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

CSE201- JAVA PROGRAMMING PRACTICAL LIST

	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

CSE201- JAVA PROGRAMMING PRACTICAL LIST

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

CSE201- JAVA PROGRAMMING PRACTICAL LIST

	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

CSE201- JAVA PROGRAMMING PRACTICAL LIST

	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

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

	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

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

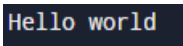Department of Computer Science & Engineering
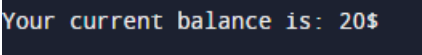
Subject Name: JAVA PROGRAMMING

Semester: 3

Subject Code: CSE201

Academic year: 2024-25

PART-1

NO.	AIM OF THE PRACTICAL

1.	Demonstration of installation steps of Java,Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.
	
	 PROGRAM:
	
	 import java.util.*;
	
	 public class hello{
	    public static void main(String[] args){
	      System.out.println("Hello world");
	    }
	 }
	
	 OUTPUT:
	
	 `Hello world`
	
	 CONCLUSION:
	
	 Introduction to Object-Oriented Concepts
	 Classes and Objects: A class is a blueprint for objects. An object is an instance of a class.
	 Encapsulation: Bundling data (variables) and methods that operate on the data into a single unit or class.
	 Inheritance: Mechanism where one class acquires the properties (fields and methods) of another.

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

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

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

NO.	AIM OF THE PRACTICAL

2.	Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20. Write a java program to store this balance in a variable and then display it to the user.
	
	PROGRAM:
	
	``java
	import java.util.*;
	
	public class Input {
	    public static void main(String[] args) {
	        int a=20;
	        System.out.println("Your current balance is: "+ a +"$");
	    }
	}
	``
	
	OUTPUT:
	
	`Your current balance is: 20$`

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

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

PROGRAM:

```java
import java.util.Scanner;

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

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

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

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

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

OUTPUT:

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

CONCLUSION:

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

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

PROGRAM:

```java
import java.util.Scanner;

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

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

        double[] dailyExpenses = new double[days];

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

```
        double totalExpenses = 0.0;
        for (double expense : dailyExpenses) {
            totalExpenses = totalExpenses + expense;
        }

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

OUTPUT:

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


Total expense for the month is: 1850.0
```

CONCLUSION:

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

NO.	AIM OF THE PRACTICAL

5.	An electric appliance shop assigns code 1 to motor,2 to fan,3 to tube and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor,12% to fan,5% to tube light,7.5% to wires and 3% for all other items is charged. A list containing the product code and price in two different arrays. Write a java program using switch statement to prepare the bill.
	
	PROGRAM:
	
	import java.util.Scanner;
	
	public class Shop {
	    public static void main(String[] args) {

```java
        Scanner scanner = new Scanner(System.in);

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

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

        double taxRate = 0.0;
        String productName;

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

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

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

OUTPUT:

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

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

CONCLUSION:

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

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

PROGRAM:

```java
import java.util.Scanner;

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

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

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

```
        int nextTerm = firstTerm + secondTerm;
        firstTerm = secondTerm;
        secondTerm = nextTerm;
      }
    }
}
```

OUTPUT:

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

CONCLUSION:

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

PART-2

NO.	AIM OF THE PRACTICAL

7.	Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;
	front_times('Chocolate', 2) → 'ChoCho'
	front_times('Chocolate', 3) → 'ChoChoCho'
	front_times('Abc', 3) → 'AbcAbcAbc'
	
	PROGRAM:
	
	import java.util.*;
	
	public class Main {
	
	   public static void main(String[] args) {
	      String st1 = "Chocolate";
	      String st2 = "Abc";
	      int n, m;
	      Scanner obj = new Scanner(System.in);
	      System.out.print("Which string's characters do you want to print");
	      System.out.print("\n1.Chocolate\n2.Abc");
	      System.out.print("\nEnter: ");
	      int choice = obj.nextInt();
	      if(choice==1){
	         System.out.print("Enter the number of times you want to print the first 3 characters: ");
	         n = obj.nextInt();
	         System.out.println("--------------------");
	         for (int i = 0; i < n; i++) {
	         System.out.print(st1.substring(0, 3));
	        }
	         System.out.println("\n--------------------");
	      }
	      else{
	      System.out.print("Enter the number of times you want to print the first 3 characters:");
	      m = obj.nextInt();
	      System.out.println("--------------------");
	      for (int i = 0; i < m; i++) {
	         System.out.print(st2.substring(0, 3));
	      }
	      System.out.println("\n--------------------");
	      }
	   }
	}

OUTPUT:

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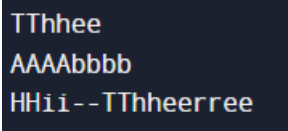
CONCLUSION:

The Java program allows users to choose between two strings, "Chocolate" and "Abc," and print the first three characters of the chosen string a specified number of times. The user is prompted to enter a choice (1 or 2) and then asked for the number of repetitions. Based on the input, the program uses a loop to print the first three characters of the selected string the desired number of times. This demonstrates basic user input handling, string manipulation, and loop control in Java.
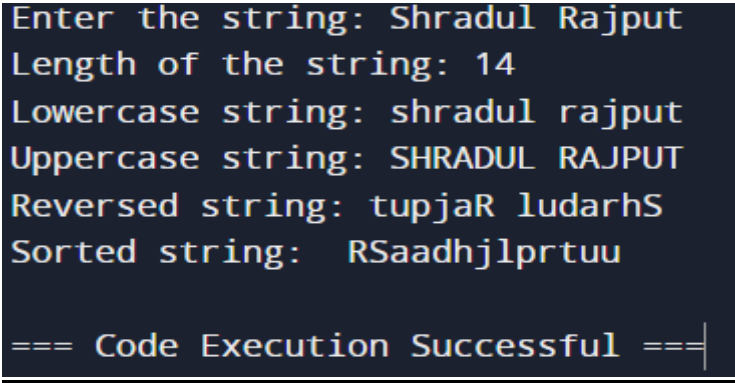
NO.	AIM OF THE PRACTICAL

8.	Given an array of ints, return the number of 9's in the array. array_count9([1, 2, 9]) → 1
	 array_count9([1, 9, 9]) → 2
	 array_count9([1, 9, 9, 3, 9]) → 3
	
	 PROGRAM:
	
	 public class Main {
	  public static void main(String[] args) {
	     int arr1[]={1,2,9};
	     int arr2[]={1,9,9};
	     int arr3[]={1,9,9,3,9};
	     array_count9(arr1);
	     array_count9(arr2);
	     array_count9(arr3);
	   }
	   static void array_count9(int arr[]){
	    int count=0;
	    for(int i=0;i<arr.length;i++){
	     if(arr[i]==9){
	      count++;
	     }
	    }

```
        System.out.println("Count of 9's in array :"+count);
    }
    }
```

OUTPUT:

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

CONCLUSION:

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

NO.	AIM OF THE PRACTICAL

9.	Given a string, return a string where for every char in the original, there are two chars.
	double_char('The') → 'TThhee'
	double_char('AAbb') → 'AAAAbbbb'
	double_char('Hi-There') → 'HHii--TThheerree'
	
	PROGRAM:
	
	public class Main{
	    public String doubleChar(String str) {
	        StringBuilder result = new StringBuilder();
	        for (int i = 0; i < str.length(); i++) {
	            result.append(str.charAt(i));
	            result.append(str.charAt(i));
	        }
	        return result.toString();
	    }
	    public static void main(String[] args) {
	        Main obj = new Main();
	        System.out.println(obj.doubleChar("The"));
	        System.out.println(obj.doubleChar("AAbb"));
	        System.out.println(obj.doubleChar("Hi-There"));
	    }
	}

OUTPUT:

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

CONCLUSION:

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

NO.	AIM OF THE PRACTICAL
10.	Perform following functionalities of the string:

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

PROGRAM:

```java
import java.util.*;

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

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

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

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

        char[] charArray = str.toCharArray();
        Arrays.sort(charArray);
```

```
        String sortedStr = new String(charArray);
        System.out.println("Sorted string: " + sortedStr);
    }
}
```

OUTPUT:

```
Enter the string: Shradul Rajput
Length of the string: 14
Lowercase string: shradul rajput
Uppercase string: SHRADUL RAJPUT
Reversed string: tupjaR ludarhS
Sorted string:  RSaadhjlprtuu


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

CONCLUSION:

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

NO.	AIM OF THE PRACTICAL

11.	Perform following Functionalities of the string:
	 "CHARUSAT UNIVERSITY"
	 ● Find length
	 ● Replace 'H' by 'FIRST LETTER OF YOUR NAME'
	 ● Convert all character in lowercase
	
	 PROGRAM:
	
	 public class Main {
	     public static void main(String[] args) {
	         String input = "CHARUSAT UNIVERSITY";
	
	         int length = input.length();
	         System.out.println("Length of the string: " + length);
	
	         String firstName = "Shradul";
	         char firstLetter = firstName.charAt(0);

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

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

OUTPUT:

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

CONCLUSION:

The Java program processes the string "CHARUSAT UNIVERSITY" by performing a series of operations. First, it calculates and prints the length of the string. Then, it replaces the character 'H' in the string with the first letter of the name "Shradul," resulting in "CKARUSAT UNIVERSITY." Finally, the program converts the modified string to lowercase and prints it as "ckarusat university." This demonstrates string manipulation techniques such as character replacement and case conversion in Java.
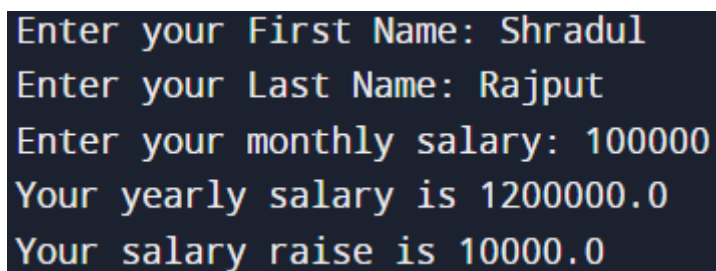
PART-3

NO.	AIM OF THE PRACTICAL

12.	Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.
	
	PROGRAM:
	
	``java
	import java.util.Scanner;
	
	public class CurrencyConverter {
	
	    private static final double CONVERSION_RATE = 100.0;
	
	    public static void main(String[] args) {
	        if (args.length > 0) {
	
	            try {
	                double pounds = Double.parseDouble(args[0]);
	                double rupees = convertPoundsToRupees(pounds);
	                System.out.printf("%.2f Pounds is %.2f Rupees%n", pounds, rupees);
	            } catch (NumberFormatException e) {
	                System.out.println("Invalid input. Please enter a valid number.");
	            }
	        } else {
	
	            Scanner scanner = new Scanner(System.in);
	            System.out.print("Enter amount in Pounds: ");
	            try {
	                double pounds = scanner.nextDouble();
	                double rupees = convertPoundsToRupees(pounds);
	                System.out.printf("%.2f Pounds is %.2f Rupees%n", pounds, rupees);
	            } catch (Exception e) {
	                System.out.println("Invalid input. Please enter a valid number.");
	            } finally {
	                scanner.close();
	            }
	        }
	    }
	``

```
        private static double convertPoundsToRupees(double pounds) {
            return pounds * CONVERSION_RATE;
        }
    }
```

OUTPUT:

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

CONCLUSION:

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

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

PROGRAM:

```java
import java.util.Scanner;

public class Main {

    String fname;
    String lname;
    double salary;

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

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

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

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

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

    }
}
```
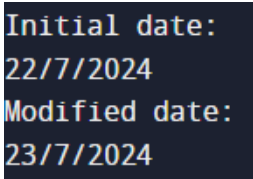
OUTPUT:

```
Enter your First Name: Shradul
Enter your Last Name: Rajput
Enter your monthly salary: 100000
Your yearly salary is 1200000.0
Your salary raise is 10000.0
```

CONCLUSION:

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

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

PROGRAM:

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

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

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

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

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

    public int getMonth() {
        return month;
    }

    public int getDay() {
        return day;
    }

    public int getYear() {
        return year;
    }

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

```
}

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

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

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

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

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

OUTPUT:

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

CONCLUSION:

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

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

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

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

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

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

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

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

        Area rectangle = new Area(length, breadth);

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

OUTPUT:

```
Enter the length of the rectangle: 4
Enter the breadth of the rectangle: 10
The area of the rectangle is: 40.0
```

CONCLUSION:

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

NO.	AIM OF THE PRACTICAL

16.	Print the sum, difference and product of two complex numbers by creating a class named 'Complex' with separate methods for each operation whose real and imaginary parts are entered by user.
	
	 PROGRAM:
	 `import java.util.Scanner;`
	
	 `class Complex`
	 `{`
	 `    double sum1, sum2, difference1, difference2, product1, product2;`
	 `    double a, b, c, d;`
	
	 `    void getData()`
	 `{`
	
	 `        Scanner scanner = new Scanner(System.in);`
	 `        System.out.print("Enter the real Part of complex number 1 : ");`
	 `        a = scanner.nextDouble();`
	 `        System.out.print("Enter the imaginary Part of complex number 1 : ");`
	 `        b = scanner.nextDouble();`
	 `        System.out.print("Enter the real Part of complex number 2 : ");`
	 `        c = scanner.nextDouble();`
	 `        System.out.print("Enter the imaginary Part of complex number 2 : ");`
	 `        d = scanner.nextDouble();`
	 `}`
	
	 `    void sum()`
	 `{`
	
	 `        sum1 = a + c;`
	 `        sum2 = b + d;`
	 `}`
	 ` void difference()`
	 `{`
	
	 `        difference1 = a - c;`
	 `        difference2 = b - d;`
	 `}`
	
	 `    void product()`
	 `{`
	
	 `        product1 = (a * c - b * d);`
	 `        product2 = (a * d + b * c);`
	 `}`
	
	 `    void display()`
	 `{`
	
	 `        System.out.println("SUM: " + sum1 + " + " + sum2 + "i");`
	 `        System.out.println("DIFFERENCE: " + difference1 + " + " + difference2`
	 `+ "i");`

```
                System.out.println("PRODUCT: " + product1 + " + " + product2 + "i");
    }

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

OUTPUT:

```
Enter the real Part of complex number 1 : 4
Enter the imaginary Part of complex number 1 : 10
Enter the real Part of complex number 2 : 3
Enter the imaginary Part of complex number 2 : 4
SUM: 7.0 + 14.0i
DIFFERENCE: 1.0 + 6.0i
PRODUCT: -28.0 + 46.0i
```

CONCLUSION:

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

PART-4

NO.	AIM OF THE PRACTICAL

17.	Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent
	
	 PROGRAM:
	 ``java
	class Parent
	{
	    void printParent()
	    {
	        System.out.println("This is parent class");
	    }
	}
	class Child extends Parent
	{
	    void printChild()
	    {
	        System.out.println("This is child class");
	    }
	}
	
	public class Main_17
	{
	    public static void main(String[] args)
	    {
	
	        Parent parent = new Parent();
	        parent.printParent();
	
	        Child child = new Child();
	        child.printChild();
	    }
	}
	``
	 OUTPUT:
	 `This is parent class`
	 `This is child class`
	
	 CONCLUSION:

	In conclusion, this program demonstrates the concept of inheritance in Java, where a child class can inherit the methods of a parent class and also have its own methods. We can create objects of both the parent and child classes and call their respective methods. The child class can also call the methods of the parent class using its own object.

NO.	AIM OF THE PRACTICAL

18.	Create a class named 'Member' having the following members: Data members
	1 - Name
	2 - Age
	3 - Phone number
	4 - Address
	5 – Salary
	It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.
	
	PROGRAM:
	
	``java
	class Member
	{
	    String name;
	    int age;
	    String phoneNumber;
	    String address;
	    double salary;
	
	    void printSalary()
	    {
	        System.out.println("Salary: " + salary);
	    }
	}
	
	class Employee extends Member {
	    String specialization;
	
	    void displayEmployeeDetails() {
	        System.out.println("Employee Details:");
	        System.out.println("Name: " + name);
	``

```
            System.out.println("Age: " + age);
            System.out.println("Phone Number: " + phoneNumber);
            System.out.println("Address: " + address);
            System.out.println("Specialization: " + specialization);
            printSalary();
        }
    }


    class Manager extends Member {
        String department;

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

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

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

            System.out.println();


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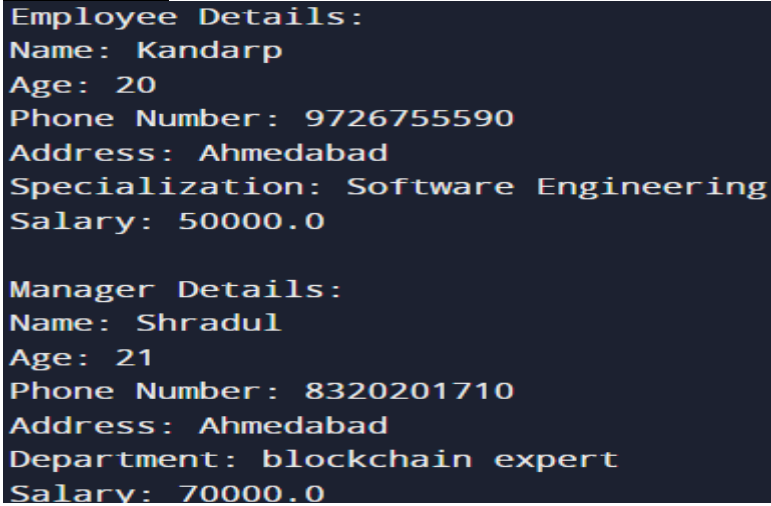
| | manager.salary = 70000.0;
manager.department = "blockchain expert";
manager.displayManagerDetails();
        }
}

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

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

CONCLUSION:
This design allows efficient management of common attributes while providing flexibility to add specialized characteristics to different types of members. |

NO.	AIM OF THE PRACTICAL

19.	Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.
	
	PROGRAM:
	``
	class Rectangle {
	    double length;
	    double breadth;
	
	    Rectangle(double length, double breadth) {
	        this.length = length;
	        this.breadth = breadth;
	``

```
      }

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

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


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

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

      Rectangle[] rectangles = new Rectangle[2];


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

      System.out.println();


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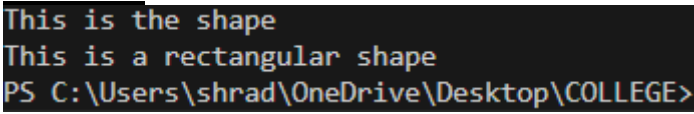
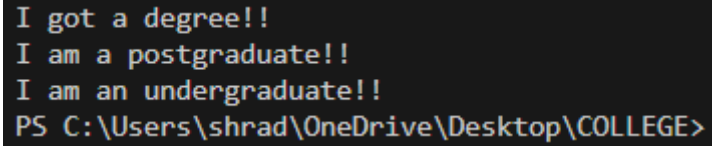**OUTPUT:**

```
Rectangle:
Area : 15.0
Perimeter : 16.0

Square:
Area : 16.0
Perimeter : 16.0
```

	CONCLUSION:
	This design highlights the benefits of inheritance, code reuse, and polymorphism in object-oriented programming.

NO.	AIM OF THE PRACTICAL
20.	Create a class named 'Shape' with a method to print "This is This is a shape". Then create two other classes named 'Rectangle', and '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 the 'Shape' and 'Rectangle' class by the object of 'Square' class.

PROGRAM:

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

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

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

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

public class prac20
{
```

	public static void main(String[] args)
	{
	Square s = new Square();
	s.display();
	s.displayRectangle();
	}
	}
	
	OUTPUT:
	
	``
	This is the shape
	This is a rectangular shape
	PS C:\Users\shrad\OneDrive\Desktop\COLLEGE>
	``
	
	CONCLUSION:
	
	The example provided illustrates core object-oriented programming principles, including inheritance, polymorphism, method overriding, and encapsulation. By structuring classes in a hierarchical manner that reflects logical relationships between shapes, the design promotes code reusability, flexibility, and maintainability. The use of polymorphism and method resolution order ensures that the correct behavior is invoked for each specific object type, showcasing the power and versatility of object-oriented design.

NO.	AIM OF THE PRACTICAL

21.	Create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.
	
	PROGRAM:
	class Degree
	{
	void getDegree()
	{
	System.out.println("I got a degree!!");
	}
	}
	
	class Postgraduate extends Degree
	{
	void getDegree()
	{
	System.out.println("I am a postgraduate!!");

```
}
}

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

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

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

OUTPUT:

```
I got a degree!!
I am a postgraduate!!
I am an undergraduate!!
PS C:\Users\shrad\OneDrive\Desktop\COLLEGE>
```

CONCLUSION:
In conclusion, this program demonstrates the concept of inheritance and method overriding in Java, where a subclass can override the methods of a parent class with its own implementation. We can create objects of each class and call their respective methods, and the correct method will be called based on the object's class. This allows for more flexibility and customization in our code.
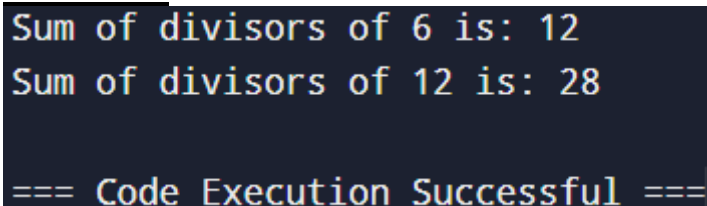
NO.	AIM OF THE PRACTICAL
22.	Write a java that implements an interface Advanced Arithmetic which contains a method signature int divisor sum(int n). You need to write a class Called MyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor_sum should return 12. The value of n will be at most 1000.

PROGRAM:

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

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

public class prac22

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

OUTPUT:

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


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

CONCLUSION:

The **MyCalculator** class implements the **Advanced Arithmetic** interface and provides a correct implementation of the **divisor sum** method, which calculates the sum of all divisors of a given integer **n**. The method is efficient and works correctly for values of **n** up to 1000.

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

PROGRAM:

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

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

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

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

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

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

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

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

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

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

class Sign {
    private Shape shape;
    private String text;

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

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

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

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

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

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

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

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

OUTPUT:

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

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

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

CONCLUSION:

The default method in the **Shape** interface allows us to provide a common implementation for all shapes, without having to repeat code in each shape class. This makes the code more concise and easier to maintain.

The **Sign** class can then use the shape's **display()** method to print the shape's information, without having to know the specific details of each shape class. This demonstrates the power of interfaces and default methods in Java.

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:
	
	``java
	import java.util.Scanner;
	
	public class DivisionWithExceptionHandling {
	    public static void main(String[] args) {
	        Scanner scanner = new Scanner(System.in);
	
	        try {
	            // Prompt the user for input
	            System.out.print("Enter the first integer (x): ");
	            int x = Integer.parseInt(scanner.nextLine());
	
	            System.out.print("Enter the second integer (y): ");
	            int y = Integer.parseInt(scanner.nextLine());
	
	            // Perform the division
	            int result = x / y;
	
	            // Display the result
	            System.out.println("Result of x/y: " + result);
	
	        } catch (NumberFormatException e) {
	            System.out.println("Error: Please enter valid integers.");
	        } catch (ArithmeticException e) {
	            System.out.println("Error: Division by zero is not allowed.");
	        } finally {
	            scanner.close();
	        }
	    }
	}
	``
	
	 OUTPUT:
	
	``
	Enter the first integer (x): 10
	Enter the second integer (y): 10
	Result of x/y: 1
	
	=== Code Execution Successful ===
	``

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

NO.	AIM OF THE PRACTICAL

25.	Write a Java program that throws an exception and catch it using a try-catch block.
	
	PROGRAM:
	<pre>public class prac25 {
	
	  public static void main(String[] args) {
	
	    try {
	      // Code that may throw an exception
	      int result = divideNumbers(10, 0); // This will throw ArithmeticException
	      System.out.println("Result: " + result);
	    } catch (ArithmeticException e) {
	      // Catch the exception and handle it
	      System.out.println("Caught Exception: Division by zero is not allowed.");
	    }
	
	    System.out.println("Program continues...");
	  }
	
	  // Method that throws ArithmeticException
	  public static int divideNumbers(int a, int b) {
	    return a / b; // Will throw ArithmeticException if b is 0
	  }
	}</pre>
	OUTPUT:
	
	``
	Caught Exception: Division by zero is not allowed.
	Program continues...
	
	=== Code Execution Successful ===
	``

<table>
<tr><td colspan="2">
CONCLUSION:
This Java program demonstrates the use of a try-catch block to handle exceptions.
The try block contains code that may throw an exception, and the catch block
catches and handles the exception. In this example, we throw
an ArithmeticException by dividing an integer by zero, and catch it using
a catch block that prints an error message and a custom message. This program
shows how to use try-catch blocks to handle exceptions and provide a more robust
and error-free program.
</td></tr>
</table>

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

PROGRAM:

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

public class prac26 {

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

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

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

```
        }
     }
   }
}
```
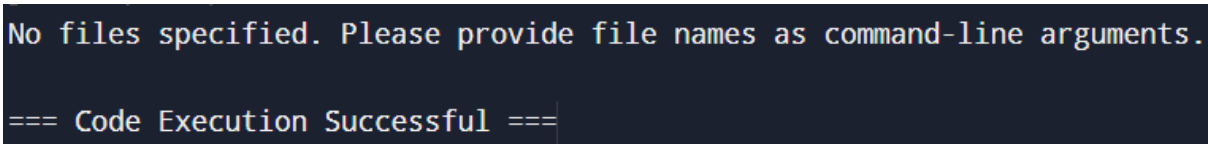
OUTPUT:

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


=== Code Execution Successful ===
```
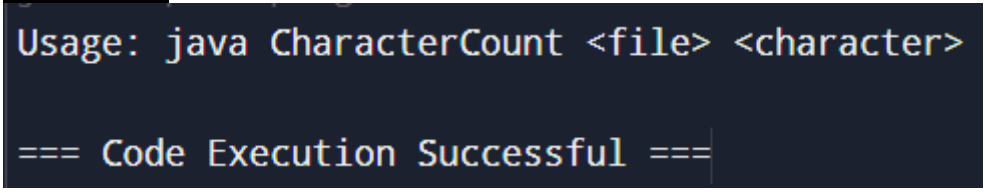
CONCLUSION:

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

PART-6

NO.	AIM OF THE PRACTICAL
| 27. | Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.<br><br>**PROGRAM:**<br><br>`import java.io.*;`<br><br>`public class LineCounts {`<br><br>`    public static void main(String[] args) {`<br>`        if (args.length == 0) {`<br>`            System.out.println("No files specified. Please provide file names as command-line arguments.");`<br>`            return;`<br>`        }`<br><br>`        for (String fileName : args) {`<br>`            File file = new File(fileName);`<br><br>`            try (BufferedReader br = new BufferedReader(new FileReader(file))) {`<br>`                int lineCount = 0;`<br>`                while (br.readLine() != null) {`<br>`                    lineCount++;`<br>`                }`<br>`                System.out.println(fileName + ": " + lineCount + " lines");`<br>`            } catch (FileNotFoundException e) {`<br>`                System.out.println("Error: File not found - " + fileName);`<br>`            } catch (IOException e) {`<br>`                System.out.println("Error: Unable to read file - " + fileName);`<br>`            }`<br>`        }`<br>`    }`<br>`}`<br><br>**OUTPUT:**<br><br>```
No files specified. Please provide file names as command-line arguments.

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

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

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

```java
import java.io.*;

public class CharacterCount {

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

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

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

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

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

}

OUTPUT:

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

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

CONCLUSION:

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

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

PROGRAM:

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

public class WordSearch {

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

        String fileName = args[0];
        String wordToFind = args[1];
        int wordCount = 0;

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

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

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

OUTPUT:

CONCLUSION:

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

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

```java
import java.io.*;

public class FileCopy {

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

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

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

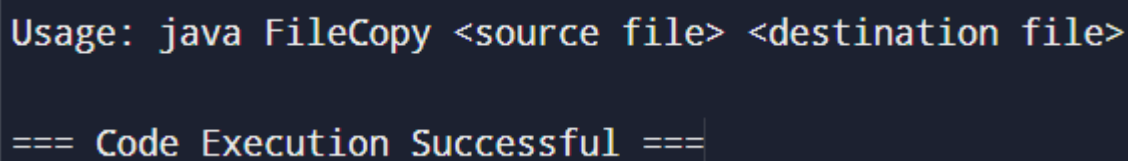            byte[] buffer = new byte[1024]; // Buffer for copying data
            int bytesRead;

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

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

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

OUTPUT:

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


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

<table>
<tr><td colspan="2">

CONCLUSION:

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

</td></tr>
</table>

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

PROGRAM:

```java
import java.io.*;

public class StreamExample {

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

            String inputLine;
            while (!(inputLine = br.readLine()).equalsIgnoreCase("exit")) {
                bw.write(inputLine);
                bw.newLine();
            }
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }

        try (FileInputStream fis = new FileInputStream("output.txt");
            FileOutputStream fos = new FileOutputStream("output_copy.txt")) {

            int byteData;
            while ((byteData = fis.read()) != -1) {
                fos.write(byteData);
            }
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
An error occurred: output.txt (Permission denied)
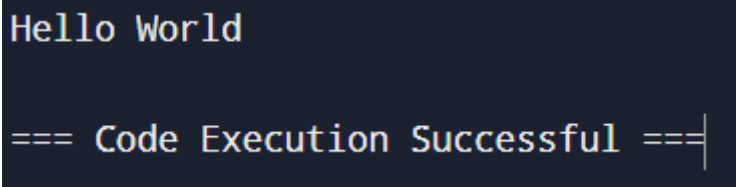An error occurred: output.txt (No such file or directory)


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

CONCLUSION:

In conclusion, this program demonstrates the use of character and byte streams in Java, as well as the use of **BufferedReader** and **BufferedWriter** to read console input and write it to a file. Character streams are used to read and write text data, while byte streams are used to read and write binary data. The **BufferedReader** and **BufferedWriter** classes are used to buffer the input/output for efficient reading and writing. This program shows how to use these classes to read input from the console and write it to a file, and how to use byte streams to write the same input to another file.

PART-7

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

PROGRAM:

```
class A extends Thread
{
   public void run()
   {
      System.out.println("Hello World");
   }

   public static void main(String[] args)
   {
      A threadA = new A();
      threadA.start();
   }
}

class B implements Runnable
{
   Thread t;

   B() {
      t = new Thread(this);
      t.start();
   }

   public void run()
   {
      System.out.println("Hello World");
   }

   public static void main(String[] args) {
      new B();
   }
}
```

OUTPUT:

```
Hello World

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

CONCLUSION:
In general, it's recommended to use the **Runnable** interface approach, as it provides more flexibility and is more scalable. Additionally, it's a better practice to separate the thread's logic from the thread's creation and management.

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

PROGRAM:
```java
import java.util.*;
class Test implements Runnable
{

   Thread t;
   int threadNumber;

   Test(int number)
   {
      threadNumber = number;
      System.out.println("Thread created for: " + threadNumber);
      t = new Thread(this, String.valueOf(threadNumber));
      t.start();
   }

   public void run()
   {
      int start = (threadNumber * 1000) + 1;
      int end = (threadNumber + 1) * 1000;

      int sum = 0;
      for (int i = start; i <= end; i++)
      {
         sum += i;
      }
```

```
        System.out.println("Thread " + threadNumber + " calculated sum from " + start +
" to " + end + ": " + sum);
    }

    public static void main(String[] args)
    {
        for (int i = 0; i < 10; i++)
        {
            new Test(i);
        }
    }
}
```

OUTPUT:

```
Thread created for: 0
Thread created for: 1
Thread created for: 2
Thread created for: 3
Thread created for: 4
Thread created for: 5
Thread created for: 6
Thread created for: 7
Thread created for: 8
Thread created for: 9
Thread 3 calculated sum from 3001 to 4000: 3500500
Thread 2 calculated sum from 2001 to 3000: 2500500
Thread 1 calculated sum from 1001 to 2000: 1500500
Thread 4 calculated sum from 4001 to 5000: 4500500
Thread 5 calculated sum from 5001 to 6000: 5500500
Thread 8 calculated sum from 8001 to 9000: 8500500
Thread 0 calculated sum from 1 to 1000: 500500
Thread 7 calculated sum from 7001 to 8000: 7500500
Thread 6 calculated sum from 6001 to 7000: 6500500
Thread 9 calculated sum from 9001 to 10000: 9500500
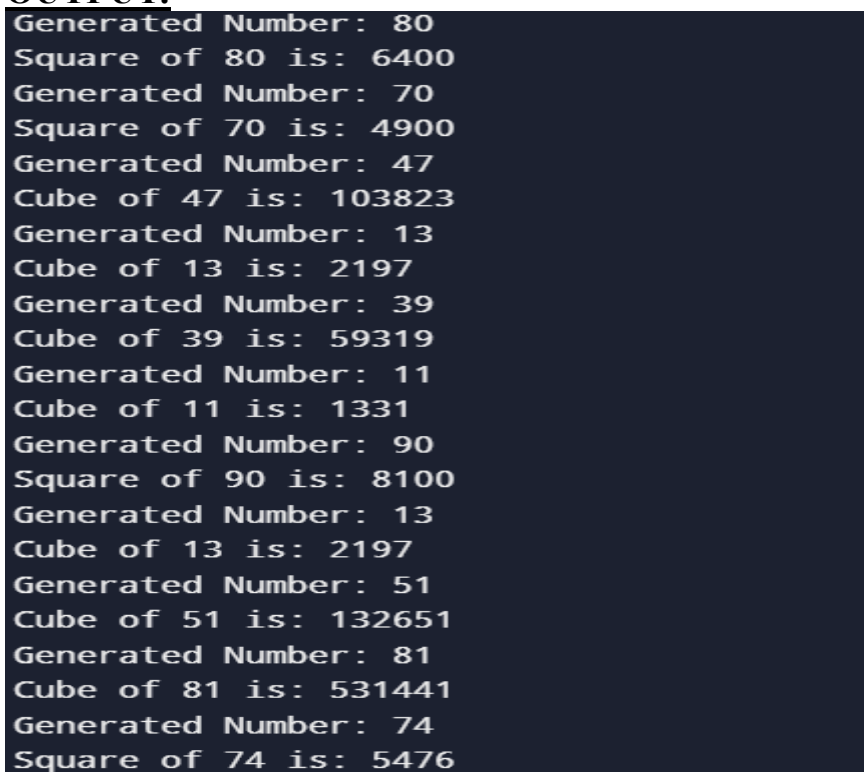
=== Code Execution Successful ===
```

CONCLUSION:
- **Concurrency**: This program demonstrates how to use multiple threads to perform a task concurrently, which can lead to improved performance, especially for large values of (N).
- **Scalability**: By distributing the workload among threads, the program can efficiently handle larger values of (N) without blocking the main thread.

- **Thread Management**: Proper management of threads (starting and joining) is crucial to ensure that all calculations are completed before displaying the final result.

This design allows for easy modifications, such as changing the number of threads or the range of numbers to be summed, making it a flexible solution for similar problems.

NO.	AIM OF THE PRACTICAL

34.	Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.
	
	PROGRAM:
	
	``java
	import java.util.*;
	
	class RandomNumberGenerator extends Thread {
	    public void run() {
	        Random random = new Random();
	        while (true) {
	            int number = random.nextInt(100); // Generates a random number
	            System.out.println("Generated Number: " + number);
	            if (number % 2 == 0) {
	                new SquareThread(number).start();
	            } else {
	                new CubeThread(number).start();
	            }
	            try {
	                Thread.sleep(1000); // Wait for 1 second
	            } catch (InterruptedException e) {
	                e.printStackTrace();
	            }
	        }
	    }
	}
	
	class SquareThread extends Thread {
	    int number;
	    SquareThread(int number) {
	        this.number = number;
	    }
	
	    public void run() {
	        System.out.println("Square of " + number + " is: " + (number * number));
	    }
	``

```
}

class CubeThread extends Thread {
    int number;
    CubeThread(int number) {
        this.number = number;
    }

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

public class MultiThreadExample {
    public static void main(String[] args) {
        new RandomNumberGenerator().start();
    }
}
```

OUTPUT:

```
Generated Number: 80
Square of 80 is: 6400
Generated Number: 70
Square of 70 is: 4900
Generated Number: 47
Cube of 47 is: 103823
Generated Number: 13
Cube of 13 is: 2197
Generated Number: 39
Cube of 39 is: 59319
Generated Number: 11
Cube of 11 is: 1331
Generated Number: 90
Square of 90 is: 8100
Generated Number: 13
Cube of 13 is: 2197
Generated Number: 51
Cube of 51 is: 132651
Generated Number: 81
Cube of 81 is: 531441
Generated Number: 74
Square of 74 is: 5476
```

CONCLUSION:
- **Concurrency**: This program demonstrates how to use multiple threads to perform different tasks concurrently, which can improve the overall performance and responsiveness of the application.

- **Thread Communication**: The program shows how threads can communicate with each other using method calls, allowing them to coordinate their actions and exchange data.
- **Thread Synchronization**: Although not explicitly shown in this example, thread synchronization mechanisms like locks, semaphores, or atomic variables would be necessary to ensure thread safety if multiple threads were accessing shared resources.

This program illustrates a simple yet effective way to divide tasks among multiple threads, making it a good starting point for more complex multi-threaded applications.

NO.	AIM OF THE PRACTICAL

35.	Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.
	PROGRAM:
	
	import java.util.*;
	
	class NumberGenerator extends Thread {
	  public void run() {
	    Random random = new Random();
	    while (true) {
	      int number = random.nextInt(100); // Generates a random number
	      System.out.println("Generated Number: " + number);
	      if (number % 2 == 0) {
	        new Square(number).start();
	      } else {
	        new Cube(number).start();
	      }
	      try {
	        Thread.sleep(1000); // Wait for 1 second before generating the next number
	      } catch (InterruptedException e) {
	        e.printStackTrace();
	      }
	    }
	  }
	}
	
	class Square extends Thread {
	  int number;
	  Square(int number) {

```java
            this.number = number;
        }

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

class Cube extends Thread {
    int number;
    Cube(int number) {
        this.number = number;
    }

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

public class MultiThreadApplication {
    public static void main(String[] args) {
        new NumberGenerator().start();
    }
}
```

OUTPUT:

```
Generated Number: 98
Square of 98 is: 9604
Generated Number: 9
Cube of 9 is: 729
Generated Number: 23
Cube of 23 is: 12167
Generated Number: 63
Cube of 63 is: 250047
Generated Number: 27
Cube of 27 is: 19683
Generated Number: 98
Square of 98 is: 9604
Generated Number: 98
Square of 98 is: 9604
```

CONCLUSION:

- **Thread-based Incrementation**: This program demonstrates how to use a thread to increment a variable and display its value at regular intervals, which can be useful in various applications, such as counters, timers, or progress indicators.
- **Sleep() Method**: The **sleep()** method is used to pause the thread's execution for a specified period, allowing the program to wait for a certain amount of time before performing the next action.

- **Concurrency**: Although this program uses a single thread, it illustrates the basic concept of concurrency, where a thread performs a task independently of the main program flow.

This program is a simple example of how threads can be used to perform repetitive tasks with a delay between each iteration.

NO.	AIM OF THE PRACTICAL
| 36. | Write a program to create three threads 'FIRST', 'SECOND', 'THIRD'. Set the priority of the 'FIRST' thread to 3, the 'SECOND' thread to 5(default) and the 'THIRD' thread to 7. <br><br> **PROGRAM:** <br><br> ```java
import java.util.*;

class FirstThread extends Thread {
   public void run() {
      System.out.println("First thread running...");
   }
}

class SecondThread extends Thread {
   public void run() {
      System.out.println("Second thread running...");
   }
}

class ThirdThread extends Thread {
   public void run() {
      System.out.println("Third thread running...");
   }
}

public class ThreadPriorityExample {
   public static void main(String[] args) {
      FirstThread first = new FirstThread();
      SecondThread second = new SecondThread();
      ThirdThread third = new ThirdThread();

      first.setPriority(3);
      second.setPriority(5); // Default
      third.setPriority(7);

      first.start();
      second.start();
      third.start();
   }
}
``` |

OUTPUT:

```
First thread running...
Third thread running...
Second thread running...

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

CONCLUSION:

- **Thread Priorities**: This program demonstrates how to set the priority of threads in Java, which can affect the order in which they are executed by the scheduler.
- **Priority Levels**: The priority levels range from 1 (lowest) to 10 (highest), with the default priority being 5. In this program, the 'FIRST' thread has a priority of 3, the 'SECOND' thread has the default priority of 5, and the 'THIRD' thread has a priority of 7.
- **Thread Scheduling**: The Java Virtual Machine (JVM) schedules threads based on their priorities, with higher-priority threads getting preference over lower-priority threads. However, the exact scheduling behavior may vary depending on the JVM implementation and system resources.

This program illustrates the basic concept of thread priorities and how they can influence the execution order of threads.

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

PROGRAM:

```java
import java.util.*;

class ProducerConsumer {
    int item;
    boolean produced = false;

    public synchronized void produce() throws InterruptedException {
        while (produced) {
            wait();
        }
        item = new Random().nextInt(100);
```

```
            System.out.println("Produced: " + item);
            produced = true;
            notify();
        }

    public synchronized void consume() throws InterruptedException {
        while (!produced) {
            wait();
        }
        System.out.println("Consumed: " + item);
        produced = false;
        notify();
    }
}

class Producer extends Thread {
    ProducerConsumer pc;

    Producer(ProducerConsumer pc) {
        this.pc = pc;
    }

    public void run() {
        while (true) {
            try {
                pc.produce();
                Thread.sleep(1000); // Simulate production delay
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Consumer extends Thread {
    ProducerConsumer pc;

    Consumer(ProducerConsumer pc) {
        this.pc = pc;
    }

    public void run() {
        while (true) {
            try {
                pc.consume();
                Thread.sleep(1000); // Simulate consumption delay
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
```

```
        }
    }

public class ProducerConsumerProblem {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();
        new Producer(pc).start();
        new Consumer(pc).start();
    }
}
```
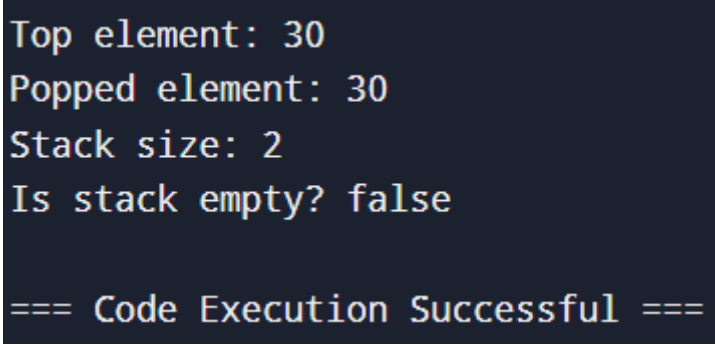
OUTPUT:

```
Produced: 32
Consumed: 32
Produced: 8
Consumed: 8
Produced: 42
Consumed: 42
Produced: 57
Consumed: 57
Produced: 6
Consumed: 6
```

CONCLUSION:

- **Thread Synchronization**: This program demonstrates how to use thread synchronization mechanisms like locks and conditions to solve the producer-consumer problem.
- **Shared Resource**: The shared queue is a critical resource that must be accessed safely by multiple threads.
- **Producer-Consumer Problem**: This program illustrates a classic problem in computer science where a producer thread produces items and a consumer thread consumes them, with the need to synchronize access to the shared queue to prevent errors.

PART-8

NO.	AIM OF THE PRACTICAL

38.	Design a Custom Stack using ArrayList class, which implements following functionalities of stack. My Stack -list ArrayList<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:
	
	``java
	import java.util.ArrayList;
	
	class MyStack {
	    private ArrayList<Object> list = new ArrayList<>();
	
	    public boolean isEmpty() {
	        return list.isEmpty();
	    }
	
	    public int getSize() {
	        return list.size();
	    }
	
	    public Object peek() {
	        if (!isEmpty()) {
	            return list.get(list.size() - 1);
	        }
	        return null;
	    }
	
	    public Object pop() {
	        if (!isEmpty()) {
	            return list.remove(list.size() - 1);
	        }
	        return null;
	    }
	
	    public void push(Object o) {
	        list.add(o);
	    }
	}
	
	public class CustomStack {
	``

```
        public static void main(String[] args) {
          MyStack stack = new MyStack();

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

          System.out.println("Top element: " + stack.peek());
          System.out.println("Popped element: " + stack.pop());
          System.out.println("Stack size: " + stack.getSize());
          System.out.println("Is stack empty? " + stack.isEmpty());
        }
      }
```

OUTPUT:

```
Top element: 30
Popped element: 30
Stack size: 2
Is stack empty? false


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

CONCLUSION:

- **Custom Stack Implementation**: This program demonstrates a custom implementation of a stack using an **ArrayList**, providing basic stack operations like **isEmpty**, **getSize**, **peek**, **pop**, and **push**.
- **ArrayList Benefits**: Using an **ArrayList** as the underlying data structure provides dynamic resizing and efficient insertion and removal of elements.
- **Stack Functionality**: The custom stack implementation provides the expected behavior of a stack, allowing users to add, remove, and inspect elements in a Last-In-First-Out (LIFO) order.

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

products, customers, or orders) as long as they implement the Comparable interface.

PROGRAM:

```java
import java.util.Arrays;

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

    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: " + price + ", Rating: " + rating;
    }
}

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

    public static void main(String[] args) {
        Product[] products = {
            new Product("Product1", 30.0, 4.5),
            new Product("Product2", 20.0, 4.7),
            new Product("Product3", 50.0, 4.3)
        };

        sortArray(products);

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

OUTPUT:

```
Product2 - Price: 20.0, Rating: 4.7
Product1 - Price: 30.0, Rating: 4.5
Product3 - Price: 50.0, Rating: 4.3

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

CONCLUSION:

- **Generic Sorting Method**: This program demonstrates a generic method that can sort arrays of **Comparable** objects, providing flexibility and reusability for sorting different types of objects.
- **Type Safety**: The use of generics ensures type safety, preventing the sorting of arrays with incompatible types.
- **Built-in Sorting**: The **Arrays.sort** method is used to sort the array, leveraging the built-in sorting functionality provided by Java.

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

PROGRAM:
```java
import java.util.*;

public class WordCount {
    public static void main(String[] args) {
        String text = "apple banana apple grape banana apple orange";

        Map<String, Integer> wordCount = new TreeMap<>();
        String[] words = text.split(" ");

        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }

        for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

OUTPUT:

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

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

CONCLUSION:

- **Word Counting**: This program effectively counts and displays the occurrences of words in a text using **Map** for counting and **Set** for sorting.
- **Flexibility**: The use of **HashMap** and **TreeSet** allows for efficient counting and sorting, making the solution adaptable to various text inputs.
- **Output**: The final output presents the words and their counts in a clear and organized manner, facilitating easy reading and analysis.

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

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

public class JavaKeywordCounter {

    public static void main(String[] args) {
        Set<String> javaKeywords = new HashSet<>();
        String[] keywords = {
            "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char",
"class", "const",
            "continue", "default", "do", "double", "else", "enum", "extends",
"final", "finally", "float",
            "for", "goto", "if", "implements", "import", "instanceof", "int",
"interface", "long", "native",
            "new", "package", "private", "protected", "public", "return", "short",
"static", "strictfp",
```

```
                    "super", "switch", "synchronized", "this", "throw", "throws",
            "transient", "try", "void",
                    "volatile", "while"
                };

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

                int keywordCount = 0;
                String fileName = "YourJavaFile.java";
                File file = new File(fileName);
                if (file.exists()) {
                    try (BufferedReader reader = new BufferedReader(new
            FileReader(file))) {
                        String line;
                        while ((line = reader.readLine()) != null) {
                            String[] words = line.split("\\W+");
                            for (String word : words) {
                                if (javaKeywords.contains(word)) {
                                    keywordCount++;
                                }
                            }
                        }
                    } catch (IOException e) {
                        System.err.println("Error reading file: " + e.getMessage());
                    }
                } else {
                    System.err.println("File not found: " + fileName);
                }

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

OUTPUT:


```
File not found: YourJavaFile.java
Number of keywords found: 0

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

CONCLUSION:

- **Keyword Counting**: This program effectively counts the number of keywords in a Java source file using a **HashSet** for efficient keyword lookup.

	• **Efficiency**: The use of a **HashSet** allows for fast lookup of keywords, making the program efficient for large source files. • **Flexibility**: The program can be easily modified to count keywords in different programming languages by updating the keyword set.

udemy

CERTIFICATE OF COMPLETION

Learn JAVA Programming - Beginner to Master

Instructors **Abdul Bari**

Shradul Rajput

Date **Oct. 16, 2024**

Length **61.5 total hours**

GROCERY SHOP MANAGEMENT SYSTEM

Code:

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

// Fundamental Product Class
class Product {
    private int id;
    private String name;
    private double price;
    private int stock;

    public Product(int id, String name, double price, int stock) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.stock = stock;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public int getStock() {
        return stock;
    }
```

```java
    public void setStock(int stock) {
        this.stock = stock;
    }

    @Override
    public String toString() {
        return String.format("ID: %d | Name: %s | Price: $%.2f | Stock: %d", id, name,
price, stock);
    }
}

// Inheritance, Interfaces & Packages
interface Billable {
    double calculateTotal();
}

class CartItem extends Product implements Billable {
    private int quantity;

    public CartItem(int id, String name, double price, int stock, int quantity) {
        super(id, name, price, stock);
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    @Override
    public double calculateTotal() {
        return getQuantity() * getPrice();
    }

    @Override
    public String toString() {
        return super.toString() + String.format(" | Quantity: %d | Total: $%.2f", quantity,
calculateTotal());
    }
}
```

```java
// Exceptions Handling
class InsufficientStockException extends Exception {
    public InsufficientStockException(String message) {
        super(message);
    }
}

// Store Class with Collections, File Handling, and Multithreading
class Store {
    private ArrayList<Product> productList;
    private ArrayList<CartItem> cart;

    public Store() {
        productList = new ArrayList<>();
        cart = new ArrayList<>();
        loadProducts(); // Load products from file
    }

    // Add new product to store
    public synchronized void addProduct(Product product) {
        productList.add(product);
    }

    // Display all products
    public void displayProducts() {
        for (Product product : productList) {
            System.out.println(product);
        }
    }

    // Search product by ID
    public Product searchProductById(int id) {
        for (Product product : productList) {
            if (product.getId() == id) {
                return product;
            }
        }
        return null;
    }
```

```java
    // Add product to cart
    public void addToCart(int productId, int quantity) throws
InsufficientStockException {
        Product product = searchProductById(productId);
        if (product != null) {
            if (product.getStock() >= quantity) {
                CartItem cartItem = new CartItem(product.getId(), product.getName(),
product.getPrice(), product.getStock(), quantity);
                cart.add(cartItem);
                product.setStock(product.getStock() - quantity);
                System.out.println("Added to cart: " + cartItem);
            } else {
                throw new InsufficientStockException("Not enough stock available.");
            }
        } else {
            System.out.println("Product not found!");
        }
    }

    // Display Cart Items
    public void displayCart() {
        if (cart.isEmpty()) {
            System.out.println("Cart is empty.");
        } else {
            for (CartItem item : cart) {
                System.out.println(item);
            }
        }
    }

    // Save products to file
    public void saveProducts() {
        try (FileWriter fw = new FileWriter("products.txt", false);
             BufferedWriter bw = new BufferedWriter(fw)) {
            for (Product product : productList) {
                bw.write(product.getId() + "," + product.getName() + "," +
product.getPrice() + "," + product.getStock() + "\n");
            }
            System.out.println("Products saved successfully.");
        } catch (IOException e) {
```

```java
            System.out.println("Error saving products: " + e.getMessage());
        }
    }


    // Load products from file
    public void loadProducts() {
        try (BufferedReader br = new BufferedReader(new FileReader("products.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                String[] data = line.split(",");
                int id = Integer.parseInt(data[0]);
                String name = data[1];
                double price = Double.parseDouble(data[2]);
                int stock = Integer.parseInt(data[3]);
                addProduct(new Product(id, name, price, stock));
            }
        } catch (FileNotFoundException e) {
            System.out.println("Product file not found. Starting with an empty store.");
        } catch (IOException e) {
            System.out.println("Error loading products: " + e.getMessage());
        }
    }


    // Thread to simulate adding new products
    public void simulateAddingProducts() {
        Runnable addTask = () -> {
            try {
                Thread.sleep(2000);
                addProduct(new Product(101, "Banana", 0.99, 100));
                System.out.println("New product added to the store!");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        };
        new Thread(addTask).start();
    }
}

// Main Class for User Interaction
public class GroceryStore {
```

```java
public static void main(String[] args) {
    Store store = new Store();
    Scanner scanner = new Scanner(System.in);
    int choice;

    store.simulateAddingProducts(); // Start background thread

    do {
        System.out.println("\n==== Grocery Store Menu ====");
        System.out.println("1. Display Products");
        System.out.println("2. Add Product to Cart");
        System.out.println("3. View Cart");
        System.out.println("4. Add New Product to Store");
        System.out.println("5. Save Products to File");
        System.out.println("6. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();

        switch (choice) {
            case 1:
                store.displayProducts();
                break;

            case 2:
                System.out.print("Enter product ID to add to cart: ");
                int id = scanner.nextInt();
                System.out.print("Enter quantity: ");
                int quantity = scanner.nextInt();

                try {
                    store.addToCart(id, quantity);
                } catch (InsufficientStockException e) {
                    System.out.println("Error: " + e.getMessage());
                }
                break;

            case 3:
                store.displayCart();
                break;
```

```java
        case 4:
            System.out.print("Enter new product ID: ");
            int newId = scanner.nextInt();
            scanner.nextLine(); // consume the newline character
            System.out.print("Enter new product name: ");
            String name = scanner.nextLine();
            System.out.print("Enter new product price: ");
            double price = scanner.nextDouble();
            System.out.print("Enter new product stock: ");
            int stock = scanner.nextInt();

            Product newProduct = new Product(newId, name, price, stock);
            store.addProduct(newProduct);
            System.out.println("New product added: " + newProduct);
            break;

        case 5:
            store.saveProducts();
            break;

        case 6:
            System.out.println("Exiting the store. Goodbye!");
            break;

        default:
            System.out.println("Invalid choice! Please try again.");
        }
    } while (choice != 6);

    scanner.close();
    }
}
```

Concepts:

This Java program utilizes several key programming concepts and features, which are outlined below:

1. Object-Oriented Programming (OOP) Concepts:

- Classes and Objects:

  - Product, CartItem, Store, and GroceryStore are defined as separate classes to encapsulate related data and behaviors.

- Inheritance:

  - CartItem inherits from the Product class, allowing it to reuse the properties and methods defined in Product.

- Polymorphism:

- o CartItem overrides the toString() method from Product to customize its string representation.

- Encapsulation:

  - o Class fields like id, name, price, and stock are marked as private and are accessed through public getters and setters to control visibility and maintain data integrity.

- Interfaces:

  - o The Billable interface is used to define the calculateTotal() method, which is then implemented in the CartItem class, demonstrating polymorphic behavior.

2. Exception Handling:

- A custom exception class, InsufficientStockException, is created to handle cases where the requested quantity of a product exceeds its available stock.

- try-catch blocks are used to manage exceptions when adding items to the cart, ensuring that invalid operations are handled gracefully.

3. File Handling:

- The program uses FileReader, BufferedReader, FileWriter, and BufferedWriter to read from and write to a text file (products.txt), demonstrating basic file I/O operations.

- This feature enables persistence, allowing the program to load products from a file when the store is initialized and save changes to the file before exiting.

4. Collections Framework:

- The ArrayList class is used to store collections of products (productList) and cart items (cart).

- It provides dynamic resizing and various utility methods for managing and manipulating collections.

5. Multithreading:

- The simulateAddingProducts() method in the Store class spawns a new thread using Runnable to add a product asynchronously, simulating concurrent operations in the store.

- This showcases basic usage of threads in Java.

6. Synchronization:

- The addProduct() method in the Store class is marked with synchronized to prevent concurrent modifications to the productList when products are being added in a multithreaded environment.

7. User Input Handling:

- The Scanner class is used to read input from the user for selecting options from the menu, adding products to the store, and managing the cart.

8. Looping and Control Structures:

- for loops are used to iterate through the collections when displaying products and cart items.

- A do-while loop is used in the main() method to repeatedly display the menu until the user chooses to exit.

9. String Formatting:

- The String.format() method is used to format strings for displaying product and cart item details in a structured format.

10. Static Typing and Method Overloading:

- The program leverages Java's static typing system to ensure type safety.

- Method overriding is demonstrated in the CartItem class with the toString() and calculateTotal() methods.

These concepts work together to create a well-structured, interactive program that simulates a basic grocery store management system with products and a shopping cart.

BY-23DCS 114,101,103