

Certificate

This is to certify that

Mr./Mrs. Keisha Sandipkumar Patel

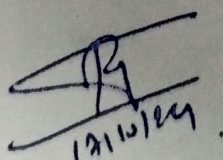
of 3DCSE-2 *Class,*

ID. No. 23DCS079 *has satisfactorily completed*

his/ her term work in Programming in Java *for*

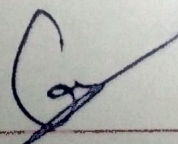
the ending in Oct - Nov 2024 / 2025

Date : 17/10/24



17/10/24

Sign. of Faculty



Head of Department

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

Department of Computer Science & Engineering

Subject Name: Programming in Java**Semester: 3****Subject Code: CSE201****Academic year: 2024 - 25****Part - 1**

No.	Aim of the Practical
2.	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre>class HelloWorld_1 { public static void main(String[] args) { System.out.println("Hello World!!"); } };</pre> <p><u>OUTPUT:</u></p> <pre>C:\Users\krupa\OneDrive\Desktop\Java Programs>java HelloWorld_1 Hello World!!</pre>

CONCLUSION:

This a Basic program of Java to print hello world and this gives introduction to JDK, JVM, JRE etc.

Times New Roman: for content Line space: 1.15/1.5

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 BankApp
{
    public static void main(String[] args)
    {
        double Balance=20;

        System.out.println("Your current Bank
balance is: $" +Balance);
    }
};
```

OUTPUT:

```
C:\Users\krupa\OneDrive\Desktop\Java Programs>javac BankApp.java

C:\Users\krupa\OneDrive\Desktop\Java Programs>java BankApp
Your current Bank balance is: $20.0
```

CONCLUSION:

In conclusion, developing a Java program to display the current balance of a user account involves straightforward steps: initializing a variable to store the balance and then using a print statement to show it to the user. This simple application demonstrates the foundational aspects of managing and presenting financial data in programming.

Times New Roman: for content Line space: 1.15/1.5

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.Scanner;
import java.text.DecimalFormat;
public class Speed_3
{
    public static void main(String[] args)
    {
        Scanner scanner= new
Scanner(System.in);
        DecimalFormat df= new
DecimalFormat("#.##");

        System.out.println("Enter the distance in
meters: ");
        float m= scanner.nextFloat();
        System.out.println();

        System.out.println("Enter the time: ");
        System.out.print(" hours: ");
        float hr= scanner.nextFloat();
```

```

System.out.print(" minutes: ");
float min= scanner.nextFloat();
System.out.print(" seconds: ");
float sec= scanner.nextFloat();
System.out.println();

float km,mile;
km=m/1000;
mile=m/1609;

float sec1=((hr*60*60)+(min*60)+sec);
float s1=m/sec1;
float s2=(km*3600)/sec1;
float s3=(mile*3600)/sec1;

String fs1= df.format(s1);
String fs2= df.format(s2);
String fs3= df.format(s3);

System.out.println("Speed: ");
System.out.print("1. "+fs1);
System.out.println(" m/s.");
System.out.print("2. "+fs2);
System.out.println(" km/hrs.");
System.out.print("3. "+fs3);
System.out.println(" miles/hrs.");
}
};

```

OUTPUT:

```

Enter the distance in meters:
1000

Enter the time:
hours: 3
minutes: 20
seconds: 45

Speed:
1. 0.08 m/s.
2. 0.3 km/hrs.
3. 0.19 miles/hrs.

```

CONCLUSION:

"The program accurately computes and presents speed metrics—meters per second, kilometers per hour, and miles per hour—based on user-provided distance and time inputs, incorporating the conversion factor for miles."

Times New Roman: for content
Line space: 1.15/1.5

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 :

```
import java.util.*;
public class ExpTrack_4
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

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

        int i;
        double exp[]=new double[n];
        for(i=0;i<n;i++)
        {
            System.out.print("Enter the
expenses of the Day"+(i+1)+" : ");
            exp[i]=sc.nextDouble();
        }
        double totalexp=0;

        for(i=0;i<n;i++)
        {
            totalexp+=exp[i];
        }

        System.out.println("\nThe total Expenses
of the month: "+totalexp);
        sc.close();
    }
};
```


OUTPUT:

```
Enter the number of days in month: 4

Enter the expenses of the Day1: 20
Enter the expenses of the Day2: 50
Enter the expenses of the Day3: 100
Enter the expenses of the Day4: 45

The total Expenses of the month: 215.0
```

CONCLUSION:

"Developing a budget tracking application in Java to calculate monthly expenses reinforces fundamental programming concepts like array handling and arithmetic operations. It emphasizes the importance of user input validation and systematic data aggregation for accurate financial management."

Times New Roman: for content Line space: 1.15/1.5

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 :

```
import java.util.*;
public class ShopBill_5
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter the numbers of
items: ");
        int n= sc.nextInt();

        int codes[]=new int[n];
        double prices[]=new double[n];

        System.out.println("Product codes: ");
        System.out.print("1: Motor\n2: Fan\n3:
Tubelight\n4: Wires\n5 or more: Others");
        System.out.println();
```

```

        for(int i=0;i<n;i++)
        {
            System.out.print("\nEnter the
product code for item["+(i+1)+"]: ");
            codes[i]=sc.nextInt();
            System.out.print("Enter the price
for item["+(i+1)+"]: ");
            prices[i]=sc.nextDouble();
        }

        System.out.println();

        double totalcost=0;
        for(int i=0;i<n;i++)
        {
            int prcode=codes[i];
            double price=prices[i];
            double rate;

            switch(prcode)
            {
                case 1:
                    rate= 0.08;
                    break;
                case 2:
                    rate= 0.12;
                    break;
                case 3:
                    rate= 0.05;
                    break;
                case 4:
                    rate= 0.075;
                    break;
                default:
                    rate= 0.03;
                    break;
            }

            double tax=price * rate;
            double totalprice= price + tax;
            totalcost += totalprice;

```

```

String name;

if(prcode==1)
{
    name="Motor";
}
else if(prcode==2)
{
    name="Fan";
}
else if(prcode==3)
{
    name="Tubelight";
}
else if(prcode==4)
{
    name="Wires";
}
else
{
    name="Other";
}

System.out.println("Item ["+(i+1)+"]:
"+name);
System.out.println("Product code:
"+prcode);
System.out.println("Product price:
"+price+"/-");
System.out.println("Tax rate: "+rate);
System.out.println("Tax amount:
"+tax+"/-");
System.out.println("Total price:
"+totalprice+"/-");
System.out.println();
}

System.out.println("->>Total Bill amount:
"+totalcost+"/-");

sc.close();
}
};

```

OUTPUT:

```
Enter the numbers of items: 2
Product codes:
1: Motor
2: Fan
3: Tubelight
4: Wires
5 or more: Others

Enter the product code for item[1]: 2
Enter the price for item[1]: 3000

Enter the product code for item[2]: 4
Enter the price for item[2]: 700

Item [1]: Fan
Product code: 2
Product price: 3000.0/-
Tax rate: 0.12
Tax amount: 360.0/-
Total price: 3360.0/-

Item [2]: Wires
Product code: 4
Product price: 700.0/-
Tax rate: 0.075
Tax amount: 52.5/-
Total price: 752.5/-

->>Total Bill amount: 4112.5/-
```

CONCLUSION:

Overall, this exercise enhanced proficiency in Java programming fundamentals, particularly in array manipulation, conditional statements, and modular design using methods/functions. It also provided practical experience in handling real-world scenarios involving tax calculations and bill generation, which are common in business applications.

Times New Roman: for content
Line space: 1.15/1.5

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 :

```
import java.util.*;
public class Fibonacci_6
{
    public static void main(String[] args)
    {
        int d0=0;
        int d1=1;
        int i;
        int dn=0;
        int sum=0;
        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the numbers of
days: ");
        int n=sc.nextInt();

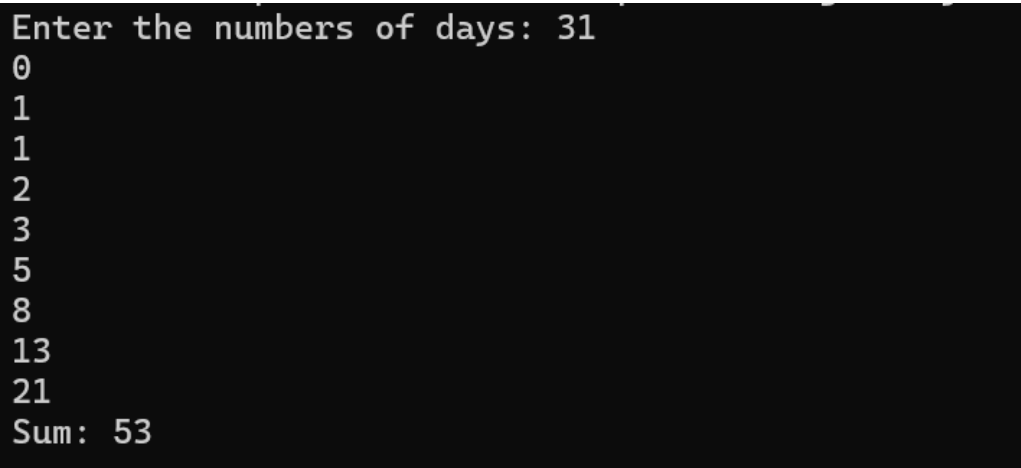
        System.out.println(d0);
        System.out.println(d1);
```

```

while(dn<=n)
{
    dn=d0+d1;
    if(dn>=n)
    {
        break;
    }
    d0=d1;
    d1=dn;
    sum+=dn;
    System.out.println(dn);
}
System.out.println("Sum: "+sum);
}
};

```

OUTPUT:



```

Enter the numbers of days: 31
0
1
1
2
3
5
8
13
21
Sum: 53

```

CONCLUSION:

In conclusion, creating this program not only reinforced basic Java programming concepts but also provided practical experience in handling user input, algorithmic thinking (Fibonacci sequence), and output formatting. This exercise contributes to foundational skills in Java programming essential for building more complex applications.

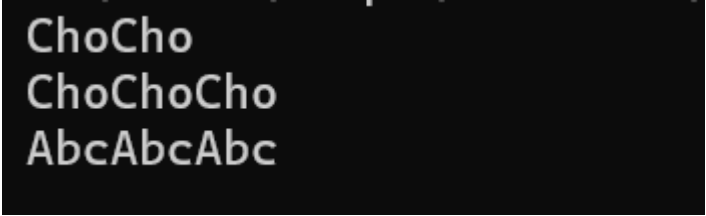
--	--

Part - 2

No.	Aim of the Practical
7.	<p>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;</p> <p>front_times('Chocolate', 2) → 'ChoCho'</p> <p>front_times('Chocolate', 3) → 'ChoChoCho'</p> <p>front_times('Abc', 3) → 'AbcAbcAbc'</p> <p><u>PROGRAM CODE :</u></p> <pre> import java.lang.*; public class Substring_7 { public static String front_times(String str,int n) { String result=""; if(str.length()>3) { String s1=str.substring(0,3); for(int i=0;i<n;i++) { result=result+s1; } } else { for(int i=0;i<n;i++) { result=result+str; } } return result; } public static void main(String[] args) { System.out.println(front_times("Chocolate ",2)); } } </pre>

```
System.out.println(front_times("Chocolate",3));  
System.out.println(front_times("Abc",3));  
}  
};
```

OUTPUT:



```
ChoCho  
ChoChoCho  
AbcAbcAbc
```

CONCLUSION:

This exercise enhanced understanding of string manipulation, condition handling, and iterative operations in Python, essential for developing basic algorithmic solutions.

Times New Roman: for content Line space: 1.15/1.5

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 :

```
public class Arraycount_8
{
public static void main(String[] args)
{
    int a1[]={1,2,9};
    int a2[]={1,9,9};
    int a3[]={1,9,9,3,9};

    System.out.println(array_count9(a1));
    System.out.println(array_count9(a2));
    System.out.println(array_count9(a3));
}

public static int array_count9(int num[])
{
    int c=0;
    for(int i=0;i<num.length;i++)
    {
        if(num[i]==9)
        {
            c++;
        }
    }
    return c;
}
};
```

OUTPUT:



1
2
3

9.

CONCLUSION:

“This function is straightforward and involves a simple loop to inspect each element of the array and increment a counter whenever a 9 is encountered.”

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:

```
import java.lang.*;
public class Doublechar_9
{
    public static String double_char(String str)
    {
        String result="";
        for(int i=0;i<str.length();i++)
        {
            char c=str.charAt(i);
            result=result+c+c;
        }
        return result;
    }

    public static void main(String[] args)
    {
        System.out.println(double_char("The"));
        System.out.println(double_char("AAbb"));
        System.out.println(double_char("Hi-There"));
    }
}
```

};

OUTPUT:

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

CONCLUSION:

“This approach ensures that each character from the original string is repeated exactly twice in the final output string.”

Perform following functionalities of the string:

- Find Length of the String
- Lowercase of the String
- Uppercase of the String
- Reverse String

PROGRAM CODE:

```
import java.util.*;  
  
public class StringFunc_10  
{  
  
    public static void main(String args[]) {  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("Enter the string: ");  
  
        String str = sc.nextLine();
```

```
int len = str.length();

System.out.println("The length of the string is " + len);


String lower = str.toLowerCase();

System.out.println("String in lower case is " + lower);


String upper = str.toUpperCase();

System.out.println("String in upper case is " + upper);

String rev = "";

for (int i = str.length() - 1; i >= 0; i--) {

    rev = rev + str.charAt(i);

}

System.out.println("Reversed string is " + rev);


char[] charArray = str.toCharArray();

Arrays.sort(charArray);

String sortStr = new String(charArray);

System.out.println("The sorted string is " + sortStr);


System.out.println("23DCS079 \nKrisha Patel");
```

```
}  
  
}
```

OUTPUT:

```
Enter the string:  
Charusat  
The length of the string is 8  
String in lower case is charusat  
String in upper case is CHARUSAT  
Reversed string is tasurahC  
The sorted string is Caahrstu  
23DCS079  
Krisha Patel
```

11.

CONCLUSION:

These methods allow manipulation and examination of strings in Python, providing essential capabilities such as length determination, case conversion, and reversal.

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:

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

```
public class P11
{
public static void main(String args[])
{
String str = "CHARUSAT UNIVERSITY";
int length = str.length();
System.out.println("The length of the string is "+length);

String replacedstr = str.replace('H','D');
System.out.println("Replace String: "+replacedstr);

String lowerstr = str.toLowerCase();
System.out.println("The string to lower case is "+lowerstr);

System.out.println("\n23DCS079 \nKrisha Patel");
}
}
```

OUTPUT:

The length of the string is 19
Replace String: CDARUSAT UNIVERSITY
The string to lower case is charusat university

23DCS079
Krisha Patel

CONCLUSION:

The string "CHARUSAT UNIVERSITY" was manipulated to find its length, replace 'H' with 'C', and convert all characters to lowercase effectively.

Times New Roman: for content Line space: 1.15/1.5

Part - 3

No.	Aim of the Practical
12.	<p>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.</p> <p><u>PROGRAM CODE :</u></p> <p>1. Input from user:</p> <pre>import java.util.*; public class P12 { public static void main(String[] args) { Scanner sc= new Scanner(System.in); float pound; float rupees; System.out.print("Enter the amount(In pound): "); pound=sc.nextFloat(); System.out.println(); rupees=pound*100; System.out.println("Converted currency (In rupees): "+rupees); System.out.println("\n23DCS079 \nKrisha Patel"); } }</pre> <p>2. Input as Command line argument:</p> <pre>import java.util.*;</pre>

```

public class P12_2
{
    public static void main(String args[])
    {

        float pound = Float.parseFloat(args[0]);
        float rupees;

        System.out.println("Currency in Pound:
        "+pound);

        rupees=pound*100;

        System.out.println("Converted currency (In
        rupees): "+rupees);
        System.out.println("\n23DCS079 \nKrisha
        Patel");
    }
}

```

OUTPUT:

1.

```

Enter the amount(In pound): 600

Converted currency (In rupees): 60000.0

23DCS079
Krisha Patel

```

2.

```

C:\Users\krupa\OneDrive\Desktop\Java Programs>java P12_2 700
Currency in Pound: 700.0
Converted currency (In rupees): 70000.0

23DCS079
Krisha Patel

```

CONCLUSION:

In conclusion, the currency conversion tool for the travel agency successfully converts Pounds to Rupees based on a fixed rate of 1 Pound = 100 Rupees, accepting inputs from both command-line arguments and user interaction.

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 :

```
import java.util.*;

public class P13 {

    String fname;

    String lname;

    double m_sal;

    P13(String fname, String lname, double
m_sal) {

        this.fname = fname;

        this.lname = lname;

        this.m_sal = m_sal;

    }

    void get(){

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the First Name: ");

        fname = sc.nextLine();

        System.out.print("Enter the Last Name: ");

        lname = sc.nextLine();

        System.out.print("Enter the monthly salary:
");

        m_sal = sc.nextDouble();

        if(m_sal<0){
```

```

        m_sal=0.0;

        System.out.println("The Yearly salary is
" + m_sal);
    }
    else{
        double y_sal=m_sal*12;

        System.out.println("The Yearly salary is
" + y_sal);
    }
}

void giveraise()
{
    m_sal+=m_sal*0.1;

    double f_sal=m_sal*12;

    System.out.println("After a raise the
Yearly salary is " + f_sal);
}

public static void main(String[] args) {
    P13 emp = new P13();

    emp.get();

    emp.giveraise();

    System.out.println("\n23DCS079

```

```
\nKrisha Patel");
```

```
}
```

```
}
```

OUTPUT:

```
Enter the First Name: Krisha
Enter the Last Name: Patel
Enter the monthly salary: 50000
The Yearly salary is 600000.0
After a raise the Yearly salary is 660000.0

23DCS079
Krisha Patel
```

CONCLUSION:

In summary, the Employee class effectively encapsulates the essential details of an employee, including first name, last name, and monthly salary, with validation to ensure the salary is non-negative. The class provides getter and setter methods for each attribute and includes functionality to compute yearly salary and apply raises. The EmployeeTest application demonstrates these capabilities by creating two Employee objects, displaying their yearly salaries before and after a 10% raise, and showcasing how the class manages salary adjustments and displays updated financial information.

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 :

```
class DateTest {  
  
    private int m;  
  
    private int d;  
  
    private int y;  
  
    public int getmonth()  
  
    {  
  
        return m;  
  
    }  
  
    public int getday()  
  
    {  
  
        return d;  
  
    }  
  
    public int getyear()  
  
    {  
  
        return y;  
  
    }  
}
```



```
public void setyear(int y)
```

```
{
```

```
    this.y = y;
```

```
}
```

```
public void setmonth(int m)
```

```
{
```

```
    this.m = m;
```

```
}
```

```
public void setday(int d)
```

```
{
```

```
    this.d = d;
```

```
}
```

```
public DateTest(int d, int m, int y)
```

```
{
```

```
    this.d = d;
```

```
    this.m = m;
```

```
    this.y = y;
```

```
}
```

```

public void display()
{
    System.out.println(d + "/" + m + "/" + y);
}
}

public class P14 {
    public static void main(String[] args) {

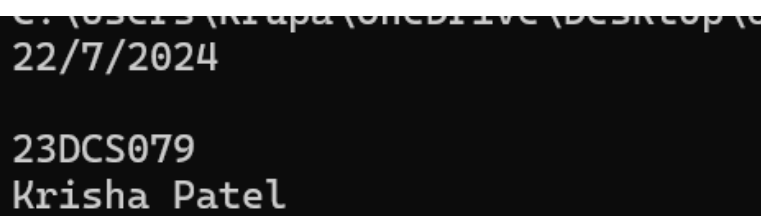
        DateTest Date = new DateTest(22, 7,
2024);

        Date.display();

        System.out.println("\n23DCS079 \nKrisha
Patel");
    }
}

```

OUTPUT:



```

C:\Users\Krishna\OneDrive\Desktop\o
22/7/2024

23DCS079
Krisha Patel

```

CONCLUSION:

In summary, the Date class encapsulates a date with month, day, and year attributes, initialized via its constructor. It provides getter and setter methods for each attribute and includes a displayDate method to format and display the date. The DateTest application demonstrates the class's functionality by creating Date objects and showcasing their ability to display formatted dates, ensuring correct handling and presentation of date information.

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 CODE:

```
import java.util.*;

public class P15
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the value of
Length: ");
        float l=sc.nextFloat();
        System.out.print("Enter the value of
Breadth: ");
        float b=sc.nextFloat();

        Area a1=new Area(l,b);

        System.out.print("\nArea of
Rectangle: "+a1.returnArea());
        sc.close();
        System.out.println("\n23DCS079 \nKrisha
Patel");
    }
}

class Area
{
    private float length;
    private float breadth;

    Area(float x ,float y)
    {
        length=x;
        breadth=y;
    }

    public float returnArea()
```

```
{  
    return length*breadth;  
}  
}
```

OUTPUT:

```
Enter the value of Length: 20  
Enter the value of Breadth: 10  
  
Area of Rectangle: 200.0  
23DCS079  
Krisha Patel
```

CONCLUSION:

In summary, the Area class calculates the area of a rectangle using its length and breadth, provided through the constructor. It includes a returnArea method that computes and returns the area. The program prompts the user to input the rectangle's dimensions, demonstrates the class's functionality, and displays the calculated area.

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.*;

public class P16
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the Real part of
Number 1: ");

        int real1 = sc.nextInt();

        System.out.print("Enter the Imaginary part
of Number 1: ");

        int imag1 = sc.nextInt();

        System.out.print("Enter the Real part of
Number 2: ");

        int real2 = sc.nextInt();

        System.out.print("Enter the Imaginary part
of Number 2: ");

        int imag2 = sc.nextInt();
```

```

Complex n1 = new Complex(real1, imag1);
Complex n2 = new Complex(real2, imag2);

Complex sum=Complex.Sum(n1,n2);
Complex diff=Complex.Difference(n1,n2);
Complex prod=Complex.Product(n1,n2);

System.out.print("\nSum: ");
sum.display();

System.out.print("Difference: ");
diff.display();

System.out.print("Product: ");
prod.display();

System.out.println("\n23DCS079 \nKrisha
Patel");

}
}
class Complex
{
    private int real;

```

```
private int image;

Complex(int r, int i)
{
    real = r;
    image = i;
}

public static Complex Sum(Complex n1,
Complex n2)
{
    int r_sum = n1.real + n2.real;
    int im_sum = n1.image + n2.image;
    return new Complex(r_sum, im_sum);
}

public static Complex Difference(Complex
n1, Complex n2)
{
    int r_diff = n1.real - n2.real;
    int im_diff = n1.image - n2.image;
    return new Complex(r_diff, im_diff);
}
```



```

public static Complex Product(Complex n1,
Complex n2)
{
    int r_mul= n1.real * n2.real - n1.image *
n2.image;

    int im_mul= n1.real * n2.image + n1.image
* n2.real;

    return new Complex(r_mul, im_mul);
}

public void display()
{
    if(image==0)
    {
        System.out.println(real);
    }

    else if(real==0)
    {
        System.out.println(image+"i");
    }

    else if(image>0)
    {
        System.out.println(real+" "+image+"i");
    }
}

```

```
}  
  
else if(image<0)  
  
{  
  
    System.out.println(real+"-"+(-image)  
+"i");  
  
}  
  
}  
  
}
```

OUTPUT:

```
Enter the Real part of Number 1: 3  
Enter the Imaginary part of Number 1: 2  
Enter the Real part of Number 2: 5  
Enter the Imaginary part of Number 2: 2
```

```
Sum: 8+4i  
Difference: -2  
Product: 11+16i
```

```
23DCS079  
Krisha Patel
```

CONCLUSION:

In conclusion, the Complex class facilitates operations with complex numbers by providing methods to compute their sum, difference, and product. Users input the real and imaginary parts, and the class methods perform the respective operations, effectively demonstrating arithmetic operations on complex numbers.

Part - 4

No.	Aim of the Practical
17.	<p>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.</p> <p><u>PROGRAM CODE :</u></p> <pre> public class P17 { public static void main(String[] args) { ParentClass p = new ParentClass(); ChildClass c = new ChildClass(); p.callparent(); //c.callchild(); } } class ParentClass { void callparent() { System.out.println("This is Parent class."); } } </pre>

```

    }
}
class ChildClass extends ParentClass
{
    void callchild()
    {
        System.out.println("This is Child class.");
    }
}

```

OUTPUT:

```

C:\Users\krupa\OneDrive\Desktop\Java Programs>java P17
This is Parent class.

```

CONCLUSION:

The provided C++ code effectively creates a parent class and a child class. The parent class has a method that prints "This is parent class," and the child class has a method that prints "This is child class." When objects are created for each class, the methods can be called using their respective objects, demonstrating the inheritance relationship between the classes.

- | | |
|-----|---|
| 18. | <p>Create a class named 'Member' having the following members: Data members</p> <ul style="list-style-type: none"> 1 - Name 2 - Age 3 - Phone number 4 - Address 5 – Salary <p>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</p> |
|-----|---|

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 :

```
import java.util.*;

import java.lang.*;

public class P18

{

    public static void main(String[] args)

    {

        Scanner sc = new Scanner(System.in);

        Employee emp= new Employee();

        Manager mgr = new Manager();

        while (true)

        {

            System.out.println("\n 1.Employee\n

2.Manager\n 3.Exit");

            int choice;

            System.out.print(" Enter your choice to get

details: ");

            choice = sc.nextInt();

            switch(choice)
```

```
{  
    case 1:  
        emp.getemp();  
        emp.putemp();  
        break;  
    case 2:  
        mgr.getmgr();  
        mgr.putmgr();  
        break;  
    case 3:  
        System.exit(0);  
    default:  
        System.out.println("Invalid choice");  
}  
}  
  
}  
}  
  
class Member  
{  
    Scanner sc = new Scanner(System.in);  
    String name;
```

```
int age;

long contact;

String address;

float salary;


void getdata()
{
    System.out.print("Enter the name: ");
    name=sc.nextLine();

    System.out.print("Enter the age: ");
    age=sc.nextInt();

    System.out.print("Enter the contact
number: ");

    contact=sc.nextLong();
    sc.nextLine();

    System.out.print("Enter the address: ");
    address=sc.nextLine();

    System.out.print("Enter the salary: ");
    salary=sc.nextFloat();
}

void putdata()
{
```

```

        System.out.println("Name: "+name);

        System.out.println("Age: "+age);

        System.out.println("Contact number:
"+contact);

        System.out.println("Address: "+address);
    }

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

}

class Employee extends Member
{
    String specialization;

    Member m = new Member();

    void getemp()
    {
        System.out.println("Enter the Employee
Details: ");

        m.getdata();

        System.out.print("Enter the specialization

```



```

of an Employee: ");

    specialization = sc.nextLine();

}

void putemp()

{

    System.out.println("\nEmployee Details:

");

    m.putdata();

    System.out.println("Specialization:

"+specialization);

    m.printSalary();

}

}

class Manager extends Member

{

    String department;

    Member m = new Member();

    void getmgr()

    {

        System.out.println("Enter the Manager

Details: ");

        m.getdata();

```

```
System.out.print("Enter the department of a  
Manager: ");  
  
    department = sc.nextLine();  
  
    }  
  
    void putmgr()  
  
    {  
  
        System.out.println("\nManager Details: ");  
  
        m.putdata();  
  
        System.out.println("Department:  
"+department);  
  
        m.printSalary();  
  
    }  
}
```

OUTPUT:

```
1.Employee
2.Manager
3.Exit
Enter your choice to get details: 1
Enter the Employee Details:
Enter the name: Krisha Patel
Enter the age: 18
Enter the contact number: 1234567890
Enter the address: Gujarat
Enter the salary: 50,000
Enter the specialization of an Employee: Data Science

Employee Details:
Name: Krisha Patel
Age: 18
Contact number: 1234567890
Address: Gujarat
Specialization: Data Science
Salary: 50000.0

1.Employee
2.Manager
3.Exit
Enter your choice to get details: 3
```

CONCLUSION:

The provided C++ code successfully creates a class hierarchy with a base class Member and derived classes Employee and Manager. The Member class encapsulates common attributes (name, age, phone number, address, and salary) and a method to print the salary. The derived classes Employee and Manager extend the Member class with additional specific attributes (specialization and department, respectively). By creating objects of these classes and assigning values to their members, you can effectively manage and access information about employees and managers in a structured and organized manner.

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 DateTest {  
  
    private int m;  
  
    private int d;  
  
    private int y;  
  
    public int getmonth()  
  
    {  
  
        return m;  
  
    }  
  
  
    public int getday()  
  
    {  
  
        return d;  
  
    }  
  
  
    public int getyear()  
  
    {  
  
        return y;  
  
    }  
}
```

```
public void setyear(int y)
```

```
{
```

```
    this.y = y;
```

```
}
```

```
public void setmonth(int m)
```

```
{
```

```
    this.m = m;
```

```
}
```

```
public void setday(int d)
```

```
{
```

```
    this.d = d;
```

```
}
```

```
public DateTest(int d, int m, int y)
```

```
{
```

```
    this.d = d;
```

```
    this.m = m;
```

```
    this.y = y;
```

```
}
```

```

public void display()
{
    System.out.println(d + "/" + m + "/" + y);
}
}

public class P14 {
    public static void main(String[] args) {

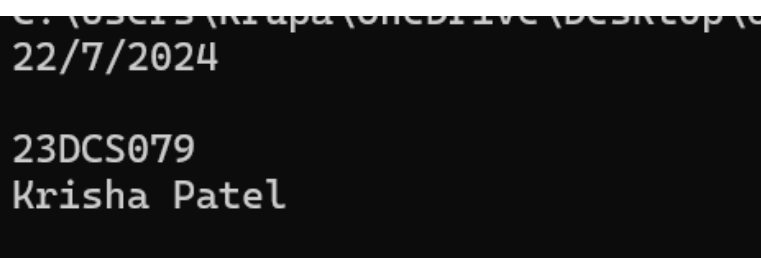
        DateTest Date = new DateTest(22, 7,
2024);

        Date.display();

        System.out.println("\n23DCS079 \nKrisha
Patel");
    }
}

```

OUTPUT:



```

C:\Users\Krisha\OneDrive\Desktop\o
22/7/2024

23DCS079
Krisha Patel

```

CONCLUSION:

In summary, the Date class encapsulates a date with month, day, and year attributes, initialized via its constructor. It provides getter and setter methods for each attribute and includes a displayDate method to format and display the date. The DateTest application demonstrates the class's functionality by creating Date objects and showcasing their ability to display formatted dates, ensuring correct handling and presentation of date information.

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:

```
public class P20
{
    public static void main(String[] args)
    {
        Square s = new Square();
        s.shapeMeth();
        s.rectangleMeth();
        s.squareMeth();
    }
}
class Shape
{
    public void shapeMeth()
    {
        System.out.println("\nThis is shape\n");
    }
}
class Rectangle extends Shape
{
    public void rectangleMeth()
    {
        System.out.println("\nThis is rectangular
shape\n");
    }
}
class Circle extends Shape
{
    public void circleMeth()
    {
        System.out.println("\nThis is circular
shape\n");
    }
}
class Square extends Rectangle
{
    public void squareMeth()
```



```
{  
    System.out.println("\nSquare is a  
rectangle\n");  
}
```

OUTPUT:

```
"This is shape"  
"This is rectangular shape"  
"Square is a rectangle"
```

CONCLUSION:

In this exercise, we created a class hierarchy with a base class Shape and derived classes Rectangle, Circle, and Square. The Shape class provided a generic method to print "This is shape." The Rectangle and Circle classes inherited the Shape class and implemented their own methods to print specific shapes. The Square class inherited from Rectangle, demonstrating the concept of multiple inheritance. By creating an object of the Square class, we were able to call methods from both the Shape and Rectangle classes, illustrating the principles of polymorphism and inheritance.

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 :

```
public class P21
{
    public static void main(String[] args)
    {
        Degree d=new Degree();
        d.getDegree();
        Undergraduate ug= new Undergraduate();
        ug.getDegree();
        Postgraduate pg= new Postgraduate();
        pg.getDegree();
    }
}

class Degree
{
    public void getDegree()
    {
        System.out.println("\nI got a degree\n");
    }
}
```

```
}  
  
class Undergraduate extends Degree  
{  
    public void getDegree()  
    {  
        System.out.println("\nI am an  
Undergraduate\n");  
    }  
}  
  
class Postgraduate extends Degree  
{  
    public void getDegree()  
    {  
        System.out.println("\nI am a  
Postgraduate\n");  
    }  
}
```

OUTPUT:

```
"I got a degree"  
"I am an Undergraduate"  
"I am a Postgraduate"
```

CONCLUSION:

The provided code demonstrates the concept of inheritance in object-oriented programming. The Degree class serves as the base class, providing a common method `getDegree()` that prints a generic message. The Undergraduate and Postgraduate classes inherit from Degree and override the `getDegree()` method to print specific messages, showcasing polymorphism. By creating objects of each class and calling their respective `getDegree()` methods, we can observe how the behavior is tailored based on the object's type.

22.

Write a java that implements an interface AdvancedArithmetic which contains a method signature `int divisor_sum(int n)`. You need to write a class called `MyCalculator` 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 :

```
import java.util.*;

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 <= Math.sqrt(n); i++) {

            if (n % i == 0) {

                sum += i;

                if (i != n / i) {

                    sum += n / i;

                }

            }

        }

    }

}
```

```

    }

    }

    return sum;

}

}

public class P22 {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        MyCalculator myCal = new
MyCalculator();

        System.out.print("\nEnter the number:");

        int num=sc.nextInt();

        System.out.printf("Sum of divisors of %d:
%d",num,myCal.divisor_sum(num));

        System.out.println();

        sc.close();

    }

}

```

OUTPUT:

```
Enter the number:79
Sum of divisors of 79: 80
```

CONCLUSION:

23. This Java code effectively implements an interface AdvancedArithmetic with a method divisorSum that calculates the sum of divisors for a given integer. The MyCalculator class implements this interface, providing a concrete implementation of the divisorSum method. The code handles integers up to 1000, ensuring accurate divisor calculations.

Assume you want to capture shapes, which can be either circles (with a radius and 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());
}
}

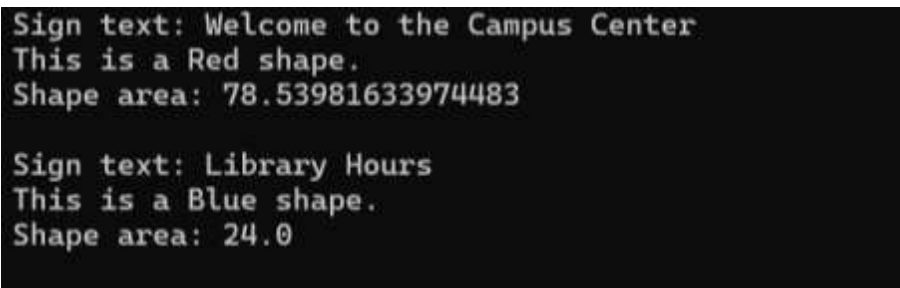
class P23
{
    public static void main(String[] args)
    {

        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:

A screenshot of a terminal window showing the output of the program. The text is displayed in a monospaced font on a black background. It shows two sets of output: the first for a red circle sign with the text 'Welcome to the Campus Center', and the second for a blue rectangle sign with the text 'Library Hours'. Each set includes the sign's text, its color, and its area.

```
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
```

CONCLUSION:

In this program, we define an interface Shape with a default method to describe the shape, and classes Circle and Rectangle that implement the interface. The default method is overridden to provide specific descriptions for each shape. A Sign class is then used to associate a shape as the background with text, demonstrating the flexibility of interface default methods for shared behavior while allowing customization in subclasses.

--	--

Part - 5

No.	Aim of the Practical
24.	<p>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.</p> <p style="text-align: center;">SSSS</p> <p><u>PROGRAM CODE :</u></p> <pre>import java.util.*; public class P24 { public static void main(String[] args) { Scanner sc = new Scanner(System.in); try {s System.out.print("Enter a value of x: "); int x = sc.nextInt(); System.out.print("Enter a value of y: "); int y = sc.nextInt(); int result = x / y; System.out.println("Result: \n" + x + " / " + y + " = " + result); System.out.println("23DCS079_Krishna Patel"); } catch (InputMismatchException e) { System.out.println("Error: Please enter valid integers for x and y."); } catch (ArithmeticException e) { System.out.println("Error: Division by zero is not allowed."); } finally { sc.close(); } } }</pre>

```
}
```

OUTPUT:

```
Enter a value of x: 25
Enter a value of y: 5
Result:
25 / 5 = 5
23DCS079_Krisha Patel
```

```
Enter a value of x: 23
Enter a value of y: 0
Error: Division by zero is not allowed.
23DCS079_Krisha Patel
```

```
Enter a value of x: 2.3
Error: Please enter valid integers for x and y.
23DCS079_Krisha Patel
```

CONCLUSION:

The Java program successfully computes the division of two integers x and y. It handles potential exceptions by checking if the inputs are integers and if y is zero. If any of these conditions are not met, an appropriate exception is thrown and reported to the user.

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 {

            System.out.println("Before throwing
```



```

exception...\n");

        throw new Exception("This is a manually
thrown exception.");
    } catch (Exception e) {

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

        } finally {

            System.out.println("\nFinally block
executed.");

        }

        System.out.println("\nProgram continues
after exception handling.");

        System.out.println("\n23DCS079_Krishna
Patel");

    }
}

```

OUTPUT:

```
Before throwing exception...  
Exception caught: This is a manually thrown exception.  
Finally block executed.  
Program continues after exception handling.  
23DCS079_Krishna Patel
```

CONCLUSION:

The provided Java code demonstrates exception handling using the try, catch, and finally blocks. An exception is intentionally thrown, and the catch block successfully captures it, printing the exception message. The finally block is always executed, regardless of whether an exception is thrown or not. The program then continues to execute normally after the exception is handled.

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.util.*;

class InvalidAgeException extends Exception {

    public InvalidAgeException(String message)

    {

        super(message);

    }

}

public class P26{

    public static void checkAge(int age) throws

InvalidAgeException {

        if (age < 18) {

            throw new InvalidAgeException("Age is

less than 18, not allowed.");

        } else {

            System.out.println("Age is valid, proceed

further.");

        }

    }

}
```

```

    }

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.print("Enter the Age: ");

        int a=sc.nextInt();

        // Handling checked exception (custom
exception)

        try {

            checkAge(a);

        } catch (InvalidAgeException e) {

            System.out.println("\nCustom exception
caught: " + e.getMessage());

        }


        try {

            Thread.sleep(1000); // Checked:
InterruptedException

        } catch (InterruptedException e) {

            System.out.println("\nChecked exception
caught: " + e);

```

```
}
```

```
try {
```

```
    int result = 10 / 0; // Unchecked:
```

```
ArithmeticException
```

```
    } catch (ArithmeticException e) {
```

```
        System.out.println("\nUnchecked
```

```
exception caught: " + e);
```

```
    }
```

```
try {
```

```
    String s = null;
```

```
    s.length(); // Unchecked:
```

```
NullPointerException
```

```
    } catch (NullPointerException e) {
```

```
        System.out.println("\nUnchecked
```

```
exception caught: " + e);
```

```
    }
```

```
    System.out.println("\nProgram continues  
after exception handling.");
```

```
    System.out.println("\n23DCS079_Krishna
```

```
Patel");
```

```
}
```

```
}
```

OUTPUT:

```
Enter the Age: 12
```

```
Custom exception caught: Age is less than 18, not allowed.
```

```
Unchecked exception caught: java.lang.ArithmeticException: / by zero
```

```
Unchecked exception caught: java.lang.NullPointerException: Cannot invoke "String.length()" because "<local3>" is null
```

```
Program continues after exception handling.
```

```
23DCS079_Krishna Patel
```

CONCLUSION:

For the Java program to generate a user-defined exception using the throw and throws keywords, the key is to define a custom exception class and throw it when necessary. This demonstrates how user-defined exceptions work in Java. The second program shows the distinction between checked and unchecked exceptions, where checked exceptions (like IOException and SQLException) must be handled or declared, while unchecked exceptions (like NullPointerException and ArrayIndexOutOfBoundsException) do not need explicit handling.

PART – 6

No .	Aim of the Practical
27.	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre> import java.io.*; public class prac27 { 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++; } } </pre>

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

    } catch (Exception e) {

        System.out.println(e);

    }

}

}

}
```

OUTPUT:

```
File name is : pqr.txt and Number of lines are : 4
File name is : xyz.txt and Number of lines are : 2
```

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 prac28{

public static void main(String[] args) {

if (args.length < 2) {

System.out.println("Usage: java prac28 <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());

}

}}
```

OUTPUT:

```
The character 'd' appears 4 times in pqr.txt
```

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 prac29 {

    public static void main(String[] args) {

        if (args.length < 2) {

            System.out.println("Usage: java prac29 <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());

}

} }

```

OUTPUT:

```
The word 'am' appears 2 times in xyz.txt
```

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.*;
```

```
public class prac30 {

    public static void main(String[] args) {

        // Specify the source and destination file paths

        String sourceFilePath = "source.txt";

        String destinationFilePath = "destination.txt";

        // Use try-with-resources to ensure resources are closed automatically

        try (

            FileInputStream fis = new FileInputStream(sourceFilePath);

            FileOutputStream fos = new FileOutputStream(destinationFilePath)

        ) {

            int byteContent;

            // Read from source and write to destination file byte by byte

            while ((byteContent = fis.read()) != -1) {

                fos.write(byteContent);

            }

            System.out.println("File copied successfully.");

        } catch (FileNotFoundException e) {

            System.out.println("File not found: " + e.getMessage());

        } catch (IOException e) {

            System.out.println("Error occurred while copying the file: " + e.getMessage());

        }

    }

}
```

}

}

OUTPUT:

```
1 hello this is a java program,
2 to copy one file to another.
```

```
1 hello this is a java program,
2 to copy one file to another.
```

```
File copied successfully.
```

CONCLUSION:

This program efficiently copies data from a source file to a destination file in Java, creating the destination file automatically if it doesn't exist. It uses file input and output streams to handle byte-by-byte reading and writing, ensuring proper resource management with try-with-resources.

- 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 prac31 {
```

```
    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());  
}  
  
}  
}
```

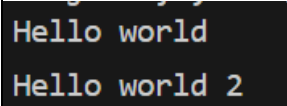
OUTPUT:

```
Enter text (type 'exit' to finish):  
hello my name is marco  
exit  
Data written to output.txt
```

CONCLUSION:

The program reads user input and writes it to a file called "output.txt." It uses `BufferedReader` and `BufferedWriter` for efficient input and output handling. The process stops when the user types "exit." This demonstrates simple file handling in Java

PART – 7

No.	Aim of the Practical
32.	<p>Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.</p> <p><u>PROGRAM CODE:</u></p> <pre> class Myclass extends Thread{ public void run(){ System.out.println("Hello world"); } } class Myclass1 implements Runnable{ public void run(){ System.out.println("Hello world 2"); } } public class prac32 { public static void main(String[] args) { Myclass thread = new Myclass(); thread.start(); Myclass1 runnable = new Myclass1(); Thread thread1 = new Thread(runnable); thread1.start(); } } </pre> <p><u>OUTPUT:</u></p>  <p><u>CONCLUSION:</u></p> <p>The code demonstrates two ways to create a thread: by extending the Thread class and by implementing the Runnable interface. Extending Thread directly ties the class to thread behavior, while using Runnable allows more flexibility since it separates task logic from</p>

	thread management. The use of <code>thread1.run()</code> calls the method on the main thread, but <code>thread1.start()</code> should be used to run it in a new thread.
33.	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre>import java.util.*; class SumTask implements Runnable { private int start; private int end; private int[] result; private int index; public SumTask(int start, int end, int[] result, int index) { this.start = start; this.end = end; this.result = result; this.index = index; } @Override public void run() { int sum = 0; for (int i = start; i <= end; i++) { sum += i; } result[index] = sum; } } public class prac33 { public static void main(String[] args) { if (args.length < 2) { System.out.println("Please provide two arguments: N and the number of threads."); return; } int N = Integer.parseInt(args[0]); int numThreads = Integer.parseInt(args[1]);</pre>

```
int[] result = new int[numThreads];

int range = N / numThreads;
int remainder = N % numThreads;

Thread[] threads = new Thread[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, result, i));
    threads[i].start();

    start = end + 1;
}

try {
    for (Thread thread : threads) {
        thread.join();
    }
} catch (InterruptedException e) {
    System.out.println("Thread interrupted: " + e.getMessage());
}

int finalSum = 0;
for (int sum : result) {
    finalSum += sum;
}

System.out.println("The sum of the first " + N + " numbers is: " + finalSum);

System.out.println(" ");

}
}
```

OUTPUT:

The sum of the first 100 numbers is: 5050

CONCLUSION:

The program calculates the sum of the first N numbers using multiple threads, each handling a specific range. It efficiently divides the task, synchronizes thread completion using join(), and combines the partial results to get the final sum. This showcases the use of multithreading for improved performance.

- 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:

```
class Thread1 extends Thread {
    public void run() {
        int i=0;
        while (i<4) {
            int n = (int) (Math.random() * 100);
            System.out.println("Generated number: " + n);
            if (n % 2 == 0) {
                // Create and start Thread2 for even number
                new Thread2(n).start();
            } else {
                // Create and start Thread3 for odd number
                new Thread3(n).start();
            }
            try {
                Thread.sleep(1000); // Sleep for 1 second
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            i++;
        }
    }
}

class Thread2 extends Thread {
    private int n;
```

```
Thread2(int n) {
    this.n = n;
    setName("EvenThread"); // Set a meaningful name
}

public void run() {
    System.out.println(getName() + ": Square of " + n + " is " + (n * n));
}
}

class Thread3 extends Thread {
    private int n;

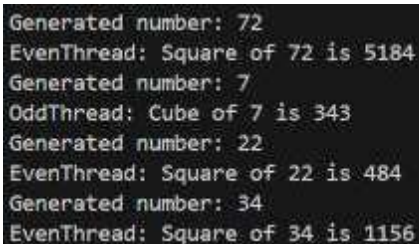
    Thread3(int n) {
        this.n = n;
        setName("OddThread"); // Set a meaningful name
    }

    public void run() {
        System.out.println(getName() + ": Cube of " + n + " is " + (n * n * n));
    }
}

public class prac34 {
    public static void main(String[] args) {
        Thread1 t1 = new Thread1();
        t1.start(); // Start the first thread

    }
}
```

OUTPUT:



```
Generated number: 72
EvenThread: Square of 72 is 5184
Generated number: 7
OddThread: Cube of 7 is 343
Generated number: 22
EvenThread: Square of 22 is 484
Generated number: 34
EvenThread: Square of 34 is 1156
```

CONCLUSION:

The program generates random numbers and processes them in separate threads based on whether they are even or odd. For even numbers, a thread calculates their square, and for

	odd numbers, another thread calculates their cube. This demonstrates the use of multithreading to handle different tasks concurrently, improving efficiency and organizing logic based on specific conditions.
35.	<p>Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method</p> <p><u>PROGRAM CODE:</u></p> <pre>class Thread1 extends Thread { public void run() { int i=0,n = 0; while (i<7) { n++; System.out.println(n); try { Thread.sleep(1000); // Sleep for 1 second } catch (InterruptedException e) { System.out.println("Thread interrupted."); } i++; } } } public class prac35 { public static void main(String[] args) { Thread1 t1 = new Thread1(); t1.start(); // Start the thread } }</pre> <p><u>OUTPUT:</u></p>



1
2
3
4
5
6
7

CONCLUSION:

The program creates a thread that prints numbers from 1 to 7, pausing for 1 second between each. The Thread.sleep() method is used to introduce the delay, and any interruption is handled by printing a message. This example demonstrates the basic concept of multithreading, where a thread runs concurrently with the main program, allowing for independent execution.

- 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 T extends Thread {
    public T(String s) {
        super(s);
    }

    public void run() {
        System.out.println("This is run method :" + getName());
    }
}

public class prac36 {
    public static void main(String[] args) {
        T FIRST = new T("FIRST");
        T SECOND = new T("SECOND");
        T THIRD = new T("THIRD");

        System.out.println("default priority:");
        System.out.println("FIRST:" + FIRST.getPriority());
        System.out.println("SECOND:" + SECOND.getPriority());
        System.out.println("THIRD:" + THIRD.getPriority());
    }
}
```

```

FIRST.setPriority(3);
SECOND.setPriority(5);
THIRD.setPriority(7);

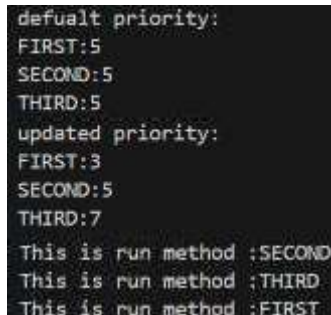
System.out.println("updated priority:");
System.out.println("FIRST:" + FIRST.getPriority());
System.out.println("SECOND:" + SECOND.getPriority());
System.out.println("THIRD:" + THIRD.getPriority());

THIRD.start();
SECOND.start();
FIRST.start();

    }
}

```

OUTPUT:



```

default priority:
FIRST:5
SECOND:5
THIRD:5
updated priority:
FIRST:3
SECOND:5
THIRD:7
This is run method :SECOND
This is run method :THIRD
This is run method :FIRST

```

CONCLUSION:

The program demonstrates thread prioritization in Java. It creates three threads (FIRST, SECOND, and THIRD), displays their default priorities, and then updates them. The `setPriority()` method is used to assign different priorities to the threads, and they are started in descending order of priority. The program showcases how thread priorities can influence the scheduling and execution order, although thread execution is ultimately managed by the JVM and may not strictly follow the set priorities.

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

PROGRAM CODE:

```
class SharedBuffer {
    int item; // A shared place for the item
    boolean isProduced = false; // Whether the item is produced or not

    public synchronized void produce() throws InterruptedException {
        if (isProduced) {
            return; // If an item is already produced, do nothing
        }
        item = (int) (Math.random() * 100); // Produce a random item
        System.out.println("Produced: " + item);
        isProduced = true; // Mark the item as produced
        notify(); // Notify the consumer that the item is ready
    }

    public synchronized void consume() throws InterruptedException {
        if (!isProduced) {
            return; // If no item is produced, do nothing
        }
        System.out.println("Consumed: " + item); // Consume the item
        isProduced = false; // Mark that the item has been consumed
        notify(); // Notify the producer that the buffer is now empty
    }
}

class Producer extends Thread {
    SharedBuffer buffer;

    public Producer(SharedBuffer buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        try {
            for (int i = 0; i < 10; i++) {
                buffer.produce(); // Produce an item
                Thread.sleep(1000); // Simulate some delay
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```



```
}

class Consumer extends Thread {
    SharedBuffer buffer;

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

    @Override
    public void run() {
        try {
            for (int i = 0; i < 10; i++) {
                buffer.consume(); // Consume an item
                Thread.sleep(1000); // Simulate some delay
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class prac37 {
    public static void main(String[] args) throws InterruptedException {
        SharedBuffer buffer = new SharedBuffer(); // Shared buffer

        // Create producer and consumer threads by extending Thread
        Producer producerThread = new Producer(buffer);
        Consumer consumerThread = new Consumer(buffer);

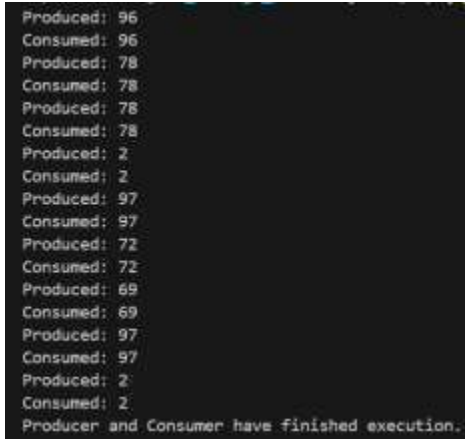
        // Start the threads
        producerThread.start();
        consumerThread.start();

        // Wait for both threads to complete
        producerThread.join();
        consumerThread.join();

        System.out.println("Producer and Consumer have finished execution.");

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

```
}
```

OUTPUT:

```
Produced: 96  
Consumed: 96  
Produced: 78  
Consumed: 78  
Produced: 78  
Consumed: 78  
Produced: 2  
Consumed: 2  
Produced: 97  
Consumed: 97  
Produced: 72  
Consumed: 72  
Produced: 69  
Consumed: 69  
Produced: 97  
Consumed: 97  
Produced: 2  
Consumed: 2  
Producer and Consumer have finished execution.
```

CONCLUSION:

The program implements a producer-consumer scenario using a shared buffer. The Producer thread generates random numbers and the Consumer thread retrieves them, with synchronization ensuring safe access to the shared resource. The use of `notify()` facilitates communication between the producer and consumer, preventing race conditions. This example effectively demonstrates inter-thread coordination in Java.

PART – 8

No.	Aim of the Practical
38.	<p>Design a Custom Stack using ArrayList class, which implements following functionalities of stack. My Stack</p> <ul style="list-style-type: none"> -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. <p><u>PROGRAM CODE:</u></p> <pre>import java.util.*; class StackList{ private ArrayList<Integer> STL; public StackList(){ STL = new ArrayList<>(); } public boolean isEmpty() { return STL.isEmpty(); } public void push(Integer value) { STL.add(value); //updateFile(); } public void pop(){ if(STL.isEmpty()){ System.out.println("Stack is empty"); //return null; } else{</pre>

```
        STL.remove(STL.size()-1);
    }
}

public Integer peek(){
    if(STL.isEmpty()){
        System.out.println("Stack is empty");
        return null;
    }
    else{

        return STL.get(STL.size()-1);
    }
}

public void display(){
    System.out.println(STL);
}

public void clr(){
    STL.clear();
}

public void size1(){
    System.out.println(STL.size());
}

public void updatafile(){

}

}
public class prac38 {
    public static void main(String[] args) {

        StackList stack = new StackList();
        stack.push(10);
        stack.push(20);
        stack.push(70);
        stack.push(40);
        stack.push(30);
        stack.display();
        System.out.println("Top element: " + stack.peek()); // Outputs: 30
    }
}
```

```
stack.pop();
stack.display();
```

```
// System.out.println("Popped element: " + stack.pop()); // Outputs: 30
System.out.println("Top element after pop: " + stack.peek()); // Outputs: 20
}
}
```

OUTPUT:

```
[10, 20, 70, 40, 30]
Top element: 30
[10, 20, 70, 40]
Top element after pop: 40
```

CONCLUSION:

This program implements a stack using an ArrayList, providing basic stack operations like push, pop, peek, and clear. It allows adding and removing elements from the stack and displays the current stack contents. The program efficiently handles stack operations and checks for underflow when the stack is empty.

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;

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;

}

}

public class prac39 {

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

        Arrays.sort(array);

    }

    public static void main(String[] args) {

        Integer[] numbers = { 8, 3, 19, 13, 7 ,2};

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

```
sortArray(numbers);

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

String[] names = { "Cristiano", "Alice", "Marco", "Messi" };

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", 700),

    new Product("Phone", 550),

    new Product("Tablet", 540),

    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);

}
```

```
}
}
```

OUTPUT:

```
Before sorting (Integers): [8, 3, 19, 13, 7, 2]
After sorting (Integers): [2, 3, 7, 8, 13, 19]

Before sorting (Strings): [Cristiano, Alice, Marco, Messi]
After sorting (Strings): [Alice, Cristiano, Marco, Messi]

Before sorting (Products by price):
Laptop: $700
Phone: $550
Tablet: $540
Smartwatch: $200

After sorting (Products by price):
Smartwatch: $200
Tablet: $540
Phone: $550
Laptop: $700
```

CONCLUSION:

This program demonstrates generic sorting by using Java's Comparable interface. It sorts arrays of integers, strings, and custom Product objects based on price in ascending order. By leveraging the Arrays.sort() method, it efficiently arranges elements and displays the sorted results. It provides a versatile approach to sorting different types of objects.

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 prac40 {

    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) {

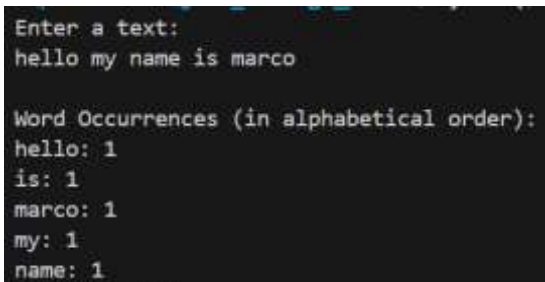
    System.out.println(entry.getKey() + ": " + entry.getValue());

    System.out.println("23DSC083_MARK");

} } }

```

OUTPUT:



```

Enter a text:
hello my name is marco

Word Occurrences (in alphabetical order):
hello: 1
is: 1
marco: 1
my: 1
name: 1

```

CONCLUSION:

This program takes a text input from the user, counts the occurrences of each word, and displays the results in alphabetical order. It uses a TreeMap to store words, ensuring automatic sorting by key. The program efficiently processes text by splitting it into words and counting their frequency.

- 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", "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);

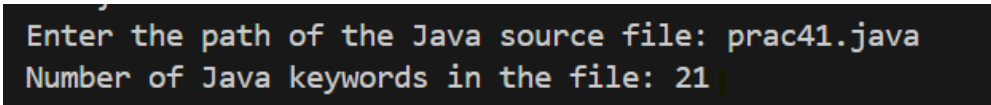
int keywordCount = 0;

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:A screenshot of a terminal window showing the program's output. The first line is a prompt 'Enter the path of the Java source file: ' followed by the user input 'prac41.java'. The second line shows the output 'Number of Java keywords in the file: 21'.

```
Enter the path of the Java source file: prac41.java  
Number of Java keywords in the file: 21
```

CONCLUSION:

This program reads a Java source file and counts the number of Java keywords it contains. By utilizing a predefined set of keywords, it efficiently scans through the file and outputs the total count. The program also handles file not found errors gracefully.