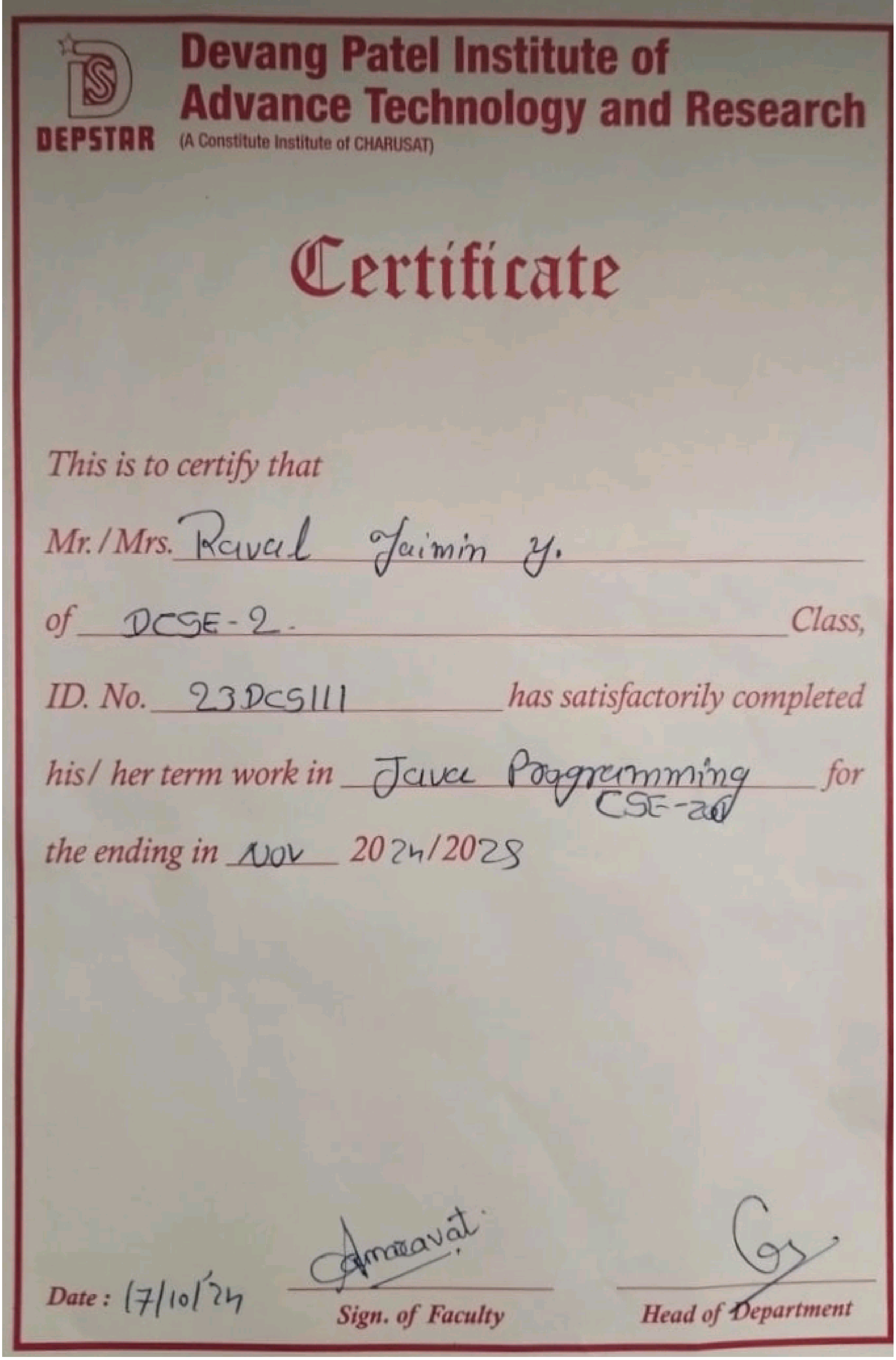


DEPSTAR

Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr. / Mrs. Raval Jaimin Y.

of DCSE-2. *Class,*

ID. No. 23DCS111 *has satisfactorily completed*

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

the ending in Nov 2024/2025

Date : 17/10/24

[illegible]

Sign. of Faculty

[illegible]

Head of Department

CHAROTAR UNIVERSITY OF SCIENCE TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: java programming

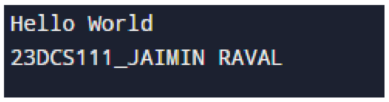
Semester: 3rd

Subject Code: CSE201

Academic year: 2024

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 CODE :
	```
	public class simple{
	    public static void main(String[] args) {
	        int a;
	        System.err.println("Hello World");
	        System.out.println("23DCS111_JAIMIN RAVAL");
	    }
	}
	```
	OUTPUT:
	
	```
	Hello World
	23DCS111_JAIMIN RAVAL
	```
	CONCLUSION:
	
	it’s a basic program to display Hello, World!.

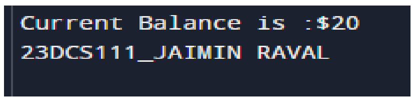
2.	Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's

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 CODE :

```
public class pra_2{
  public static void main(String[] args) {
     int a = 20;
     System.err.print("Current Balance is :$");
     System.err.println(a);
     System.out.println("23DCS111_JAIMIN RAVAL");
  }
}
```

OUTPUT:



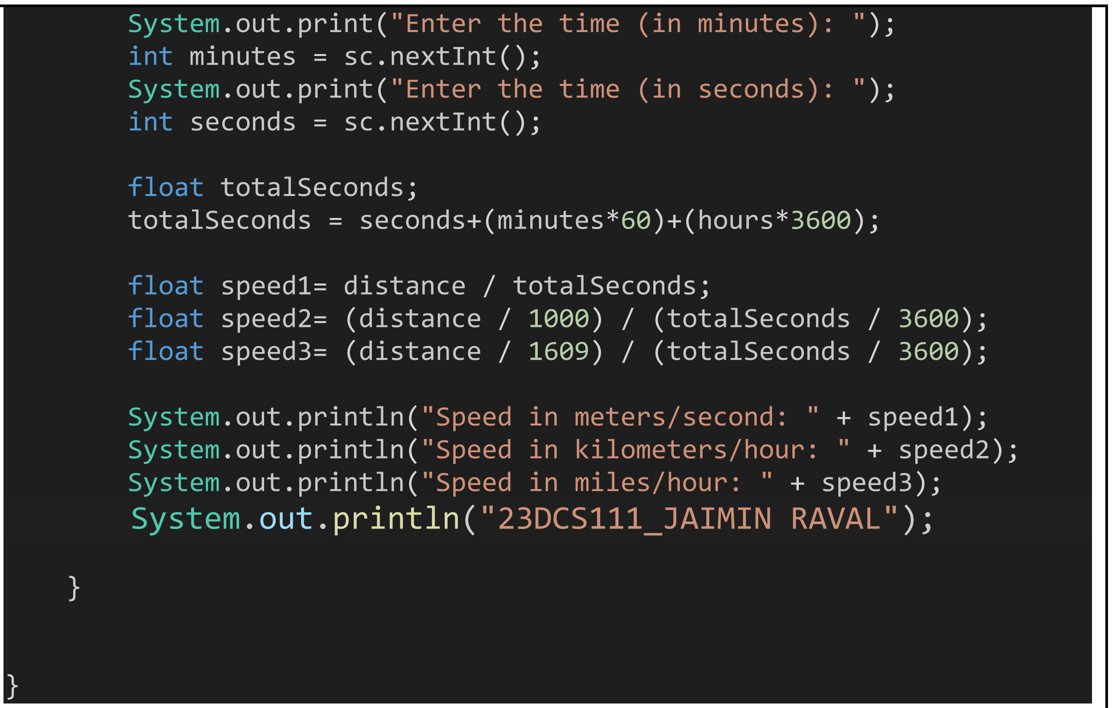
```
Current Balance is :$20
23DCS111_JAIMIN RAVAL
```

CONCLUSION:
In this example, we created a basic Java program that simulates a banking application by storing a user's account balance in a variable and then displaying it. This demonstrates the fundamental concept of variable storage and output in Java, which is essential for developing more complex banking systems.

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).

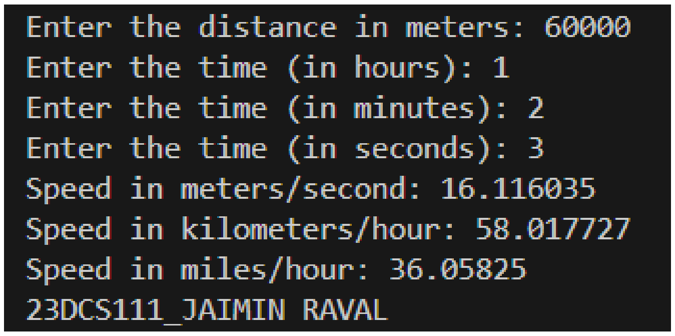
PROGRAM CODE :

```
import java.util.*;

public class Pra3 {

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

        System.out.print("Enter the distance in meters: ");
        float distance = sc.nextFloat();
        System.out.print("Enter the time (in hours): ");
        int hours = sc.nextInt();
```



```
        System.out.print("Enter the time (in minutes): ");
        int minutes = sc.nextInt();
        System.out.print("Enter the time (in seconds): ");
        int seconds = sc.nextInt();

        float totalSeconds;
        totalSeconds = seconds+(minutes*60)+(hours*3600);

        float speed1= distance / totalSeconds;
        float speed2= (distance / 1000) / (totalSeconds / 3600);
        float speed3= (distance / 1609) / (totalSeconds / 3600);

        System.out.println("Speed in meters/second: " + speed1);
        System.out.println("Speed in kilometers/hour: " + speed2);
        System.out.println("Speed in miles/hour: " + speed3);
        System.out.println("23DCS111_JAIMIN RAVAL");

    }


}
```

<u>OUTPUT:</u>



```
Enter the distance in meters: 60000
Enter the time (in hours): 1
Enter the time (in minutes): 2
Enter the time (in seconds): 3
Speed in meters/second: 16.116035
Speed in kilometers/hour: 58.017727
Speed in miles/hour: 36.05825
23DCS111_JAIMIN RAVAL
```

<u>CONCLUSION:</u>

In this program, we used Java's Scanner class to take user input for distance in meters and time in hours, minutes, and seconds. We then converted the time to total seconds to simplify the calculations. The program calculates and displays the speed in three different units: meters per second, kilometers per hour, and miles per hour. This example demonstrates how to handle user input, perform unit conversions, and output results in Java.

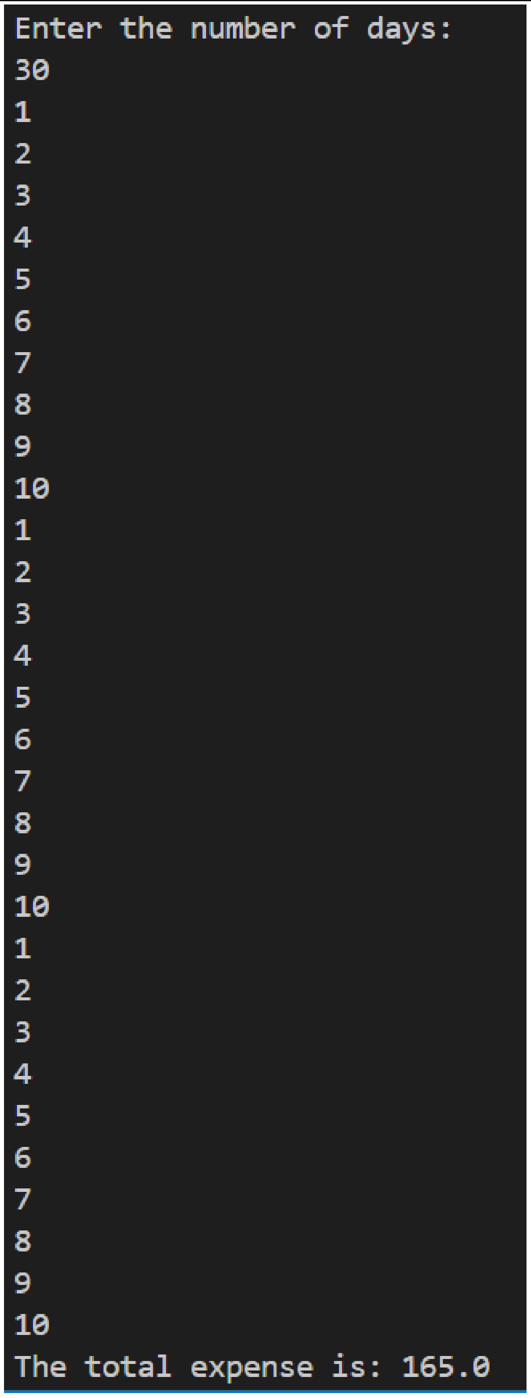
4.

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 CODE :

```java
import java.util.*;

public class pra_4 {
    public static void main(String[] args) {
        float sum = 0;
        int n;
        int[] arr = new int[30];
        System.out.println("Enter the number of days:");
        Scanner b = new Scanner(System.in);
        n = b.nextInt();
        for (int i = 0; i < n; i++) {
            arr[i] = b.nextInt();
            sum = sum + arr[i];
        }
        System.out.println("The total expense is: " + sum);
       System.out.println("23DCS111_JAIMIN RAVAL");
    }
}
```

OUTPUT:



```
Enter the number of days:
30
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
The total expense is: 165.0
```

CONCLUSION:

This Java program provides a simple yet effective way to track and calculate monthly expenses based on daily inputs from the user. It demonstrates how to use arrays to store data, take user input through a loop, and perform a summation of array elements.

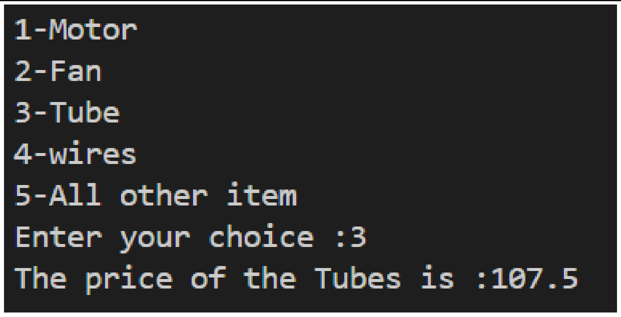
5.

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 CODE :

```java
import java.util.Scanner;

public class pra5 {
    public static void main(String[] args) {
        int a;
       // int code[] = {1,2,3,4,5};
        int price[] = {200,550,100,170,220};
        System.out.println("1-Motor\n2-Fan\n3-Tube\n4-wires\n5-All
other item");
        System.out.print("Enter your choice :");
        Scanner b = new Scanner(System.in);
        a = b.nextInt();

        switch (a) {
            case 1 -> System.out.println("The price of the Motor is
:"+(price[0]+price[0]*0.08f));

            case 2 -> System.out.println("The price of the Fan is
:"+(price[1]+price[1]*0.12f));
            case 3 -> System.out.println("The price of the Tubes is
:"+(price[2]+price[2]*0.075f));
            case 4 -> System.out.println("The price of the wires is
:"+(price[3]+price[3]*0.03f));
             default -> System.out.println("Invalid choice");
        }
    }
}
```

OUTPUT:



```
1-Motor
2-Fan
3-Tube
4-wires
5-All other item
Enter your choice :3
The price of the Tubes is :107.5
```

CONCLUSION:

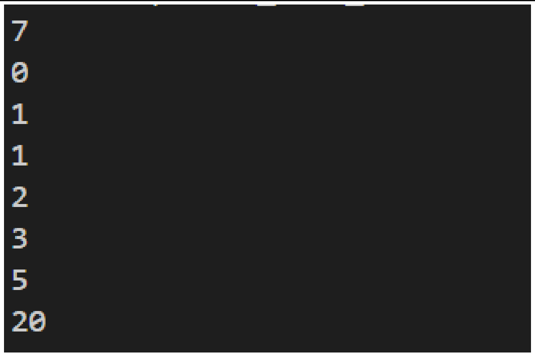
This practical exercise reinforced the fundamentals of array handling, control structures, and basic arithmetic operations in Java

6. Create a Java program that prompts the user to enter the number of days (n) for which they want to generate their exercise routine. The program should then calculate and display the first n terms of the Fibonacci series, representing the exercise duration for each day.

PROGRAM CODE :

```java
import java.util.Scanner;
public class Pr6
{
    public static void main(String[] args) {
        int j = 1, k, temp = 0, sum = 0;
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for (int i = 0; temp <=n; i++) {
            System.out.println(temp);
            k = j;
            j = temp;
            temp = j + k;

            sum += temp;
        }
        System.out.println(sum);
        sc.close();
    }
}
```

OUTPUT:



```
7
0
1
1
2
3
5
20
```

<u>CONCLUSION:</u>

This program provides an easy way to generate an exercise routine based on the Fibonacci series, which is often used to progressively increase workout durations. By following this routine, you can ensure a gradual increase in your exercise time, which can help improve endurance and overall fitness. The Fibonacci series starts with 0 and 1, and each subsequent term is the sum of the previous two terms. This pattern creates a naturally progressive routine, making it a useful tool for designing exercise plans.

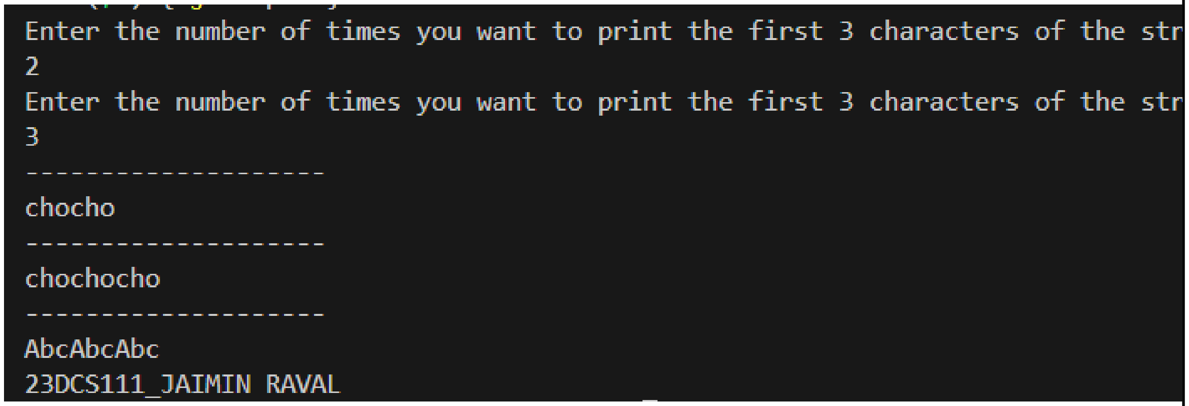
SET - 2

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 CODE:

```java
import java.util.*;

public class pra_7 {

    public static void main(String[] args) {
        String st = "chocolate";
        String st1 = "Abc";
        int n, m;
        Scanner b = new Scanner(System.in);
        System.out.println("Enter the number of times you want to print the
first 3 characters of the string");
        n = b.nextInt();
        System.out.println("Enter the number of times you want to print the
first 3 characters of the string");
        m = b.nextInt();
        System.out.println("--------------------");
        for (int i = 0; i < n; i++) {

            System.out.print(st.substring(0, 3));
        }
        System.out.println("\n--------------------");
        for (int i = 0; i < m; i++) {

            System.out.print(st.substring(0, 3));
        }
        System.out.println("\n--------------------");
        for (int i = 0; i < m; i++) {
            System.out.print(st1.substring(0, 3));
        }
       System.out.println("\n23DCS111_JAIMIN RAVAL");

    }
}
```

OUTPUT:



```
Enter the number of times you want to print the first 3 characters of the str
2
Enter the number of times you want to print the first 3 characters of the str
3
--------------------
chocho
--------------------
chochocho
--------------------
AbcAbcAbc
23DCS111_JAIMIN RAVAL
```

CONCLUSION:

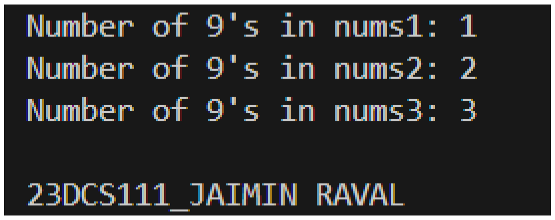
This Java program prompts the user to enter two integers, n and m, then prints the first 3 characters of the string "chocolate" n times and m times consecutively. Additionally, it prints the first 3 characters of the string "Abc" m times. The program separates different outputs with dashes and concludes by displaying a footer message. The use of loops ensures repetitive printing based on user input, demonstrating basic input-output operations and string manipulation in Java.

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 CODE :

```java
public class Pra_8 {
    public static int array_count9(int[] nums) {
        int count = 0;
        for (int num : nums) {
            if (num == 9) {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        int[] nums1 = {1, 2, 9};
        int[] nums2 = {1, 9, 9};
        int[] nums3 = {1, 9, 9, 3, 9};

        System.out.println("Number of 9's in nums1: " +
array_count9(nums1));
        System.out.println("Number of 9's in nums2: " +
array_count9(nums2));
        System.out.println("Number of 9's in nums3: " +
array_count9(nums3));

    }
}
```

OUTPUT:



```
Number of 9's in nums1: 1
Number of 9's in nums2: 2
Number of 9's in nums3: 3

23DCS111_JAIMIN RAVAL
```

CONCLUSION:

This Java program defines a method `array_count9` that calculates and returns the number of times the integer `9` appears in an input integer array. It then demonstrates the functionality by applying the method to three different arrays (`nums1`, `nums2`, `nums3`) and prints the results. Finally, it includes a footer message `"23DCS089_Samarth Patel"` for identification. The program effectively showcases basic array manipulation and function usage in Java.

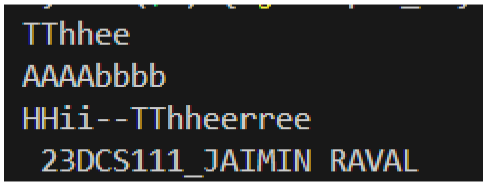
9. Given a string, return a string where for every char in the

original, there are two chars.
double_char('The') → 'TThhee'
double_char('AAbb') → 'AAAAbbbb'
double_char('Hi-There') → 'HHii--TThheerree'

PROGRAM CODE :

```
public  class pra_9{
    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) {
        pra_9 obj = new pra_9();
        System.out.println(obj.doubleChar("The"));
        System.out.println(obj.doubleChar("AAbb"));
        System.out.println(obj.doubleChar("Hi-There"));
       System.out.println(" 23DCS111_JAIMIN RAVAL");
    }
}
```

OUTPUT:



```
TThhee
AAAAbbbb
HHii--TThheerree
 23DCS111_JAIMIN RAVAL
```

CONCLUSION:
This Java program defines a method `doubleChar` within the class `pra_9`, which takes a string `str` as input and doubles each character in the string. It uses a `StringBuilder` to efficiently build the resultant string by appending each character twice in a loop. The `main` method demonstrates the functionality of `doubleChar` by creating an instance of `pra_9`, calling the method with different input strings ("The", "AAbb", "Hi-There"), and printing the transformed strings. The program concludes with a footer.

10. Perform following functionalities of the string:
● Find Length of the String

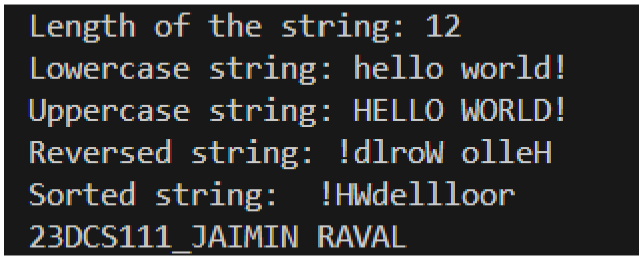
- Lowercase of the String
- Uppercase of the String
- Reverse String

  Sort the string

PROGRAM CODE :

```java
import java.util.Arrays;

public class pra_10 {
    public static void main(String[] args) {
        String str = "Hello World!";

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

        String lowercase = str.toLowerCase();
        System.out.println("Lowercase string: " + lowercase);

        String uppercase = str.toUpperCase();
        System.out.println("Uppercase string: " + uppercase);

        String reverse = "";
        for (int i = length - 1; i >= 0; i--) {
            reverse = reverse + str.charAt(i);
        }
        System.out.println("Reversed string: " + reverse);

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

        System.out.println("23DCS111_JAIMIN RAVAL");
    }
}
```

OUTPUT:



```
Length of the string: 12
Lowercase string: hello world!
Uppercase string: HELLO WORLD!
Reversed string: !dlroW olleH
Sorted string:  !HWdellloor
23DCS111_JAIMIN RAVAL
```

CONCLUSION:

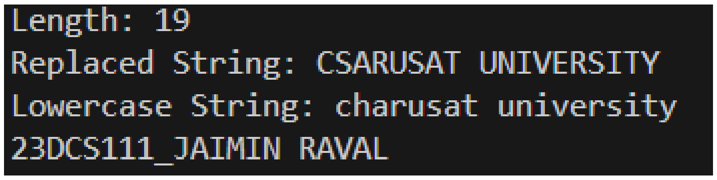
The Java program provided demonstrates several string manipulation techniques. It calculates the length of a string, converts it to lowercase and uppercase, reverses the string, and sorts its characters alphabetically. These operations showcase fundamental string handling capabilities in Java using built-in methods and loops. Additionally, the program concludes by printing a fixed message. Overall, it serves as a basic example of how to perform common string operations and utilize array manipulation functions in Java.

11 Perform following Functionalities of the string:
"CHARUSAT UNIVERSITY"
- Find length
- Replace 'H' by 'FIRST LATTER OF YOUR NAME'
- Convert all character in lowercase

PROGRAM CODE :

```
public class pra_11 {
    public static void main(String[] args) {
        String str = "CHARUSAT UNIVERSITY";

        int length = str.length();
        System.out.println("Length: " + length);

        String replacedStr = str.replace('H', 'S');
        System.out.println("Replaced String: " + replacedStr);

        String lowercaseStr = str.toLowerCase();
        System.out.println("Lowercase String: " + lowercaseStr);

        System.out.println("23DCS111_JAIMIN RAVAL ");
    }
}
```

OUTPUT:



```
Length: 19
Replaced String: CSARUSAT UNIVERSITY
Lowercase String: charusat university
23DCS111_JAIMIN RAVAL
```

CONCLUSION:

The Java code performs various string manipulations on the input "CHARUSAT UNIVERSITY". It first calculates and prints the length of the string. Then, it replaces the character 'H' with 'S' and prints the modified string. Next, the code converts the entire string to lowercase and prints the result. These operations demonstrate basic string handling techniques in Java.

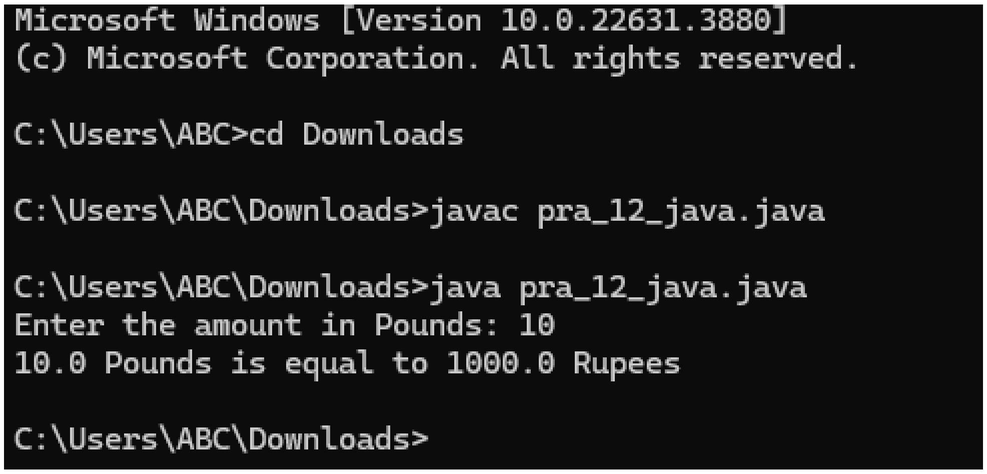
SET - 3

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 CODE:

```
import java.util.*;
public class p12{
   public static void main(String[] args){
      double pounds;
      double rupees = 0;
      Scanner sc = new Scanner(System.in);

      if(args.length > 0 ){
         pounds = Double.parseDouble(args[0]);
         rupees = 100*pounds;
              System.out.println("The amount in pound converted into " +
rupees " rupees.");
         return;

      }
      else{

         System.out.println("Enter the amount in pounds: ");
         pounds = sc.nextDouble();
         rupees = 100*pounds;

         System.out.println("The amount in rupees is: "+rupees);
      }

   }

}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ABC>cd Downloads

C:\Users\ABC\Downloads>javac pra_12_java.java

C:\Users\ABC\Downloads>java pra_12_java.java
Enter the amount in Pounds: 10
10.0 Pounds is equal to 1000.0 Rupees

C:\Users\ABC\Downloads>
```

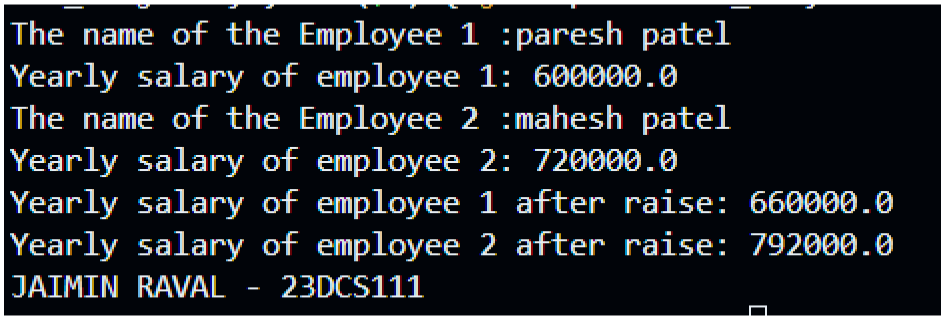
CONCLUSION:

The provided Java program converts an amount in pounds to rupees, using a conversion rate of 100 rupees per pound. If the amount is given as a command-line argument, it directly converts and prints the result. If not, it prompts the user to enter the amount, performs the conversion, and then displays the result.

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 CODE :

```
public class practical_13 {
    public static void main(String[] args) {

        Employee em1 = new Employee("paresh", "patel", 50000.0);
        Employee em2 = new Employee("mahesh", "patel", 60000.0);

        System.out.print("The name of the Employee 1 :");
        em1.displayname();
                System.out.println("Yearly salary of employee 1: " +
em1.getYearlySalary());

        System.out.print("The name of the Employee 2 :");
        em2.displayname();
                System.out.println("Yearly salary of employee 2: " +
em2.getYearlySalary());

        em1.giveRaise(10);
        em2.giveRaise(10);

            System.out.println("Yearly salary of employee 1 after raise: " +
em1.getYearlySalary());
            System.out.println("Yearly salary of employee 2 after raise: " +
em2.getYearlySalary());
    }
}

class Employee {
    String fname;
    String lname;
    double salary;

    public Employee(String f, String l, double s) {
        fname = f;
        lname = l;
        if (salary < 0) {
            salary = 0.0;
        } else {
            salary = s;
        }
    }

    double getYearlySalary() {
        return salary * 12;
    }
```

	```
	    void giveRaise(int percent) {
	        salary = salary + (salary * percent / 100);
	    }

	    void displayname() {
	        System.out.println(fname + " " + lname);
	    }

	    void displaysalary() {
	        System.out.println("Salary: " + salary);
	    }
	}
	```
	 OUTPUT: 
	```
	The name of the Employee 1 :paresh patel
	Yearly salary of employee 1: 600000.0
	The name of the Employee 2 :mahesh patel
	Yearly salary of employee 2: 720000.0
	Yearly salary of employee 1 after raise: 660000.0
	Yearly salary of employee 2 after raise: 792000.0
	JAIMIN RAVAL - 23DCS111
	```
	 CONCLUSION: The provided Java program defines an Employee class with attributes for first name, last name, and salary, including methods to calculate yearly salary, give a raise, and display the employee's name. The practical_13 class creates two Employee objects, prints their names and yearly salaries, gives them a 10% raise, and then prints their updated yearly salaries.

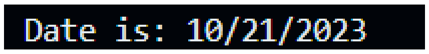
14.	Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set 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 CODE :

```java
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 int getMonth() {
        return month;
    }

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

    public int getDay() {
        return day;
    }

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

    public int getYear() {
        return year;
    }

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

public class pra_14 {
    public static void main(String[] args) {

        Date date = new Date(10, 21, 2023);
        System.out.print("Date is: ");
        date.displayDate();


    }
}
```

OUTPUT:



```
Date is: 10/21/2023
```

CONCLUSION:

The provided Java program consists of a date class that stores day, month, and year, with methods to get data from the user and display the date. In the practical_14 class, an instance of date is created with initial values, the user is prompted to enter a date, and the entered date is displayed.

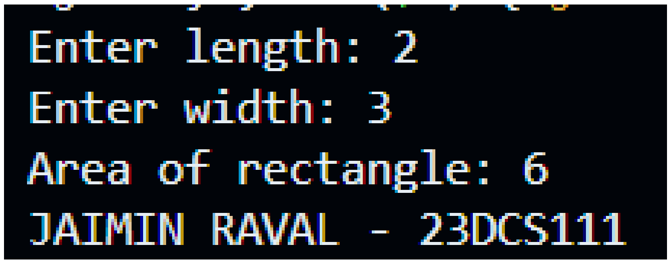
15. Perform following functionalities of the string:
- Find Length of the String
- Lowercase of the String
- Uppercase of the String
- Reverse String

  Sort the string

PROGRAM CODE :

```
import java.util.Scanner;

public class pra_15 {
    private int length;
    private int width;
    private int area;

    public pra_15(int length, int width) {
        this.length = length;
        this.width = width;
    }

    public int calculateArea() {
        area = length * width;
        return area;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter length: ");
        int length = scanner.nextInt();
        System.out.print("Enter width: ");
        int width = scanner.nextInt();
        pra_15 rectangle = new pra_15(length, width);
        int area = rectangle.calculateArea();
        System.out.println("Area of rectangle: " + area);

        System.out.println("JAIMIN RAVAL - 23DCS111");
    }
}
```

OUTPUT:



```
Enter length: 2
Enter width: 3
Area of rectangle: 6
JAIMIN RAVAL - 23DCS111
```

CONCLUSION:

The Java code performs a series of string manipulations on the input "My name is Shubh". It calculates and prints the length of the string, then converts and prints the string in both uppercase and lowercase formats. The code also reverses the string and prints the reversed version. Additionally, it converts the string to a character array, sorts the array, and prints the sorted string. Finally, These operations illustrate various methods of string handling

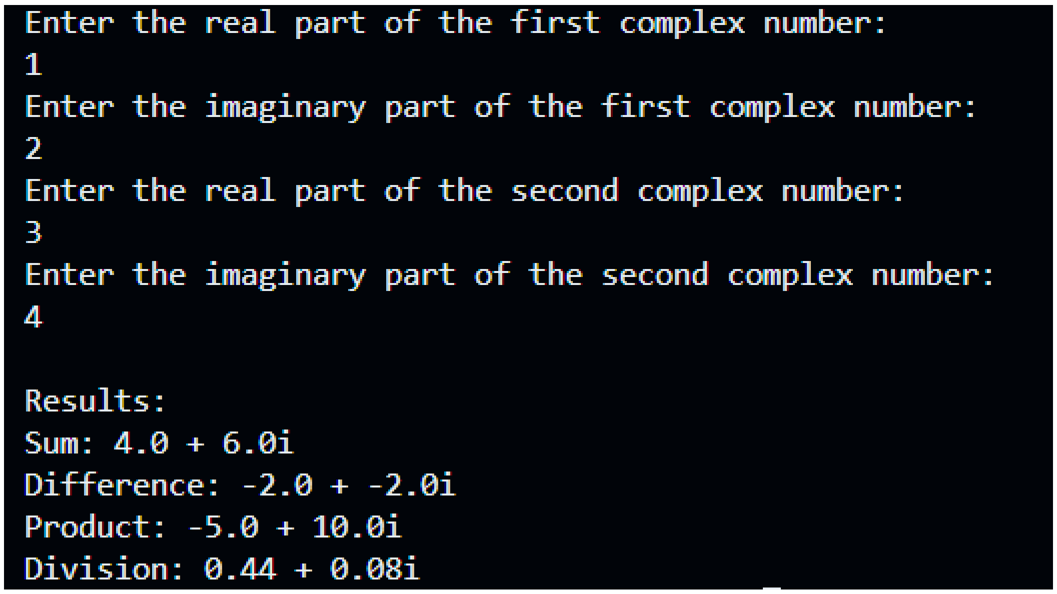
16. Print the sum , difference and product of two complex numbers by creating a class named ‘ Complex ’ with separate methods for each operation whose real and imaginary parts are entered by user.

PROGRAM CODE :

```
import java.util.Scanner;

class Complex {
    private double real;
    private double imag;

    public Complex(double real, double imag) {
        this.real = real;
        this.imag = imag;
    }

    public Complex add(Complex other) {
        return new Complex(this.real + other.real, this.imag +
other.imag);
    }

    public Complex subtract(Complex other) {
        return new Complex(this.real - other.real, this.imag -
other.imag);
    }

    public Complex multiply(Complex other) {
        double realPart = this.real * other.real - this.imag *
other.imag;
        double imagPart = this.real * other.imag + this.imag *
other.real;
        return new Complex(realPart, imagPart);
    }

    public Complex divide(Complex other) {
        double denominator = other.real * other.real + other.imag *
other.imag;
        double realPart = (this.real * other.real + this.imag *
other.imag) / denominator;
        double imagPart = (this.imag * other.real - this.real *
other.imag) / denominator;
        return new Complex(realPart, imagPart);
    }

    public void display() {
        System.out.println(this.real + " + " + this.imag + "i");
    }
}

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

        System.out.println("Enter the real part of the first complex
number:");
        double real1 = scanner.nextDouble();
        System.out.println("Enter the imaginary part of the first
complex number:");
        double imag1 = scanner.nextDouble();
```

```java
        System.out.println("Enter the real part of the second complex
number:");
        double real2 = scanner.nextDouble();
        System.out.println("Enter the imaginary part of the second
complex number:");
        double imag2 = scanner.nextDouble();

        Complex complex1 = new Complex(real1, imag1);
        Complex complex2 = new Complex(real2, imag2);

        Complex sum = complex1.add(complex2);
        Complex difference = complex1.subtract(complex2);
        Complex product = complex1.multiply(complex2);
        Complex quotient = complex1.divide(complex2);

        System.out.println("\nResults:");
        System.out.print("Sum: ");
        sum.display();
        System.out.print("Difference: ");
        difference.display();
        System.out.print("Product: ");
        product.display();
        System.out.print("Division: ");
        quotient.display();
    }
}
```

OUTPUT:



```
Enter the real part of the first complex number:
1
Enter the imaginary part of the first complex number:
2
Enter the real part of the second complex number:
3
Enter the imaginary part of the second complex number:
4

Results:
Sum: 4.0 + 6.0i
Difference: -2.0 + -2.0i
Product: -5.0 + 10.0i
Division: 0.44 + 0.08i
```

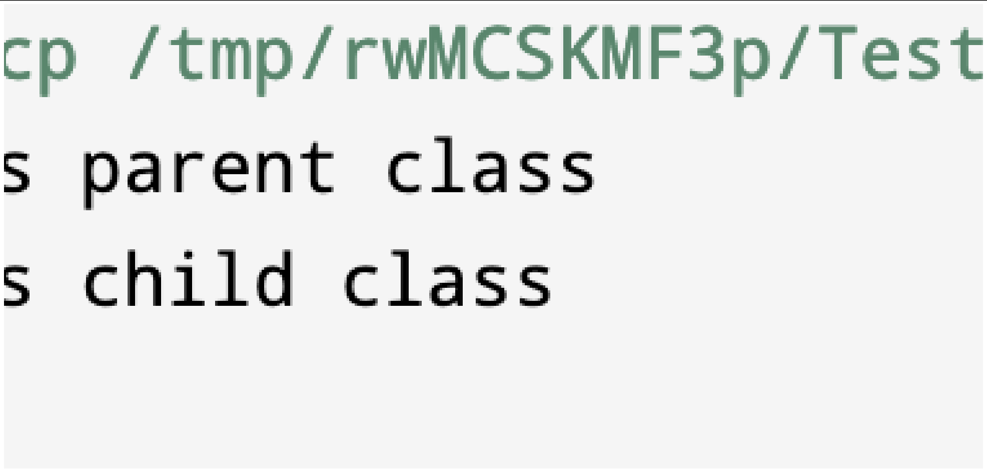
CONCLUSION:

	The provided Java program defines a complex class to represent and manipulate complex numbers. It includes methods to compute and display the sum, difference, and product of two complex numbers. The main method takes user input for two complex numbers and demonstrates these operations, showcasing the functionality of the complex class.

SET - 4

No.	Aim of the Practical

17.	ICreate 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 CODE:
	```
	class Test1
	{
	public static void main(String []args)
	{
	Child c1=new Child();
	c1.mssg1();
	c1.mssg2();
	}
	}
	class Parent
	{
	void mssg1()
	{
	System.out.println("This is parent class");
	}
	}
	class Child extends Parent
	{
	void mssg2()
	{
	System.out.println("This is child class");
	}
	}
	```
	OUTPUT:



```
cp /tmp/rwMCSKMF3p/Test
s parent class
s child class
```

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.

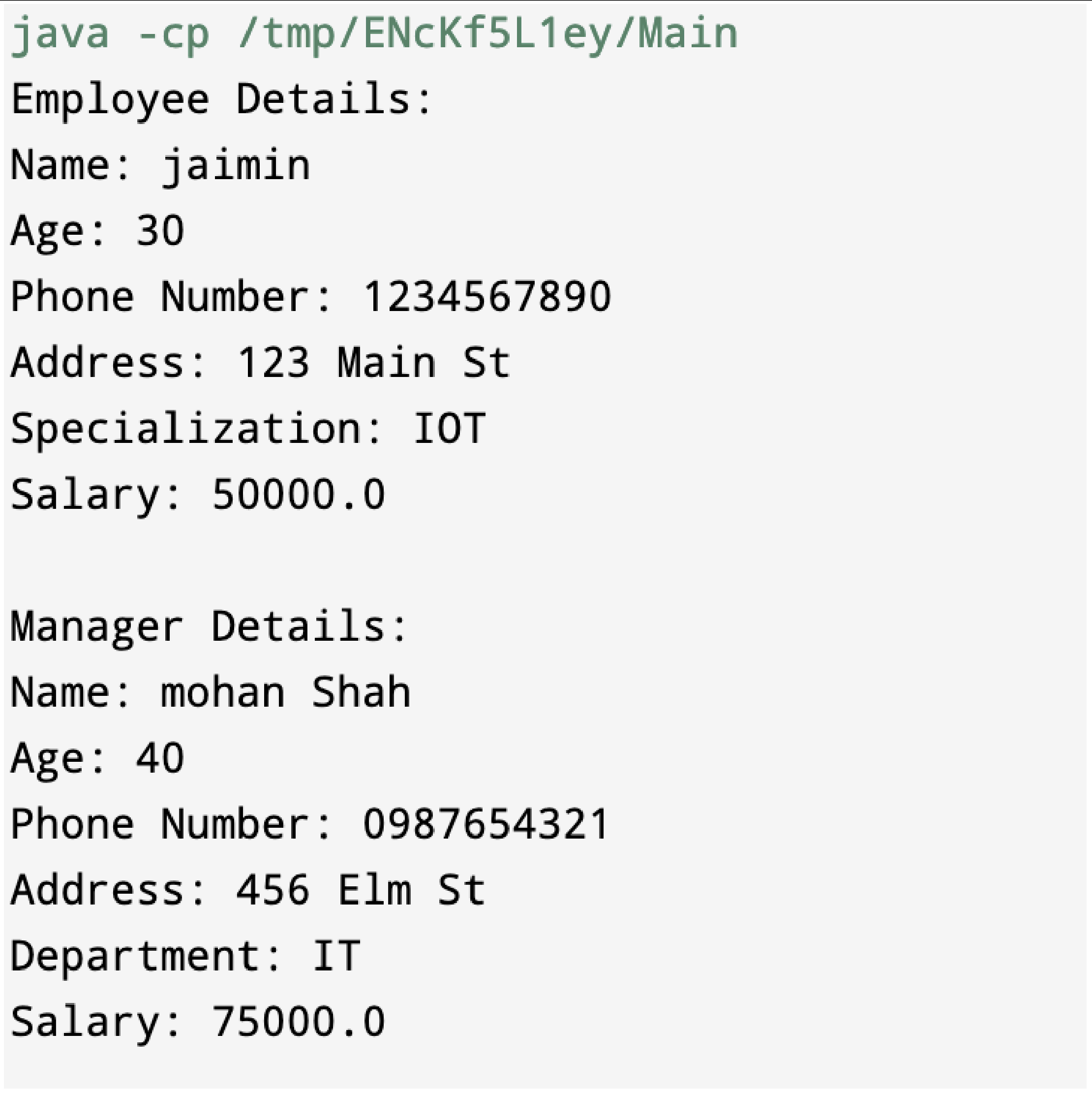
PROGRAM CODE :

```
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;

Employee(String name, int age, String phoneNumber, String address,
double salary, String
specialization) {
this.name = name;
this.age = age;
this.phoneNumber = phoneNumber;
this.address = address;
this.salary = salary;
this.specialization = specialization;
}

void display() {
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;

Manager(String name, int age, String phoneNumber, String address,
double salary, String
department) {
this.name = name;
this.age = age;
this.phoneNumber = phoneNumber;
this.address = address;
this.salary = salary;
this.department = department;
```

```
}

void display() {
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 e1 = new Employee("Tisha Rana", 30, "1234567890", "123
Main St", 50000,
"IOT");

Manager m1 = new Manager("Hetvi Shah", 40, "0987654321", "456 Elm
St", 75000, "IT");

e1.display();
System.out.println();
m1.display();
}
}
```

OUTPUT:



```
java -cp /tmp/ENcKf5L1ey/Main
Employee Details:
Name: jaimin
Age: 30
Phone Number: 1234567890
Address: 123 Main St
Specialization: IOT
Salary: 50000.0

Manager Details:
Name: mohan Shah
Age: 40
Phone Number: 0987654321
Address: 456 Elm St
Department: IT
Salary: 75000.0
```

CONCLUSION:

In the above practical, we understood concept of Hierarchical Inheritance. We made a parent class “Member” which extends two child classes “Manager” and “Employee”. By Objects of these class we called methods of parent class also.

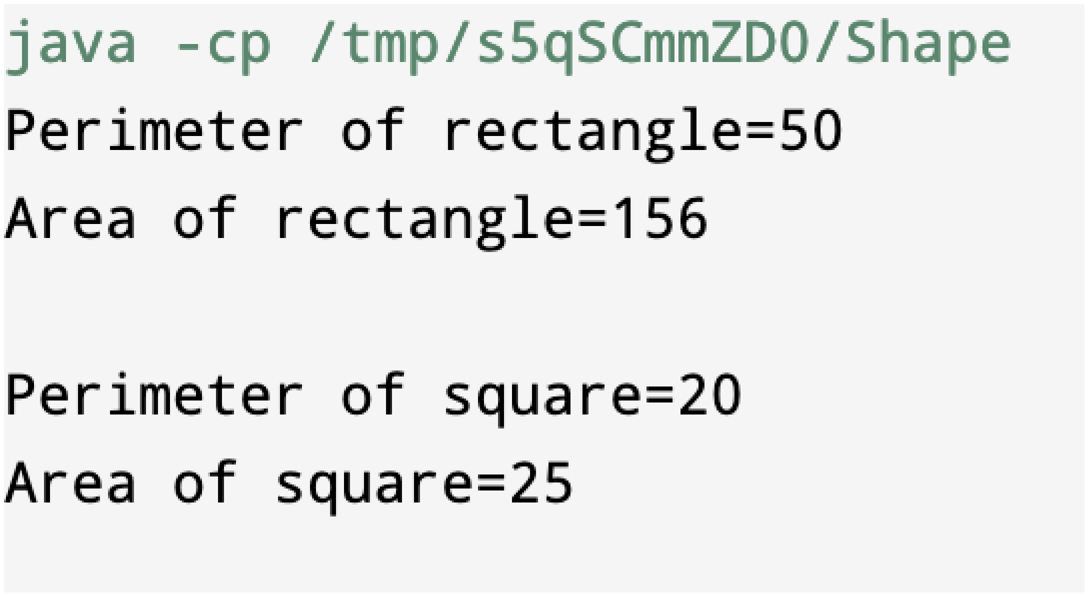
19. Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit 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 CODE :

```
class Shape
{
public static void main(String []args)
{
Rectangle []shape=new Rectangle[2];
shape[0]=new Rectangle(12,13);
shape[1]=new Square(5);
for(int i=0;i<2;i++)
{
shape[i].perimeter();
shape[i].area();
System.out.println();
}
}
}
class Rectangle
{
int length;
int width;
void area()
{
System.out.println("Area of rectangle="+(length*width));
}
void perimeter()
{
System.out.println("Perimeter of rectangle="+(2*(length+width)));
}
Rectangle(int l,int w)
{
length=l;
width=w;
}
}
class Square extends Rectangle
{
```

```
Square(int side)
{
super(side,side);
}
void perimeter()
{
System.out.println("Perimeter of square="+(4*length));

}
void area()
{
System.out.println("Area of square="+(length*length));
}
}
```

OUTPUT:



```
java -cp /tmp/s5qSCmmZDO/Shape
Perimeter of rectangle=50
Area of rectangle=156

Perimeter of square=20
Area of square=25
```

CONCLUSION:

In the above practical, we understood the concept of inheriting the constructors of Parent class to child class. Here the constructors of class Rectangle are extended in class Square also and respective methods are called by their objects in Class Shape.

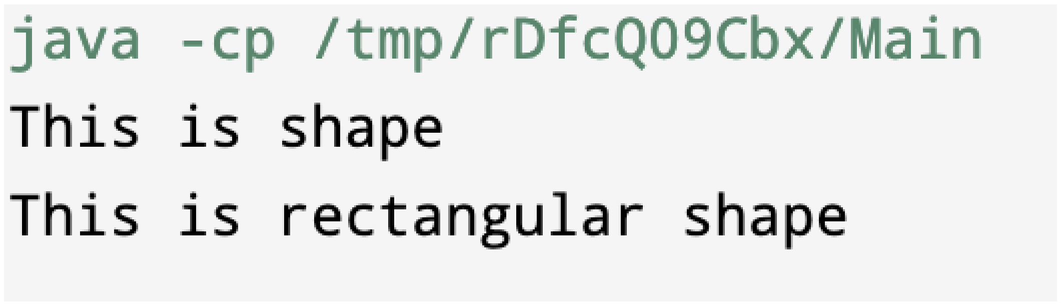
20. Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' 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 CODE :

```
class Shape {
  void printShape() {
     System.out.println("This is shape");
  }
}


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

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

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


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

     Square square = new Square();
```

```
        square.printShape();
        square.printRectangle();
    }
}
```

OUTPUT:



```
java -cp /tmp/rDfcQO9Cbx/Main
This is shape
This is rectangular shape
```

CONCLUSION:

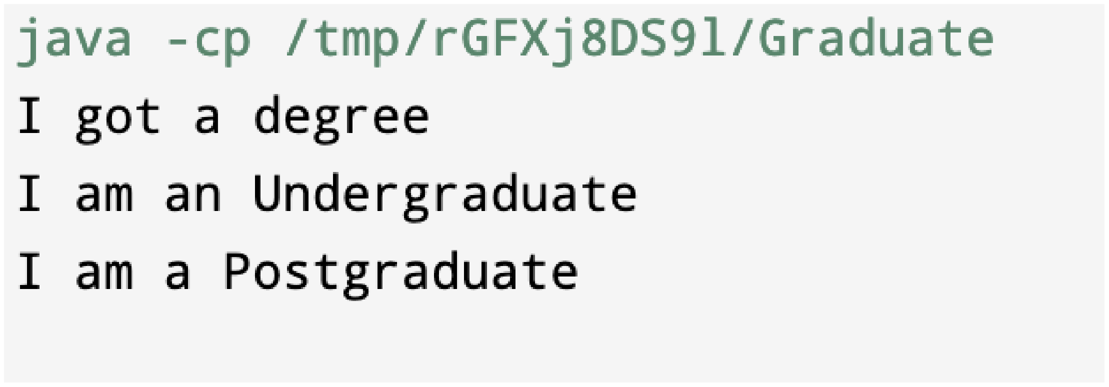
The Java code performs a series of string manipulations on the input "My name is Shubh". It calculates and prints the length of the string, then converts and prints the string in both uppercase and lowercase formats. The code also reverses the string and prints the reversed version. Additionally, it converts the string to a character array, sorts the array, and prints the sorted string. Finally, These operations illustrate various methods of string handling

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 CODE :

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


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

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


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

     Degree d = new Degree();
     Undergraduate ug = new Undergraduate();
     Postgraduate pg = new Postgraduate();


     d.getDegree();
     ug.getDegree();
     pg.getDegree();
  }
}
```

OUTPUT:



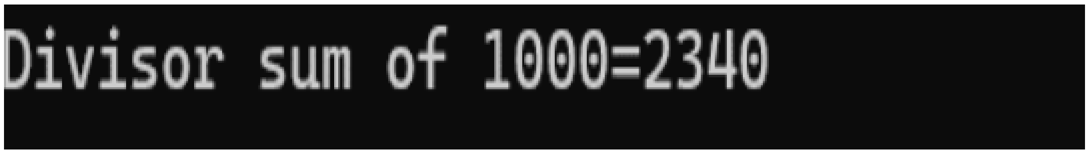
```
java -cp /tmp/rGFXj8DS9l/Graduate
I got a degree
I am an Undergraduate
I am a Postgraduate
```

22. Write a java that implements an interface AdvancedArithmetic which contains amethodsignature 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 CODE :

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

}

}
return s;
}

}
class Divisor
{
public static void main(String [] args)
{
MyCalculator a=new MyCalculator();
int sum=a.divisor_sum(1000);
System.out.println("Divisor sum="+sum);


}
}
```

OUTPUT:



```
Divisor sum of 1000=2340
```

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 CODE:

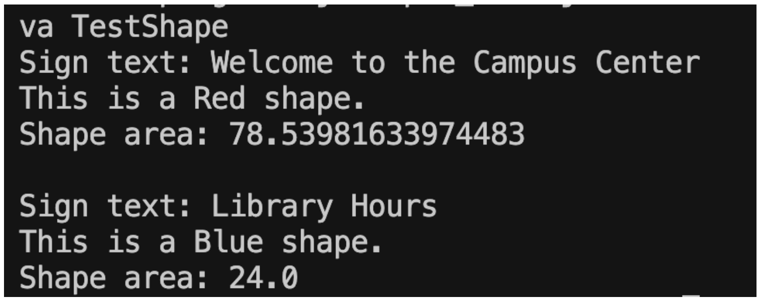
```
interface Shape
{
   String getColor();


   default void describe()
   {
      System.out.println("This is a " + getColor() + " shape.");
   }


   double calculateArea();
}


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 calculateArea()
  {
     return Math.PI * radius * radius;
  }


  public double getRadius()
  {
     return 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 calculateArea()
    {
        return length * width;
    }


    public double getLength()
    {
        return length;
    }

    public double getWidth()
    {
        return width;
    }
}


class Sign
{
    private Shape shape;
    private String text;


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

```
  public void displaySign()
  {
     System.out.println("Sign text: " + text);
     shape.describe();
     System.out.println("Shape area: " + shape.calculateArea());
  }
}

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

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


     Shape rectangle = new Rectangle(4, 6, "Blue");


     Sign circleSign = new Sign(circle, "Welcome to the Campus
Center");
     Sign rectangleSign = new Sign(rectangle, "Library Hours");


     circleSign.displaySign();
     System.out.println();
     rectangleSign.displaySign();
  }
}
```

OUTPUT:



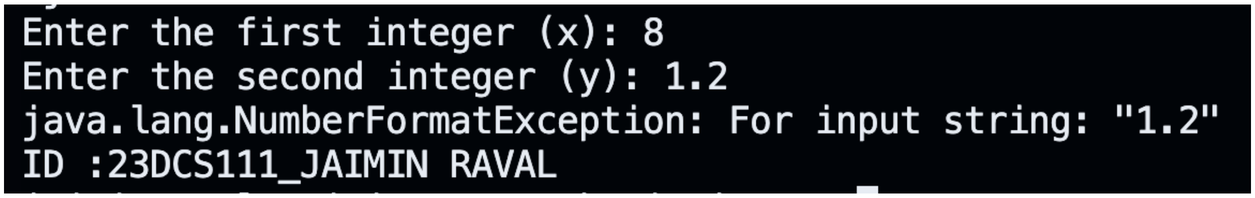
```
va TestShape
Sign text: Welcome to the Campus Center
This is a Red shape.
Shape area: 78.53981633974483

Sign text: Library Hours
This is a Blue shape.
Shape area: 24.0
```

No.	Data Types, Variables, String, Control Statements, Operators, Arrays
	PART-V Exception Handling
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 CODE:

```
import java.util.Scanner;

public class P24 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

System.out.print("Enter the first integer (x): ");

int x = Integer.parseInt(sc.nextLine()); // Parse integer input

System.out.print("Enter the second integer (y): ");

int y = Integer.parseInt(sc.nextLine()); // Parse integer input

int result = x / y;

System.out.println("Result of " + x + " / " + y + " = " + result);

}

catch (NumberFormatException e) {

System.out.println(e);

}
catch (ArithmeticException e) {

System.out.println(e);

}

System.out.println("ID :23DCS111_JAIMIN RAVAL");

sc.close();

} }
```

OUTPUT:



```
Enter the first integer (x): 8
Enter the second integer (y): 1.2
java.lang.NumberFormatException: For input string: "1.2"
ID :23DCS111_JAIMIN RAVAL
```

CONCLUSION:

This program handles user input for two integers and performs division while using exception handling to catch invalid input (`NumberFormatException`) and division by zero (`ArithmeticException`). It ensures the program doesn't crash due to errors and prints the result or the exception message. Finally, it closes the scanner and displays the student ID.

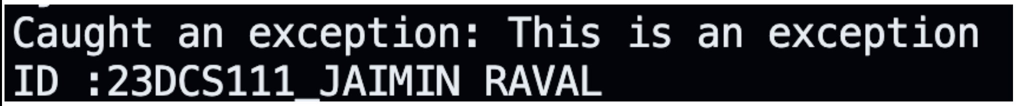
25 Write a Java program that throws an exception and catch it using a try-catch block.

PROGRAM CODE:

```
public class P25 {

public static void main(String[] args) {

try {

throw new Exception("This is an exception");

} catch (Exception e) {
```

```
System.out.println("Caught an exception: " + e.getMessage());
 System.out.println("ID :23DCS111_JAIMIN RAVAL");
}
}}
```

OUTPUT:



```
Caught an exception: This is an exception
ID :23DCS111_JAIMIN RAVAL
```

CONCLUSION:

In this program an exception is deliberately created by use of the keyword throw and then all lines after this statement in the catch block catch it. Here is how the Java programming language is made useful in handling runtime errors.

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).

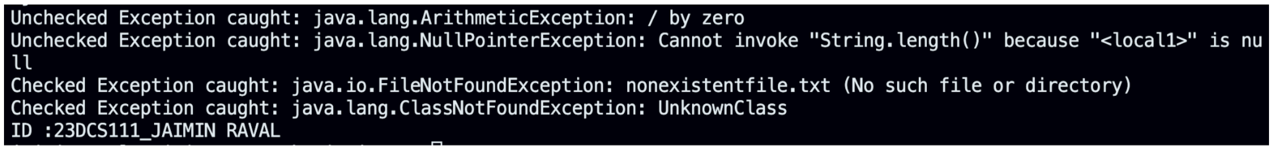
PROGRAM CODE:

```
import java.io.*;
public class P26 {
public static void readFile(String filePath) throws IOException {
FileReader file = new FileReader(filePath);
BufferedReader br = new BufferedReader(file);
System.out.println(br.readLine());
br.close();
```

```
}
public static void main(String[] args) {
try {
int a = 10 / 0;
} catch (ArithmeticException e) {
System.out.println("Unchecked Exception caught: " + e);
}
try {
String str = null;
System.out.println(str.length());
} catch (NullPointerException e) {
System.out.println("Unchecked Exception caught: " + e);
}
try {
readFile("nonexistentfile.txt");
} catch (IOException e) {
System.out.println("Checked Exception caught: " + e);
}
try {
Class.forName("UnknownClass");
} catch (ClassNotFoundException e) {
```

```
System.out.println("Checked Exception caught: " + e);

}

System.out.println("ID :23DCS111_JAIMIN RAVAL");

} }
```

OUTPUT:



```
Unchecked Exception caught: java.lang.ArithmeticException: / by zero
Unchecked Exception caught: java.lang.NullPointerException: Cannot invoke "String.length()" because "<local1>" is nu
ll
Checked Exception caught: java.io.FileNotFoundException: nonexistentfile.txt (No such file or directory)
Checked Exception caught: java.lang.ClassNotFoundException: UnknownClass
ID :23DCS111_JAIMIN RAVAL
```

CONCLUSION:

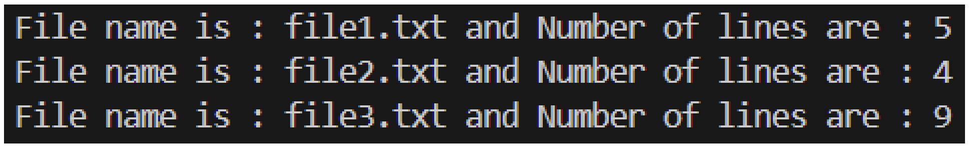
This Java program manages unchecked as well as checked exceptions. Arithmetic, null pointer file not found, and class not found exceptions are caught using the try-catch blocks, therefore the program runs quite seamlessly. Exception handling is quite important in building robust applications that can handle errors with grace.

PART-VI File Handling & Streams

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 CODE:

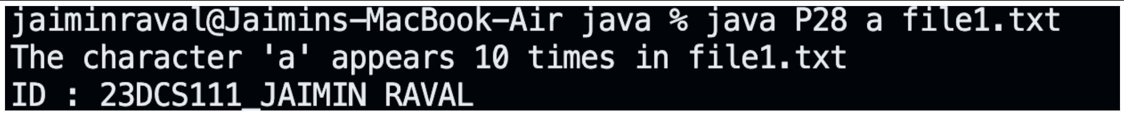
```
import java.io.*;

public class P27 {

public static void main(String[] args) throws Exception {

if (args.length == 0) {

System.out.println("No file Found!");

} else {

for (int i = 0; i < args.length; i++) {

try {

BufferedReader f = new BufferedReader(new FileReader(args[i]));

String j;

int count = 0;

while ((j = f.readLine()) != null) {

count++;

}

System.out.println("File name is : " + args[i] + " and Number of lines are : " + count);
```

	} catch (Exception e) { System.out.println(e); } } } System.out.println("ID :23DCS111_JAIMIN RAVAL"); } } **OUTPUT:**  File name is : file1.txt and Number of lines are : 5 File name is : file2.txt and Number of lines are : 4 File name is : file3.txt and Number of lines are : 9 **CONCLUSION:** This Java program reads several files named by the command line arguments and counts the number of lines in each. If no files are provided as command-line arguments, it will print out the appropriate message. Exception handling ensures graceful error management during file reading, thus a stable program.

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 CODE:
	import java.io.BufferedReader;
	import java.io.FileReader;
	import java.io.IOException;
	public class P28{
	public static void main(String[] args) {

```
if (args.length < 2) {
System.out.println("Usage: java P28 <character> <filename>");
return; }
char targetChar = args[0].charAt(0);
String fileName = args[1];
int count = 0;
try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
int ch;
while ((ch = reader.read()) != -1) {
if (ch == targetChar) {
count++;
} }
System.out.println("The character '" + targetChar + "' appears " + count + " times in " +
fileName);
} catch (IOException e) {
System.out.println("Error reading " + fileName + ": " + e.getMessage());
}
System.out.println("ID : 23DCS111_JAIMIN RAVAL");
}}
```

OUTPUT:



```
jaiminraval@Jaimins-MacBook-Air java % java P28 a file1.txt
The character 'a' appears 10 times in file1.txt
ID : 23DCS111_JAIMIN RAVAL
```

CONCLUSION:

The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access.

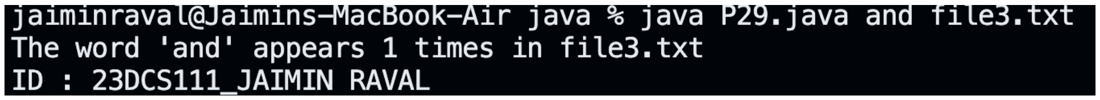
29 Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.

PROGRAM CODE:

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class P29 {

public static void main(String[] args) {

if (args.length < 2) {

System.out.println("Usage: java P29 <word> <filename>");

return;

}

String searchWord = args[0];

String fileName = args[1];

Integer count = 0;

try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
```

```
String line;
while ((line = reader.readLine()) != null) {
String[] words = line.split("\\W+");
for (String word : words) {
if (word.equalsIgnoreCase(searchWord)) {
count++;
} } }
System.out.println("The word '" + searchWord + "' appears " + count + " times in " +
fileName);
} catch (IOException e) {
System.out.println("Error reading " + fileName + ": " + e.getMessage());
}
System.out.println("ID : 23DCS111_JAIMIN RAVAL");
} }
```

OUTPUT:



```
jaiminraval@Jaimins-MacBook-Air java % java P29.java and file3.txt
The word 'and' appears 1 times in file3.txt
ID : 23DCS111_JAIMIN RAVAL
```

CONCLUSION:

This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java.

30	Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.

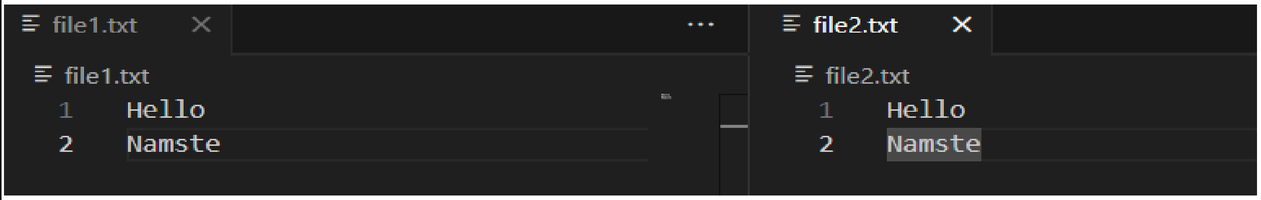
PROGRAM CODE:

```
import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class P30 {

public static void main(String[] args) {

if (args.length < 2) {

System.out.println("Usage: java P30 <source file> <destination file>");

return;

}

String sourceFile = args[0];

String destinationFile = args[1];

try (FileReader fr = new FileReader(sourceFile);

FileWriter fw = new FileWriter(destinationFile)) {

int ch;

while ((ch = fr.read()) != -1) {

fw.write(ch); }

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

} catch (IOException e) {

System.out.println("Error: " + e.getMessage());

}
```

System.out.println("ID : 23DCS111_JAIMIN RAVAL");

} }

OUTPUT:

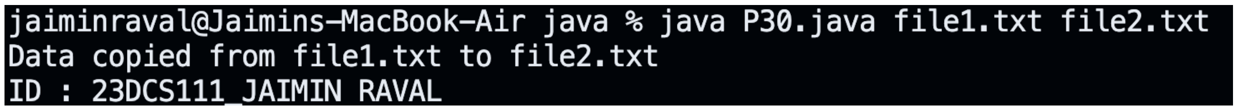


file1.txt

```
Hello
Namste
```

file2.txt

```
Hello
Namste
```



```
jaiminraval@Jaimins-MacBook-Air java % java P30.java file1.txt file2.txt
Data copied from file1.txt to file2.txt
ID : 23DCS111_JAIMIN RAVAL
```

CONCLUSION:

This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

31 Write a program to show use of character and byte stream. Also show use of BufferedReader / BufferedWriter to read console input and write them into a file.

PROGRAM CODE:

import java.io.*;

public class P31 {

public static void main(String[] args) {

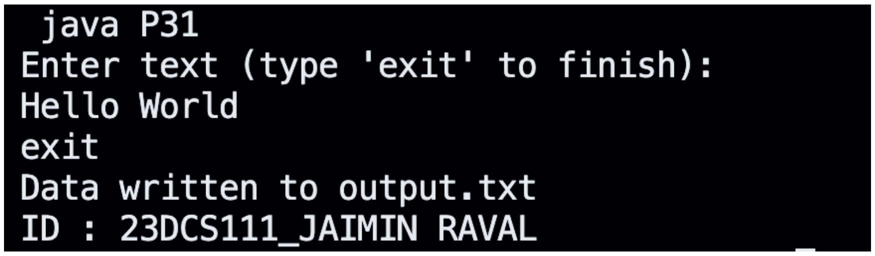
BufferedReader consoleReader = new BufferedReader(new InputStreamReader (System.in));

String fileName = "output.txt";

try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {

```
System.out.println("Enter text (type 'exit' to finish):");

String input;

while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit")) {

fileWriter.write(input);

fileWriter.newLine();

}

System.out.println("Data written to " + fileName);

} catch (IOException e) {

System.out.println("Error: " + e.getMessage());

}

System.out.println("ID : 23DCS111_JAIMIN RAVAL");

} }
```

OUTPUT:



```
 java P31
Enter text (type 'exit' to finish):
Hello World
exit
Data written to output.txt
ID : 23DCS111_JAIMIN RAVAL
```

CONCLUSION:

This program effectively demonstrates the use of character streams via BufferedReader and BufferedWriter for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with try-with-resources.

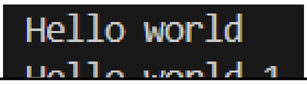
PART-VII Multithreading

32 Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.

PROGRAM CODE:

```
class Thread1 extends Thread{  // by extending Thread class

public void run(){

System.out.println("Hello world");

}}

class Thread2 implements Runnable{ //by using Runnable interface.

public void run(){

System.out.println("Hello world 1");

} }

public class P32 {

public static void main(String[] args) {

Thread1 thread = new Thread1();

thread.start();

Thread2 obj2 = new Thread2();

Thread t1 = new Thread(obj2);

t1.start();

} }
```

OUTPUT:



```
Hello world
```

CONCLUSION:

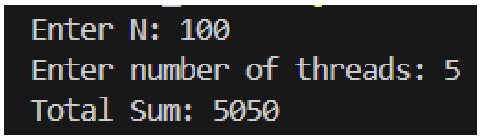
This program demonstrates two approaches to creating threads in Java: extending the Thread class and implementing the Runnable interface. Both methods effectively print "Hello World," showcasing the flexibility of Java's concurrency model.

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 CODE:

```
import java.util.Scanner;

class SumTask implements Runnable {

private int start;

private int end;

private static int totalSum = 0;

public SumTask(int start, int end) {

this.start = start;

this.end = end;

}

public void run() {

int partialSum = 0;

for (int i = start; i <= end; i++) {

partialSum += i;

}
```

```
synchronized (SumTask.class) {
totalSum += partialSum;
} }
public static int getTotalSum() {
return totalSum;
} }
public class P33 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter N: ");
int N = scanner.nextInt();
System.out.print("Enter number of threads: ");
int numThreads = scanner.nextInt();
Thread[] threads = new Thread[numThreads];
int range = N / numThreads;
int remainder = N % numThreads;
int start = 1;
for (int i = 0; i < numThreads; i++) {
int end = start + range - 1;
if (i == numThreads - 1) {
end += remainder;
}
```

	threads[i] = new Thread(new SumTask(start, end));
	threads[i].start();
	start = end + 1;
	}
	for (Thread thread : threads) {
	try {
	thread.join();
	} catch (InterruptedException e) {
	e.printStackTrace();
	} }
	System.out.println("Total Sum: " + SumTask.getTotalSum());
	} }
	OUTPUT:
	
	Enter N: 100
	Enter number of threads: 5
	Total Sum: 5050
	CONCLUSION:
	This program effectively demonstrates how to utilize multiple threads in Java to perform a summation task concurrently. By distributing the workload among threads, it showcases improved efficiency in computation, making it a practical example of multithreading in action.

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.

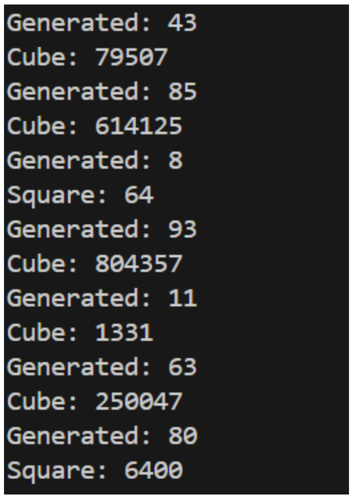
PROGRAM CODE:

```
import java.util.Random;
class RandomNumberGenerator extends Thread {
private final Object lock;
public RandomNumberGenerator(Object lock) {
this.lock = lock;
}
public void run() {
Random random = new Random();
while (true) {
int number = random.nextInt(100);
synchronized (lock) {
P34.lastNumber = number;
lock.notifyAll();
System.out.println("Generated: " + number);
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
} } } } }
class EvenNumberProcessor extends Thread {
private final Object lock;
```

```
public EvenNumberProcessor(Object lock) {
this.lock = lock;
}
public void run() {
while (true) {
synchronized (lock) {
try {
lock.wait();
} catch (InterruptedException e) {
e.printStackTrace();
}
if (P34.lastNumber % 2 == 0) {
int square = P34.lastNumber * P34.lastNumber;
System.out.println("Square: " + square);
} } } } }
class OddNumberProcessor extends Thread {
private final Object lock;
public OddNumberProcessor(Object lock) {
this.lock = lock;
}
public void run() {
while (true) {
```

```
synchronized (lock) {
try {
lock.wait();
} catch (InterruptedException e) {
e.printStackTrace();
}
if (P34.lastNumber % 2 != 0) {
int cube = P34.lastNumber * P34.lastNumber * P34.lastNumber;
System.out.println("Cube: " + cube);
} } } } }
public class P34 {
public static int lastNumber;
public static void main(String[] args) {
Object lock = new Object();
RandomNumberGenerator generator = new RandomNumberGenerator(lock);
EvenNumberProcessor evenProcessor = new EvenNumberProcessor(lock);
OddNumberProcessor oddProcessor = new OddNumberProcessor(lock);
generator.start();
evenProcessor.start();
oddProcessor.start();
}}
```

OUTPUT:



```
Generated: 43
Cube: 79507
Generated: 85
Cube: 614125
Generated: 8
Square: 64
Generated: 93
Cube: 804357
Generated: 11
Cube: 1331
Generated: 63
Cube: 250047
Generated: 80
Square: 6400
```

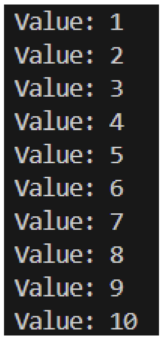
CONCLUSION:

This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency can be leveraged for efficient task distribution.

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 CODE:

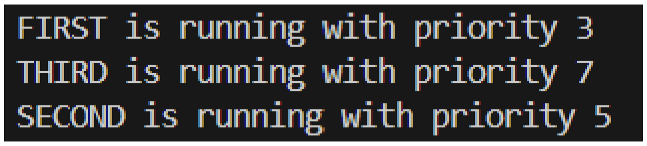
```
public class P35 extends Thread {

private int value = 0;

public void run() {

while (true) {

value++;

System.out.println("Value: " + value);

try {

Thread.sleep(1000);

} catch (InterruptedException e) {
```

	e.printStackTrace();
	} } }
	public static void main(String[] args) {
	P35 incrementer = new P35();
	incrementer.start();
	} }
	OUTPUT:
	
	Value: 1
	Value: 2
	Value: 3
	Value: 4
	Value: 5
	Value: 6
	Value: 7
	Value: 8
	Value: 9
	Value: 10
	CONCLUSION:
	This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java.

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 CODE:
	class MyThread extends Thread {
	MyThread(String name) {
	super(name);

```
}
public void run() {
System.out.println(getName() + " is running with priority " + getPriority());
} }
public class P36 {
public static void main(String[] args) {
MyThread first = new MyThread("FIRST");
MyThread second = new MyThread("SECOND");
MyThread third = new MyThread("THIRD");
first.setPriority(3);
first.start();
second.setPriority(Thread.NORM_PRIORITY);
second.start();
third.setPriority(7);
third.start();
} }
```

OUTPUT:



```
FIRST is running with priority 3
THIRD is running with priority 7
SECOND is running with priority 5
```

CONCLUSION:

This program demonstrates thread creation and priority setting in Java by extending the Thread class. Each thread prints its name and priority when executed. Different priority levels (3, 5, 7) are set using setPriority(), showcasing the influence of priority on execution order. However, actual execution may vary due to the system's thread scheduling.

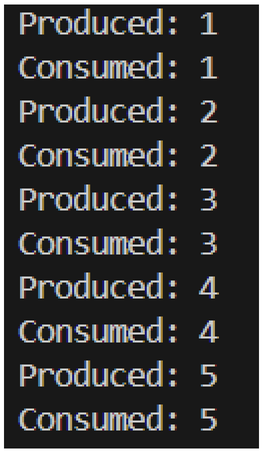
37 Write a program to solve producer-consumer problem using thread synchronization.

PROGRAM CODE:

```
class Buffer {
private int data;
private boolean isEmpty = true;
public synchronized void produce(int value) {
while (!isEmpty) {
try {
wait();
} catch (InterruptedException e) {
e.printStackTrace();
} }
data = value;
isEmpty = false;
System.out.println("Produced: " + data);
notify();
}
public synchronized void consume() {
while (isEmpty) {
try {
wait();
} catch (InterruptedException e) {
e.printStackTrace();
} }
System.out.println("Consumed: " + data);
isEmpty = true;
notify();
} }
class Producer extends Thread {
private Buffer buffer;
public Producer(Buffer buffer) {
this.buffer = buffer;
}
public void run() {
for (int i = 1; i <= 5; i++) {
buffer.produce(i);  // Produce values from 1 to 5
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
} } } }
class Consumer extends Thread {
private Buffer buffer;
```

```
public Consumer(Buffer buffer) {
this.buffer = buffer;
}
public void run() {
for (int i = 1; i <= 5; i++) {
buffer.consume();
try {
Thread.sleep(1500);
} catch (InterruptedException e) {
e.printStackTrace();
} } } }
public class P37 {
public static void main(String[] args) {
Buffer buffer = new Buffer();
Producer producer = new Producer(buffer);
Consumer consumer = new Consumer(buffer);
producer.start();
consumer.start();
} }
```

OUTPUT:



```
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5
```

CONCLUSION:

This program demonstrates producer-consumer synchronization in Java using the wait() and notify() methods. The producer thread generates data, while the consumer thread consumes it, both synchronized to avoid race conditions. The use of wait() and notify() ensures proper coordination between the threads, allowing for controlled data production and consumption.

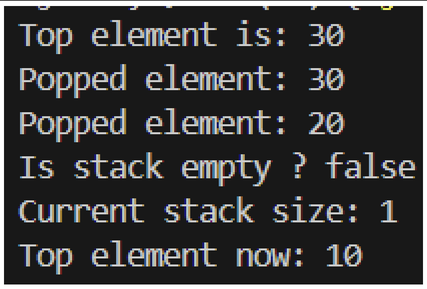
PART-VIII Collection Framework and Generic

38 Design a Custom Stack using ArrayList class, which implements following functionalities of stack.
My Stack -list ArrayList<Object>: A list to store elements.
isEmpty: boolean: Returns true if this stack is empty.
getSize(): int: Returns number of elements in this stack.
peek(): Object: Returns top element in this stack without removing it.
pop(): Object: Returns and Removes the top elements in this stack.
push(o: object): Adds new element to the top of this stack.

PROGRAM CODE:

```
import java.util.ArrayList;
class MyStack {
private ArrayList<Object> list = new ArrayList<>();
public boolean isEmpty() {
return list.isEmpty();
}
public int getSize() {
return list.size();
}
public Object peek() {
if (isEmpty()) {
return "Stack is empty";
}
return list.get(list.size() - 1);
}
public Object pop() {
if (isEmpty()) {
return "Stack is empty";
}
return list.remove(list.size() - 1);
}
public void push(Object o) {
list.add(o);
} }
public class P38 {
public static void main(String[] args) {
MyStack stack = new MyStack();
stack.push(10);
stack.push(20);
stack.push(30);
System.out.println("Top element is: " + stack.peek());
System.out.println("Popped element: " + stack.pop());
System.out.println("Popped element: " + stack.pop());
System.out.println("Is stack empty ? " + stack.isEmpty());
System.out.println("Current stack size: " + stack.getSize());
System.out.println("Top element now: " + stack.peek());
} }
```

OUTPUT:



```
Top element is: 30
Popped element: 30
Popped element: 20
Is stack empty ? false
Current stack size: 1
Top element now: 10
```

CONCLUSION:

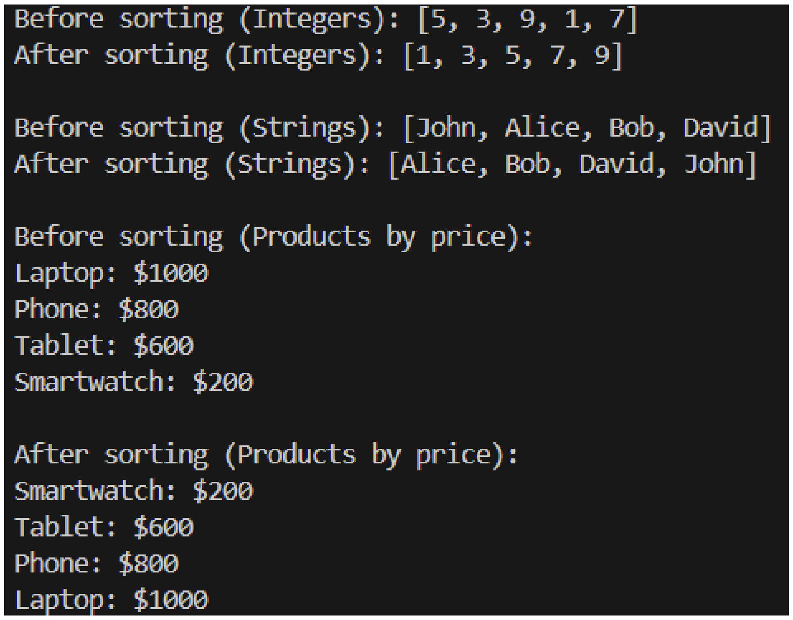
This program demonstrates the implementation of a custom stack using the ArrayList class in Java. It provides functionalities to push, pop, peek, check if the stack is empty, and get the current size of the stack. The program effectively showcases how to manage a dynamic collection of elements while adhering to stack principles.

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 CODE:

```
import java.util.Arrays;

public class P39 {

public static <T extends Comparable<T>> void sortArray(T[] array) {

Arrays.sort(array);

}

public static void main(String[] args) {

Integer[] numbers = {5, 3, 9, 1, 7};

System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));

sortArray(numbers);
System.out.println("After sorting (Integers): " + Arrays.toString(numbers));
String[] names = {"John", "Alice", "Bob", "David"};
System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));
sortArray(names);
System.out.println("After sorting (Strings): " + Arrays.toString(names));
Product[] products = {
new Product("Laptop", 1000),
new Product("Phone", 800),
new Product("Tablet", 600),
new Product("Smartwatch", 200)
};
System.out.println("\nBefore sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
}
sortArray(products);
System.out.println("\nAfter sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
} } }
class Product implements Comparable<Product> {
private String name;
private int price;

public Product(String name, int price) {

this.name = name;

this.price = price;

}

@Override

public int compareTo(Product other) {

return this.price - other.price;

}

@Override

public String toString() {

return name + ": $" + price;

} }
```

OUTPUT:



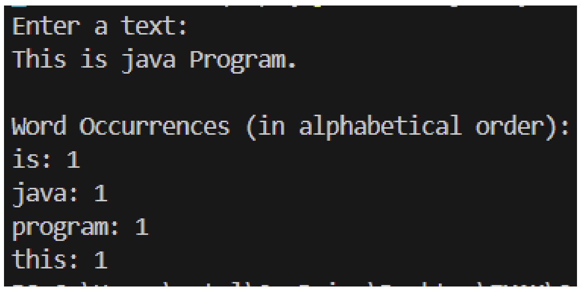
```
Before sorting (Integers): [5, 3, 9, 1, 7]
After sorting (Integers): [1, 3, 5, 7, 9]

Before sorting (Strings): [John, Alice, Bob, David]
After sorting (Strings): [Alice, Bob, David, John]

Before sorting (Products by price):
Laptop: $1000
Phone: $800
Tablet: $600
Smartwatch: $200

After sorting (Products by price):
Smartwatch: $200
Tablet: $600
Phone: $800
Laptop: $1000
```

	CONCLUSION: This program demonstrates the use of generics in Java to create a versatile sorting method for arrays of different types. By implementing the Comparable interface in the Product class, it enables sorting of custom objects based on specific criteria, such as price. The output shows the effective sorting of integers, strings, and products, highlighting the flexibility and reusability of the generic sorting method.

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 CODE:
	
	`import java.util.*;`
	`public class P40 {`
	`public static void main(String[] args) {`
	`Map<String, Integer> wordMap = new TreeMap<>();`
	`Scanner scanner = new Scanner(System.in);`
	`System.out.println("Enter a text:");`
	`String text = scanner.nextLine();`
	`String[] words = text.toLowerCase().split("\\W+");`
	`for (String word : words) {`
	`if (!word.isEmpty()) {`
	`wordMap.put(word, wordMap.getOrDefault(word, 0) + 1);`
	`} }`
	`System.out.println("\nWord Occurrences (in alphabetical order):");`
	`Set<Map.Entry<String, Integer>> entrySet = wordMap.entrySet();`
	`for (Map.Entry<String, Integer> entry : entrySet) {`
	`System.out.println(entry.getKey() + ": " + entry.getValue());`

```
} } }
```

OUTPUT:



```
Enter a text:
This is java Program.

Word Occurrences (in alphabetical order):
is: 1
java: 1
program: 1
this: 1
```

CONCLUSION:

This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of getOrDefault() simplifies the counting process, showcasing efficient word frequency analysis.

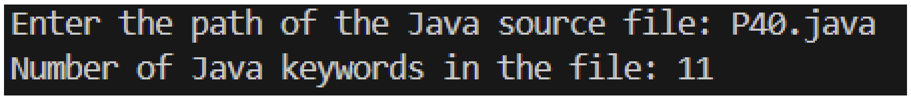
41 Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the contains () method to test if a word is in the keyword set.

PROGRAM CODE:

```
import java.io.*;

import java.util.*;

public class P41 {

private static final HashSet<String> keywords = new HashSet<>();

static {

String[] keywordArray = {

"abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",

"const", "continue", "default", "do", "double", "else", "enum", "extends", "final",

"finally", "float", "for", "goto", "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"
};
for (String keyword : keywordArray) {
keywords.add(keyword);
} }
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter the path of the Java source file: ");
String filePath = scanner.nextLine();
try {
File file = new File(filePath);
Scanner fileScanner = new Scanner(file);
int keywordCount = 0;
while (fileScanner.hasNext()) {
String word = fileScanner.next();
if (keywords.contains(word)) {
keywordCount++;
} }
System.out.println("Number of Java keywords in the file: " + keywordCount);
fileScanner.close();
```

```
} catch (FileNotFoundException e) {
System.out.println("File not found: " + filePath);
} } }
```

OUTPUT:



```
Enter the path of the Java source file: P40.java
Number of Java keywords in the file: 11
```

CONCLUSION:

This program demonstrates the use of a HashSet to efficiently count Java keywords in a source file. By reading each word from the file and checking for its presence in the set of keywords, it showcases how to utilize collections for rapid lookups. The result is the total number of keywords, providing a simple yet effective tool for analyzing Java code.