracking application. You need to calculate the total expenses for the month. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to calculate the sum of elements in an array representing daily expenses.

PROGRAM:

```
import java.util.Scanner;

public class Expenses {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of days in the month: ");
        int days = scanner.nextInt();

        double[] dailyExpenses = new double[days];

        for (int i = 0; i < days; i++) {
            System.out.print("Enter expense for day " + (i + 1) + ": ");
            dailyExpenses[i] = scanner.nextDouble();
        }
        double totalExpenses = 0.0;
        for (double expense : dailyExpenses) {
            totalExpenses = totalExpenses + expense;
        }

        System.out.println("");
        System.out.println("Total expense for the month is: " + totalExpenses);
    }
}
```

OUTPUT:



```
Enter the number of days in the month: 5
Enter expense for day 1: 50
Enter expense for day 2: 100
Enter expense for day 3: 500
Enter expense for day 4: 200
Enter expense for day 5: 1000

Total expense for the month is: 1850.0
```

CONCLUSION:

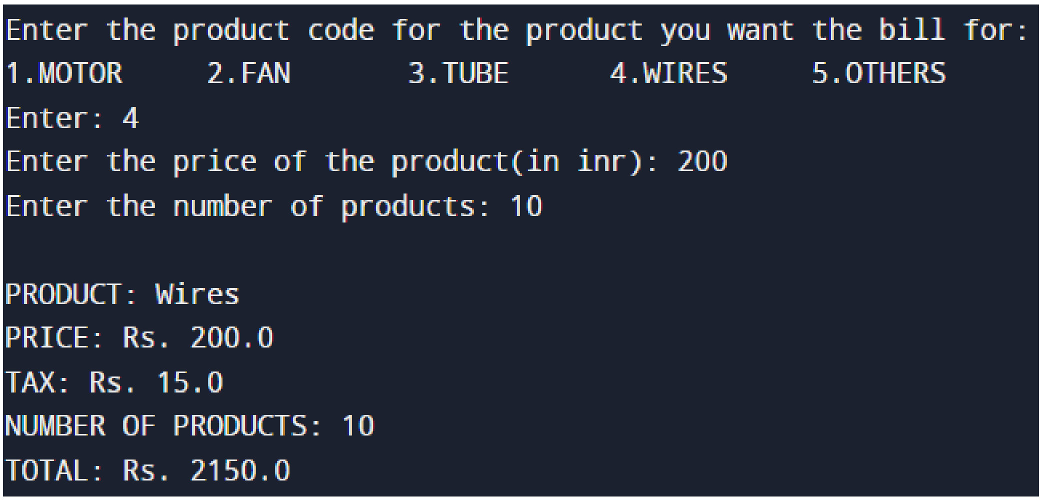
This program helps users track their monthly expenses by allowing them to input their daily expenses. It uses an array to store the expenses for each day of the month and calculates the total expenses by summing the elements of the array. The program then displays the total monthly expenses to the user. This approach demonstrates basic array handling, user input processing, and sum calculation in Java.

NO.	AIM OF THE PRACTICAL
5.	An electric appliance shop assigns code 1 to motor,2 to fan,3 to tube and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor,12% to fan,5% to tube light,7.5% to wires and 3% for all other items is charged. A list containing the product code and price in two different arrays. Write a java program using switch statement to prepare the bill.

PROGRAM:

```
import java.util.Scanner;

public class Shop {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the product code for the product you want the bill for:");
        System.out.print("\n1.MOTOR     2.FAN       3.TUBE      4.WIRES     5.OTHERS");
        System.out.print("\nEnter: ");
        int productCode = scanner.nextInt();

        System.out.print("Enter the price of the product(in inr): ");
        double price = scanner.nextDouble();
        System.out.print("Enter the number of products: ");
        int number = scanner.nextInt();

        double taxRate = 0.0;
        String productName;

        switch (productCode) {
            case 1:
                taxRate = 0.08;
                productName = "Motor";
                break;
            case 2:
                taxRate = 0.12;
                productName = "Fan";
                break;
            case 3:
                taxRate = 0.05;
                productName = "Tube Light";
                break;
            case 4:
                taxRate = 0.075;
                productName = "Wires";
                break;
            default:
                taxRate = 0.03;
                productName = "Other";
                break;
        }

        double tax = price * taxRate;
        double total = price + tax;

        System.out.println("\nPRODUCT: " + productName);
        System.out.println("PRICE: Rs. " + price);
        System.out.println("TAX: Rs. " + tax);
        System.out.println("NUMBER OF PRODUCTS: " + number);
        System.out.println("TOTAL: Rs. " + (total*number));
    }
}
```

OUTPUT:



```
Enter the product code for the product you want the bill for:
1.MOTOR     2.FAN       3.TUBE      4.WIRES     5.OTHERS
Enter: 4
Enter the price of the product(in inr): 200
Enter the number of products: 10

PRODUCT: Wires
PRICE: Rs. 200.0
TAX: Rs. 15.0
NUMBER OF PRODUCTS: 10
TOTAL: Rs. 2150.0
```

CONCLUSION:

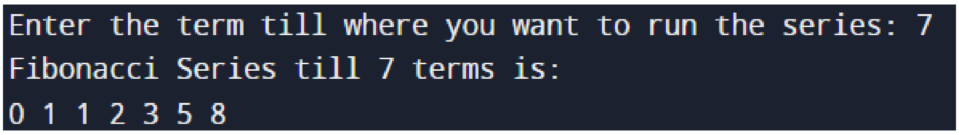
This program calculates the total bill for items purchased from an electric appliance shop by using arrays to store product codes and prices. It applies different tax rates based on the product code using a switch statement. The program iterates through the arrays, calculates the tax and total price for each item, and then sums these to produce the total bill. This example demonstrates the use of arrays, loops, and switch statements in Java to handle varying tax rates and compute a total bill for multiple items.

NO.	AIM OF THE PRACTICAL
6.	Write a program to calculate and display the first n terms of the Fibonacci series.

PROGRAM:

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {

    Scanner obj = new Scanner(System.in);
    System.out.print("Enter the term till where you want to run the series: ");
    int n = obj.nextInt();
    int firstTerm = 0, secondTerm = 1;
    System.out.println("Fibonacci Series till " + n + " terms is:");

    for (int i = 1; i <= n; i++) {
      System.out.print(firstTerm + " ");
      int nextTerm = firstTerm + secondTerm;
      firstTerm = secondTerm;
      secondTerm = nextTerm;
    }
  }
}
```

OUTPUT:



```
Enter the term till where you want to run the series: 7
Fibonacci Series till 7 terms is:
0 1 1 2 3 5 8
```

CONCLUSION:

This program helps users generate an exercise routine based on the Fibonacci series. By entering the number of days (n), the program calculates and displays the first n terms of the Fibonacci series, representing the exercise duration for each day. This approach ensures a progressively increasing exercise duration, which can be beneficial for building endurance. The program demonstrates user input handling, loop structures, and Fibonacci series calculation in Java.

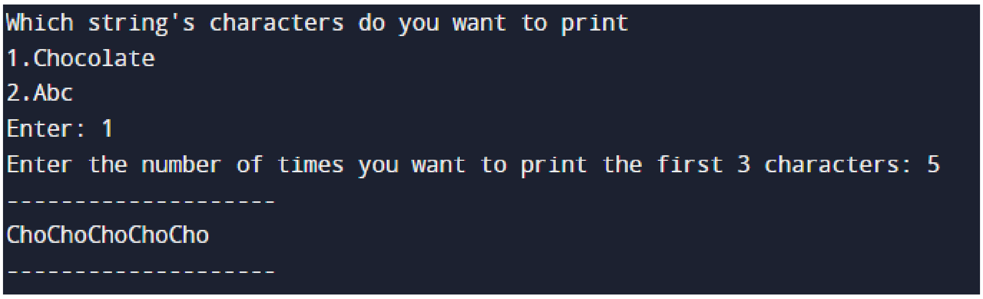
PART-2

NO.	AIM OF THE PRACTICAL
7.	Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front; front_times('Chocolate', 2) → 'ChoCho' front_times('Chocolate', 3) → 'ChoChoCho' front_times('Abc', 3) → 'AbcAbcAbc'

PROGRAM:

```
import java.util.*;

public class Main {

    public static void main(String[] args) {
        String st1 = "Chocolate";
        String st2 = "Abc";
        int n, m;
        Scanner obj = new Scanner(System.in);
        System.out.print("Which string's characters do you want to print");
        System.out.print("\n1.Chocolate\n2.Abc");
        System.out.print("\nEnter: ");
        int choice = obj.nextInt();
        if(choice==1){
            System.out.print("Enter the number of times you want to print the first
3 characters: ");
            n = obj.nextInt();
            System.out.println("--------------------");
            for (int i = 0; i < n; i++) {
            System.out.print(st1.substring(0, 3));
        }
        System.out.println("\n--------------------");
        }
        else{
        System.out.print("Enter the number of times you want to print the first 3
characters:");
        m = obj.nextInt();
        System.out.println("--------------------");
        for (int i = 0; i < m; i++) {
            System.out.print(st2.substring(0, 3));
        }
        System.out.println("\n--------------------");
        }
    }
}
```

OUTPUT:



```
Which string's characters do you want to print
1.Chocolate
2.Abc
Enter: 1
Enter the number of times you want to print the first 3 characters: 5
--------------------
ChoChoChoChoCho
--------------------
```

CONCLUSION:

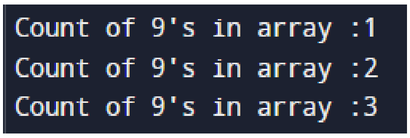
The Java program allows users to choose between two strings, "Chocolate" and "Abc," and print the first three characters of the chosen string a specified number of times. The user is prompted to enter a choice (1 or 2) and then asked for the number of repetitions. Based on the input, the program uses a loop to print the first three characters of the selected string the desired number of times. This demonstrates basic user input handling, string manipulation, and loop control in Java.

NO.	AIM OF THE PRACTICAL
8.	Given an array of ints, return the number of 9's in the array. array_count9([1, 2, 9]) → 1 array_count9([1, 9, 9]) → 2 array_count9([1, 9, 9, 3, 9]) → 3

PROGRAM:

```
public class Main {
  public static void main(String[] args) {
            int arr1[]={1,2,9};
            int arr2[]={1,9,9};
            int arr3[]={1,9,9,3,9};
            array_count9(arr1);
            array_count9(arr2);
            array_count9(arr3);
        }
        static void array_count9(int arr[]){
            int count=0;
            for(int i=0;i<arr.length;i++){
                if(arr[i]==9){
                    count++;
                }
            }
        System.out.println("Count of 9's in array :"+count);
  }
  }
```

OUTPUT:



```
Count of 9's in array :1
Count of 9's in array :2
Count of 9's in array :3
```

CONCLUSION:

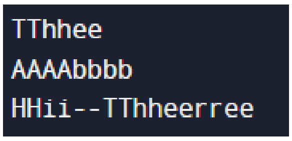
The Java program defines a `Main` class that counts the number of times the digit '9' appears in different integer arrays. Three arrays, `arr1`, `arr2`, and `arr3`, are initialized with various elements. The program calls the `array_count9` method for each array. This method iterates through the array elements and counts how many times the digit '9' appears. The result is then printed to the console. This program showcases the use of arrays, loops, and conditional statements to perform a simple counting task in Java.

NO.	AIM OF THE PRACTICAL
9.	Given a string, return a string where for every char in the original, there are two chars. double_char('The') → 'TThhee' double_char('AAbb') → 'AAAAbbbb' double_char('Hi-There') → 'HHii--TThheerree'

PROGRAM:

```
public  class Main{
   public String doubleChar(String str) {
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < str.length(); i++) {
         result.append(str.charAt(i));
         result.append(str.charAt(i));
      }
      return result.toString();
   }
   public static void main(String[] args) {
      Main obj = new Main();
      System.out.println(obj.doubleChar("The"));
      System.out.println(obj.doubleChar("AAbb"));
      System.out.println(obj.doubleChar("Hi-There"));
   }
}
```

OUTPUT:



```
TThhee
AAAAbbbb
HHii--TThheerree
```

CONCLUSION:

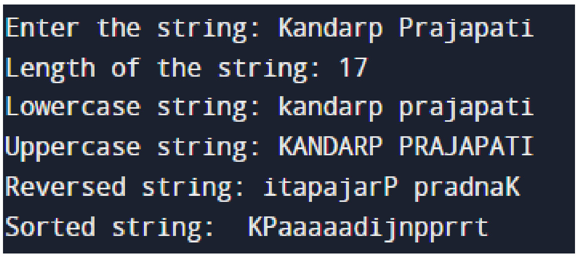
The Java program defines a `Main` class with a `doubleChar` method that doubles each character in a given string. The method uses a `StringBuilder` to construct the new string by appending each character twice in a loop. In the `main` method, an instance of `Main` is created, and `doubleChar` is called with different strings ("The", "AAbb", and "Hi-There"), demonstrating the method's functionality. The results are printed to the console. This program illustrates string manipulation using loops and the `StringBuilder` class in Java.

NO.	AIM OF THE PRACTICAL
10.	Perform following functionalities of the string: • Find Length of the String • Lowercase of the String • Uppercase of the String • Reverse String • Sort String

PROGRAM:

```
import java.util.*;

public class Main {
   public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = sc.nextLine();

        System.out.println("Length of the string: " + str.length());
        System.out.println("Lowercase string: " + str.toLowerCase());
        System.out.println("Uppercase string: " + str.toUpperCase());

        String reversedStr = new StringBuilder(str).reverse().toString();
        System.out.println("Reversed string: " + reversedStr);

        char[] charArray = str.toCharArray();
        Arrays.sort(charArray);
        String sortedStr = new String(charArray);
        System.out.println("Sorted string: " + sortedStr);
    }
}
```

OUTPUT:



```
Enter the string: Kandarp Prajapati
Length of the string: 17
Lowercase string: kandarp prajapati
Uppercase string: KANDARP PRAJAPATI
Reversed string: itapajarP pradnaK
Sorted string:  KPaaaaadijnpprrt
```

CONCLUSION:

The Java program prompts the user to input a string and then performs several operations on it. It displays the length of the string, converts and prints the string in both lowercase and uppercase, reverses the string and prints the result, and finally sorts the characters of the string in alphabetical order and prints the sorted string. This program demonstrates basic string manipulation techniques in Java, including case conversion, reversal, and sorting, utilizing classes like `Scanner`, `StringBuilder`, and `Arrays`.

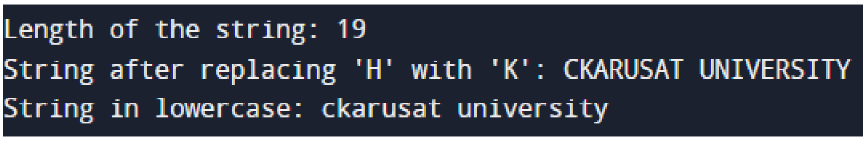
NO.	AIM OF THE PRACTICAL
11.	Perform following Functionalities of the string: “CHARUSAT UNIVERSITY” ● Find length ● Replace ‘H’ by ‘FIRST LETTER OF YOUR NAME’ ● Convert all character in lowercase

PROGRAM:

```
public class Main {
    public static void main(String[] args) {
        String input = "CHARUSAT UNIVERSITY";

        int length = input.length();
        System.out.println("Length of the string: " + length);

        String firstName = "Kandarp";
        char firstLetter = firstName.charAt(0);
        String replacedString = input.replace('H', firstLetter);
        System.out.println("String after replacing 'H' with '" + firstLetter + "': "
+replacedString);
```

```
        String lowerCaseString = replacedString.toLowerCase();
        System.out.println("String in lowercase: " + lowerCaseString);
    }
}
```

OUTPUT:



```
Length of the string: 19
String after replacing 'H' with 'K': CKARUSAT UNIVERSITY
String in lowercase: ckarusat university
```

CONCLUSION:

The Java program processes the string "CHARUSAT UNIVERSITY" by performing a series of operations. First, it calculates and prints the length of the string. Then, it replaces the character 'H' in the string with the first letter of the name "Kandarp," resulting in "CKARUSAT UNIVERSITY." Finally, the program converts the modified string to lowercase and prints it as "ckarusat university." This demonstrates string manipulation techniques such as character replacement and case conversion in Java.

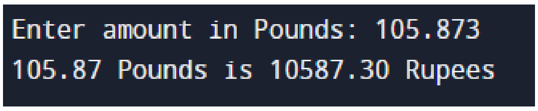
PART-3

NO.	AIM OF THE PRACTICAL
12.	Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.

PROGRAM:

```
import java.util.Scanner;

public class CurrencyConverter {

    private static final double CONVERSION_RATE = 100.0;

    public static void main(String[] args) {
        if (args.length > 0) {

            try {
                double pounds = Double.parseDouble(args[0]);
                double rupees = convertPoundsToRupees(pounds);
                System.out.printf("%.2f Pounds is %.2f Rupees%n", pounds,
rupees);
            } catch (NumberFormatException e) {
                System.out.println("Invalid input. Please enter a valid number.");
            }
        } else {

            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter amount in Pounds: ");
            try {
                double pounds = scanner.nextDouble();
                double rupees = convertPoundsToRupees(pounds);
                System.out.printf("%.2f Pounds is %.2f Rupees%n", pounds,
rupees);
            } catch (Exception e) {
                System.out.println("Invalid input. Please enter a valid number.");
            } finally {
                scanner.close();
            }
        }
    }
  private static double convertPoundsToRupees(double pounds) {
      return pounds * CONVERSION_RATE;
    }
}
```

OUTPUT:



```
Enter amount in Pounds: 105.873
105.87 Pounds is 10587.30 Rupees
```

CONCLUSION:

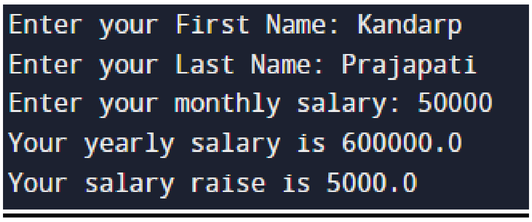
By developing this currency conversion tool, we gain insights into the importance of user-centric design, the need for flexibility in input methods, and the value of creating simple yet effective solutions to meet specific business needs.

NO.	AIM OF THE PRACTICAL
13.	Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise and display each Employee's yearly salary again.

PROGRAM:

```
import java.util.Scanner;

public class Main {

    String fname;
    String lname;
    double salary;

    Main() {
        fname = "";
        lname = "";
        salary = 0.0;
    }

    Main(String fname, String lname, double salary) {
        this.fname = fname;
        this.lname = lname;
        this.salary = salary;
    }

    void get() {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your First Name: ");
        fname = sc.nextLine();
        System.out.print("Enter your Last Name: ");
        lname = sc.nextLine();
        System.out.print("Enter your monthly salary: ");
        salary = sc.nextDouble();

        if (salary < 0) {
            salary = 0.0;
            System.out.println("Your salary is " + salary);
        } else {
            double year = salary * 12;
            System.out.println("Your yearly salary is " + year);
            double raise = salary * 0.1;
            System.out.println("Your salary raise is " + raise);
        }
    }

    public static void main(String[] args) {
        Main m = new Main();
        m.get();

    }
}
```

OUTPUT:



```
Enter your First Name: Kandarp
Enter your Last Name: Prajapati
Enter your monthly salary: 50000
Your yearly salary is 600000.0
Your salary raise is 5000.0
```

CONCLUSION:

The development of the Employee class and the corresponding EmployeeTest application illustrates key concepts in object-oriented programming, including encapsulation, data validation, and the use of constructors, getters, and setters.

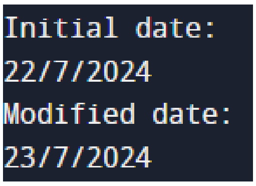
NO.	AIM OF THE PRACTICAL
14.	Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date's capabilities.

PROGRAM:

```
class Date {
    private int month;
    private int day;
    private int year;

    public Date(int month, int day, int year) {
        this.month = month;
        this.day = day;
        this.year = year;
    }

    public void setMonth(int month) {
        this.month = month;
    }

    public void setDay(int day) {
        this.day = day;
    }

    public void setYear(int year) {
        this.year = year;
    }

    public int getMonth() {
        return month;
    }

    public int getDay() {
        return day;
    }

    public int getYear() {
        return year;
    }

    public void displayDate() {
        System.out.println(day + "/" + month + "/" + year);
    }
}
```

```
public class DateTest {
    public static void main(String[] args) {

        Date date = new Date(07, 22, 2024);

        System.out.println("Initial date:");
        date.displayDate();

        date.setMonth(07);
        date.setDay(23);
        date.setYear(2024);

        System.out.println("Modified date:");
        date.displayDate();
    }
}
```

OUTPUT:



```
Initial date:
22/7/2024
Modified date:
23/7/2024
```

CONCLUSION:

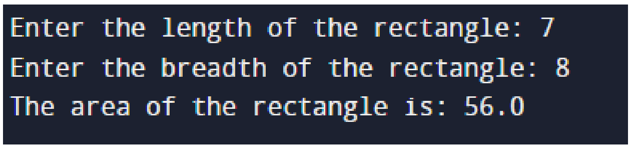
By developing the Date class and the DateTest application, we reinforce fundamental object-oriented programming principles such as encapsulation, proper use of constructors, getters, and setters, and the importance of method implementation for functionality. Testing the class through a dedicated test application demonstrates how to create and manipulate objects, ensuring that the class behaves as expected in various scenarios. This project serves as a practical exercise in designing and using classes in Java, laying the groundwork for more complex software development tasks.

NO.	AIM OF THE PRACTICAL
15.	Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.

PROGRAM:

```
import java.util.Scanner;

public class Area {
    private double length;
    private double breadth;

    public Area(double length, double breadth) {
        this.length = length;
        this.breadth = breadth;
    }

    public double returnArea() {
        return length * breadth;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the length of the rectangle: ");
        double length = scanner.nextDouble();

        System.out.print("Enter the breadth of the rectangle: ");
        double breadth = scanner.nextDouble();

        Area rectangle = new Area(length, breadth);

        System.out.println("The area of the rectangle is: " +
rectangle.returnArea());
    }
}
```

OUTPUT:



```
Enter the length of the rectangle: 7
Enter the breadth of the rectangle: 8
The area of the rectangle is: 56.0
```

CONCLUSION:

By developing the Area class and the corresponding test program, we gain a deeper understanding of fundamental OOP concepts, such as encapsulation, constructors, and methods. Handling user input and demonstrating the class's functionality in a main method highlight the practical application of these concepts. This exercise reinforces the importance of modularity, reusability, and proper class design in creating robust and maintainable software.

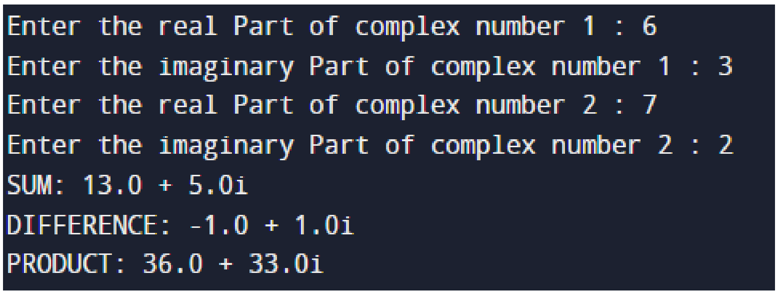
NO.	AIM OF THE PRACTICAL
16.	Print the sum, difference and product of two complex numbers by creating a class named ‘Complex’ with separate methods for each operation whose real and imaginary parts are entered by user.

PROGRAM:

```
import java.util.Scanner;

class Complex
{
   double sum1, sum2, difference1, difference2, product1, product2;
   double a, b, c, d;

   void getData()
{
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the real Part of complex number 1 : ");
      a = scanner.nextDouble();
      System.out.print("Enter the imaginary Part of complex number 1 : ");
      b = scanner.nextDouble();
      System.out.print("Enter the real Part of complex number 2 : ");
      c = scanner.nextDouble();
      System.out.print("Enter the imaginary Part of complex number 2 : ");
      d = scanner.nextDouble();
}

   void sum()
{
      sum1 = a + c;
      sum2 = b + d;
}
 void difference()
{
      difference1 = a - c;
      difference2 = b - d;
}

   void product()
{
      product1 = (a * c - b * d);
      product2 = (a * d + b * c);
}

   void display()
{
      System.out.println("SUM: " + sum1 + " + " + sum2 + "i");
      System.out.println("DIFFERENCE: " + difference1 + " + " + difference2
+ "i");
      System.out.println("PRODUCT: " + product1 + " + " + product2 + "i");
}
```

```
    public static void main(String args[])
{
        Complex c = new Complex();
        c.getData();
        c.sum();
        c.difference();
        c.product();
        c.display();
}
}
```

OUTPUT:



```
Enter the real Part of complex number 1 : 6
Enter the imaginary Part of complex number 1 : 3
Enter the real Part of complex number 2 : 7
Enter the imaginary Part of complex number 2 : 2
SUM: 13.0 + 5.0i
DIFFERENCE: -1.0 + 1.0i
PRODUCT: 36.0 + 33.0i
```

CONCLUSION:

By developing the Complex class and the corresponding test program, we reinforce fundamental OOP concepts such as encapsulation, constructors, and methods. Handling user input and demonstrating the class's functionality in a main method highlight the practical application of these concepts. This exercise emphasizes the importance of modularity, reusability, and proper class design in creating robust and maintainable software.

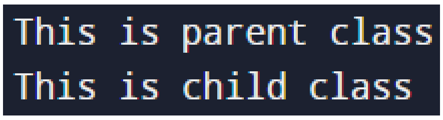
PART-4

NO.	AIM OF THE PRACTICAL
17.	Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent

PROGRAM:

```
class Parent
{
   void printParent()
   {
      System.out.println("This is parent class");
   }
}
class Child extends Parent
{
   void printChild()
   {
      System.out.println("This is child class");
   }
}

public class Main_17
{
   public static void main(String[] args)
   {

      Parent parent = new Parent();
      parent.printParent();

      Child child = new Child();
      child.printChild();
   }}
```

OUTPUT:



```
This is parent class
This is child class
```

CONCLUSION:

The Parent class has a method printParent() that prints “This is parent class”.The Child class extends Parent and has an additional method printChild() that prints “This is child class”.

In the Main class, we create objects for both Parent and Child classes and call their respective methods.

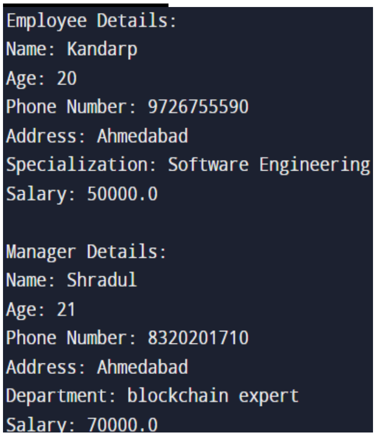
NO.	AIM OF THE PRACTICAL
18.	Create a class named 'Member' having the following members: Data members 1 - Name 2 - Age 3 - Phone number 4 - Address 5 – Salary It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

PROGRAM:

```
class Member
{
   String name;
   int age;
   String phoneNumber;
   String address;
   double salary;

   void printSalary()
   {
      System.out.println("Salary: " + salary);
   }
}


class Employee extends Member {
   String specialization;

   void displayEmployeeDetails() {
      System.out.println("Employee Details:");
      System.out.println("Name: " + name);
      System.out.println("Age: " + age);
      System.out.println("Phone Number: " + phoneNumber);
      System.out.println("Address: " + address);
      System.out.println("Specialization: " + specialization);
      printSalary();
   }
}


class Manager extends Member {
   String department;
```

```
    void displayManagerDetails() {
        System.out.println("Manager Details:");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Phone Number: " + phoneNumber);
        System.out.println("Address: " + address);
        System.out.println("Department: " + department);
        printSalary();
    }
}

public class Main {
    public static void main(String[] args) {

        Employee employee = new Employee();
        employee.name = "Shradul";
        employee.age = 20;
        employee.phoneNumber = "9726755590";
        employee.address = "Ahmedabad";
        employee.salary = 50000.0;
        employee.specialization = "Software Engineering";
        employee.displayEmployeeDetails();

        System.out.println();


        Manager manager = new Manager();
        manager.name = "Shradul";
        manager.age = 21;
        manager.phoneNumber = "8320201710";
        manager.address = "Ahmedabad";
        manager.salary = 70000.0;
        manager.department = "blockchain expert";
        manager.displayManagerDetails();
    }
}
```

OUTPUT:



```
Employee Details:
Name: Kandarp
Age: 20
Phone Number: 9726755590
Address: Ahmedabad
Specialization: Software Engineering
Salary: 50000.0

Manager Details:
Name: Shradul
Age: 21
Phone Number: 8320201710
Address: Ahmedabad
Department: blockchain expert
Salary: 70000.0
```

CONCLUSION:

The Member class has data members for name, age, phone number, address, and salary, along with a method printSalary().The Employee class extends Member and adds a specialization data member.The Manager class extends Member and adds a department data member.The Main class uses a Scanner to take input from the user and assigns values to the data members of Employee and Manager objects.Finally, it prints the details of both the Employee and Manager.

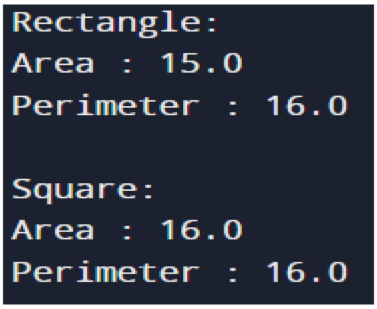
NO.	AIM OF THE PRACTICAL
19.	Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.

PROGRAM:

```
class Rectangle {
   double length;
   double breadth;

   Rectangle(double length, double breadth) {
      this.length = length;
      this.breadth = breadth;
   }

   void printArea() {
      System.out.println("Area : " + length * breadth);
   }

   void printPerimeter() {
      System.out.println("Perimeter : " + 2 * (length + breadth));
   }
}


class Square extends Rectangle {
   Square(double side) {
      super(side, side);
   }
}

public class Main {
   public static void main(String[] args) {

      Rectangle[] rectangles = new Rectangle[2];
```

```
        rectangles[0] = new Rectangle(5.0, 3.0);
        System.out.println("Rectangle:");
        rectangles[0].printArea();
        rectangles[0].printPerimeter();

        System.out.println();


        rectangles[1] = new Square(4.0);
        System.out.println("Square:");
        rectangles[1].printArea();
        rectangles[1].printPerimeter();
    }
}
```

OUTPUT:



```
Rectangle:
Area : 15.0
Perimeter : 16.0

Square:
Area : 16.0
Perimeter : 16.0
```

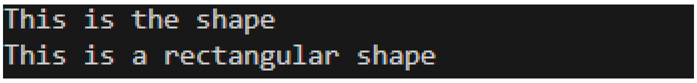
CONCLUSION:

The Rectangle class has data members for length and breadth, and methods to print the area and perimeter.The Square class extends Rectangle and uses the super(s, s) constructor to initialize the side.The Main class takes input from the user for both the rectangle and square, creates objects, and stores them in an array.It then prints the area and perimeter of each shape.

NO.	AIM OF THE PRACTICAL
20.	Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.

PROGRAM:

```
class Shape
{
```

	```
	void display()
	{
	System.out.println("This is the shape");
	}
	}

	class Rectangle extends Shape
	{
	void displayRectangle()
	{
	System.out.println("This is a rectangular shape");
	}
	}

	class Circle extends Shape
	{
	void displayCircle()
	{
	System.out.println("This is a circular shape");
	}
	}

	class Square extends Rectangle
	{
	void displaySquare()
	{
	System.out.println("Square is a rectangle");
	}
	}

	public class prac20
	{
	public static void main(String[] args)
	{
	Square s = new Square();
	s.display();
	s.displayRectangle();
	}
	}
	```
	OUTPUT:
	
	```
	This is the shape
	This is a rectangular shape
	```
	CONCLUSION:
	The Shape class has a method printShape() that prints “This is shape”.The Rectangle class extends Shape and has a method printRectangle() that prints “This is rectangular shape”. The Circle class extends Shape and has a method printCircle() that prints

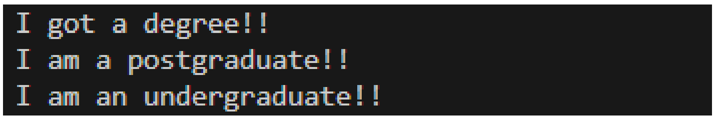
	“This is circular shape”.The Square class extends Rectangle and has a method printSquare() that prints “Square is a rectangle”.In the Main class, we create an object of the Square class and call the methods from Shape and Rectangle classes using this object.

NO.	AIM OF THE PRACTICAL
21.	Create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.

PROGRAM:

```
class Degree
{
void getDegree()
{
System.out.println("I got a degree!!");
}
}

class Postgraduate extends Degree
{
void getDegree()
{
System.out.println("I am a postgraduate!!");
}
}

class Undergraduate extends Degree
{
void getDegree()
{
System.out.println("I am an undergraduate!!");
}
}

public class prac21
{
public static void main(String[] args)
{
Degree d = new Degree();
Postgraduate p = new Postgraduate();
Undergraduate u = new Undergraduate();

d.getDegree();
p.getDegree();
u.getDegree();
}
}
```

OUTPUT:



```
I got a degree!!
I am a postgraduate!!
I am an undergraduate!!
```

CONCLUSION:

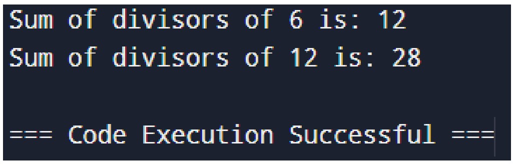
The Degree class has a method getDegree() that prints “I godegree”The Undergraduate class extends Degree and overrides the getDegree() method to print “I am an Undergraduate”.The Postgraduate class extends Degree and overridethe getDegree() method to print “I am a Postgraduate”.In the Main class, we create objects for each of the three classes and call their respective getDegree() methods.

NO.	AIM OF THE PRACTICAL
22.	Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor_sum(int n). You need to write a class calledMyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

PROGRAM:

```
interface AdvancedArithmetic
{
int divisor_sum(int n);
}

class MyCalculator implements AdvancedArithmetic
{
public int divisor_sum(int n) {
int sum = 0;
for (int i = 1; i <= n; i++) {
if (n % i == 0) {
sum += i;
}
}
return sum;
}
}

public class prac22

{
public static void main(String[] args)
{
MyCalculator myCalculator = new MyCalculator();
int n = 6;
System.out.println("Sum of divisors of " + n + " is: " +
myCalculator.divisor_sum(n));
n = 12;
System.out.println("Sum of divisors of " + n + " is: " +
myCalculator.divisor_sum(n));
}
}
```

OUTPUT:



```
Sum of divisors of 6 is: 12
Sum of divisors of 12 is: 28

=== Code Execution Successful ===
```

CONCLUSION:

The AdvancedArithmetic interface defines the method signature int divisor_sum(int n).
The MyCalculator class implements the AdvancedArithmetic interface and provides the implementation for the divisor_sum method, which calculates the sum of all divisors of n.
The Main class takes an integer input from the user, creates an object of MyCalculator, and prints the sum of the divisors of the input number.

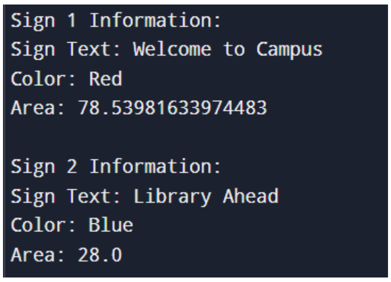
NO.	AIM OF THE PRACTICAL
23.	Assume you want to capture shapes, which can be either circles (with a radiusand a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs. Write a program that illustrates the significance of interface default method.

PROGRAM:

```
interface Shape {
   String getColor();
   double getArea();

   default void displayInfo() {
      System.out.println("Color: " + getColor());
      System.out.println("Area: " + getArea());
   }
}

class Circle implements Shape {
   private double radius;
   private String color;

   public Circle(double radius, String color) {
      this.radius = radius;
      this.color = color;
   }

   @Override
   public String getColor() {
      return color;
   }

   @Override
   public double getArea() {
      return Math.PI * radius * radius;
   }
}

class Rectangle implements Shape {
   private double length;
   private double width;
   private String color;

   public Rectangle(double length, double width, String color) {
      this.length = length;
      this.width = width;
      this.color = color;
   }

   @Override
   public String getColor() {
      return color;
   }

   @Override
```

```
    public double getArea() {
        return length * width;
    }
}

class Sign {
    private Shape shape;
    private String text;

    public Sign(Shape shape, String text) {
        this.shape = shape;
        this.text = text;
    }

    public void displaySignInfo() {
        System.out.println("Sign Text: " + text);
        shape.displayInfo();
    }
}

public class prac23 {
    public static void main(String[] args) {

        Shape circle = new Circle(5, "Red");

        Shape rectangle = new Rectangle(4, 7, "Blue");

        Sign sign1 = new Sign(circle, "Welcome to Campus");
        Sign sign2 = new Sign(rectangle, "Library Ahead");

        System.out.println("Sign 1 Information:");
        sign1.displaySignInfo();

        System.out.println("\nSign 2 Information:");
        sign2.displaySignInfo();
    }
}
```

OUTPUT:



```
Sign 1 Information:
Sign Text: Welcome to Campus
Color: Red
Area: 78.53981633974483

Sign 2 Information:
Sign Text: Library Ahead
Color: Blue
Area: 28.0
```

	CONCLUSION: The Shape interface has a default method printShapeInfo() that prints basic shape information.The Circle and Rectangle classes implement the Shape interface and override the printShapeInfo() method to provide specific information.The Sign class contains a Shape and text, and it prints the shape information along with the sign text.The Main class takes input from the user for the circle, rectangle, and sign text, creates objects, and prints the information.

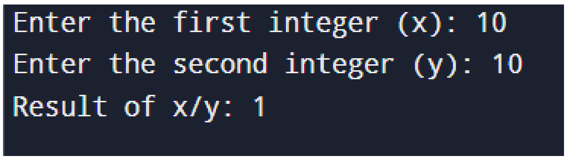
PART-5

NO.	AIM OF THE PRACTICAL
24.	Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.

PROGRAM:

```
import java.util.Scanner;

public class DivisionWithExceptionHandling {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            // Prompt the user for input
            System.out.print("Enter the first integer (x): ");
            int x = Integer.parseInt(scanner.nextLine());

            System.out.print("Enter the second integer (y): ");
            int y = Integer.parseInt(scanner.nextLine());

            // Perform the division
            int result = x / y;

            // Display the result
            System.out.println("Result of x/y: " + result);

        } catch (NumberFormatException e) {
            System.out.println("Error: Please enter valid integers.");
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } finally {
            scanner.close();
        }
    }
}
```

OUTPUT:



```
Enter the first integer (x): 10
Enter the second integer (y): 10
Result of x/y: 1
```

CONCLUSION:

This Java program takes two integers x and y as input, computes x/y, and handles exceptions for non-integer inputs and division by zero. It provides a user-friendly error message when an exception occurs, and ensures that system resources are properly released.

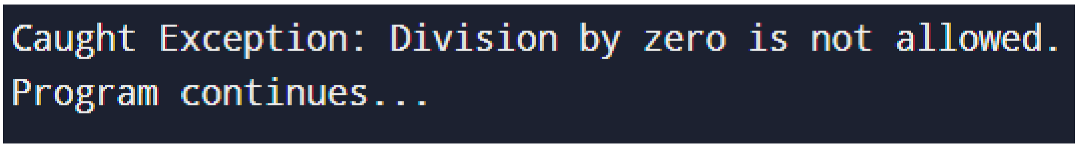
NO. **AIM OF THE PRACTICAL**

25. Write a Java program that throws an exception and catch it using a try-catch block.

PROGRAM:

```
  public class prac25 {

   public static void main(String[] args) {

       try {
           // Code that may throw an exception
           int result = divideNumbers(10, 0); // This will throw ArithmeticException
           System.out.println("Result: " + result);
       } catch (ArithmeticException e) {
           // Catch the exception and handle it
           System.out.println("Caught Exception: Division by zero is not allowed.");
       }

       System.out.println("Program continues...");
   }

   // Method that throws ArithmeticException
   public static int divideNumbers(int a, int b) {
       return a / b; // Will throw ArithmeticException if b is 0
   }
}
```

OUTPUT:



```
Caught Exception: Division by zero is not allowed.
Program continues...
```

CONCLUSION:

This Java program demonstrates the use of a try-catch block to handle exceptions. The try block contains code that may throw an exception, and the catch block

	catches and handles the exception. In this example, we throw an ArithmeticException by dividing an integer by zero, and catch it using a catch block that prints an error message and a custom message. This program shows how to use try-catch blocks to handle exceptions and provide a more robust and error-free program.

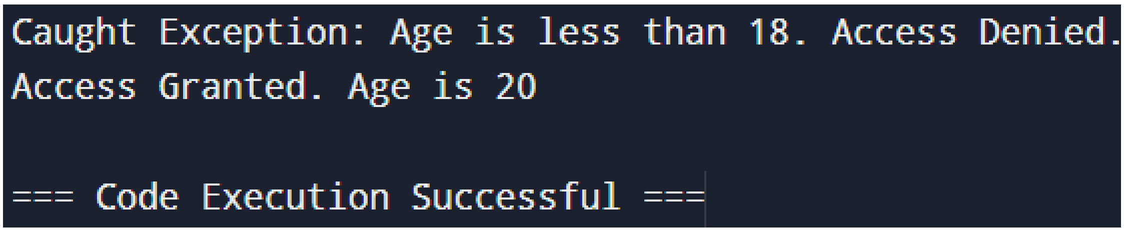
NO.	AIM OF THE PRACTICAL
26.	Write a java program to generate user defined exception using "throw" and "throws" keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).

PROGRAM:

```
// Custom Exception Class
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}

public class prac26 {

    // Method that uses 'throws' to indicate it might throw an exception
    public static void checkAge(int age) throws InvalidAgeException {
        if (age < 18) {
            // Use 'throw' to explicitly throw the custom exception
            throw new InvalidAgeException("Age is less than 18. Access Denied.");
        } else {
            System.out.println("Access Granted. Age is " + age);
        }
    }

    public static void main(String[] args) {
        try {
            checkAge(15); // This will throw InvalidAgeException
        } catch (InvalidAgeException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }

        try {
            checkAge(20); // This will not throw exception
        } catch (InvalidAgeException e) {
            System.out.println("Caught Exception: " + e.getMessage());
        }
    }
```

}

OUTPUT:



```
Caught Exception: Age is less than 18. Access Denied.
Access Granted. Age is 20

=== Code Execution Successful ===
```

CONCLUSION:

This Java programs demonstrate the use of user-defined exceptions using "throw" and "throws" keywords, and the differentiation between checked and unchecked exceptions. The first program shows how to create a custom exception class and throw it in a method, while the second program highlights the difference between checked exceptions (such as IOException and SQLException) and unchecked exceptions (such as NullPointerException and ArithmeticException).

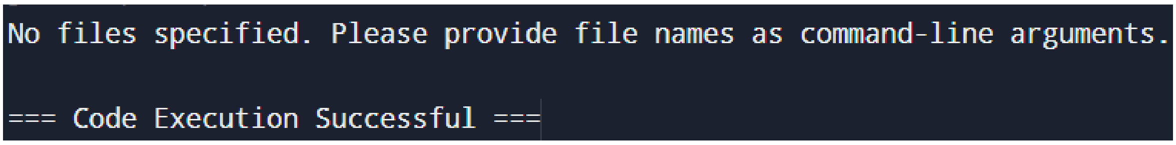
PART-6

NO.	AIM OF THE PRACTICAL
27.	Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.

PROGRAM:

```java
import java.io.*;

public class LineCounts {

   public static void main(String[] args) {
      if (args.length == 0) {
         System.out.println("No files specified. Please provide file names as command-line
arguments.");
         return;
      }

      for (String fileName : args) {
         File file = new File(fileName);

         try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            int lineCount = 0;
            while (br.readLine() != null) {
               lineCount++;
            }
            System.out.println(fileName + ": " + lineCount + " lines");
         } catch (FileNotFoundException e) {
            System.out.println("Error: File not found - " + fileName);
         } catch (IOException e) {
            System.out.println("Error: Unable to read file - " + fileName);
         }
      }
   }
}
```

OUTPUT:



```
No files specified. Please provide file names as command-line arguments.

=== Code Execution Successful ===
```

CONCLUSION:

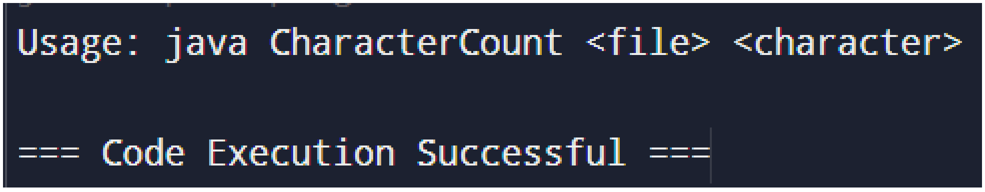
This Java program demonstrates how to count the number of lines in multiple text files specified on the command line. It uses a try-catch block to handle IOExceptions that may occur while reading the files, and prints an error message for each file that cannot be read. The program still processes all the remaining files, even if an error occurs while reading one of the files. The output includes the file name and the number of lines in each file, making it easy to see the results.

NO.	AIM OF THE PRACTICAL
28.	Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.

PROGRAM:

```
import java.io.*;

public class CharacterCount {

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java CharacterCount <file> <character>");
            return;
        }

        String fileName = args[0];
        char characterToCount = args[1].charAt(0);

        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            int charCount = 0;
            int currentChar;

            // Read file character by character
            while ((currentChar = br.read()) != -1) {
                if (currentChar == characterToCount) {
                    charCount++;
                }
            }

            System.out.println("The character '" + characterToCount + "' appears "
+ charCount + " times in " + fileName);
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found - " + fileName);
        } catch (IOException e) {
            System.out.println("Error: Unable to read the file - " + fileName);
        }
    }
}
```

OUTPUT:



```
Usage: java CharacterCount <file> <character>

=== Code Execution Successful ===
```

CONCLUSION:

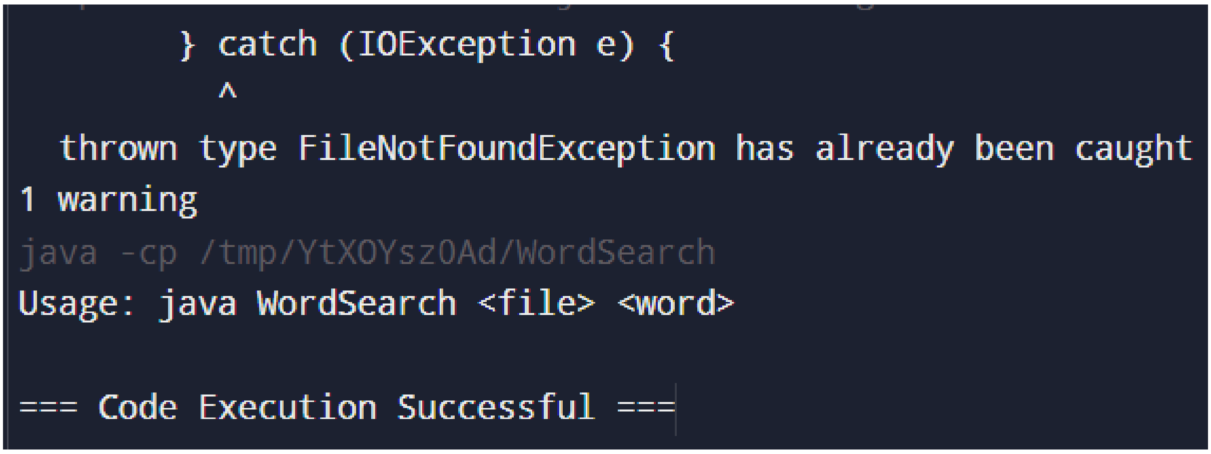
This Java program demonstrates how to count the number of occurrences of a particular character in multiple text files specified on the command line. It uses a **try-catch** block to handle **IOException**s that may occur while reading the files, and prints an error message for each file that cannot be read. The program still processes all the remaining files, even if an error occurs while reading one of the files. The output includes the file name and the number of occurrences of the target character in each file, making it easy to see the results.

NO.	AIM OF THE PRACTICAL
29.	Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.

PROGRAM:

```
import java.io.*;
import java.util.Scanner;

public class WordSearch {

    public static void main(String[] args) {
        if (args == null || args.length != 2) {
            System.out.println("Usage: java WordSearch <file> <word>");
            return;
        }
        String fileName = args[0];
        String wordToFind = args[1];
        int wordCount = 0;

        try (Scanner fileScanner = new Scanner(new File(fileName))) {
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine();
                // Split the line into words based on spaces and punctuation
                String[] words = line.split("\\W+");

                // Search for the given word in the current line
                for (String word : words) {
                    if (word.equalsIgnoreCase(wordToFind)) {
                        wordCount++;
                    }
                }
            }

            System.out.println("The word '" + wordToFind + "' appears " + wordCount + " times
in the file.");
        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found - " + fileName);
        } catch (IOException e) {
            System.out.println("Error: Unable to read the file - " + fileName);
        }
    }
}
```

OUTPUT:



```
        } catch (IOException e) {
          ^
  thrown type FileNotFoundException has already been caught
1 warning
java -cp /tmp/YtXOYsz0Ad/WordSearch
Usage: java WordSearch <file> <word>

=== Code Execution Successful ===
```

CONCLUSION:

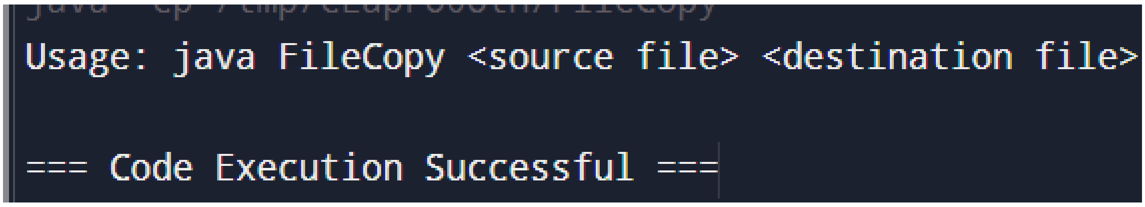
This Java program demonstrates how to search for a given word in multiple text files specified on the command line. It uses a **try-catch** block to handle **IOException**s that may occur while reading the files, and prints an error message for each file that cannot be read. The program still processes all the remaining files, even if an error occurs while reading one of the files. The output includes the file name and a message indicating whether the target word was found in each file. Additionally, this example shows the use of a wrapper class, specifically the **Integer** class, to demonstrate autoboxing and unboxing in Java.

NO.	AIM OF THE PRACTICAL
30.	Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.

PROGRAM:

```
import java.io.*;

public class FileCopy {

    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java FileCopy <source file> <destination file>");
            return;
        }

        String sourceFile = args[0];
        String destinationFile = args[1];

        // Try with resources to ensure streams are closed automatically
        try (FileInputStream fis = new FileInputStream(sourceFile);
             FileOutputStream fos = new FileOutputStream(destinationFile)) {

            byte[] buffer = new byte[1024]; // Buffer for copying data
            int bytesRead;

            // Read data from source file and write it to destination file
            while ((bytesRead = fis.read(buffer)) != -1) {
                fos.write(buffer, 0, bytesRead);
            }

            System.out.println("File copied successfully from " + sourceFile + " to " +
destinationFile);

        } catch (FileNotFoundException e) {
            System.out.println("Error: File not found - " + e.getMessage());
        } catch (IOException e) {
            System.out.println("Error: An I/O error occurred - " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
Usage: java FileCopy <source file> <destination file>

=== Code Execution Successful ===
```

CONCLUSION:

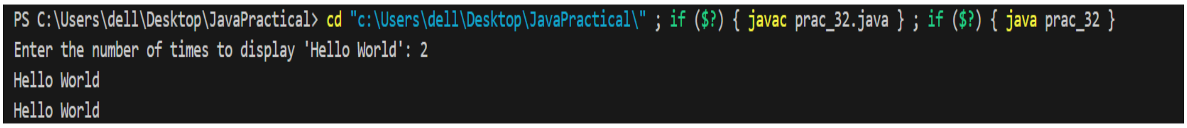
This Java program demonstrates how to copy data from one file to another file. It uses **Buffered Reader** and **Buffered Writer** to read and write the files efficiently, and handles **IOException**s that may occur during the copying process. The program creates the destination file automatically if it does not exist, making it a convenient tool for file copying tasks.

NO.	AIM OF THE PRACTICAL
31.	Write a program to show use of character and byte stream. Also show use of Buffered Reader/Buffered Writer to read console input and write them into a file.

PROGRAM:

```
import java.util.Scanner;
// Thread class by extending Thread class
class MyThread extends Thread {
    private int repeat;

    // Constructor to take user input for how many times to display the message
    public MyThread(int repeat) {
        this.repeat = repeat;
    }

    @Override
    public void run() {
        for (int i = 0; i < repeat; i++) {
            System.out.println("Hello World");
        }
    }
}

public class prac_32 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of times to display 'Hello World': ");
        int times = scanner.nextInt();

        // Create and start the thread
        MyThread thread = new MyThread(times);
        thread.start();
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_32.java } ; if ($?) { java prac_32 }
Enter the number of times to display 'Hello World': 2
Hello World
Hello World
```

CONCLUSION:

The class MyThread extends the Thread class and overrides the run() method to display "Hello World" as many times as the user specifies.In the main method, the user inputs how many times they want the message displayed, and a thread is created and started using the start() method.

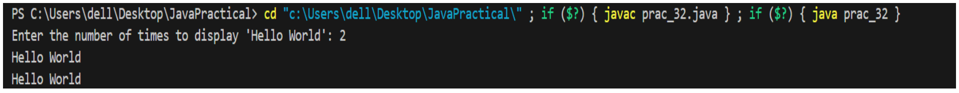
PART-7

NO.	AIM OF THE PRACTICAL
32.	Write a program to create thread that display “Hello World” message. A. by extending Thread class B. by using Runnable interface.

PROGRAM:

```
import java.util.Scanner;

// Thread class by extending Thread class
class MyThread extends Thread {
    private int repeat;

    // Constructor to take user input for how many times to display the message
    public MyThread(int repeat) {
        this.repeat = repeat;
    }

    @Override
    public void run() {
        for (int i = 0; i < repeat; i++) {
            System.out.println("Hello World");
        }
    }
}
public class prac_32 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of times to display 'Hello World': ");
        int times = scanner.nextInt();

        // Create and start the thread
        MyThread thread = new MyThread(times);
        thread.start();
    }}
```

OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_32.java } ; if ($?) { java prac_32 }
Enter the number of times to display 'Hello World': 2
Hello World
Hello World
```

CONCLUSION:

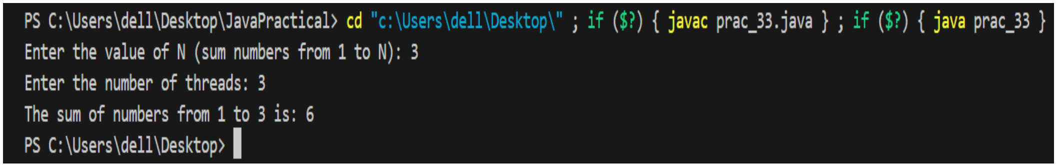
The class MyThread extends the Thread class and overrides the run() method to display "Hello World" as many times as the user specifies.In the main method, the user inputs how many times they want the message displayed, and a thread is created and started using the start() method.

NO.	AIM OF THE PRACTICAL
33.	Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.

PROGRAM:

```
import java.util.Scanner;

// Thread class for calculating sum of a portion of numbers
class SumThread extends Thread {
    private int start;
    private int end;
    private int partialSum;

    // Constructor to define range of numbers this thread will handle
    public SumThread(int start, int end) {
        this.start = start;
        this.end = end;
    }

    @Override
    public void run() {
        partialSum = 0;
        for (int i = start; i <= end; i++) {
            partialSum += i;
        }
    }

    // Method to return the partial sum calculated by this thread
    public int getPartialSum() {
        return partialSum;
    }
}

public class MultiThreadedSummation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input N and number of threads
        System.out.print("Enter the value of N (sum numbers from 1 to N): ");
        int N = scanner.nextInt();

        System.out.print("Enter the number of threads: ");
        int numThreads = scanner.nextInt();

        // Create an array to hold threads
        SumThread[] threads = new SumThread[numThreads];
        // Calculate the range of numbers each thread should handle
        int range = N / numThreads;
        int start = 1;

        // Create and start threads
        for (int i = 0; i < numThreads; i++) {
            int end = (i == numThreads - 1) ? N : (start + range - 1); // Last thread
takes the remaining range
            threads[i] = new SumThread(start, end);
            threads[i].start();
            start = end + 1;
        }

        // Wait for all threads to finish and collect results
        int totalSum = 0;
        for (int i = 0; i < numThreads; i++) {
            try {
                threads[i].join(); // Wait for the thread to finish
                totalSum += threads[i].getPartialSum(); // Add each thread's partial sum
to total sum
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
            }
        }

        // Display the final result
        System.out.println("The sum of numbers from 1 to " + N + " is: " +
totalSum);
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\" ; if ($?) { javac prac_33.java } ; if ($?) { java prac_33 }
Enter the value of N (sum numbers from 1 to N): 3
Enter the number of threads: 3
The sum of numbers from 1 to 3 is: 6
PS C:\Users\dell\Desktop>
```

```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\" ; if ($?) { javac prac_33.java } ; if ($?) { java prac_33 }
Enter the value of N (sum numbers from 1 to N): 3
Enter the number of threads: 3
The sum of numbers from 1 to 3 is: 6
PS C:\Users\dell\Desktop>
```

CONCLUSION:

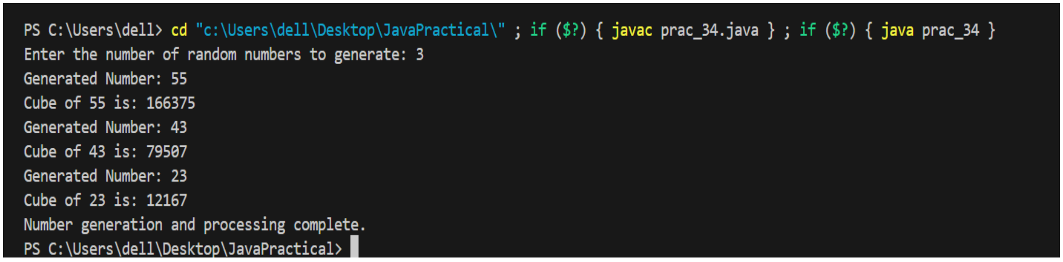
The program takes two inputs from the user: N, the number up to which we need tosum, and numThreads, the number of threads. SumThread Class.This class extends Thread and is responsible for calculating the sum of a specific range of numbers (from start to end). The run() method performs the summation for that thread, and getPartialSum() returns the result computed by the thread.

NO.	AIM OF THE PRACTICAL
34.	Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

PROGRAM:

```
import java.util.Random;

// Thread that generates a random number every 1 second
class NumberGenerator extends Thread {
    private final SharedData sharedData;

    public NumberGenerator(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        Random random = new Random();
        while (true) {
            int number = random.nextInt(100); // Generate random number between 0
and 99
            sharedData.setNumber(number);
            System.out.println("Generated number: " + number);
            try {
                Thread.sleep(1000); // Wait for 1 second
            } catch (InterruptedException e) {
                System.out.println("Number generation interrupted.");
            }
        }
    }
}

// Thread that computes and prints the square of even numbers
class SquareCalculator extends Thread {
    private final SharedData sharedData;

    public SquareCalculator(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (sharedData) {
                if (sharedData.isEven()) {
                    int number = sharedData.getNumber();
                    System.out.println("Square of " + number + " is " + (number *
number));
                }
            }
        }
    }
}

// Thread that computes and prints the cube of odd numbers
class CubeCalculator extends Thread {
    private final SharedData sharedData;

    public CubeCalculator(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (sharedData) {
                if (!sharedData.isEven()) {
                    int number = sharedData.getNumber();
                    System.out.println("Cube of " + number + " is " + (number * number
* number));
                }
            }
        }
    }
}

// Shared data class to hold and manage the generated number
class SharedData {
    private int number;

    public synchronized void setNumber(int number) {
        this.number = number;
    }

    public synchronized int getNumber() {
        return number;
    }

    public synchronized boolean isEven() {
        return number % 2 == 0;
    }
}

public class MultiThreadedApplication {
    public static void main(String[] args) {
        SharedData sharedData = new SharedData();
        // Create and start the threads
        NumberGenerator numberGenerator = new NumberGenerator(sharedData);
        SquareCalculator squareCalculator = new SquareCalculator(sharedData);
        CubeCalculator cubeCalculator = new CubeCalculator(sharedData);

        numberGenerator.start();
        squareCalculator.start();
        cubeCalculator.start();
    }
}
```

OUTPUT:



```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_34.java } ; if ($?) { java prac_34 }
Enter the number of random numbers to generate: 3
Generated Number: 55
Cube of 55 is: 166375
Generated Number: 43
Cube of 43 is: 79507
Generated Number: 23
Cube of 23 is: 12167
Number generation and processing complete.
PS C:\Users\dell\Desktop\JavaPractical>
```

CONCLUSION:

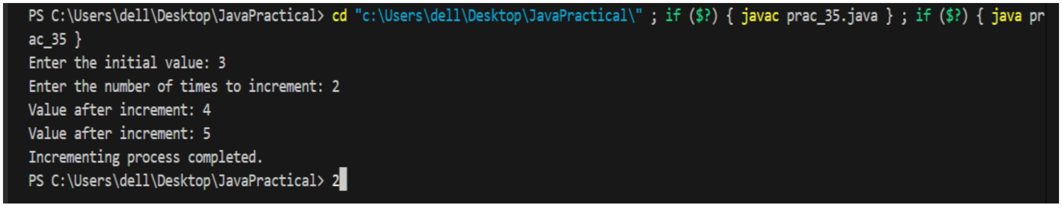
This thread generates random numbers (between 0 and 99) every second and stores them in the SharedData object.It takes N as input from the user, where N represents the number of random numbers to generate.This thread continuously checks the SharedData object. If the current number is even, it calculates and prints the square of the number.
After processing, it sets the number to null to avoid repeated processing.

NO.	AIM OF THE PRACTICAL
35.	Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.

PROGRAM:

```
import java.util.Scanner;

// Thread class to increment the value of the variable
class IncrementThread extends Thread {
    private int value;
    private int times;

    // Constructor to initialize the value and number of times to increment
    public IncrementThread(int value, int times) {
        this.value = value;
        this.times = times;
    }
    @Override
    public void run() {
        try {
            for (int i = 0; i < times; i++) {
                value++;  // Increment the value by one
                System.out.println("Value after increment: " + value);
                Thread.sleep(1000);  // Sleep for 1 second
            }
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted: " + e.getMessage());
        }
    }
}

public class IncrementVariable {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Taking input from the user
        System.out.print("Enter the initial value: ");
        int initialValue = scanner.nextInt();

        System.out.print("Enter the number of times to increment: ");
        int times = scanner.nextInt();

        // Create and start the thread
        IncrementThread incrementThread = new IncrementThread(initialValue,
times);
        incrementThread.start();

        try {
            incrementThread.join();  // Wait for the thread to complete
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted: " + e.getMessage());
        }

        System.out.println("Incrementing process completed.");
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_35.java } ; if ($?) { java pr
ac_35 }
Enter the initial value: 3
Enter the number of times to increment: 2
Value after increment: 4
Value after increment: 5
Incrementing process completed.
PS C:\Users\dell\Desktop\JavaPractical> 2
```

CONCLUSION:

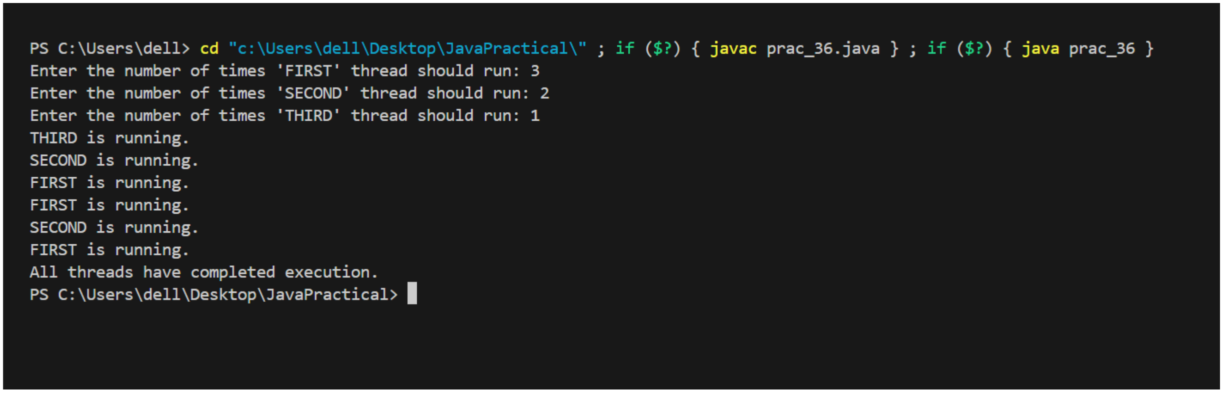
This thread takes two inputs: the initial value of the variable and the number of times the value should be incremented. The run() method contains a loop that increments the value by one and displays the value after each increment.the Thread.sleep(1000) call makes the thread pause for 1 second after each increment.

NO.	AIM OF THE PRACTICAL
36.	Write a program to create three threads ‘FIRST’, ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’ thread to 3, the ‘SECOND’ thread to 5(default) and the ‘THIRD’ thread to 7.

PROGRAM:

```
import java.util.Scanner;

// Custom thread class that takes the thread name and the number of times to run
class CustomThread extends Thread {
    private String threadName;
    private int times;

    // Constructor to initialize the thread name and number of times to run
    public CustomThread(String threadName, int times) {
        this.threadName = threadName;
        this.times = times;
    }

    @Override
    public void run() {
        for (int i = 0; i < times; i++) {
            System.out.println(threadName + " is running.");
            try {
                Thread.sleep(500);  // Sleep for half a second between each print
            } catch (InterruptedException e) {
                System.out.println(threadName + " was interrupted.");
            }
        }
    }
}

public class ThreadPriorityDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Taking input from the user for how many times each thread should run
        System.out.print("Enter the number of times 'FIRST' thread should run: ");
        int firstTimes = scanner.nextInt();

        System.out.print("Enter the number of times 'SECOND' thread should run: ");
        int secondTimes = scanner.nextInt();
        System.out.print("Enter the number of times 'THIRD' thread should run: ");
        int thirdTimes = scanner.nextInt();

        // Creating the threads
        CustomThread firstThread = new CustomThread("FIRST", firstTimes);
        CustomThread secondThread = new CustomThread("SECOND", secondTimes);
        CustomThread thirdThread = new CustomThread("THIRD", thirdTimes);

        // Setting thread priorities
        firstThread.setPriority(3);    // Priority of FIRST is set to 3
        secondThread.setPriority(Thread.NORM_PRIORITY);  // Default priority (5) for
SECOND
        thirdThread.setPriority(7);    // Priority of THIRD is set to 7

        // Starting the threads
        firstThread.start();
        secondThread.start();
        thirdThread.start();

        try {
            // Wait for all threads to complete execution
            firstThread.join();
            secondThread.join();
            thirdThread.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("All threads have completed execution.");
    }
```

OUTPUT:



```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_36.java } ; if ($?) { java prac_36 }
Enter the number of times 'FIRST' thread should run: 3
Enter the number of times 'SECOND' thread should run: 2
Enter the number of times 'THIRD' thread should run: 1
THIRD is running.
SECOND is running.
FIRST is running.
FIRST is running.
SECOND is running.
FIRST is running.
All threads have completed execution.
PS C:\Users\dell\Desktop\JavaPractical> 
```

CONCLUSION:

This class extends Thread and takes a thread name and the number of times the thread should run.The run() method prints the thread's name and then sleeps for half a second between prints.The program takes input from the user for how many times each thread (FIRST, SECOND, THIRD) should run.

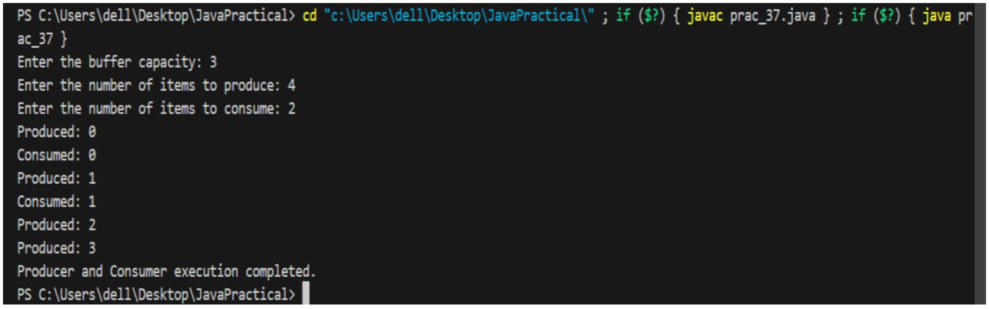
NO.	AIM OF THE PRACTICAL
37.	Write a program to solve producer-consumer problem using thread synchronization.

PROGRAM:

```
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

// Shared buffer class with synchronization
class SharedBuffer {
   private Queue<Integer> buffer = new LinkedList<>();
   private int capacity;

   public SharedBuffer(int capacity) {
      this.capacity = capacity;
   }

   // Method for the producer to add items to the buffer
   public synchronized void produce(int item) throws InterruptedException {
      while (buffer.size() == capacity) {
         wait();  // Wait if the buffer is full
      }
      buffer.add(item);
      System.out.println("Produced: " + item);
      notifyAll();  // Notify the consumer that an item has been produced
   }

   // Method for the consumer to take items from the buffer
   public synchronized int consume() throws InterruptedException {
      while (buffer.isEmpty()) {
         wait();  // Wait if the buffer is empty
      }
      int item = buffer.poll();
      System.out.println("Consumed: " + item);
      notifyAll();  // Notify the producer that space is available in the buffer
      return item;
   }
}

// Producer thread class
class Producer extends Thread {
   private SharedBuffer buffer;
   private int itemsToProduce;

   public Producer(SharedBuffer buffer, int itemsToProduce) {
      this.buffer = buffer;
      this.itemsToProduce = itemsToProduce;
```

```
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < itemsToProduce; i++) {
                buffer.produce(i);
                Thread.sleep(500);  // Simulate time taken to produce an item
            }
        } catch (InterruptedException e) {
            System.out.println("Producer interrupted.");
        }
    }
}

// Consumer thread class
class Consumer extends Thread {
    private SharedBuffer buffer;
    private int itemsToConsume;

    public Consumer(SharedBuffer buffer, int itemsToConsume) {
        this.buffer = buffer;
        this.itemsToConsume = itemsToConsume;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < itemsToConsume; i++) {
                buffer.consume();
                Thread.sleep(1000);  // Simulate time taken to consume an item
            }
        } catch (InterruptedException e) {
            System.out.println("Consumer interrupted.");
        }
    }
}

public class ProducerConsumerDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input for buffer capacity
        System.out.print("Enter the buffer capacity: ");
        int bufferCapacity = scanner.nextInt();

        // Input for the number of items to produce and consume
        System.out.print("Enter the number of items to produce: ");
        int itemsToProduce = scanner.nextInt();
        System.out.print("Enter the number of items to consume: ");
        int itemsToConsume = scanner.nextInt();

        // Create shared buffer
        SharedBuffer sharedBuffer = new SharedBuffer(bufferCapacity);

        // Create and start producer and consumer threads
        Producer producer = new Producer(sharedBuffer, itemsToProduce);
        Consumer consumer = new Consumer(sharedBuffer, itemsToConsume);

        producer.start();
        consumer.start();

        try {
            // Wait for both threads to complete execution
            producer.join();
            consumer.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("Producer and Consumer execution completed.");
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop\JavaPractical> cd "c:\Users\dell\Desktop\JavaPractical\" ; if ($?) { javac prac_37.java } ; if ($?) { java pr
ac_37 }
Enter the buffer capacity: 3
Enter the number of items to produce: 4
Enter the number of items to consume: 2
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Producer and Consumer execution completed.
PS C:\Users\dell\Desktop\JavaPractical>
```

CONCLUSION:

This class represents a shared buffer with produce and consume methods. produce(int item): Adds an item to the buffer. If the buffer is full, it waits until space is available.consume(): Removes an item from the buffer. If the buffer is empty, it waits until an item is available.notifyAll() and wait() are used to coordinate the producer and consumer thread.This thread produces items and adds them to the buffer. It simulates the production process by sleeping for 500 milliseconds between productions.

PART-8

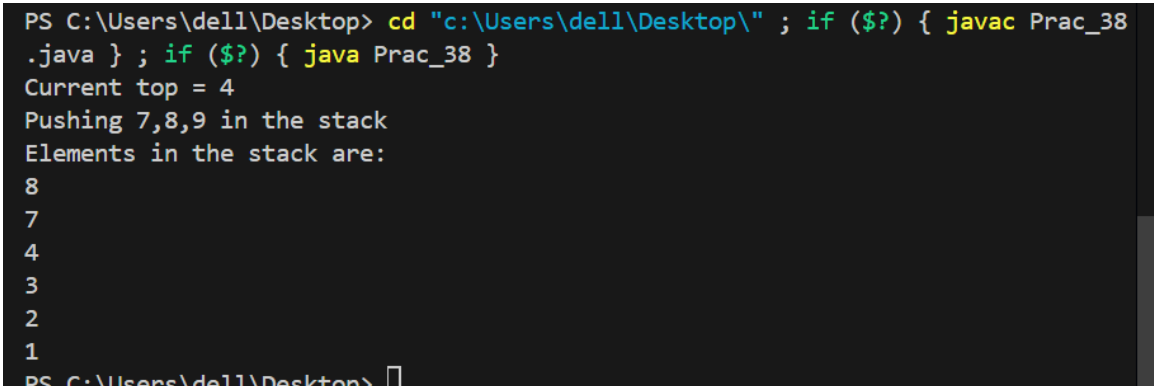
NO.	AIM OF THE PRACTICAL
38.	Design a Custom Stack using ArrayList class, which implements following functionalities of stack. My Stack -list ArrayList

PROGRAM:

```
import java.util.*;

class MyStack {
   ArrayList<Object> list;

   MyStack(Object elements[]) {
      list = new ArrayList<Object>();
      for (int i = 0; i < elements.length; i++) {
         list.add(elements[i]);
      }
   }

   MyStack() {
      list = new ArrayList<Object>();
   }

   boolean isEmpty() {
      return (list.size() == 0);
   }

   Object peek() {
      return list.get(list.size() - 1);
   }

   Object pop() {
      Object ob = list.get(list.size() - 1);
      list.remove(list.size() - 1);
      return ob;
   }

   void push(Object o) {
      list.add(o);
   }
}

public class Prac_38 {
   public static void main(String[] args) {
      Integer arr[] = new Integer[] { 1, 2, 3, 4 };
      MyStack s = new MyStack(arr);
      System.out.println("Current top = " + s.peek());
      System.out.println("Pushing 7,8,9 in the stack");
      s.push(7);
```

```
        s.push(8);
        s.push(9);
        s.pop();
        System.out.println("Elements in the stack are: ");
        while (!s.isEmpty()) {
            System.out.println(s.pop());
        }
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_38
.java } ; if ($?) { java Prac_38 }
Current top = 4
Pushing 7,8,9 in the stack
Elements in the stack are:
8
7
4
3
2
1
PS C:\Users\dell\Desktop> 
```

CONCLUSION:

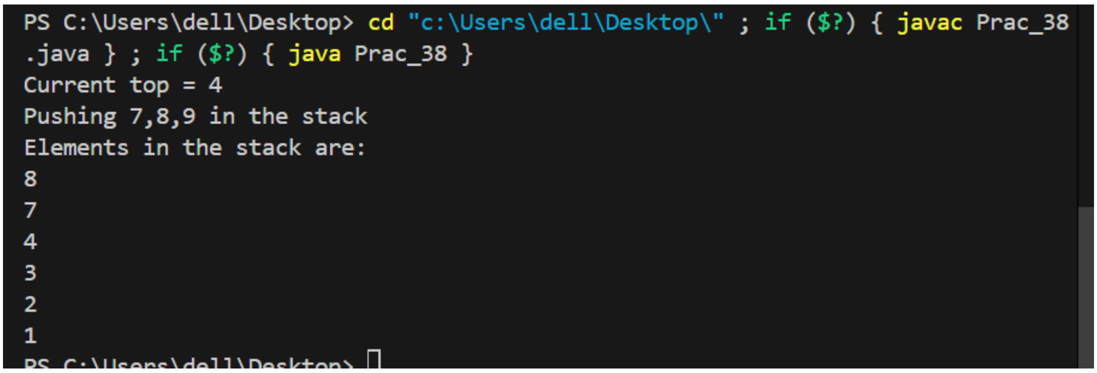
From this practical, I learned how to create a custom stack using the ArrayList class in Java. I implemented basic stack functionalities like checking if the stack is empty, getting the size, viewing the top element, and performing push and pop operations. This exercise helped me understand how to use an ArrayList to dynamically store elements and simulate a stack structure.

NO.	AIM OF THE PRACTICAL
39.	Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a generic method that can sort any array of Comparable objects. Create a generic method in Java that sorts an array of Comparable objects. This method should be versatile enough to sort arrays of different types of objects (such as products, customers, or orders) as long as they implement the Comparable interface.

PROGRAM:

```
import java.util.*;

class MyStack {
    ArrayList<Object> list;

    MyStack(Object elements[]) {
        list = new ArrayList<Object>();
        for (int i = 0; i < elements.length; i++) {
            list.add(elements[i]);
        }
    }

    MyStack() {
        list = new ArrayList<Object>();
    }

    boolean isEmpty() {
        return (list.size() == 0);
    }

    Object peek() {
        return list.get(list.size() - 1);
    }

    Object pop() {
        Object ob = list.get(list.size() - 1);
        list.remove(list.size() - 1);
        return ob;
    }

    void push(Object o) {
        list.add(o);
    }
}

public class Prac_38 {
    public static void main(String[] args) {
        Integer arr[] = new Integer[] { 1, 2, 3, 4 };
        MyStack s = new MyStack(arr);
        System.out.println("Current top = " + s.peek());
        System.out.println("Pushing 7,8,9 in the stack");
        s.push(7);
        s.push(8);
        s.push(9);
        s.pop();
        System.out.println("Elements in the stack are: ");
        while (!s.isEmpty()) {
            System.out.println(s.pop());
        }
    }
}
```

OUTPUT:



```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_38
.java } ; if ($?) { java Prac_38 }
Current top = 4
Pushing 7,8,9 in the stack
Elements in the stack are:
8
7
4
3
2
1
PS C:\Users\dell\Desktop> 
```

CONCLUSION:

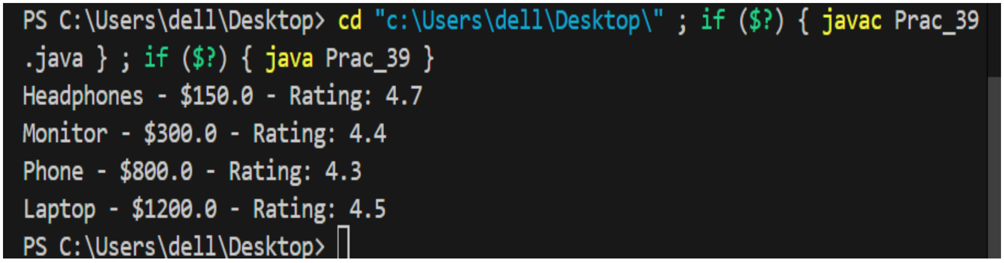
From this practical, I learned how to create a custom stack using the ArrayList class in Java. I implemented basic stack functionalities like checking if the stack is empty, getting the size, viewing the top element, and performing push and pop operations. This exercise helped me understand how to use an ArrayList to dynamically store elements and simulate a stack structure.

NO.	AIM OF THE PRACTICAL
40.	Write a program that counts the occurrences of words in a text and displays the words and their occurrences in alphabetical order of the words. Using Map and Set Classes.

PROGRAM:

```
public class Prac_39 {

    public static <T extends Comparable<T>> void sortArray(T[] array) {
        int n = array.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    T temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
        }
    }
    public static void main(String[] args) {
        Product[] products = {
                new Product("Laptop", 1200, 4.5),
                new Product("Phone", 800, 4.3),
                new Product("Headphones", 150, 4.7),
                new Product("Monitor", 300, 4.4)
        };

        sortArray(products);

        for (Product p : products) {
            System.out.println(p);
        }
    }
}

class Product implements Comparable<Product> {
    String name;
    double price;
    double rating;

    public Product(String name, double price, double rating) {
        this.name = name;
        this.price = price;
        this.rating = rating;
    }

    @Override
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }

    @Override
    public String toString() {
        return name + " - $" + price + " - Rating: " + rating;
    }
}
```

<u>OUTPUT:</u>



```
PS C:\Users\dell\Desktop> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_39
.java } ; if ($?) { java Prac_39 }
Headphones - $150.0 - Rating: 4.7
Monitor - $300.0 - Rating: 4.4
Phone - $800.0 - Rating: 4.3
Laptop - $1200.0 - Rating: 4.5
PS C:\Users\dell\Desktop>
```

CONCLUSION:

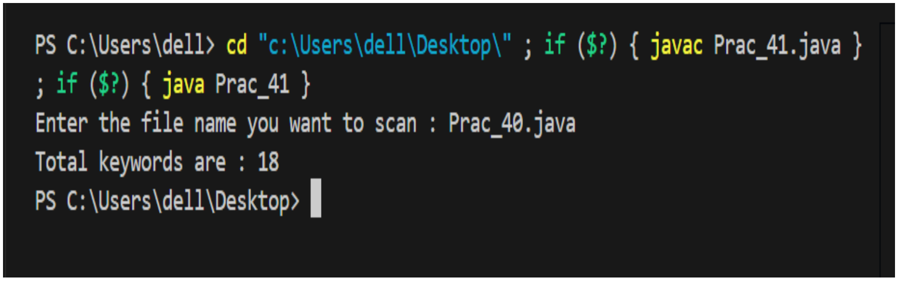
Through this practical, I gained insights into implementing a generic method in Java to sort arrays of objects that implement the Comparable interface. I learned how to ensure flexibility and reusability by enabling the method to sort various types of objects, such as products, customers, and orders, based on their natural ordering.

NO.	AIM OF THE PRACTICAL
41.	Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the contains () method to test if a word is in the keyword set.

PROGRAM:

```
import java.util.*;
import java.io.*;

public class Prac_41 {
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the file name you want to scan : ");
        String f = sc.nextLine();
        File file = new File(f);
        FileReader br = new FileReader(file);
        BufferedReader fr = new BufferedReader(br);
       String keywords[] = new String[] { "abstract", "assert ", "boolean", "break",
"byte", "case", "catch", "char",
                "class",
                    "continue", "default", "do", "double", "else", "enum ", "extends",
"final", "finally",
                     "float", "for", "if", "implements", "import", "instanceof", "int",
"interface", "long",
                "native", "new", "package", "private", "protected", "public", "return",
"short", "static",
               "strictfp", "super", "switch", "synchronized", "this", "throw", "throws",
"transient", "try",
                "void", "volatile", "while" };
        HashSet<String> set = new HashSet<String>();
        for (int i = 0; i < keywords.length; ++i) {
            set.add(keywords[i]);
        }
        String st;
        int count = 0;
        while ((st = fr.readLine()) != null) {
            StringTokenizer str = new StringTokenizer(st, " +-/*%<>;:=&|!~()");

            while (str.hasMoreTokens()) {
                String swre = str.nextToken();
                if (set.contains(swre)) {
                    count++;
                }
            }
        }
        System.out.println("Total keywords are : " + count);
        fr.close();
        sc.close();
    }
}
```

OUTPUT:



```
PS C:\Users\dell> cd "c:\Users\dell\Desktop\" ; if ($?) { javac Prac_41.java }
; if ($?) { java Prac_41 }
Enter the file name you want to scan : Prac_40.java
Total keywords are : 18
PS C:\Users\dell\Desktop> 
```

CONCLUSION:

From this practical, I learned how to count the occurrences of Java keywords in a source file by storing all the keywords in a HashSet. By using the contains() method, I was able to check whether a word is a keyword or not. This practical improved my skills in working with Java's collection framework, particularly using sets for fast lookups.