Devang Patel Institute of Advance Technology and Research

DEPSTAR (A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./ Mrs. _VENISHA THAKKER_

of _DCSE2_ Class,

ID. No. _23DCS134_ has satisfactorily completed

his/ her term work in _JAVA_ for

the ending in _OCT_ 20 _24/20 25_

Date : 17/10/24

Sign. of Faculty   Head of Department

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY (CHARUSAT)
DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH
DEPSTAR

Subject : JAVA PROGRAMMING　　　　　　　　　　　**Semester:** 3

Subject Code: CSE201　　　　　　　　　　　　　　**Academic Year :** 2024-25

Course Outcome (COs):

At the end of the course, the students will be able to:

CO1	Comprehend Java Virtual Machine architecture and Java Programming Fundamentals.
CO2	Demonstrate basic problem-solving skills: analyzing problems, modelling a problem as a system of objects, creating algorithms, and implementing models and algorithms in an object-oriented computer language (classes, objects, methods with parameters)
CO3	Design applications involving Object Oriented Programming concepts such as inheritance, polymorphism, abstract classes and interfaces.
CO4	Build and test program using exception handling
CO5	Design and build multi-threaded Java Applications.
CO6	Build software using concepts such as files and collection frameworks.

Bloom's Taxonomy:

Level 1- Remembering
Level 2- Understanding
Level 3- Applying
Level 4- Analyzing
Level 5- Evaluating
Level 6- Creating

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY (CHARUSAT)
DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY AND RESEARCH
DEPSTAR

Practical List

Sr No.	AIM	Hrs.	CO	Bloom's Taxonomy
PART-I Data Types, Variables, String, Control Statements, Operators, Arrays				
1	Demonstration of installation steps of Java, Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.	2	1	1
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.	1	1	2,3,4
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).	1	1	2,3,4
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. **Supplementary Experiment:** You are creating a library management system. The library has two separate lists of books for fiction and non-fiction. The system should merge these lists into a single list for inventory purposes. Write a Java program to merge two arrays.	1	1, 2	2,3
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.	1	1, 2	2
6	Create a Java program that prompts the user to enter the	1	1, 2	2,3,4

	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. **Supplementary Experiment:** Imagine you are developing a classroom management system. You need to keep track of the grades of students in a class. After collecting the grades, you want to display each student's grade along with a message indicating if they have passed or failed. Let's assume the passing grade is 50.			
PART-II Strings				
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'	1	1, 2	2,3,4
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 **Supplementary Experiment:** 1. Write a Java program to replace each substring of a given string that matches the given regular expression with the given replacement. Sample string : "The quick brown fox jumps over the lazy dog." **In the above string replace all the fox with cat.**	1	1, 2	2,3
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'	1	1, 2	2
10	Perform following functionalities of the string: ● Find Length of the String ● Lowercase of the String ● Uppercase of the String ● Reverse String	1	1, 2	2,3,4

	Sort the string			
11	Perform following Functionalities of the string: "CHARUSAT UNIVERSITY" ● Find length ● Replace 'H' by 'FIRST LATTER OF YOUR NAME' ● Convert all character in lowercase **Supplementary Experiment:** 1. Write a Java program to count and print all duplicates in the input string. Sample Output: The given string is: resource The duplicate characters and counts are: e appears 2 times r appears 2 times	1	1, 2	4
	PART-III Object Oriented Programming: Classes, Methods, Constructors			
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.	1	2	3
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.	2	1, 2	3
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.	2	1, 2	**3**

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. **Supplementary Experiment:** 1.Write a Java program to create a class called "Airplane" with a flight number, destination, and departure time attributes, and methods to check flight status and delay. [L:M]	1	1, 2	3
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.	1	1, 2	2,3
PART-IV Inheritance, Interface, Package				
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	1	1, 2, 3	3
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.	2	1, 2, 3	3
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	1	2,3	3

	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. **Supplementary Experiment:** 1.Write a Java program to create a vehicle class hierarchy. The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance traveled, and maximum speed. [L:A]			
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.	**2**	**2,3**	**3**
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.	**1**	**2,3**	**3**
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. **Supplementary Experiment:** 1.Write a Java programming to create a banking system with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw,	**2**	**2,3**	**2,3**

	calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods. [L:A]			
23	Assume you want to capture shapes, which can be either circles (with a radiusand a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs. Write a program that illustrates the significance of interface default method.	2	2,3	6
PART-V Exception Handling				
24	Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.	1	4	3
25	Write a Java program that throws an exception and catch it using a try-catch block.	1	4	3
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). **Supplementary Experiment:** 1.Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates. [L:M]	2	4	2,3
PART-VI File Handling & Streams				
27	Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.	1	4,6	3
28	Write an example that counts the number of times a	1	4,6	3

CSE201- JAVA PROGRAMMING PRACTICAL LIST

	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.			
29	Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.	**2**	**4,6**	**3**
30	Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically. **Supplementary Experiment:** 1.Write a Java program to sort a list of strings in alphabetical order, ascending and descending using streams.	**2**	**4,6**	**3**
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.	**2**	**4,6**	**2,3**
PART-VII Multithreading				
32	Write a program to create thread which display "Hello World" message. A. by extending Thread class B. by using Runnable interface.	**1**	**5,6**	**3**
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.	**1**	**5,6**	**3**
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.	**2**	**5,6**	**3**
35	Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.	**2**	**5,6**	**2,3**
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.	**2**	**5,6**	**2,3**
37	Write a program to solve producer-consumer problem using thread synchronization.	**2**	**5,6**	**3**
PART-VIII Collection Framework and Generic				
38	Design a Custom Stack using ArrayList class, which	**2**	**5**	**3**

	implements following functionalities of stack. My Stack -list ArrayList<Object>: A list to store elements. +isEmpty: boolean: Returns true if this stack is empty. +getSize(): int: Returns number of elements in this stack. +peek(): Object: Returns top element in this stack without removing it. +pop(): Object: Returns and Removes the top elements in this stack. +push(o: object): Adds new element to the top of this stack.			
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.	2	5	6
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.	2	5	3
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.	2	5	2,3

CSE201- JAVA PROGRAMMING PRACTICAL LIST

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,or BlueJ and Console Programming. **OUTPUT:** Installation Steps of Java 1. Download the JDK 2. Install the JDK 3. Set Up Environment Variables (Windows) 4. Verify the Installation

Introduction to Object-Oriented Concepts

1. Class and Object

Class: A blueprint for creating objects.

Object: An instance of a class.

2. Inheritance

- Mechanism where one class inherits the fields

and methods of another.

3. Polymorphism

- Ability of different classes to respond to the

same method call in different ways.

4. Encapsulation

- Wrapping of data and methods into a single

unit (class), and restricting access to some of the

object's components.

5. Abstraction

- Hiding complex implementation details and

showing only the necessary features.

:: Java vs. C++

- Memory Management: Java has automatic

garbage collection, whereas C++ requires

manual memory management.

- Platform Independence: Java is platform-

independent at the source level (write once, run

anywhere), while C++ is platform-dependent.

- Pointers: Java does not support pointers

explicitly for security reasons, whereas C++

supports pointers.

:: Introduction to JDK, JRE, JVM, Javadoc,

  JDK (Java Development Kit)

- A software development kit used to develop

Java applications.

- Includes JRE, an interpreter/loader (Java), a

compiler (javac), an archiver (jar), a

documentation generator (Javadoc), and other

tools.

JRE (Java Runtime Environment)

- Provides the libraries, Java Virtual Machine

(JVM), and other components to run

applications written in Java.

JVM (Java Virtual Machine)
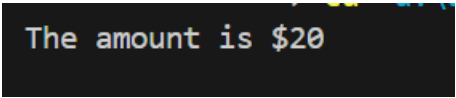
- An abstract machine that enables your

computer to run a Java program.

- Converts bytecode into machine code,

ensuring Java programs can run on any

platform.

| 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 :

```java
class pract2 {
    public static void main(String[] args) {
        int a=20;
        System.out.println("The amount is $"+ a);
    }
}
```

OUTPUT:

```
The amount is $20
```

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) 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 :

```java
import java.util.Scanner;
class pract3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner (System.in);
        double distance;
        int hours,min,sec;

        System.out.println("Enter distance in metres: ");
        distance = sc.nextDouble();
        System.out.println("Enter hours: ");
        hours = sc.nextInt();
        System.out.println("Enter minutes: ");
        min = sc.nextInt();
        System.out.println("Enter seconds: ");
        sec = sc.nextInt();

        int totalTime = (hours * 3600) + (min * 60) + sec;
        double speedMetersPerSecond = distance / totalTime;
        double speedKilometersPerHour = (distance / 1000) / (totalTime / 3600);
        double speedMilesPerHour = (distance / 1609) / (totalTime / 3600);

        System.out.printf("Speed in meters per second: %.2f%n", speedMetersPerSecond);
        System.out.printf("Speed in kilometers per hour: %.2f%n", speedKilometersPerHour);
```

```java
        System.out.printf("Speed in miles per hour: %.2f%n", speedMilesPerHour);


    }
}
```

OUTPUT:

```
Enter distance in metres:
10000
Enter hours:
1
Enter minutes:
30
Enter seconds:
0
Speed in meters per second: 1.85
Speed in kilometers per hour: 10.00
Speed in miles per hour: 6.22
```

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. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to calculate the sum of elements in an array representing daily expenses.

PROGRAM CODE :

```java
import java.util.Scanner;
class pract4 {
    public static void main(String args[]) {
        double total_exp = 0;
        System.out.print("Enter number of days in a month: ");
        Scanner sc = new Scanner(System.in);
        int days = sc.nextInt();

        double arr[] = new double[days];
        for (int i = 0; i < days; i++) {
            System.out.print("Enter expense of day " + (i + 1) + ":");
            arr[i] = sc.nextDouble();
            total_exp = total_exp + arr[i];
        }
        System.out.println("Total expenses for the month : " + total_exp);
    }
}
```

OUTPUT:

```
Enter number of days in a month: 30
Enter expense of day 1:1
Enter expense of day 2:1
Enter expense of day 3:1
Enter expense of day 4:1
Enter expense of day 5:1
Enter expense of day 6:1
Enter expense of day 7:1
Enter expense of day 8:1
Enter expense of day 9:1
Enter expense of day 10:1
Enter expense of day 11:1
Enter expense of day 12:1
Enter expense of day 13:1
Enter expense of day 14:1
Enter expense of day 15:1
Enter expense of day 16:1
Enter expense of day 17:1
Enter expense of day 18:1
Enter expense of day 19:1
Enter expense of day 20:1
Enter expense of day 21:1
Enter expense of day 22:1
Enter expense of day 23:1
Enter expense of day 24:1
Enter expense of day 25:1
Enter expense of day 26:1
Enter expense of day 27:1
Enter expense of day 28:1
Enter expense of day 29:1
Enter expense of day 30:1
Total expenses for the month : 30.0
```

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 to fan,3 to tube and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor,12% to fan,5% to tube light,7.5% to wires and 3% for all other items is charged. A list containing the product code and price in two different arrays. Write a java program using switch statement to prepare the bill.

PROGRAM CODE :

```java
import java.util.Scanner;
public class prac5 {
  public static void main(String[] args)
  {
    Scanner sc= new Scanner(System.in);
    int code[]={1,2,3,4,5,6};
    double price[]=new double[5];
    double tax,total=0;
    int no;
    do{

    System.out.println("1.Motor\n2.Fan\n3.Tube\n4.Wires\n5.Other items\n6.Exit");
    System.out.println("Enter code:");
     no= sc.nextInt();
    switch(code[no-1])
    {
      case 1:
      {
        System.out.print("Enter Price of motor:");
        price[0]=sc.nextDouble();
        tax=price[0]*(8/100);
        total=total+(price[0]+tax);
```

```java
        }
        break;
        case 2:
        {
          System.out.print("Enter Price of fan:");
          price[1]=sc.nextDouble();
          tax=price[1]*(12/100);
          total=total+(price[1]+tax);


        }
        break;
        case 3:
        {
          System.out.print("Enter Price of tube:");
          price[2]=sc.nextDouble();
          tax=price[2]*(5/100);
          total=total+(price[2]+tax);
        }
        break;
        case 4:
        {
          System.out.print("Enter Price of wires:");
          price[3]=sc.nextDouble();
          tax=price[3]*(7.5/100);
          total=total+(price[3]+tax);
        }
        break;
```

```java
        case 5:
        {
            System.out.print("Enter Price of other:");
            price[4]=sc.nextDouble();
            tax=price[4]*(3/100);
            total=total+(price[4]+tax);
        }
        break;
        case 6:
        break;
        default:
        System.out.println("wrong");
        break;
    }
  }while(no!=6);
  System.out.println("TOTAL BILL PRICE:"+total);
}
}
```

OUTPUT:

```
1.Motor
2.Fan
3.Tube
4.Wires
5.Other items
6.Exit
Enter code:
1
Enter Price of motor:1000
1.Motor
2.Fan
3.Tube
4.Wires
5.Other items
6.Exit
Enter code:
2
Enter Price of fan:1000
1.Motor
2.Fan
3.Tube
4.Wires
5.Other items
6.Exit
Enter code:
3
Enter Price of tube:500
```

```
1.Motor
2.Fan
3.Tube
4.Wires
5.Other items
6.Exit
Enter code:
4
Enter Price of wires:500
1.Motor
2.Fan
3.Tube
4.Wires
5.Other items
6.Exit
Enter code:
5                      .
Enter Price of other:500
1.Motor
2.Fan
3.Tube
4.Wires
4.Wires
5.Other items
6.Exit
Enter code:
6
TOTAL BILL PRICE:3537.5
```

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 and demonstrates formatted output for a detailed bill summary.

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

PROGRAM CODE :

```java
import java.util.*;
class pract6 {
    public static void main(String []args){
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter the number of days : ");
        int n=sc.nextInt();
        int n1=0,n2=1,n3;
        System.out.println("Your exercise routine for the next " + n + " days:");
        for (int i = 1; i <= n; i++) {
            if (i == 1) {
                System.out.println("Day " + i + ": " + n1 + " minutes");
            } else if (i == 2) {
                System.out.println("Day " + i + ": " + n2 + " minutes");
            } else {
                n3 =n1 + n2;
                n1 = n2;
                n2 = n3;
                System.out.println("Day " + i + ": " + n3 + " minutes");
            }
        }
    }
}
```

OUTPUT:

```
Enter the number of days : 10
Your exercise routine for the next 10 days:
Day 1: 0 minutes
Day 2: 1 minutes
Day 3: 1 minutes
Day 4: 2 minutes
Day 5: 3 minutes
Day 6: 5 minutes
Day 7: 8 minutes
Day 8: 13 minutes
Day 9: 21 minutes
Day 10: 34 minutes
```
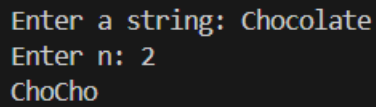
CONCLUSION:

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.util.Scanner;```
	
	 ``` class prac7 { ... ```

PROGRAM CODE :

```java
import java.util.Scanner;

class prac7 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a string: ");
    String s1 = sc.nextLine();

    System.out.print("Enter n: ");
    int n = sc.nextInt();

    String result = front_times(s1, n);
    System.out.println(result);
  }

  static String front_times(String s1, int n) {
    String s;
    if (s1.length() < 3) {
      s = s1;
    } else {
      s = s1.substring(0, 3);
    }
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < n; i++) {
      sb.append(s);
    }
```

```
        return sb.toString();
    }
}
```

OUTPUT:

```
Enter a string: Chocolate
Enter n: 2
ChoCho
```

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. The program concludes by displaying a signature line.

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

PROGRAM CODE :

```java
class prac8 {
  public static void main(String[] args){
    int arr1[]={1,2,9};
    int arr2[]={1,9,9};
    int arr3[]={1,9,9,3,9};
    array_count(arr1);
    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("Num of times 9 appear
in given array :"+count);

  }

}
```

OUTPUT:

```
Num of times 9 appear in given array  :1
Num of times 9 appear in given array  :2
Num of times 9 appear in given array  :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 theoriginal, there are two chars.

double_char('The') → 'TThhee'

double_char('AAbb') → 'AAAAbbbb'

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

PROGRAM CODE :

```java
import java.util.Scanner;
 class pract9 {
   public static void main(String[] args) {
      Scanner sc= new Scanner (System.in);
    System.out.print("Enter string : ");
      String s1 = sc.nextLine();
      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));
      }
   }
}
```
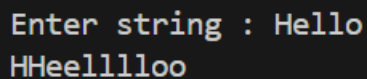
OUTPUT:

```
Enter string : Hello
HHeelllloo
```

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 :

```java
import java.util.Arrays;
import java.util.Scanner;
class prac10 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter string : ");
    String s = sc.nextLine();
    System.out.println("Length of String: " + s.length());
    System.out.println("Lowercase of the string: " + s.toLowerCase());
    System.out.println("Uppercase of the string: " + s.toUpperCase());
    System.out.print("String in reverse order :");
    for (int i = s.length() - 1; i >= 0; i--) {
      System.out.print(s.charAt(i));
    }
    System.out.println();
    char[] temp = s.toCharArray();
    Arrays.sort(temp);
    String sorted_string = new String(temp);
    System.out.print("Sorted String:");
    System.out.println(sorted_string);
```

```
    }

}
```

OUTPUT:

```
Enter string : venisha
Length of String: 7
Lowercase of the string: venisha
Uppercase of the string: VENISHA
String in reverse order :ahsinev
Sorted String:aehinsv
```

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 :

```java
public class prac11 {
    public static void main(String [] args){
        String s1="CHARUSAT UNIVERSITY";
        System.out.println("Length of string is : " + s1.length());
        System.out.println("Lower case : "+ s1.toLowerCase());
        System.out.println("Replace character : " + s1.replace('H', 'K'))

    }
}
```

OUTPUT:

```
Length of string is : 19
Lower case : charusat university
Replace character : CKARUSAT UNIVERSITY
```

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. <br><br> **PROGRAM CODE :** <br><br> ```java
import java.util.*;

class Currency {

   void convert(double p) {
      double rupee = p * 100;
      System.out.println("Amount in rupees :" + rupee);
   }
}

public class Pract12 {

   public static void main(String[] args) {
      double p = Double.parseDouble(args[0]);
      Currency p1 = new Currency();
      System.out.println("Amount in pound : "+p);
      p1.convert(p);
   }
}
``` |

OUTPUT:

```
C:\Windows\System32>D:

D:\>cd Java lang

D:\Java lang>javac Pract12.java

D:\Java lang>java Pract12 1
Amount in pound : 1.0
Amount in rupees :100.0
```

CONCLUSION:

This Java program converts an input value in pounds to rupees using a fixed conversion rate of 100 rupees per pound. It demonstrates basic command-line argument handling and arithmetic operations in Java for currency conversion.

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:

```java
import java.util.*;
class Employee{
double salary;
String fname,lname;

Employee(){
fname = null;
lname = null;
salary = 0.0;
}

void get()
{
Scanner sc = new Scanner(System.in);
System.out.print("Enter your first name:");
fname = sc.nextLine();
System.out.print("Enter your last name:");
lname = sc.nextLine();
System.out.print("Enter your salary:");
salary = sc.nextDouble();
}

void display(){
 System.out.println("First name :" + fname);
System.out.println("Last name :" + lname);
if(salary<0.0){
salary = 0.0;
System.out.println("Please enter valid salary.");
}
```

```java
else{
 System.out.println("Yearly salary :" + salary * 12);
System.out.println("Increased yearly salary is :" + (salary + (salary *12 * 0.1)));
}
}
}
class Pract13{
public static void main(String [] args){
System.out.println("Employee 1:");
Employee e1 = new Employee();
e1.get();
e1.display();
System.out.println("\nEmployee 2:");
Employee e2 = new Employee();
e2.get();
e2.display();
}
}
```

OUTPUT:

```
Employee 1:
Enter your first name:Keya
Enter your last name:Sonaiya
Enter your salary:100000
First name :Keya
Last name :Sonaiya
Yearly salary :1200000.0
Increased yearly salary is :220000.0

Employee 2:
Enter your first name:Vishva
Enter your last name:Valand
Enter your salary:200000
First name :Vishva
Last name :Valand
Yearly salary :2400000.0
Increased yearly salary is :440000.0
```

CONCLUSION:

The Java program efficiently manages employee details by capturing user input for name and salary, then calculating and displaying the yearly salary. It handles salary validation and applies a 10% increase for output.

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

PROGRAM CODE:
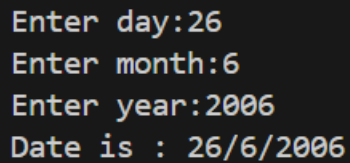
```java
import java.util.*;
class Date {
int month, day, year;
Date()
{
}
Date(int d, int m, int y) {
 day = d;
 month = m;
 year = y;
}
void get()
{
Scanner sc = new Scanner(System.in);
System.out.print("Enter day:");
day = sc.nextInt();
System.out.print("Enter month:");
month = sc.nextInt();
System.out.print("Enter year:");
year = sc.nextInt();
}
```

```java
void display() {
System.out.println("Date is : " + day + "/" + month + "/" + year);
}
}
class Pract14{
public static void main(String [] args){
Date d1 = new Date();
d1.get();
d1.display();
}
}
```

OUTPUT:

```
Enter day:26
Enter month:6
Enter year:2006
Date is : 26/6/2006
```

CONCLUSION:

The Java program defines a Date class with methods to input and display a date. It allows the user to enter a date interactively and then prints it in day/month/year format. The program demonstrates basic object-oriented principles by using encapsulation for the Date class and provides a straightforward approach to managing 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:

```java
import java.util.*;
class area{
    double length;
    double breadth;

    area(double l, double b){
        length=l;
        breadth=b;
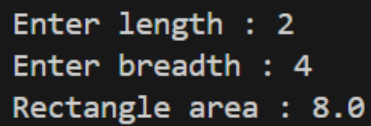    }

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

public class Pract15 {

    public static void main(String[] args){
        Scanner sc= new Scanner(System.in);
        System.out.print("Enter length : ");
        double l= sc.nextDouble();
        System.out.print("E2nter breadth : ");
        double b= sc.nextDouble();
        area a1= new area(l, b);
        System.out.println("Rectangle area : " + a1.returnArea());
```

```
    }


}
```

OUTPUT:

```
                          > c
 Enter length : 2
 Enter breadth : 4
 Rectangle area : 8.0
```

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:

```java
import java.util.Scanner;
class Complex {
    int real, img;

    Complex() {}

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

    Complex add(Complex c) {
        return new Complex(real + c.real, img + c.img);
    }

    Complex sub(Complex c) {
        return new Complex(real - c.real, img - c.img);
    }

    Complex mul(Complex c) {
        int realPart = real * c.real - img * c.img;
        int imgPart = real * c.img + img * c.real;
        return new Complex(realPart, imgPart);
    }

    void display() {
```

```java
      if (img >= 0) {
        System.out.println(real + " + " + img + "i");
      } else {
        System.out.println(real + " - " + (-img) + "i");
      }
   }
}


public class Pract16 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter real part of first complex number: ");
      int r1 = sc.nextInt();
      System.out.print("Enter imaginary part of first complex number: ");
      int i1 = sc.nextInt();
      System.out.print("Enter real part of second complex number: ");
      int r2 = sc.nextInt();
      System.out.print("Enter imaginary part of second complex number: ");
      int i2 = sc.nextInt();

      Complex c1 = new Complex(r1, i1);
      Complex c2 = new Complex(r2, i2);

      System.out.print("First complex number: ");
      c1.display();
      System.out.print("Second complex number: ");
      c2.display();

      Complex s = c1.add(c2);
      System.out.print("Addition of two complex numbers: ");
```

```
        s.display();


    Complex diff = c1.sub(c2);
    System.out.print("Subtraction of two complex numbers:");
    diff.display();


    Complex prod = c1.mul(c2);
    System.out.print("Multiplication of two complex numbers: ");
    prod.display();


  }
}
```

OUTPUT:

```
Enter imaginary part of first complex number: 1
Enter real part of second complex number: 2
Enter imaginary part of second complex number: 2
First complex number: 1 + 1i
Second complex number: 2 + 2i
Addition of two complex numbers: 3 + 3i
Subtraction of two complex numbers:-1 - 1i
Multiplication of two complex numbers: 0 + 4i
```

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 :

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

public class Pract17 {
   public static void main(String[] args) {
   Parent p = new Parent();
   p.printP();

   Child c = new Child();
   c.printC();
   }

}
```

OUTPUT:

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

CONCLUSION:

From This practical we learn about introduction of inheritance in java. we accessed parent classs method from child class object.

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

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 :

```java
class Member {

   String name, add;
   int age;
   double salary;
   long num;

   void printSalary() {
      System.out.println("Salary is: " + salary);
   }
   void display()
   {
      System.out.println("Name is: " + name);
      System.out.println("Address is: " + add);
      System.out.println("Age is: " + age);
      System.out.println("Num is: " + num );
   }
}

class Employee extends Member {

   String sp;
   void Speci() {
      System.out.println("Specificaton is: " + sp);
```

```
    }

}

class Manager extends Member {

    String dep;
    void Dep() {
        System.out.println("Department is: " + dep);
    }
}

public class Pract18 {

    public static void main(String[] args) {
        Employee e = new Employee();
        System.out.println("\nEmployee details:");
        e.name="Keya";
        e.add="Abc";
        e.age=18;
        e.salary=100000;
        e.num=1234567898;
        e.sp="xyz";
        e.display();
        e.printSalary();
        e.Speci();

        Manager m = new Manager();
        System.out.println("\nManager details:");
        m.name="Keya";
        m.add="Abc";
        m.age=18;
        m.salary=100000;
        m.num=1234567898;
        m.dep="xyz";
        m.display();
        m.printSalary();
        m.Dep();
```

```
    }
}
```

OUTPUT:

```
Employee details:
Name is: Keya
Address is: Abc
Age is: 18
Num is: 1234567898
Salary is: 100000.0
Specificaton is: xyz

Manager details:
Name is: Keya
Address is: Abc
Age is: 18
Num is: 1234567898
Salary is: 100000.0
Department is: xyz
```

CONCLUSION:

The Java code demonstrates object-oriented programming concepts through the creation of Member, Employee, and Manager classes. It allows users to input and display details of employees and managers, showcasing inheritance and polymorphism. The code serves as a basic example of an employee management system.

| 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 :

```java
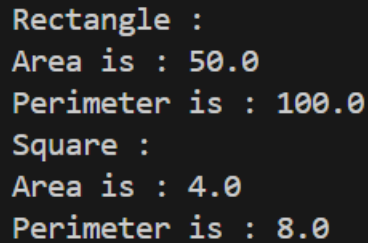class Rectangle {

    double length, breadth;
    Rectangle(double l, double b) {
        length = l;
        breadth = b;
    }

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

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

class Square extends Rectangle {

    Square(double s) {
        super(s, s);
    }
    void area()
    {
        super.area();
    }
    void perimeter()
    {
        super.perimeter();
    }
```

```
    }

 class Pract19 {

    public static void main(String[] args) {
       Rectangle[] arr = new Rectangle[2];
       arr[0] = new Rectangle(10.0, 5.0);
       arr[1] = new Square(2.0);
       System.out.println("Rectangle : ");
       arr[0].area();
       arr[0].perimeter();
       System.out.println("Square : ");
       arr[1].area();
       arr[1].perimeter();


    }
 }
```

OUTPUT:

```
Rectangle :
Area is : 50.0
Perimeter is : 100.0
Square :
Area is : 4.0
Perimeter is : 8.0
```

CONCLUSION:

The code demonstrates the concept of inheritance in Java, where a Square class extends the Rectangle class and overrides its methods to calculate perimeter and area. The Square class also has its own methods to calculate its perimeter and area. The code creates an array of Square objects and demonstrates polymorphism by calling the methods of both Rectangle and Square classes on the same object.

| 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 :

```java
class Shape
{
    void print1()
    {
        System.out.println("This is shape.");
    }
}
class Rectangle extends Shape{

    void print2()
    {
        System.out.println("This is rectangular shape.");
    }
}
class Circle extends Shape{

    void print3()
    {
        System.out.println("This is circular shape.");
    }
}
class Square extends Rectangle{
    void print4()
    {
        System.out.println("Square is rectangle.");
    }
}
public class Pract20 {
    public static void main(String[] args) {
        Square s = new Square();
        s.print1();
```

```
        s.print2();


     }
}
```

OUTPUT:

```
This is shape.
This is rectangular shape.
```

CONCLUSION:

In this practical, we demonstrated the concept of inheritance in Java by creating a hierarchy of shapes, where Square inherits properties from Rectangle, which in turn inherits from Shape. We successfully called methods from parent classes using an object of the child class, showcasing the power of inheritance in object-oriented programming. This exercise helped us understand the relationships between classes and how to reuse code through 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 :

```java
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 d = new Degree();
    Undergraduate u = new Undergraduate();
    Postgraduate p = new Postgraduate();
    d.getDegree();
    u.getDegree();
    p.getDegree();
  }
}
```

OUTPUT:

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

CONCLUSION:

This Java code demonstrates the concept of inheritance and method overriding. The Degree class serves as the parent class, while Undergraduate and Postgraduate classes extend it and override the getdegree() method to provide their specific implementations. The code successfully showcases the execution of the overridden methods, displaying the expected output for each degree type.

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 :

```java
import java.util.Scanner;
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=sum+i;
     }
   }
   return sum;
}
}
public class Pract22 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
            System.out.print("Enter n:");
            int n = sc.nextInt();
            MyCalculator obj = new MyCalculator();
            System.out.println("Sum of divisors of "+ n + " is " + obj.divisor_sum(n));
   }
}
```

OUTPUT:

```
Enter n:6
Sum of divisors of 6 is 12
```

CONCLUSION:

In this Java program we implemented interface which is functionality of java language. As java is not supporting multiple inheritance. We can archive it by using of interface here In this program one class is implementing interface.

| 23. | Assume you want to capture shapes, which can be either circles (with a radiusand a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs. Write a program that illustrates the significance of interface default method. |

PROGRAM CODE :

```java
interface Shape{
   String getcolor();
   double area();

   default void display(){
      System.out.println("Color is : " + getcolor());
      System.out.println("Area is : " + area());
   }
}

class Circle implements Shape{
   double radius;
   String color;
   public Circle(double radius,String color){
      this.radius = radius;
      this.color = color;
   }
   public String getcolor(){
      return color;
   }
   public double area()
   {
      return 3.14 * radius * radius;
   }
}

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

```java
            this.width = width;
            this.color = color;
        }
        public String getcolor() {
            return color;
        }
        public double area() {
            return length * width;
        }
    }

    class Sign{
        Shape s;
        String text;
        public Sign(Shape s,String text){
            this.s = s;
            this.text = text;
        }
        public void displaySign()
        {
            s.display();
            System.out.println("Sign text: " + text);
        }
    }
    public class Pract23 {
        public static void main(String[] args) {

            Shape c = new Circle(5, "Red");
            Shape r = new Rectangle(4, 6, "Blue");

            Sign c1 = new Sign(c, "Welcome to the Campus!");
            Sign r1 = new Sign(r, "Library Ahead");

            System.out.println("Circle Sign:");
            c1.displaySign();

            System.out.println("\nRectangle Sign:");
            r1.displaySign();}
```

}

OUTPUT:

```
Circle Sign:
Color is : Red
Area is : 78.5
Sign text: Welcome to the Campus!

Rectangle Sign:
Color is : Blue
Area is : 24.0
Sign text: Library Ahead
```

CONCLUSION:

This Java program demonstrates the implementation of an interface with default and abstract methods. It showcases polymorphism by creating different shapes (circle and rectangle) that each implement the interface and define their unique properties while inheriting a common behavior for the color.

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 Pract24 { public static void main(String[] args) { Scanner scanner = new Scanner(System.in); try { System.out.print("Enter x: "); int x = scanner.nextInt(); System.out.print("Enter y: "); int y = scanner.nextInt(); int result = x / y; System.out.println("Result: " + x + " / " + y + " = " + result); } catch (ArithmeticException e) { System.out.println("Error: Division by zero is not allowed."); } } } ```
	
	 OUTPUT:
	
	 ``` Enter x: 4 Enter y: 0 Error: Division by zero is not allowed. ```

CONCLUSION:

This program demonstrates exception handling in Java by attempting to divide two integers input by the user. If the user enters non-integer values or if the divisor is zero, an exception will occur, which is caught and reported. This ensures the program handles errors gracefully, maintaining robustness and providing user feedback.

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

PROGRAM CODE :

```java
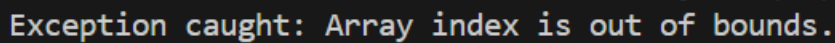public class prac25 {
    public static void main(String[] args) {
        try {
            int[] numbers = {1, 2, 3};
        System.out.println("Accessing element at index 5: " + numbers[5]);

        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception caught: Array index is out of bounds.");
}
}
}
```

OUTPUT:

```
Exception caught: Array index is out of bounds.
```

CONCLUSION:

This program tries to access an array index that doesn't exist, which causes an ArrayIndexOutOfBoundsException. By using a try-catch block, we catch this error and print a friendly message instead of crashing. This shows how exception handling helps keep Java programs running smoothly even when mistakes happen.

| 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 :

```java
class MyException extends Exception {
  public MyException() {
    System.out.println("Exception created by user");
  }
}

public class Pract26 {
  static void checkValue(int value) throws MyException {
    if (value > 10) {
      throw new MyException();
    }
    System.out.println("Value is acceptable: " + value);
  }
  public static void CheckedException() throws Exception {
    throw new Exception("This is a checked exception.");
  }
  public static void UncheckedException() {
    int result = 10 / 0;
  }

  public static void main(String[] args) {
    try {
      checkValue(3);
      checkValue(20);
    } catch (MyException e) {
      System.out.println("Caught user-defined exception.");
    }
    try {
       CheckedException();
    } catch (Exception e) {
      System.out.println("Caught checked exception: " + e.getMessage());
    }
    try {
      UncheckedException();
```

```
        } catch (ArithmeticException e) {
            System.out.println("Caught unchecked exception: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Value is acceptable: 3
Exception created by user
Caught user-defined exception.
Caught checked exception: This is a checked exception.
Caught unchecked exception: / by zero
```

CONCLUSION:

This program illustrates the difference between checked and unchecked exceptions in Java. Unchecked exceptions, like ArithmeticException and NullPointerException, occur at runtime and do not require explicit handling, while checked exceptions, such as IOException and ClassNotFoundException, must be declared or caught. By using the throw and throws keywords, we can create and manage user-defined exceptions effectively.

Part - 6

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:

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class prac27{
    public static void main(String[] args)
    {
        if (args.length == 0)
        {
            System.out.println("Please specify a filename.");
            return;
        }
        for (String fileName : args)
        {
            int linecount = 0;

            try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
            {
                while (reader.readLine() != null)
                {
                    linecount++;
                }
                System.out.println(fileName + ": " + linecount + " lines");
            } catch (IOException e)
            {
                System.out.println("Error reading line "+ fileName + " - " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

```
file.txt: 2 lines
```

CONCLUSION:

The provided Java program efficiently counts the number of lines in each text file specified on the command line. It handles multiple files, processes each file sequentially, and provides useful feedback in the event of an error. By using the try-with-resources feature, it ensures proper resource management and file handling. This program demonstrates how to handle command-line arguments, manage file I/O operations, and handle exceptions in Java, making it a robust solution for counting lines across multiple files.

| 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:

```java
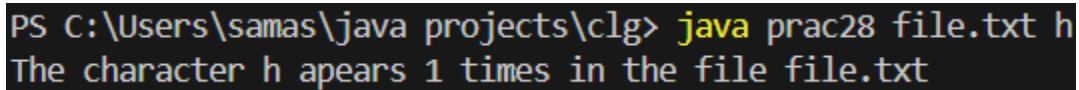import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class prac28{
    public static void main(String[] args) {
        if (args.length < 2)
        {
            System.out.println("Run : java prac28 <filename> <character>");
            return;
        }
        String filename = args[0];
        char Char = args[1].charAt(0);
        try(BufferedReader reader = new BufferedReader(new FileReader(filename)))
        {
            int count = 0;
            int curr;
```

```java
        while ((curr = reader.read()) != -1)
        {
          if(curr == Char)
          {
             count++;
          }
        }
        System.out.println("The character " + Char + " apears " + count + " times in the
file " + filename);
      }
    catch(IOException e)
    {
      System.out.println("Error reading file: " + filename + " - " + e.getMessage());
    }
   }
}
```

OUTPUT:

```
PS C:\Users\samas\java projects\clg> java prac28 file.txt h
The character h apears 1 times in the file file.txt
```

CONCLUSION:

The program successfully reads a text file and counts how many times a specified character appears in it. It accepts the filename and the character to search for as command-line arguments. By using BufferedReader for file reading and a loop to check each character in the file, it efficiently counts occurrences. The program also handles file-related errors, such as the file not being found or issues with reading, using a try-catch block, ensuring that the program exits gracefully while providing informative error messages when something goes wrong. This makes the program both robust and easy to use.

| 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:

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Scanner;
public class prac29
{
    public static void main(String[] args)
    {
        try (Scanner sc = new Scanner(System.in))
        {
            System.out.print("Enter the filename: ");
            String fileName = sc.nextLine();
            System.out.print("Enter the word to search for: ");
            String targetWord = sc.nextLine();
            Integer wordCount = 0;
            try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
            {
                String line;
                while ((line = reader.readLine()) != null)
                {
                    String[] words = line.split("\\s+");
                    for (String word : words) {
                        if (word.equals(targetWord))
                        {
                            wordCount++;
                        }
                    }
                }
                System.out.println("The word '" + targetWord + "' appears " + wordCount + " 
times in the file " + fileName);
            }
            catch (IOException e)
            {
                System.out.println("Error reading file: " + fileName + " - " + e.getMessage());
            }
        }
```

```
    }
}
```

OUTPUT:

```
Enter the filename: file.txt
Enter the word to search for: Hello
The word 'Hello' appears 1 times in the file file.txt
```

CONCLUSION:

The program efficiently reads a specified file and counts how many times a user-specified word appears in it. It utilizes a Scanner to get the filename and the target word from the user, and then a BufferedReader to read the file line by line. The words in each line are split and compared to the target word, incrementing a counter for each match. The program handles file-related errors gracefully with a try-catch block, ensuring that if the file cannot be read, an appropriate error message is displayed. This program demonstrates effective use of file handling, string manipulation, and user input 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:

```java
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
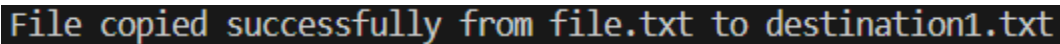
public class prac30 {

    public static void main(String[] args) {
        String sourceFile = "txt3.txt";
        String destinationFile = "destination1.txt";
        try (
            FileInputStream inputStream = new FileInputStream(sourceFile);
            FileOutputStream outputStream = new FileOutputStream(destinationFile)) {
```

```java
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = inputStream.read(buffer)) != -1) {
           outputStream.write(buffer, 0, bytesRead);
        }

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

     } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
     }
   }
}
```

OUTPUT:

```
File copied successfully from file.txt to destination1.txt
```

CONCLUSION:

This program demonstrates how to copy the contents of one file to another using Java's
FileInputStream and FileOutputStream. It reads data from the source file in chunks (using
a byte buffer) and writes it to the destination file. This efficient approach allows handling
large files without consuming excessive memory. The program handles potential
IOException errors, such as file not found or permission issues, and provides informative
feedback. Overall, it showcases how to perform basic file I/O operations in Java for
copying binary or text files between locations.

| 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:

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

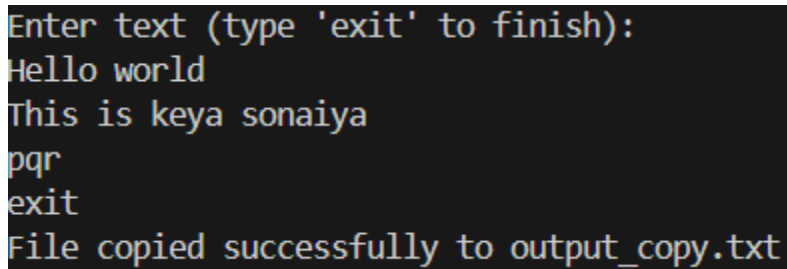public class prac31 {

    public static void main(String[] args) {
        String fileName = "output.txt";
        try (
                BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
                BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {

            System.out.println("Enter text (type 'exit' to finish):");
            String line;
            while (!(line = consoleReader.readLine()).equalsIgnoreCase("exit")) {
                fileWriter.write(line);
                fileWriter.newLine();
            }

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        try (
                FileInputStream fileInputStream = new FileInputStream(fileName);
                FileOutputStream fileOutputStream = new
FileOutputStream("output_copy.txt")) {

            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = fileInputStream.read(buffer)) != -1) {
```

```
            fileOutputStream.write(buffer, 0, bytesRead);
        }
        System.out.println("File copied successfully to output_copy.txt");
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
  }
}
```

OUTPUT:

```
Enter text (type 'exit' to finish):
Hello world
This is keya sonaiya
pqr
exit
File copied successfully to output_copy.txt
```

CONCLUSION:

This program efficiently handles two tasks: first, it captures user input from the console and writes it to a file (output.txt) until the user types "exit". Second, it reads this file and copies its contents to a new file (output_copy.txt). The program uses BufferedReader and BufferedWriter for efficient character stream handling, and FileInputStream and FileOutputStream to perform the file copy operation. It also includes error handling to manage potential I/O issues gracefully. This demonstrates basic file writing and copying operations in Java.

Part - 7

| 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 HelloWorldThread extends Thread {
    @Override
    public void run() {
        System.out.println("Hello World from Thread class!");
    }
}
class HelloWorldRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Hello World from Runnable interface!");
    }
}
public class prac32 {
    public static void main(String[] args) {
        HelloWorldThread thread1 = new HelloWorldThread();
        thread1.start();
        HelloWorldRunnable runnable = new HelloWorldRunnable();
        Thread thread2 = new Thread(runnable);
        thread2.start();
    }
}
```

OUTPUT:

```
Hello World from Thread class!
Hello World from Runnable interface!
```
|

CONCLUSION:

We demonstrated two approaches to creating threads in Java: one by extending the Thread class and another by implementing the Runnable interface. Both approaches are effective for multi-threading, but the Runnable interface is more flexible as it allows the class to extend other classes as well. The program illustrates how to use these methods, with each thread executing independently, displaying the "Hello World" message from different contexts. This provides a solid foundation for understanding thread creation and execution in Java.

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 int[] result;

    public SumTask(int start, int end, int[] result) {
        this.start = start;
        this.end = end;
        this.result = result;
    }

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

```java
            result[0] += sum;
        }
    }


public class prac33 {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java prac33 <N> <number_of_threads>");
            return;
        }

        int N = Integer.parseInt(args[0]);
        int numberOfThreads = Integer.parseInt(args[1]);

        int[] result = new int[1];
        Thread[] threads = new Thread[numberOfThreads];
        int numbersPerThread = N / numberOfThreads;

        for (int i = 0; i < numberOfThreads; i++) {
            int start = i * numbersPerThread + 1;
            int end = (i == numberOfThreads - 1) ? N : start + numbersPerThread - 1;
            threads[i] = new Thread(new SumTask(start, end, result));
            threads[i].start();
        }

        for (int i = 0; i < numberOfThreads; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Sum of first " + N + " numbers is: " + result[0]);
    }}
```

OUTPUT:

```
Sum of first 4 numbers is: 10
```

CONCLUSION:

This program effectively demonstrates how to distribute the task of summing the first NNN numbers among a specified number of threads in Java. By dividing the workload, each thread calculates a portion of the total sum, which promotes parallel processing and improves efficiency. The program uses the Runnable interface to define the summation task and manages thread execution through the Thread class. After all threads complete their execution, the program aggregates the results and displays the final sum on the console. This approach showcases fundamental concepts in multi-threading, such as task distribution, thread management, and synchronization, making it a practical example for understanding concurrent programming in Java.

---

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:

```java
import java.util.Random;

class RandomNumberGenerator extends Thread {
    private final NumberProcessor processor;

    public RandomNumberGenerator(NumberProcessor processor) {
        this.processor = processor;
    }

    @Override
    public void run() {
        Random random = new Random();
        while (true) {
            int number = random.nextInt(100);
```

```java
            System.out.println("Generated: " + number);
            processor.process(number);
            try {
               Thread.sleep(1000);
            } catch (InterruptedException e) {
               break;
            }
         }
      }
}

class NumberProcessor {
   private final Object lock = new Object();
   private int number;

   public void process(int number) {
      synchronized (lock) {
         this.number = number;
         lock.notifyAll();
      }
   }

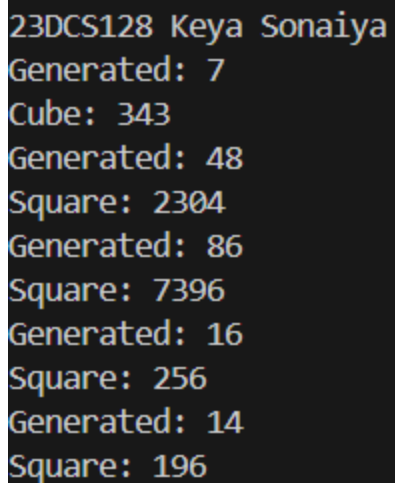   public void computeSquare() {
      while (true) {
         synchronized (lock) {
            try {
               lock.wait();
            } catch (InterruptedException e) {
               break;
            }
            if (number % 2 == 0) {
               System.out.println("Square: " + (number * number));
            }
         }
      }
   }

   public void computeCube() {
      while (true) {
         synchronized (lock) {
            try {
```

```java
                        lock.wait();
                    } catch (InterruptedException e) {
                        break;
                    }
                    if (number % 2 != 0) {
                        System.out.println("Cube: " + (number * number * number));
                    }
                }
            }
        }
    }

public class prac34 {
    public static void main(String[] args) {
        NumberProcessor processor = new NumberProcessor();
        RandomNumberGenerator generator = new RandomNumberGenerator(processor);
        Thread squareThread = new Thread(processor::computeSquare);
        Thread cubeThread = new Thread(processor::computeCube);

        generator.start();
        squareThread.start();
        cubeThread.start();
    }
}
```

OUTPUT:

```
23DCS128 Keya Sonaiya
Generated: 7
Cube: 343
Generated: 48
Square: 2304
Generated: 86
Square: 7396
Generated: 16
Square: 256
Generated: 14
Square: 196
```

CONCLUSION:

This program effectively showcases multi-threading in Java by creating a system that generates random integers and processes them based on their parity. By utilizing synchronization mechanisms, the program ensures that the generated numbers are processed safely and concurrently, demonstrating the power of threading for managing tasks that require coordination.
4o mini

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:

```java
public class prac35 {
    public static void main(String[] args) {
        Incrementer incrementer = new Incrementer();
        Thread incrementThread = new Thread(incrementer);
        incrementThread.start();
    }
}

class Incrementer implements Runnable {
    private int value = 0;
    @Override
    public void run() {
        while (true) {
            value++;
            System.out.println("Current value: " + value);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
                break; }}}}
```

OUTPUT:

```
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Current value: 6
Current value: 7
```

CONCLUSION:

This program demonstrates a simple use of threads in Java by incrementing a variable and displaying its value every second. The use of the sleep() method illustrates how to control the execution flow of a thread effectively, allowing for timed operations in multi-threaded applications.
4o mini

---

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:

```java
class FirstThread extends Thread {
    @Override
    public void run() {
        System.out.println("FIRST thread is running with priority: " + getPriority());
    }
}

class SecondThread extends Thread {
    @Override
    public void run() {
        System.out.println("SECOND thread is running with priority: " + getPriority());
    }
}
```

```java
class ThirdThread extends Thread {
   @Override
   public void run() {
      System.out.println("THIRD thread is running with priority: " + getPriority());
   }
}

public class prac36 {
   public static void main(String[] args) {
      FirstThread firstThread = new FirstThread();
      SecondThread secondThread = new SecondThread();
      ThirdThread thirdThread = new ThirdThread();

      firstThread.setPriority(3);
      secondThread.setPriority(Thread.NORM_PRIORITY);
      thirdThread.setPriority(7);

      firstThread.start();
      secondThread.start();
      thirdThread.start();
   }
}
```

OUTPUT:

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

CONCLUSION:

This program demonstrates how to create and manage multiple threads in Java with different priorities. By setting specific priorities for each thread, the program allows the Java runtime to handle the execution order based on these priorities, showcasing basic concepts of thread management in concurrent programming.

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

PROGRAM CODE:

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

class ProducerConsumer {
    private final Queue<Integer> buffer = new LinkedList<>();
    private final int capacity;

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

    public void produce(int item) throws InterruptedException {
        synchronized (this) {
            while (buffer.size() == capacity) {
                wait();
            }
            buffer.add(item);
            System.out.println("Produced: " + item);
            notifyAll();
        }
    }

    public int consume() throws InterruptedException {
        synchronized (this) {
            while (buffer.isEmpty()) {
                wait();
            }
            int item = buffer.poll();
            System.out.println("Consumed: " + item);
            notifyAll();
            return item;
        }
    }
}

class Producer extends Thread {
    private final ProducerConsumer pc;
```

```java
    public Producer(ProducerConsumer pc) {
      this.pc = pc;
    }

    @Override
    public void run() {
      for (int i = 0; i < 10; i++) {
        try {
          pc.produce(i);
          Thread.sleep(500);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
}

class Consumer extends Thread {
  private final ProducerConsumer pc;
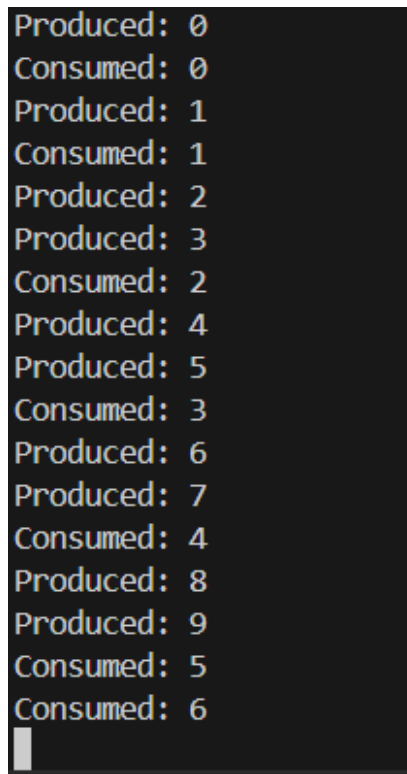
  public Consumer(ProducerConsumer pc) {
    this.pc = pc;
  }

  @Override
  public void run() {
    for (int i = 0; i < 10; i++) {
      try {
        pc.consume();
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}

public class prac37 {
  public static void main(String[] args) {
    ProducerConsumer pc = new ProducerConsumer(5);
```

```
        Producer producer = new Producer(pc);
        Consumer consumer = new Consumer(pc);

        producer.start();
        consumer.start();
    }
}
```

OUTPUT:

```
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Produced: 3
Consumed: 2
Produced: 4
Produced: 5
Consumed: 3
Produced: 6
Produced: 7
Consumed: 4
Produced: 8
Produced: 9
Consumed: 5
Consumed: 6
```

CONCLUSION:

This program effectively demonstrates the Producer-Consumer problem using thread synchronization in Java. It ensures safe communication between producer and consumer threads through proper handling of shared resources, utilizing the wait() and notifyAll() methods to synchronize access to the buffer. This implementation serves as a foundational example of handling concurrency issues in multi-threaded applications.

Part - 8

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

PROGRAM CODE:

```java
import java.util.ArrayList;

class CustomStack {
    private ArrayList<Object> stackList;

    public CustomStack() {
        stackList = new ArrayList<>();
    }

    public boolean isEmpty() {
        return stackList.isEmpty();
    }

    public int getSize() {
        return stackList.size();
    }

    public Object peek() {
        if (isEmpty()) {
            return null;
        }
        return stackList.get(stackList.size() - 1);
    }

    public Object pop() {
```

```java
        if (isEmpty()) {
            return null;
        }
        return stackList.remove(stackList.size() - 1);
    }

    public void push(Object o) {
        stackList.add(o);
    }

    public void display() {
        System.out.println("Stack: " + stackList);
    }
}

public class prac38 {
    public static void main(String[] args) {
        CustomStack stack = new CustomStack();

        stack.push(10);
        stack.push(20);
        stack.push(30);

        System.out.println("Stack size: " + stack.getSize());
        System.out.println("Top element (peek): " + stack.peek());

        stack.display();

        System.out.println("Popped element: " + stack.pop());
        System.out.println("Stack size after pop: " + stack.getSize());

        stack.display();

        System.out.println("Is stack empty? " + stack.isEmpty());
        stack.pop();
```

```
        stack.pop();
        System.out.println("Is stack empty after popping all? " + stack.isEmpty());
    }
}
```

OUTPUT:

```
Stack size: 3
Top element (peek): 30
Stack: [10, 20, 30]
Popped element: 30
Stack size after pop: 2
Stack: [10, 20]
Is stack empty? false
Is stack empty after popping all? true
```

CONCLUSION:

This program provides a functional implementation of a custom stack using the ArrayList class in Java. It encapsulates the core functionalities of a stack, such as push, pop, peek, and checking for emptiness, making it a suitable structure for managing a collection of elements with Last In, First Out (LIFO) behavior.

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 prac39 {

```java
    public static <T extends Comparable<T>> void sortArray(T[] array) {
        Arrays.sort(array);
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product("Laptop", 1500, 4.5),
            new Product("Smartphone", 800, 4.7),
            new Product("Tablet", 400, 4.2)
        };

        System.out.println("Products before sorting:");
        for (Product product : products) {
            System.out.println(product);
        }

        sortArray(products);

        System.out.println("\nProducts after sorting by natural order (price):");
        for (Product product : products) {
            System.out.println(product);
        }
    }
}

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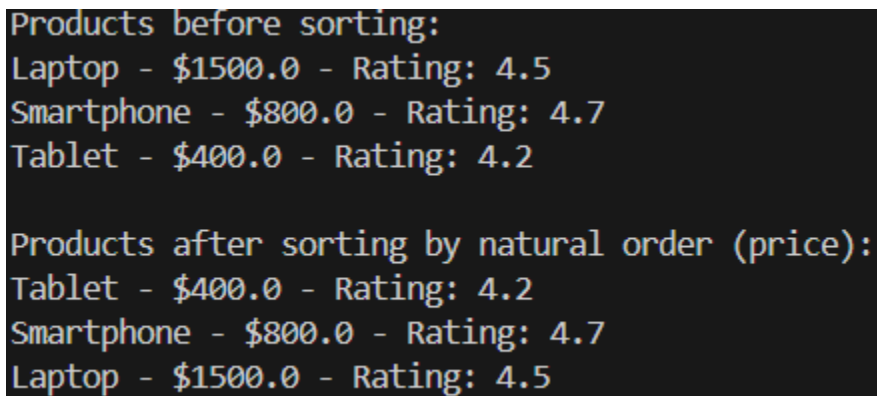    private double price;
    private double rating;

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

```
    }

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

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

OUTPUT:

```
Products before sorting:
Laptop - $1500.0 - Rating: 4.5
Smartphone - $800.0 - Rating: 4.7
Tablet - $400.0 - Rating: 4.2

Products after sorting by natural order (price):
Tablet - $400.0 - Rating: 4.2
Smartphone - $800.0 - Rating: 4.7
Laptop - $1500.0 - Rating: 4.5
```

CONCLUSION:

This program implements a generic method to sort arrays of objects that implement the
Comparable interface. It defines a Product class that represents a product with attributes
for name, price, and rating. The program demonstrates sorting an array of Product objects
based on price using the generic sorting method. This design allows for flexibility and
reusability, enabling the sorting of various object types in a type-safe manner, which is
essential in developing scalable applications like e-commerce platforms.
4o mini

| 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:

```java
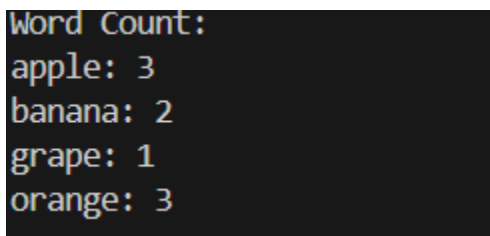import java.util.*;

public class prac40 {
    public static void main(String[] args) {
        String text = "apple banana apple orange banana apple grape orange orange";
        countWords(text);
    }
    public static void countWords(String text) {
        String[] words = text.split("\\s+");
        Map<String, Integer> wordCountMap = new HashMap<>();

        for (String word : words) {
            wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
        }
        List<String> sortedWords = new ArrayList<>(wordCountMap.keySet());
        Collections.sort(sortedWords);

        System.out.println("Word Count:");
        for (String word : sortedWords) {
            System.out.println(word + ": " + wordCountMap.get(word));
        }
}}}
```

OUTPUT:

```
Word Count:
apple: 3
banana: 2
grape: 1
orange: 3
```

CONCLUSION:

This program efficiently counts and displays the occurrences of words in a given text. By utilizing a Map to store word counts and a List to sort the keys, it ensures that the output is presented in alphabetical order. This method demonstrates how to handle word occurrences using collections, making it suitable for text processing tasks.

---

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.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;

public class prac41 {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Usage: java prac41 <source-file.java>");
            return;
        }

        String fileName = args[0];
        HashSet<String> keywords = initializeKeywords();
        int keywordCount = countKeywords(fileName, keywords);

        System.out.println("Number of keywords in " + fileName + ": " + keywordCount);
    }
```

```java
    private static HashSet<String> initializeKeywords() {
        HashSet<String> keywords = new HashSet<>();
        String[] javaKeywords = {
            "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", "null", "package", "private", "protected",
            "public", "return", "short", "static", "strictfp", "super",
            "switch", "synchronized", "this", "throw", "throws",
            "transient", "try", "void", "volatile", "while"
        };

        for (String keyword : javaKeywords) {
            keywords.add(keyword);
        }

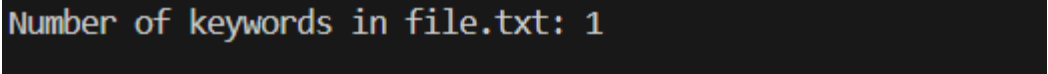        return keywords;
    }

    private static int countKeywords(String fileName, HashSet<String> keywords) {
        int count = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\W+"); // Split on non-word characters
                for (String word : words) {
                    if (keywords.contains(word)) {
                        count++;
                    }
                }
            }
        } catch (IOException e) {
```

```
            System.out.println("Error reading file: " + fileName + " - " + e.getMessage());
        }


        return count;
    }
}
```

OUTPUT:

```
Number of keywords in file.txt: 1
```

CONCLUSION:

This program effectively counts the number of Java keywords present in a specified source file. By leveraging a HashSet to store Java keywords, the program allows for efficient keyword lookups using the contains() method. The implementation reads the file line by line, splits the content into words, and checks each word against the keyword set, ultimately providing a count of how many keywords were found. This approach not only demonstrates the practical use of collections in Java but also serves as a useful tool for code analysis, aiding developers in understanding the structure and usage of keywords within Java source code.