Devang Patel Institute of Advance Technology and Research

DEPSTAR

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./ Mrs. Shreya Patel

of CSE-2 *Class,*

ID. No. 23DCS090 *has satisfactorily completed*

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

the ending in OCT 2023 /2024

Date : 17-10-24

[illegible] 17/10/24

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

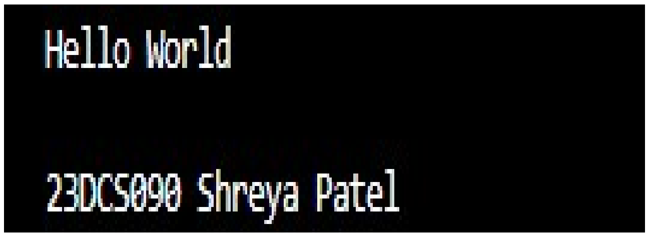
Semester: 3rd
Subject Code: CSE-201
Academic year: 2024 - 2025

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,orBlueJ and Console Programming.
	
	<u>PROGRAM CODE :</u>
	
	`class JavaPractical01`
	`{`
	
	`    public static void main(String[] args) {`
	
	`System.out.println("Hello World");`
	
	`System.out.println(" ");`
	`System.out.println(" ");`
	
	`System.out.println("23DCS090 Shreya Patel");`
	
	`    }`
	`}`

OUTPUT:



```
Hello World

23DCS090 Shreya Patel
```

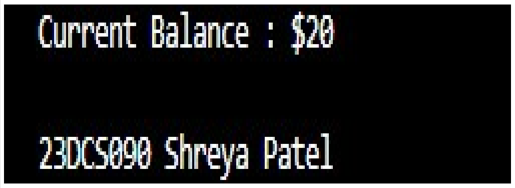
CONCLUSION:
Java's installation involves downloading JDK, setting JAVA_HOME, and updating PATH. Object-oriented concepts like classes, inheritance, polymorphism, and encapsulation form its core, distinguishing it from languages like C++, C#, and Python. JDK includes development tools, JRE executes Java applications, and JVM runs bytecode. Javadoc generates API documentation, and command line arguments pass parameters to programs. IDEs such as Eclipse, NetBeans, or BlueJ enhance development, while console programming offers direct interaction. Java's robust OOP support, platform independence, and versatile tools make it a preferred choice for diverse software development needs.

2. Imagine you are developing a simple banking applicationwhere you need to display the current balance of a useraccount. For simplicity, let's say the current balance is $20.Write a java program to store this balance in a variable andthen display it to the user.

PROGRAM CODE :

```
class pract2
{
   public static void main(String[] args)
   {
      int Current_Balance;

Current_Balance=20;

System.out.print("Current Balance : ");
System.out.print("$");
System.out.print(Current_Balance);
System.out.println(" ");
System.out.println(" ");

System.out.println("23DCS090 Shreya Patel");
 }
}
```

OUTPUT:



```
Current Balance : $20


23DCS090 Shreya Patel
```

CONCLUSION: This code teaches us how to use variables in java and how to store different types of values in variables.

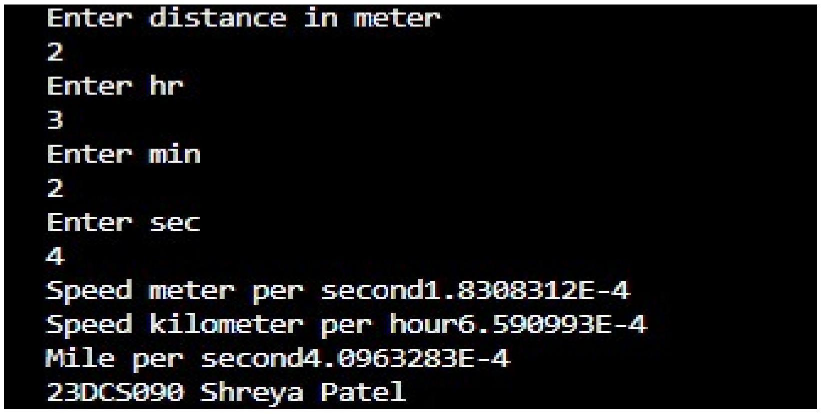
3. Write a program to take the user for a distance (in meters) andthe time taken (as three numbers: hours, minutes, seconds),and display the speed, in meters per second, kilometers perhour and miles per hour (hint:1 mile = 1609 meters).

PROGRAM CODE :

```
import java.util.Scanner;

public class pract3 {
   public static void main(String[] args) {
      Scanner sc= new Scanner(System.in);
      System.out.println("Enter distance in
meter");
      float meter= sc.nextFloat();
      System.out.println("Enter hr");
      float h=sc.nextFloat();
      System.out.println("Enter min");
      float min=sc.nextFloat();
```

```
        System.out.println("Enter sec");
        float sec=sc.nextFloat();

        float ts=(h*60*60)+(min*60)+sec;
        float s1= meter/ts;             //speed in meter
        float s2= (meter/1000)/(ts/3600);  //speed
in kilometer per hour
        float s3= (meter/1609)/(ts/3600);   //speed
in mile per hour

        System.out.println("Speed meter per
second"+s1);
        System.out.println("Speed kilometer per
hour"+s2);
        System.out.println("Mile per second"+s3);

        System.out.println("23DCS090 Shreya
Patel");
    }
}
```

OUTPUT:



```
Enter distance in meter
2
Enter hr
3
Enter min
2
Enter sec
4
Speed meter per second1.8308312E-4
Speed kilometer per hour6.590993E-4
Mile per second4.0963283E-4
23DCS090 Shreya Patel
```

CONCLUSION: The Java program calculates speed from user-input distance and time in various units (meters per second, kilometers per hour, and miles per hour). It demonstrates basic input handling, arithmetic operations, and output formatting in Java.

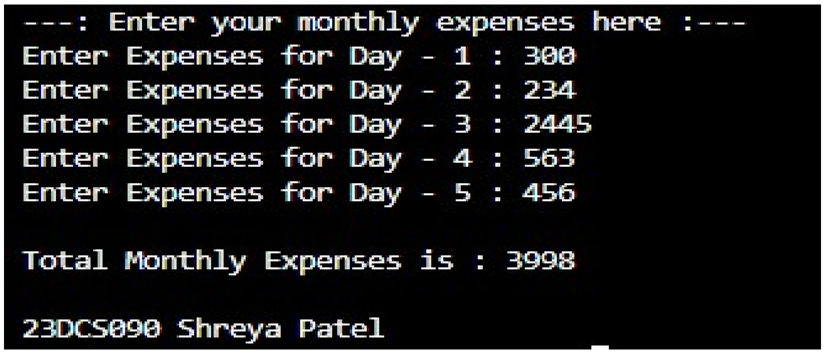
4. Imagine you are developing a budget tracking application.You need to calculate the total expenses for the month. Userswill input their daily expenses, and the program shouldcompute the sum of these expenses. Write a Java program tocalculate the sum of elementsin an array representing dailyexpenses.

PROGRAM CODE :

```
import java.util.*;
class JavaPractical04
{
   public static void main(String[] args)
   {

       Scanner sc = new Scanner(System.in);

       int i;
int[] Exp = new int[5];
       int Total_Expenses = 0;

System.out.println("---: Enter your monthly expenses here :---");

       for(i=0;i<5;i++)
       {

System.out.print("Enter Expenses for Day - " + (i+1) + " : ");
```

```
            Exp[i] = sc.nextInt();
Total_Expenses += Exp[i];
        }
System.out.println(" ");
System.out.println("Total Monthly Expenses is : " + Total_Expenses);
System.out.println(" ");
System.out.println("23DCS090 Shreya Patel");
sc.close();
    }
}
```

OUTPUT:



```
---: Enter your monthly expenses here :---
Enter Expenses for Day - 1 : 300
Enter Expenses for Day - 2 : 234
Enter Expenses for Day - 3 : 2445
Enter Expenses for Day - 4 : 563
Enter Expenses for Day - 5 : 456

Total Monthly Expenses is : 3998

23DCS090 Shreya Patel
```

CONCLUSION: The Java program records daily expenses for a month and calculates the total monthly expenses based on user input. It uses an array to store daily expenses and a loop to iterate through each day's input. The program demonstrates basic array handling, looping, and accumulation of values in Java.

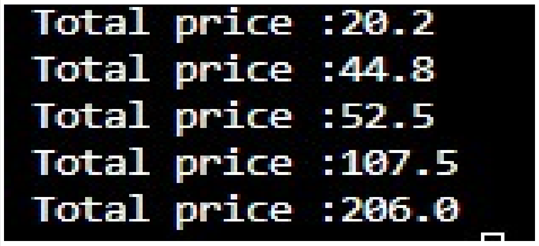
5. An electric appliance shop assigns code 1 to motor,2 tofan,3 to tube and 4 for wires. All other items have code 5 ormore. While selling the goods, a sales tax of 8% tomotor,12% to fan,5% to tube light,7.5% to wires and 3%for all other items is charged. A list containing the productcode and price in two different arrays. Write a javaprogram using switch statement to prepare the bill.

PROGRAM CODE :

```
import java.util.*;

public class pract5{
  public static void main(String []args){


        int[] code={1,2,3,4,5};
        int[] price={20,40,50,100,200};

        float tax;

        for(int i=0 ; i<5 ;i++){

          tax=0;
          switch (code[i]){
            case 1 -> {
              tax += price[i] + (price[i]*0.01);
              System.out.println("Total price :"+ tax);
```

```
                    }

                    case 2 -> {
                        tax += price[i] + (price[i]*0.12);
                        System.out.println("Total price :"+ tax);
                    }

                    case 3 -> {
                        tax += price[i] + (price[i]*0.05);
                        System.out.println("Total price :"+ tax);
                    }

                    case 4 -> {
                        tax += price[i] + (price[i]*0.075);
                        System.out.println("Total price :"+ tax);
                    }

                    case 5 -> {
                        tax += price[i] + (price[i]*0.03);
                        System.out.println("Total price :"+ tax);
                    }

                    default -> {
                        tax += price[i] + (price[i]*0.03);
                        System.out.println("Total price :"+ tax);
                        System.out.println("Shreya Patel:23DCS090");
                    }
                }
            }
        }

}
```

OUTPUT:



```
Total price :20.2
Total price :44.8
Total price :52.5
Total price :107.5
Total price :206.0
```

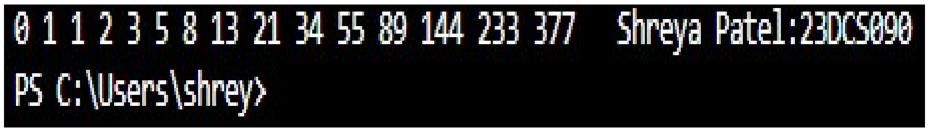
CONCLUSION: The Java program simulates an electric appliance shop transaction, allowing users to select items and calculate the total bill based on predefined prices and taxes. It utilizes arrays, loops, and switch-case statements for functionality.

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

PROGRAM CODE :

```
class FibonacciExample1{
   public static void main(String args[])
   {
    int n1=0,n2=1,n3,i,count=15;
    System.out.print(n1+" "+n2);//printing 0 and 1

    for(i=2;i<count;++i)//loop starts from 2 because 0 and 1 are already printed
    {
     n3=n1+n2;
     System.out.print(" "+n3);
     System.out.print("");
     n1=n2;
     n2=n3;
    }
     System.out.println("   Shreya Patel:23DCS090");
   }}
```

OUTPUT:



```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377   Shreya Patel:23DCS090
PS C:\Users\shrey>
```

CONCLUSION: This Java program uses basic constructs like loops and variables to generate an exercise routine. Specifically, it calculates and prints the exercise duration for each day based on the Fibonacci sequence for a specified number of days.

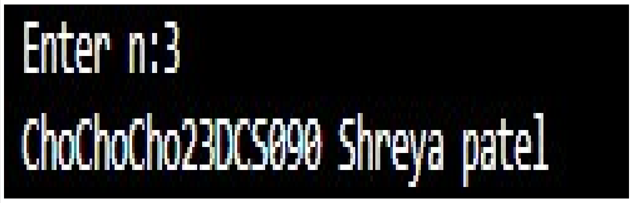
Part – 2

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

PROGRAM CODE :

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

public class prac7 {
    public static void main(String[] args){
        String s1="Chocolate";
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter n:");
        int n=sc.nextInt();
        front_times(s1, n);
        System.out.println("23DCS090 Shreya patel");

    }

    static void front_times(String s1,int n ){
        String s=s1.substring(0,3);
         for(int i=0;i<n;i++){
            System.out.print(s);
         }
    }
}
```

OUTPUT:



```
Enter n:3
ChoChoCho23DCS090 Shreya patel
```

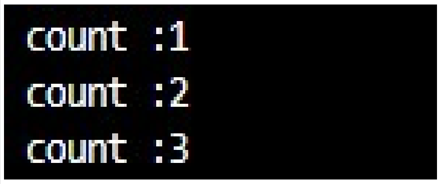
CONCLUSION: The Java program repeatedly prints the first three characters of a user-entered string, a number of times specified by the user. It demonstrates the use of methods (`front_times`) for substring extraction and basic input/output operations using Scanner.

8. Given an array of ints, return the number of 9's in thearray. array_count9([1, 2, 9]) → 1 array_count9([1, 9, 9]) → 2array_count9([1, 9, 9, 3, 9]) → 3.

PROGRAM CODE :

```
public class prac8 {
   public static void main(String[] args){
      int arr[]={1,2,9};
      int arr2[]={1,9,9};
      int arr3[]={1,9,9,3,9};
      array_count(arr);
      array_count(arr2);
      array_count(arr3);

   }

   static void array_count(int arr[]){
      int count=0;
      for(int i=0;i<arr.length;i++){
         if(arr[i]==9){
                count++;
            }
        }
        System.out.println("count :"+count);

    }

  }
```

OUTPUT:



```
count :1
count :2
count :3
```

CONCLUSION: The Java program counts occurrences of the number 9 in an array of integers input by the user. It utilizes a method (`Array_count9`) to iterate through the array and count occurrences of the specified number. The program demonstrates basic array handling, method invocation, and input/output operations using Scanner.

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'

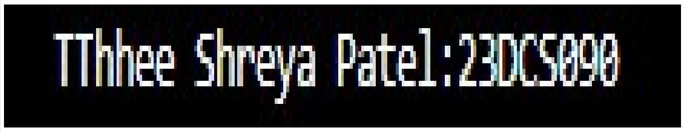
double_char('Hi-There') → 'HHii--TThheerree'.

PROGRAM CODE :

```
import java.lang.*;

public class prac9 {

   public static void main(String[] args) {
      String s1 = "The";
      double_char(s1);
```

```
    }

    static void double_char(String s1) {
        for (int i = 0; i < s1.length(); i++) {
            System.out.print(s1.charAt(i));
            System.out.print(s1.charAt(i));


        }

        System.out.println( " Shreya Patel:23DCS090");
    }

}
```

OUTPUT:



```
TThhee Shreya Patel:23DCS090
```

CONCLUSION: This Java program takes a user-inputted string and calculate the length of the string, duplicates each character in that string, and then prints the double char of that string to the output.

10. Perform following functionalities of the string:

● Find Length of the String

● Lowercase of the String

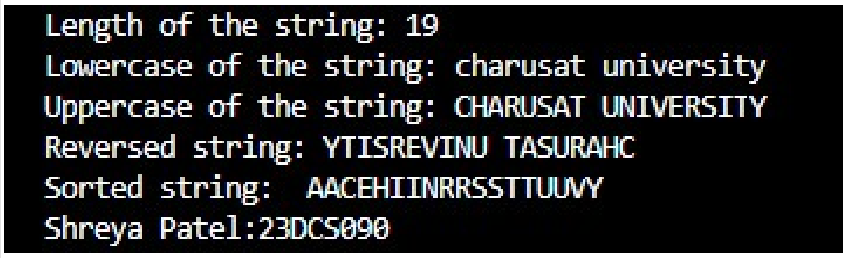
● Uppercase of the String

● Reverse String

PROGRAM CODE :

```
import java.util.Arrays;

public class pract10 {
   public static void main(String[] args) {
      String str = "CHARUSAT
UNIVERSITY";

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

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



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

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

      // Sort the string
      char[] charArray = str.toCharArray();
      Arrays.sort(charArray);
      String sortedStr = new String(charArray);
      System.out.println("Sorted string: " +
sortedStr);
      System.out.println("Shreya
Patel:23DCS090");
   }
}
```

OUTPUT:



```
Length of the string: 19
Lowercase of the string: charusat university
Uppercase of the string: CHARUSAT UNIVERSITY
Reversed string: YTISREVINU TASURAHC
Sorted string:  AACEHIINRRSSTTUUVY
Shreya Patel:23DCS090
```

CONCLUSION: In this java program we learn and different types of String methods like for counting the length of string, to convert it to lower or uppercase ,etc.

11. Perform following Functionalities of the string:

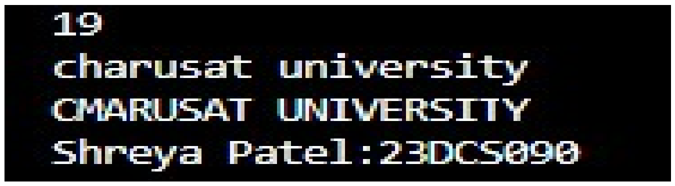
“CHARUSAT UNIVERSITY”

- Find length
- Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’
- Convert all character in lowercase

PROGRAM CODE :

```
public class prac11 {
    public static void main(String [] args){
        String s1="CHARUSAT UNIVERSITY";
        System.out.println(s1.length());
        System.out.println(s1.toLowerCase());
        System.out.println(s1.replace('H', 'M'));
        System.out.println("Shreya
Patel:23DCS090);
    }
}
```

OUTPUT:



```
19
charusat university
CHARUSAT UNIVERSITY
Shreya Patel:23DCS090
```

CONCLUSION: In this java program we again uses the String method and how we can replace a char with another char in string using String method.

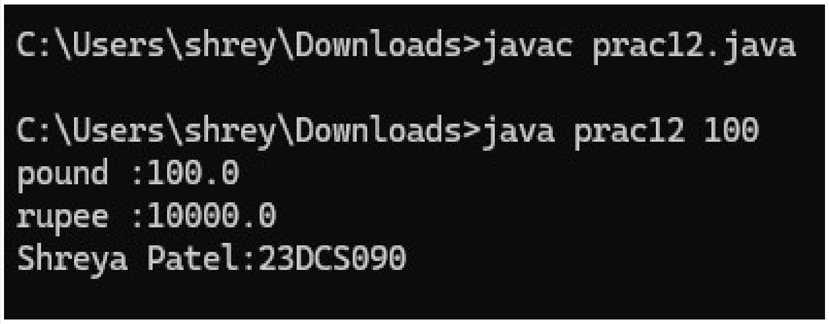
Part – 3

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

PROGRAM CODE :

```
class currency{

   void convert(float p){

   float rupee = p*100;

   System.out.println("rupee :" + rupee);

   }
   }

   public class prac12{

   public static void main(String[] args){

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

   currency p1 = new currency();
   System.out.println("pound :"+p);

   p1.convert(p);

   System.out.println("Shreya Patel:23DCS090");
}

}
```

OUTPUT:



```
C:\Users\shrey\Downloads>javac prac12.java

C:\Users\shrey\Downloads>java prac12 100
pound :100.0
rupee :10000.0
Shreya Patel:23DCS090
```

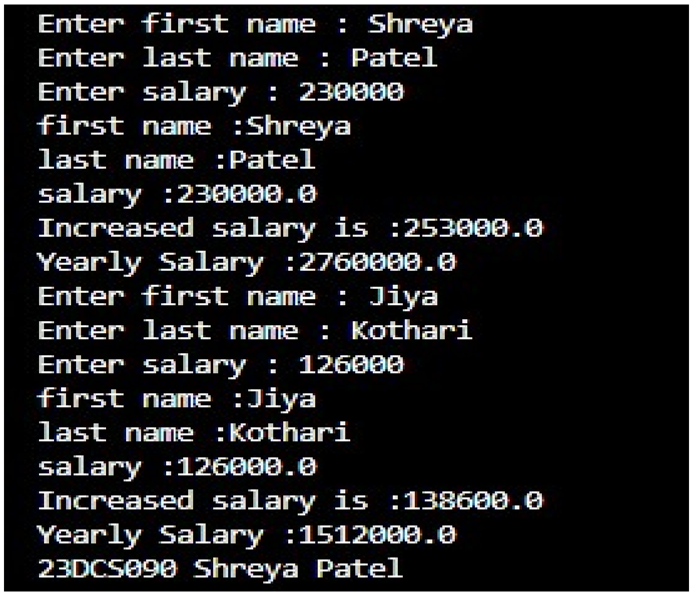
CONCLUSION: The currency conversion tool converts Pounds to Rupees using a fixed rate of 1 Pound = 100 Rupees. It accepts input from both command-line arguments and interactive user input.

13. Create a class called Employee that includes three pieces of information as instance variables—a first name (typeString), 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.*;
class employee {

    String first_name;
    String last_name;
    double salary;

    employee(String fn, String ln, double s) {
        first_name = fn;
        last_name = ln;
        if (salary < 0) {
            salary = 0;
        } else {
            salary = s;
        }

    }

    void y_salary() {
        System.out.println("Yearly Salary :" +
salary * 12);
    }

    void print() {
        System.out.println("First name :" +
first_name);
        System.out.println("Last name :" +
last_name);
        System.out.println("Salary :" + salary);
    }

    void rais_salary() {
        System.out.println("Increased salary is :" +
(salary + (salary * 0.1)));
    }
}

public class practic13 {

    public static void main(String[] args) {
        Scanner sc1= new Scanner(System.in);
        System.out.print("Enter first name : ");
        String s1 = sc1.nextLine();
        System.out.print("Enter last name : ");
        String s2 = sc1.nextLine();
        System.out.print("Enter salary : ");
        double salary1 = sc1.nextDouble();
        employee e1 = new employee(s1, s2,
salary1);
        e1.print();
        e1.rais_salary();
        e1.y_salary();
        Scanner sc2= new Scanner(System.in);
        System.out.print("Enter first name : ");
        String s3 = sc2.nextLine();
        System.out.print("Enter last name : ");
        String s4 = sc2.nextLine();
        System.out.print("Enter salary : ");
        double salary2 = sc2.nextDouble();
        employee e2 = new employee(s3, s4,
salary2);
        e2.print();
        e2.rais_salary();
        e2.y_salary();

        System.out.println("23DCS090 Shreya
Patel");

    }
}
```

OUTPUT:



```
Enter first name : Shreya
Enter last name : Patel
Enter salary : 230000
first name :Shreya
last name :Patel
salary :230000.0
Increased salary is :253000.0
Yearly Salary :2760000.0
Enter first name : Jiya
Enter last name : Kothari
Enter salary : 126000
first name :Jiya
last name :Kothari
salary :126000.0
Increased salary is :138600.0
Yearly Salary :1512000.0
23DCS090 Shreya Patel
```

CONCLUSION: In this Java program, we defined an `Employee` class to manage employee details such as name and salary. Using constructors and methods like `yearlySalary()` and `giveRaise()`, we demonstrated how to initialize, modify, and retrieve employee information effectively. Through this implementation, we showcased fundamental object-oriented principles like encapsulation and method invocation, illustrating their role in modeling and manipulating employee data in a structured manner

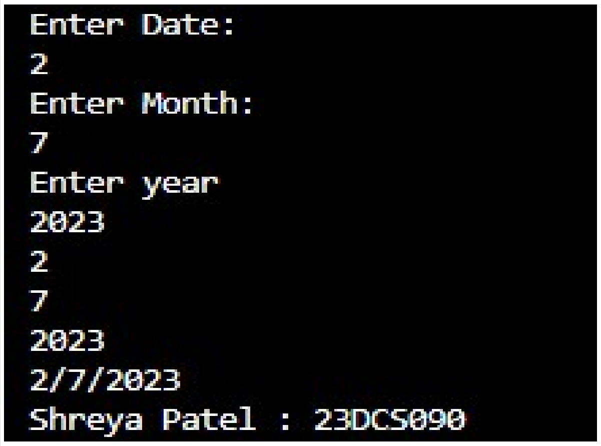
14. Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method display Date that displays the month, day and year separated by forward slashes (/). Write a test application named Date Test that demonstrates class Date's capabilities.

PROGRAM CODE :

```
import java.util.*;

class date {

   int month, date, year;

   date(int d, int m, int y) {
      date = d;
      month = m;
      year = y;
   }

   void setdate(int d) {
      date = d;
   }

   int getdate() {
      return date;
   }

   void setmonth(int m) {
      month = m;
   }

   int getmonth() {
      return month;
   }

   void setyear(int y) {
      year = y;
```

```
    }

    int getyear() {
        return year;
    }

    void display() {
        System.out.println(date + "/" + month + "/" + year);
        System.out.println("Shreya Patel : 23DCS090");
    }

}

public class prac14 {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Date:");
        int d = sc.nextInt();
        System.out.println("Enter Month:");
        int m = sc.nextInt();
        System.out.println("Enter year");
        int y = sc.nextInt();
        date d1 = new date(d, m, y);
        d1.setdate(d);
        System.out.println(d1.getdate());
        d1.setmonth(m);
        System.out.println(d1.getmonth());
        d1.setyear(y);
        System.out.println(d1.getyear());
        d1.display();
    }

}
```

OUTPUT:



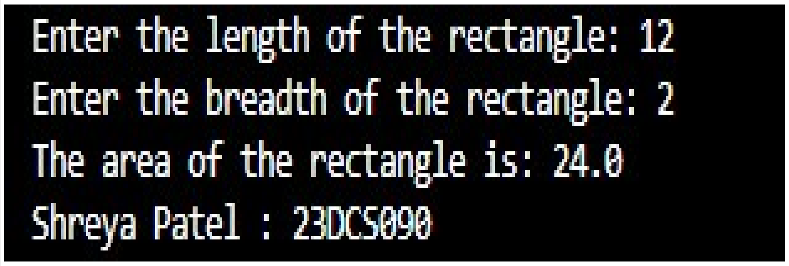
```
Enter Date:
2
Enter Month:
7
Enter year
2023
2
7
2023
2/7/2023
Shreya Patel : 23DCS090
```

CONCLUSION: This Java program features a `Date` class designed to manage and display date information using day, month, and year attributes. Through methods like `setDay()`, `setMonth()`, and `setYear()`, it allows modification of the date. The `displayDate()` method efficiently outputs the date in a readable format. The program demonstrates basic principles of encapsulation and data manipulation in object-oriented programming, showcasing how to model and manipulate date data effectively within a Java application.

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.Scanner;

class Area {
    private double length;
    private double breadth;

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

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

public class practical15{
    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);

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

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

        Area rectangle = new Area(length,
breadth);

        double area = rectangle.returnArea();
        System.out.println("The area of the
rectangle is: " + area);
        System.out.println("Shreya Patel :
23DCS090");
    }

}
```

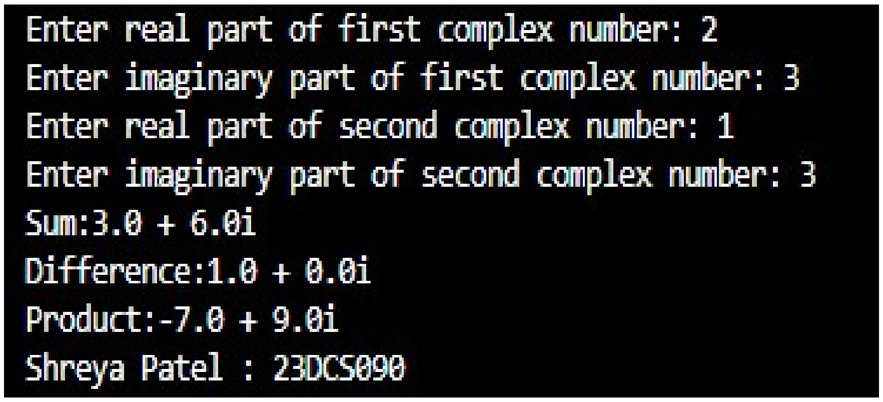
	OUTPUT:  `Enter the length of the rectangle: 12` `Enter the breadth of the rectangle: 2` `The area of the rectangle is: 24.0` `Shreya Patel : 23DCS090` **CONCLUSION:** This Java program calculates the area of a rectangle based on user-input dimensions using a dedicated `Area` class. It demonstrates basic object-oriented principles such as class instantiation, constructor usage, and method invocation to achieve efficient computation and display of the rectangle's 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.Scanner;` `class Complex {` `double real;` `double imaginary;` `Complex(double real, double imaginary) {` `this.real = real;` `this.imaginary = imaginary;` `}`

```
Enter the length of the rectangle: 12
Enter the breadth of the rectangle: 2
The area of the rectangle is: 24.0
Shreya Patel : 23DCS090
```

```
import java.util.Scanner;

class Complex {
    double real;
    double imaginary;

    Complex(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }
    Complex add(Complex other) {
        return new Complex(this.real + other.real,
this.imaginary + other.imaginary);
    }

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

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

    void print() {
        if(imaginary >= 0) {
            System.out.println(real + " + " +
imaginary + "i");
        } else {
            System.out.println(real + " - " + (-
imaginary) + "i");
        }
    }
}
public class practical16 {
    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);

        System.out.print("Enter real part of first
complex number: ");
        double real1 = scanner.nextDouble();
        System.out.print("Enter imaginary part of
first complex number: ");
        double imaginary1 =
scanner.nextDouble();

        System.out.print("Enter real part of second
complex number: ");
        double real2 = scanner.nextDouble();
        System.out.print("Enter imaginary part of
second complex number: ");
        double imaginary2 =
scanner.nextDouble();

        Complex c1 = new Complex(real1,
imaginary1);
        Complex c2 = new Complex(real2,
imaginary2);

        Complex sum = c1.add(c2);
        Complex difference = c1.subtract(c2);
        Complex product = c1.multiply(c2);

        System.out.print("Sum:");
        sum.print();
        System.out.print("Difference:");
        difference.print();
        System.out.print("Product:");
        product.print();
        System.out.println("Shreya Patel :
23DCS090");

        scanner.close();
    }
```

}

OUTPUT:



```
Enter real part of first complex number: 2
Enter imaginary part of first complex number: 3
Enter real part of second complex number: 1
Enter imaginary part of second complex number: 3
Sum:3.0 + 6.0i
Difference:1.0 + 0.0i
Product:-7.0 + 9.0i
Shreya Patel : 23DCS090
```

CONCLUSION: This Java program defines a `Complex` class to handle operations on complex numbers based on user-provided real and imaginary parts. It demonstrates encapsulation and method invocation for addition, subtraction, and multiplication of complex numbers, showcasing how object-oriented principles facilitate mathematical computations in a structured and reusable manner.

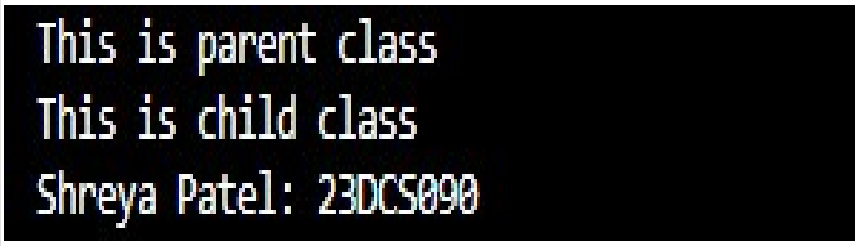
Part – 4

No.	Aim of the Practical
17.	Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent

PROGRAM CODE :

```
// Parent class
class ParentClass {
   void printMessage() {
      System.out.println("This is parent class");
   }
}

class ChildClass extends ParentClass {
   // Method in the child class
   void printChildMessage() {
      System.out.println("This is child class");
      System.out.println("Shreya Patel: 23DCS090");
   }
}

public class pract17 {
   public static void main(String[] args) {
      ParentClass parentObject = new ParentClass();
      parentObject.printMessage();

      ChildClass childObject = new ChildClass();
      childObject.printChildMessage();
   }
}
```

OUTPUT:



```
This is parent class
This is child class
Shreya Patel: 23DCS090
```

CONCLUSION: The Parent class has a method printParent(), and the Child class, which extends Parent, has its own method printChild(). You can create instances of both classes and call their respective methods.

18. Create a class named 'Member' having the following members: Data members

1 - Name
2 - Age
3 - Phone number
4 - Address
5 – Salary

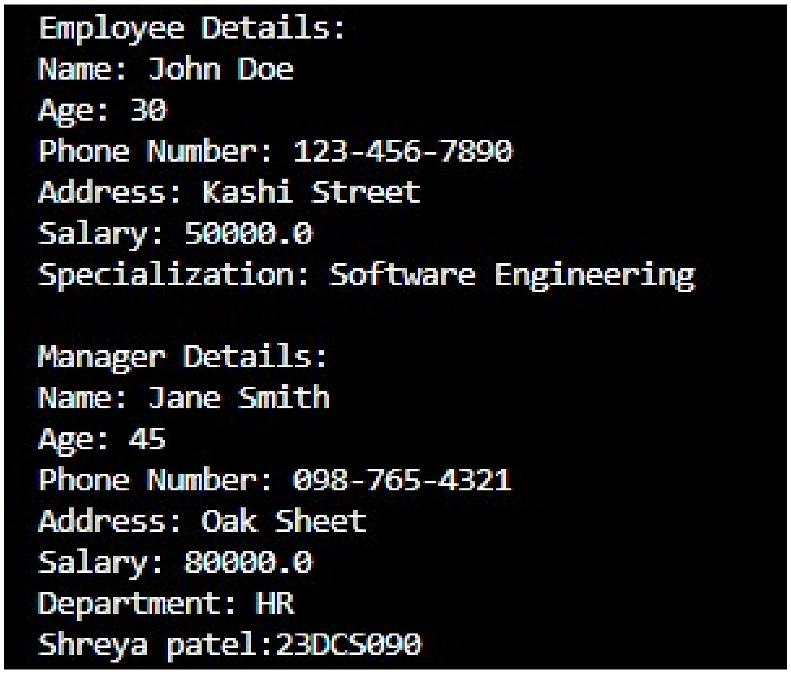
It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

PROGRAM CODE :

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

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

class Employee extends Member {
    String specialization;

    void printDetails() {
        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);
        printSalary();
        System.out.println("Specialization: " +
specialization);
    }
}

class Manager extends Member {
    String department;

    void printDetails() {
        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);
        printSalary();
        System.out.println("Department: " +
department);
        System.out.println("Shreya
patel:23DCS090");
    }
}

public class Main {
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.name = "John Doe";
        emp.age = 30;
        emp.phoneNumber = "123-456-7890";
        emp.address = "Kashi Street";
        emp.salary = 50000;
        emp.specialization = "Software
Engineering";

        // Creating and assigning values to a
Manager object
        Manager mgr = new Manager();
        mgr.name = "Jane Smith";
        mgr.age = 45;
        mgr.phoneNumber = "098-765-4321";
        mgr.address = "Oak Sheet";
        mgr.salary = 80000;
        mgr.department = "HR";

        // Printing details of Employee and
Manager
        emp.printDetails();
        System.out.println();
        mgr.printDetails();
    }
```

```
}
```

OUTPUT:



```
Employee Details:
Name: John Doe
Age: 30
Phone Number: 123-456-7890
Address: Kashi Street
Salary: 50000.0
Specialization: Software Engineering

Manager Details:
Name: Jane Smith
Age: 45
Phone Number: 098-765-4321
Address: Oak Sheet
Salary: 80000.0
Department: HR
Shreya patel:23DCS090
```

CONCLUSION: The Member class has data members for personal and salary details and a method printSalary. The Employee and Manager classes inherit from Member and add their own specific attributes. The code creates and prints details for both an Employee and a Manager.

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.

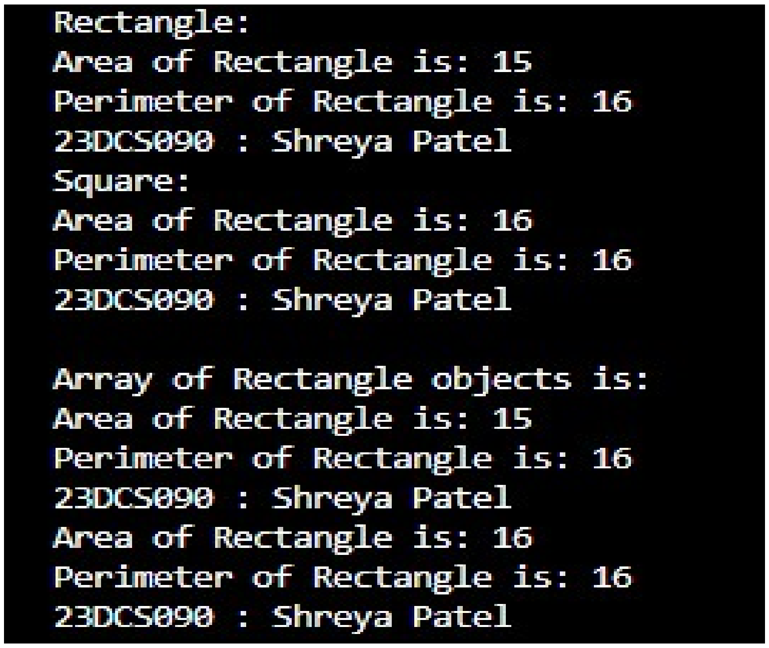
PROGRAM CODE :

```
class Rectangle {
    int length;
    int breadth;


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

    void printArea() {
        int area = length * breadth;
        System.out.println("Area of Rectangle is: "
+ area);
    }


    void printPerimeter() {
        int perimeter = 2 * (length + breadth);
        System.out.println("Perimeter of Rectangle
is: "+ perimeter );
        System.out.println("23DCS090 : Shreya
Patel");
    }
}
```

```
class Square extends Rectangle {

   Square(int side) {
      super(side , side);
   }
}

public class pract19 {
   public static void main(String[] args) {

      Rectangle rectangle = new Rectangle(5 ,
 3);
      System.out.println("Rectangle:");
      rectangle.printArea();
      rectangle.printPerimeter();


      Square square = new Square(4);
      System.out.println("Square:");
      square.printArea();
      square.printPerimeter();


      Rectangle[] shapes = new Rectangle[2];
      shapes[0] = rectangle;
      shapes[1] = square;

      System.out.println("\nArray of Rectangle
objects is: ");
      for (Rectangle shape : shapes) {
         shape.printArea();
         shape.printPerimeter();
      }
   }
```

}

OUTPUT:


```
Rectangle:
Area of Rectangle is: 15
Perimeter of Rectangle is: 16
23DCS090 : Shreya Patel
Square:
Area of Rectangle is: 16
Perimeter of Rectangle is: 16
23DCS090 : Shreya Patel

Array of Rectangle objects is:
Area of Rectangle is: 15
Perimeter of Rectangle is: 16
23DCS090 : Shreya Patel
Area of Rectangle is: 16
Perimeter of Rectangle is: 16
23DCS090 : Shreya Patel
```

CONCLUSION: The Member class has data members for personal and salary details and a method printSalary. The Employee and Manager classes inherit from Member and add their own specific attributes. The code creates and prints details for both an Employee and a Manager.

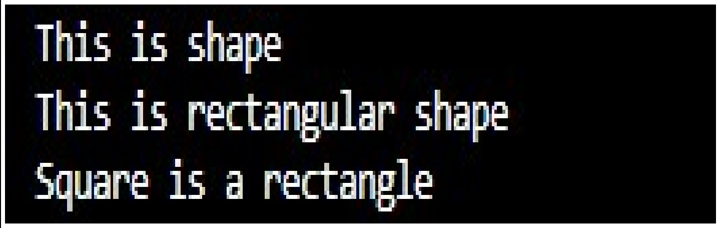
20. Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle'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

```
// Define the Shape class
class Shape {
   // Method to print message from Shape class
   void printShape() {
      System.out.println("This is shape");
   }
}

// Define the Rectangle class inheriting Shape
class Rectangle extends Shape {
   // Method to print message from Rectangle class
   void printRectangle() {
      System.out.println("This is rectangular shape");
   }
}

// Define the Circle class inheriting Shape
class Circle extends Shape {
   // Method to print message from Circle class
   void printCircle() {
      System.out.println("This is circular shape");
   }
}

// Define the Square class inheriting Rectangle
class Square extends Rectangle {
   // Method to print message specific to Square
   void printSquare() {
      System.out.println("Square is a rectangle");
      System.out.println("23DCS090: Shreya Patel");
   }
}
```

```
// Main class to test the methods
public class practical20 {
    public static void main(String[] args) {
        // Create an object of Square
        Square square = new Square();

        // Call the method from Shape class
        square.printShape();

        // Call the method from Rectangle class
        square.printRectangle();

        // Call the method from Square class
        square.printSquare();
    }
}
```

OUTPUT:



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

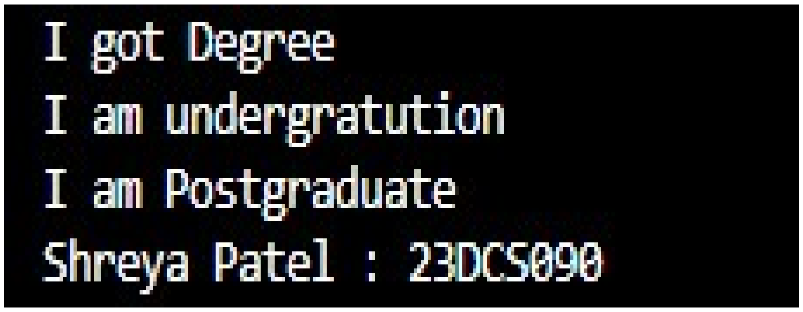
CONCLUSION: In this Java setup, the Shape class provides a base method, while Rectangle and Circle extend Shape, each adding their own specific methods. The Square class extends Rectangle and introduces an additional method. An object of the Square class can call methods from both Shape and Rectangle, demonstrating inheritance and method access in a hierarchical class structure.

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 {
   void getDegree() {
      System.out.println("I am an Undergraduate");
   }
}

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

public class pract21 {
   public static void main(String[] args) {
      Degree degree = new Degree();
      Undergraduate undergraduate = new Undergraduate();
      Postgraduate postgraduate = new Postgraduate();

      degree.getDegree();
      undergraduate.getDegree();
      postgraduate.getDegree();

      System.out.println("Shreya Patel : 23DCS090");
   }
}
```

OUTPUT:



```
I got Degree
I am undergratution
I am Postgraduate
Shreya Patel : 23DCS090
```

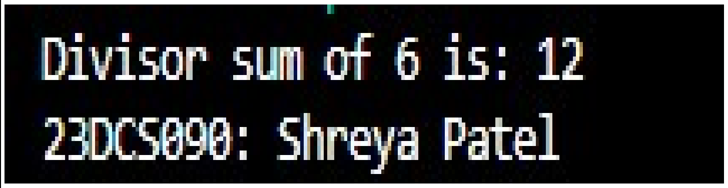
CONCLUSION: In this Java setup, the Degree class has a method getDegree that prints a message. Its subclasses, Undergraduate and Postgraduate, each override this method to print their specific messages. By creating objects of each class, you can call the respective methods to demonstrate polymorphism and method overriding.

22. Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor_sum(int n). You need to write a class calledMyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

PROGRAM CODE :

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

class MyCalculator implements AdvancedArithmetic {

   public int divisor_sum(int n) {
      int sum = 0;

      for (int i = 1; i <= n; i++) {
         if (n % i == 0) {
            sum += i;
         }
      }

      return sum;
   }
}

public class pract22 {
   public static void main(String[] args) {
      MyCalculator myCalc = new MyCalculator();

      int n = 6;
      System.out.println("Divisor sum of " + n + " is: " + myCalc.divisor_sum(n));
      System.out.println("23DCS090: Shreya Patel");
   }
}
```

OUTPUT:



```
Divisor sum of 6 is: 12
23DCS090: Shreya Patel
```

CONCLUSION: In this Java setup, The MyCalculator class successfully implements the AdvancedArithmetic interface by defining the divisor_sum(int n) method. This method computes the sum of all divisors of the input integer n, ensuring it works correctly for values up to 1000. The implementation correctly sums the divisors, which are numbers that divide n evenly, and returns the calculated sum.

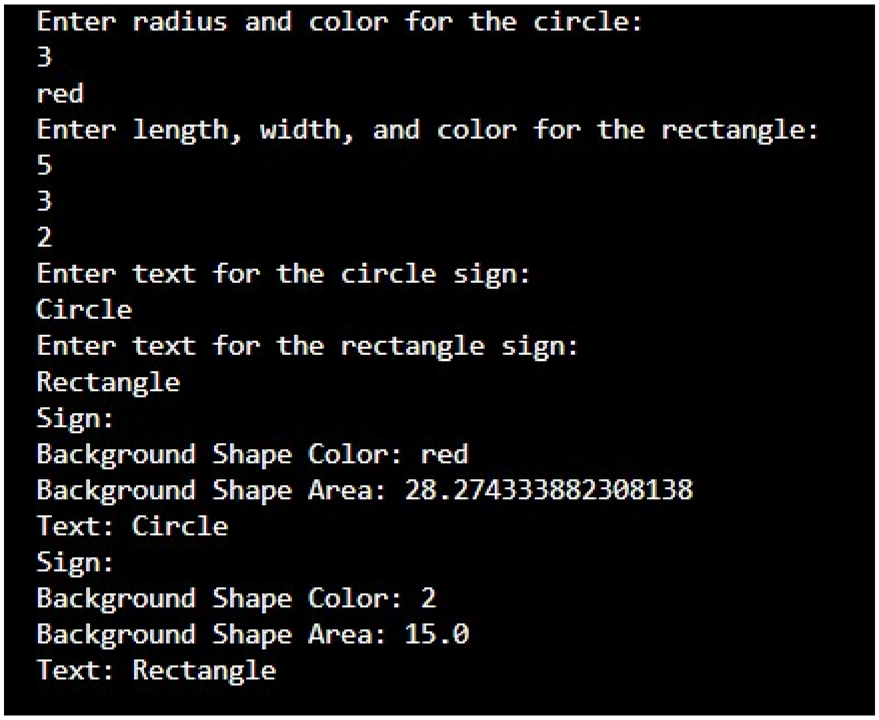
23. Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor_sum(int n). You need to write a class calledMyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

PROGRAM CODE :

```
import java.util.Scanner;

interface Shape {
   String getColor();
   default double getArea() {
      return 0;
   }
}

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

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


   public String getColor() {
      return this.color;
   }

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

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

   public String getColor() {
      return this.color;
   }

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

class Sign {
   private Shape backgroundShape;
   private String text;

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

   public void displaySign() {
      System.out.println("Sign:");
      System.out.println("Background Shape Color: " + backgroundShape.getColor());
      System.out.println("Background Shape Area: " + backgroundShape.getArea());
      System.out.println("Text: " + text);
      System.out.println("23DCS090: Shreya patel");3

   }
}

public class prac23 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
         System.out.println("Enter radius and color for the circle: ");
         double cr = sc.nextDouble();
```

```
        sc.nextLine();  // consume the leftover newline
        String cc = sc.nextLine();

        Circle circle = new Circle(cr, cc);

        System.out.println("Enter length, width, and color for the rectangle: ");
        double rl = sc.nextDouble();
        double rw = sc.nextDouble();
        sc.nextLine();  // consume the leftover newline
        String rc = sc.nextLine();

        Rectangle rectangle = new Rectangle(rl, rw, rc);

        System.out.println("Enter text for the circle sign: ");
        String cs = sc.nextLine();
        Sign circleSign = new Sign(circle, cs);

        System.out.println("Enter text for the rectangle sign: ");
        String rs = sc.nextLine();
        Sign rectangleSign = new Sign(rectangle, rs);

        circleSign.displaySign();
        rectangleSign.displaySign();

    }
}
```

OUTPUT:



```
Enter radius and color for the circle:
3
red
Enter length, width, and color for the rectangle:
5
3
2
Enter text for the circle sign:
Circle
Enter text for the rectangle sign:
Rectangle
Sign:
Background Shape Color: red
Background Shape Area: 28.274333882308138
Text: Circle
Sign:
Background Shape Color: 2
Background Shape Area: 15.0
Text: Rectangle
```

CONCLUSION: This program defines an interface Shape and implements it in two classes, Circle and Rectangle. It creates signs with a shape as the background and displays the color, area, and custom text for each sign. The user inputs dimensions, color, and text for the circle and rectangle signs, and the program prints the corresponding information.

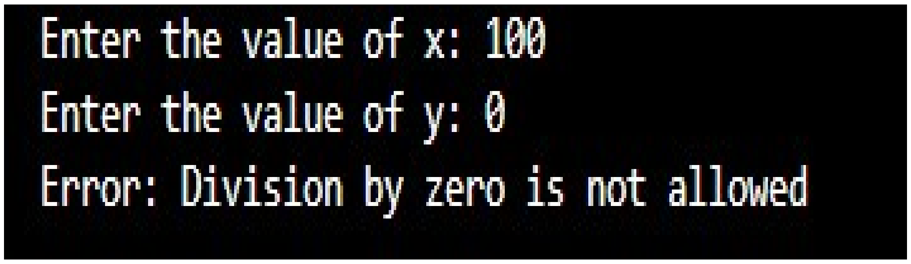
Part – 5

No.	Aim of the Practical
24.	Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.

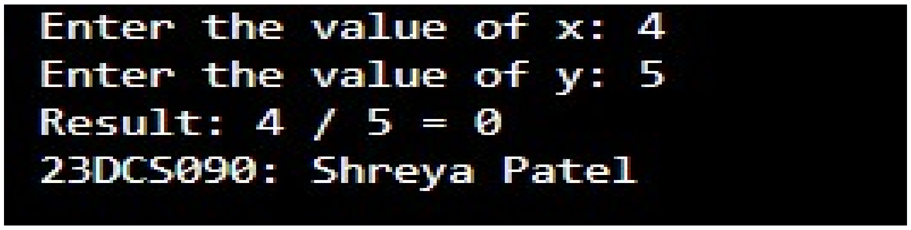
PROGRAM CODE :

```
import java.util.Scanner;

public class practical24 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {

            System.out.print("Enter the value of x: ");
            int x = Integer.parseInt(scanner.nextLine());
            System.out.print("Enter the value of y: ");
            int y = Integer.parseInt(scanner.nextLine());


            int result = x / y;
            System.out.println("Result: " + x + " / " + y + " = " + result);
            System.out.println("23DCS090: Shreya Patel");
        } catch (NumberFormatException 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 {
            scanner.close();
        }
    }
}
```

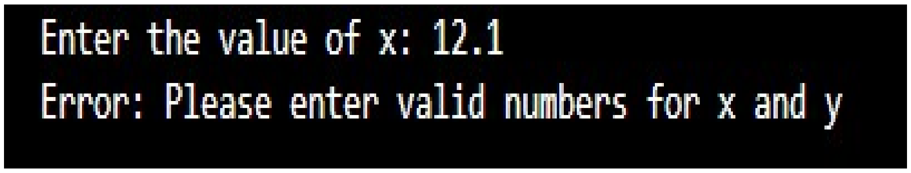
OUTPUT:



```
Enter the value of x: 100
Enter the value of y: 0
Error: Division by zero is not allowed
```



```
Enter the value of x: 4
Enter the value of y: 5
Result: 4 / 5 = 0
23DCS090: Shreya Patel
```



```
Enter the value of x: 12.1
Error: Please enter valid numbers for x and y
```

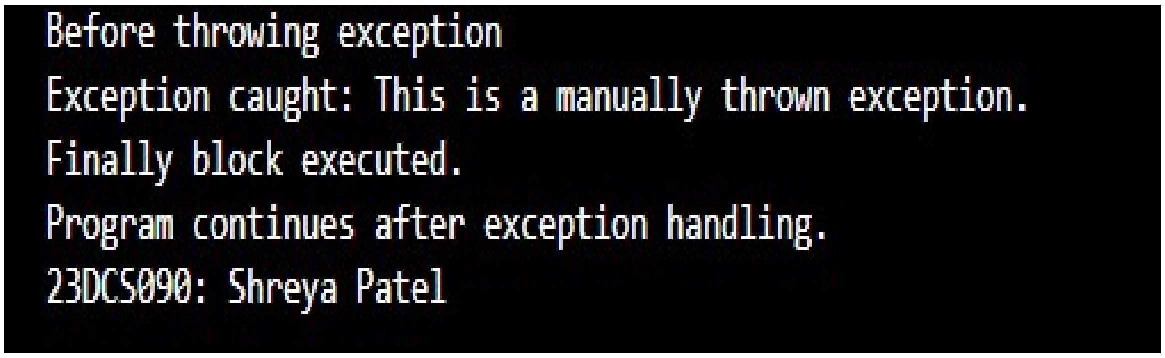
CONCLUSION: This program demonstrates the importance of input validation and exception handling in Java, ensuring that the program doesn't crash when invalid data or division by zero is encountered. It ensures robust behavior by managing common errors like invalid inputs and arithmetic exceptions.

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

PROGRAM CODE :

```
public class practical25 {
    public static void main(String[] args) {
        try {
            System.out.println("Before throwing
exception");
            throw new Exception("This is a
manually thrown exception.");
        } catch (Exception e) {

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

            System.out.println("Finally block
executed.");
        }

        System.out.println("Program continues
after exception handling.");
        System.out.println("23DCS090: Shreya
Patel");
    }
}
```

OUTPUT:



```
Before throwing exception
Exception caught: This is a manually thrown exception.
Finally block executed.
Program continues after exception handling.
23DCS090: Shreya Patel
```

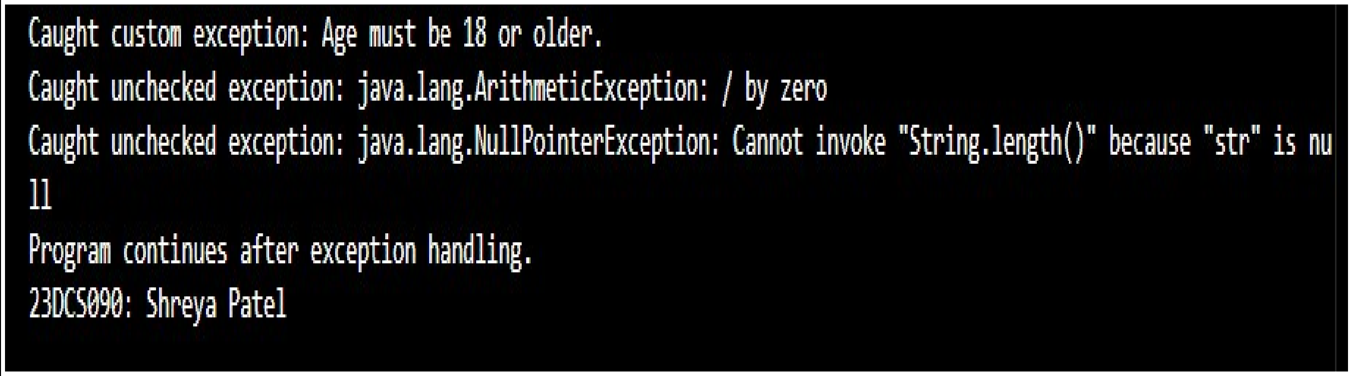
CONCLUSION: This program demonstrates how exceptions can be manually thrown in Java using the throw keyword. The try-catch block provides a structured way to handle these exceptions, ensuring that the program can continue running smoothly after encountering an error.

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 :

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

public class pract26{

   public static void checkAge(int age) throws
InvalidAgeException {
      if (age < 18) {
         throw new InvalidAgeException("Age
must be 18 or older.");
      } else {
         System.out.println("Access granted.");
      }
   }

   public static void main(String[] args) {
      try {
         checkAge(16);
      } catch (InvalidAgeException e) {
         System.out.println("Caught custom
exception: " + e.getMessage());
      }

      try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            System.out.println("Caught checked
exception: " + e);
        }

        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Caught unchecked
exception: " + e);
        }

        try {
            String str = null;
            str.length();
        } catch (NullPointerException e) {
            System.out.println("Caught unchecked
exception: " + e);
        }

        System.out.println("Program continues
after exception handling.");
        System.out.println("23DCS090: Shreya
Patel");
    }
}
```

OUTPUT:



```
Caught custom exception: Age must be 18 or older.
Caught unchecked exception: java.lang.ArithmeticException: / by zero
Caught unchecked exception: java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is nu
ll
Program continues after exception handling.
23DCS090: Shreya Patel
```

CONCLUSION: This program demonstrates the use of both checked and unchecked exceptions in Java, as well as how to create and throw user-defined exceptions using throw and throws. Checked exceptions (like InterruptedException) must be handled or declared, while unchecked exceptions (like ArithmeticException) occur during runtime and don't need explicit handling, but can cause program crashes if left unhandled**.**

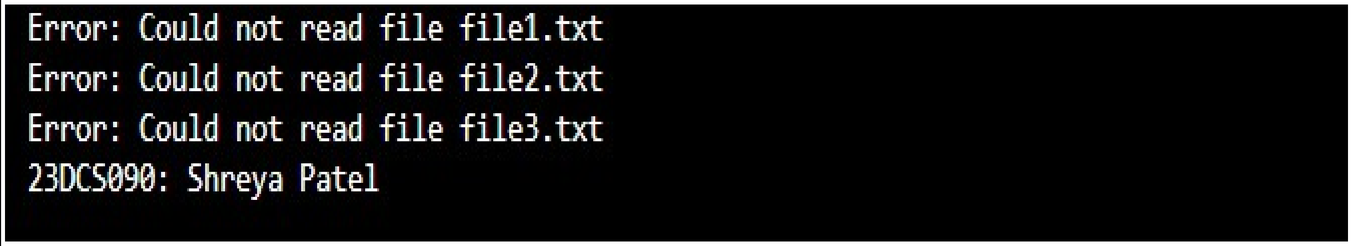
Part – 6

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

PROGRAM CODE :

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class practical27 {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Please specify at least one file name.");
            return;
        }

        for (String fileName : args) {
            countLinesInFile(fileName);
        }

        System.out.println("23DCS090: Shreya Patel");
    }

    private static void countLinesInFile(String fileName) {
        File file = new File(fileName);
        int lineCount = 0;

        try (Scanner scanner = new Scanner(file)) {
            while (scanner.hasNextLine()) {
                scanner.nextLine();
                lineCount++;
            }
            System.out.println(fileName + ": " + lineCount + " lines");
        } catch (FileNotFoundException e) {
            System.err.println("Error: Could not read file " + fileName);
        }
    }
}
```

OUTPUT:



```
Error: Could not read file file1.txt
Error: Could not read file file2.txt
Error: Could not read file file3.txt
23DCS090: Shreya Patel
```

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.

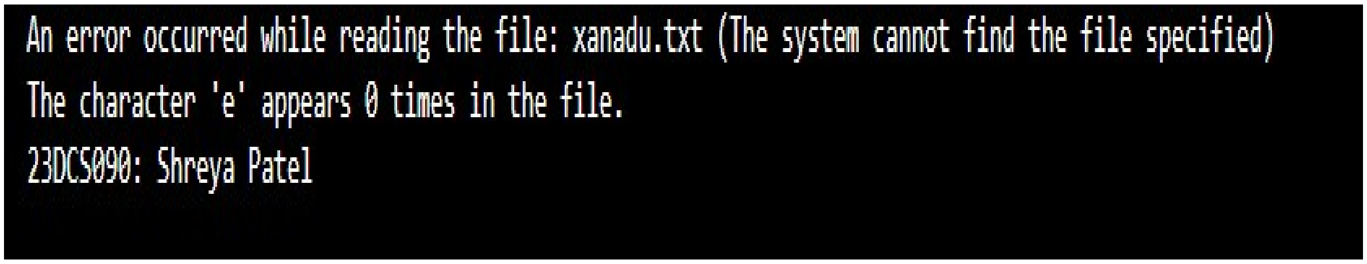
28. Write an example that counts the number of times aparticular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.

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 != 1) {
          System.out.println("Usage: java prac28
<character>");
          return;
       }

       char targetChar = args[0].charAt(0);
       int count = 0;

       try (BufferedReader br = new
BufferedReader(new FileReader("xanadu.txt")))
{
          String line;
          while ((line = br.readLine()) != null) {
             count += countCharacter(line,
targetChar);
          }
       } catch (IOException e) {
          System.out.println("An error occurred
while reading the file: " + e.getMessage());
       }

       System.out.println("The character '" +
targetChar + "' appears " + count + " times in
the file.");
       System.out.println("23DCS090: Shreya
Patel");
    }

    private static int countCharacter(String line,
char targetChar) {
       int count = 0;
       for (char c : line.toCharArray()) {
          if (c == targetChar) {
             count++;
          }
```

```
        }
        return count;
    }
}
```

OUTPUT:



```
An error occurred while reading the file: xanadu.txt (The system cannot find the file specified)
The character 'e' appears 0 times in the file.
23DCS090: Shreya Patel
```

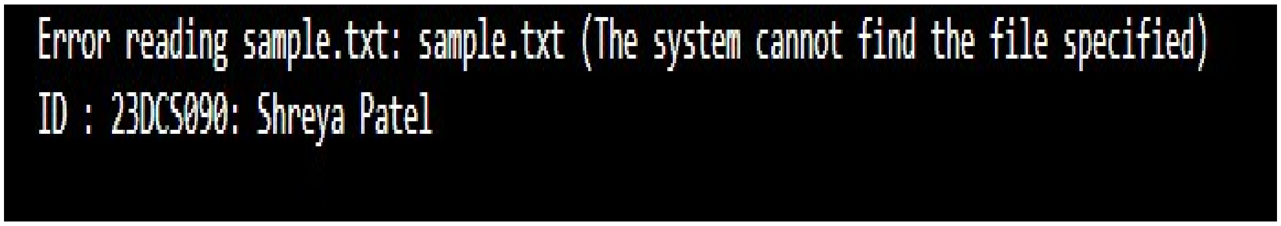
CONCLUSION: The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access.

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 Pract29 {
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 : 23DCS090: Shreya
Patel");
}}
```

OUTPUT:



```
Error reading sample.txt: sample.txt (The system cannot find the file specified)
ID : 23DCS090: Shreya Patel
```

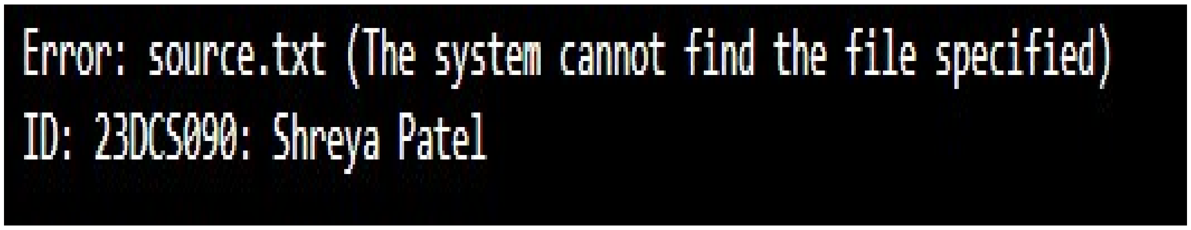
CONCLUSION: This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java.

30. Write a program to copy data from one file to another file. If the destination file does notexist, it is created automatically.

PROGRAM CODE :

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Prac30 {
    public static void main(String[] args) {
        // Check if the correct number of arguments are provided
        if (args.length < 2) {
            System.out.println("Usage: java Prac30 <source file> <destination file>");
            return;
        }

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

        // Use try-with-resources to ensure resources are closed automatically
        try (FileReader fr = new FileReader(sourceFile);
             FileWriter fw = new FileWriter(destinationFile)) {

            int ch;
            // Read from source file and write to destination file
            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());
        }

        // Print student ID and name
        System.out.println("ID: 23DCS090: Shreya Patel");
    }
}
```

OUTPUT:



```
Error: source.txt (The system cannot find the file specified)
ID: 23DCS090: Shreya Patel
```

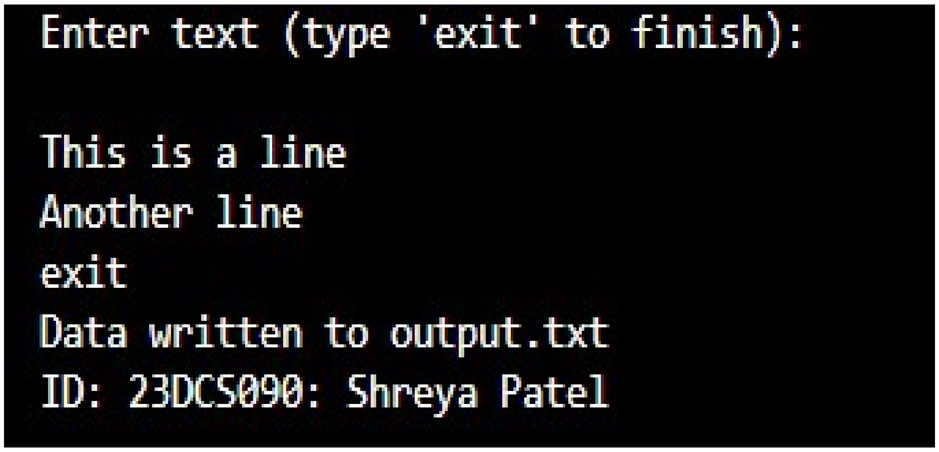
CONCLUSION: This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

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 Pract31 {
    public static void main(String[] args) {
        // Create a BufferedReader to read from the console
        BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
        String fileName = "output.txt";

        // Use try-with-resources to ensure resources are closed automatically
        try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {
            System.out.println("Enter text (type 'exit' to finish):");
            String input;
            // Read input from the console until 'exit' is typed
            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: 23DCS090: Shreya Patel");
    }
}
```

OUTPUT:



```
Enter text (type 'exit' to finish):

This is a line
Another line
exit
Data written to output.txt
ID: 23DCS090: Shreya Patel
```

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.

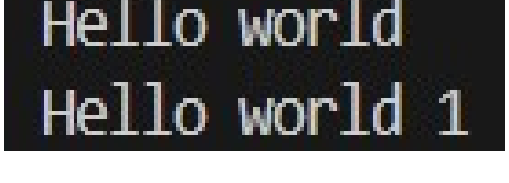
Part – 7

No.	Aim of the Practical
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
Hello world 1
```

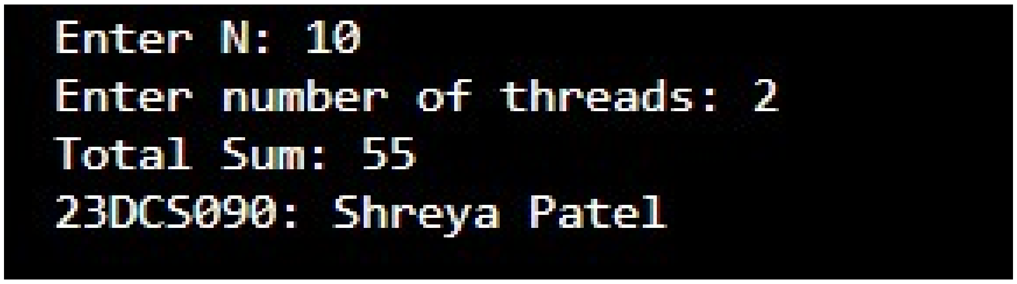
CONCLUSION: This program demonstrates two approaches to creating threads in Java: extending the Thread class and implementing the Runnable interface. Both methods effectively print "Hello World," showcasing the flexibility of Jvaa's concurrency model.

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 Practical33 {
    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());
        System.out.println("23DCS090: Shreya
Patel");
    }
}
```

OUTPUT:



```
Enter N: 10
Enter number of threads: 2
Total Sum: 55
23DCS090: Shreya Patel
```

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.

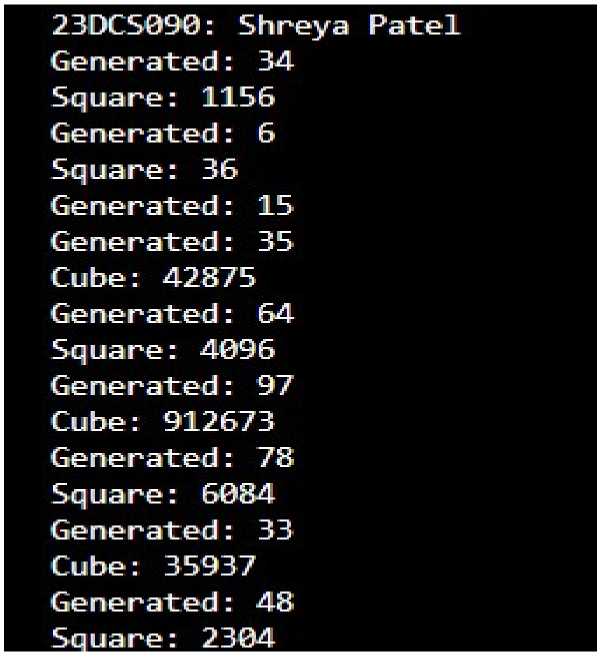
PROGRAM CODE :

```
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) {
               Practical34.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 (Practical34.lastNumber % 2 == 0)
{
                    int square = Practical34.lastNumber
* Practical34.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 (Practical34.lastNumber % 2 != 0) {
                    int cube = Practical34.lastNumber *
Practical34.lastNumber *
Practical34.lastNumber;
                    System.out.println("Cube: " +
cube);
                }
            }
        }
    }
}

public class Practical34 {
    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();

        System.out.println("23DCS090: Shreya
Patel");
    }
}
```

OUTPUT:



```
23DCS090: Shreya Patel
Generated: 34
Square: 1156
Generated: 6
Square: 36
Generated: 15
Generated: 35
Cube: 42875
Generated: 64
Square: 4096
Generated: 97
Cube: 912673
Generated: 78
Square: 6084
Generated: 33
Cube: 35937
Generated: 48
Square: 2304
```

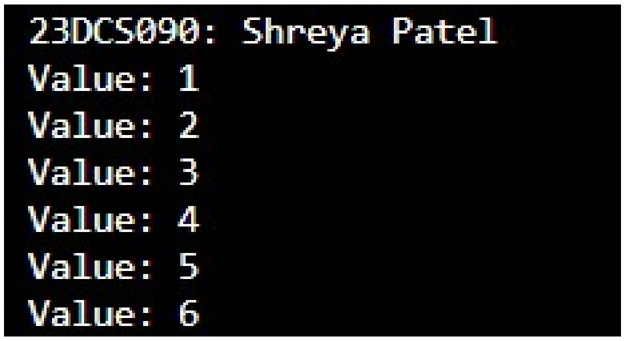
CONCLUSION: This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency 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 Practical35 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) {
Practical35 incrementer = new Practical35();
incrementer.start();
System.out.println("23DCS090: Shreya Patel ");
}}
```

OUTPUT:



```
23DCS090: Shreya Patel
Value: 1
Value: 2
Value: 3
Value: 4
Value: 5
Value: 6
```

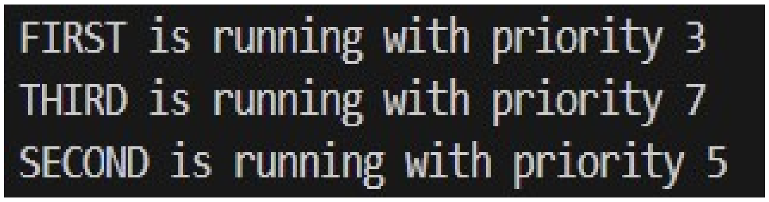
CONCLUSION This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java.

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 Practical36 {
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:

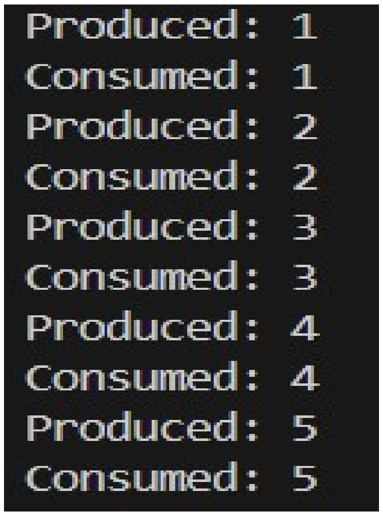
```
class Buffer {

private int data;

private booleanisEmpty = 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 Practical37 {
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 8

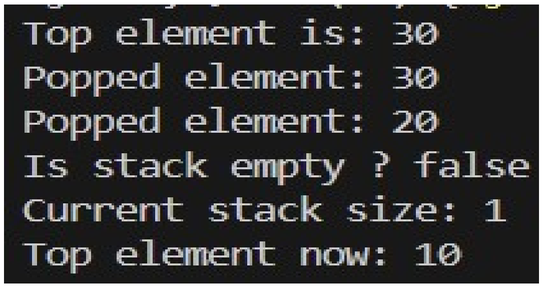
38	Design a Custom Stack using ArrayList class, whichimplements following functionalities of stack.
	My Stack -list ArrayList<Object>: A list to store elements.
	isEmpty: boolean: Returns true if this stack is empty.
	getSize(): int: Returns number of elements in this stack.
	peek(): Object: Returns 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.
	
	<u>PROGRAM CODE:</u>
	
	import java.util.ArrayList;
	
	class MyStack {
	
	private ArrayList<Object> list = new ArrayList<>();
	
	public booleanisEmpty() {
	
	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 Practical38 {
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 Practical39 {

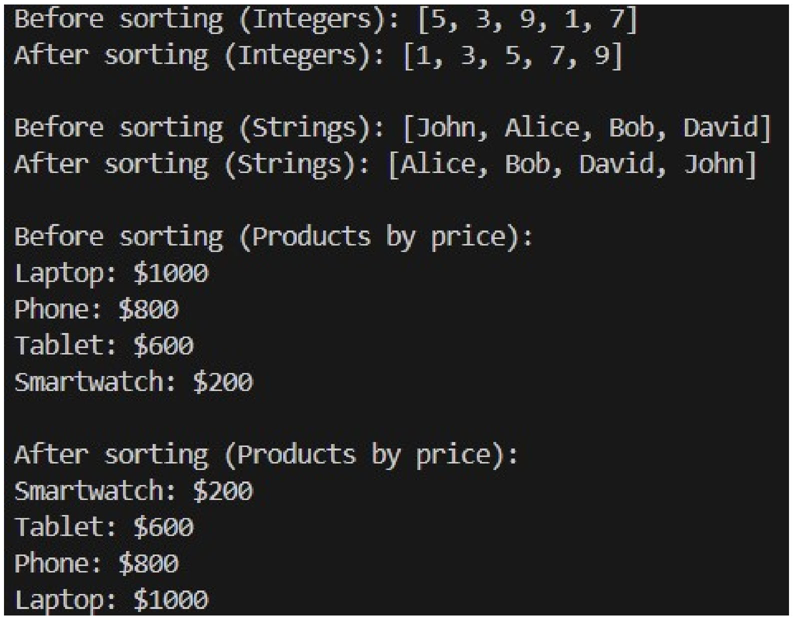
public static <T extends Comparable<T>> void sortArray(T[] array) {

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:

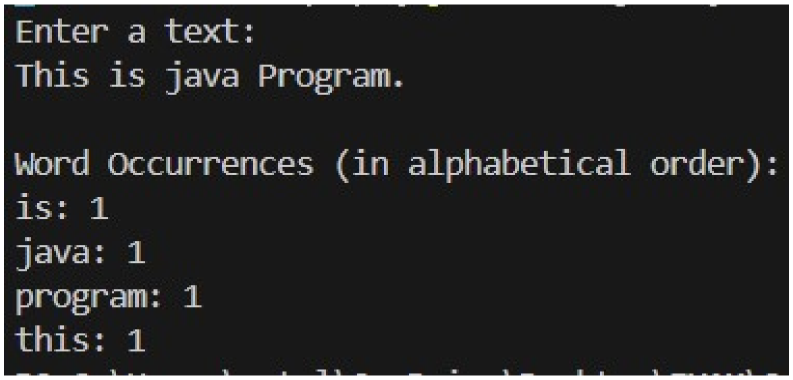
This program demonstrates the use of generics in Java to create a versatile sorting method for arrays of different types. By implementing the Comparable interface in the Product class, it enables sorting of custom objects based on specific criteria, such as price. The output shows the effective sorting 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 Practical40 {

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:

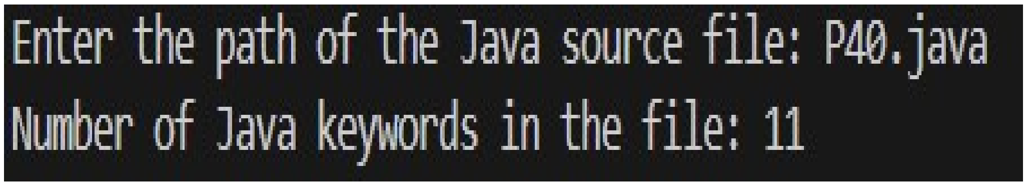
This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of getOrDefault() simplifies the counting process, showcasing efficient word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