DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Engineering/Computer Science &

Engineering/Information Technology

Subject Name: JAVA PROGRAMMING
Semester: III
Subject Code: CSE201
Academic year: 2024-25

PART-I Data Types, Variables, String, Control Statements, Operators, Arrays

No.	Aim of the Practical
1.	**AIM :** 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. **Ans**. Object Oriented Concepts: Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to design and develop applications. The key concepts of OOP are: • Class: A blueprint for creating objects. It defines a datatype by bundling data and methods that work on the data. • Object: An instance of a class. It represents a real-world entity. • Encapsulation: The wrapping of data (attributes) and methods (functions) into a single unit or class. It restricts direct access to some of the object's components, which is a means of preventing accidental interference and misuse of the data. • Inheritance: The mechanism by which one class can inherit the properties and behaviors of another class. It promotes code reusability. • Polymorphism: The ability of different classes to be treated as instances of the sameclass through inheritance. It allows one interface to be used for a general class of actions. Abstraction: The concept of hiding the complex implementation details and showing only the necessary features of an object. It simplifies the complexity of the system. Comparison of java with other object oriented programming languages:

TOPIC	C++	Java	Python
Compiled vs. Interpreted	Compiled Programming Language	Java is both Compiled and Interpreted.	Interpreted Programming Language
Platform Dependence	C++ is platform dependent	Java is platformindependent	Python is platformindependent
Operator Overloading	C++ supports operator overloading	Java does not support operator overloading	Python supports operator overloading
Inheritance	C++ provides both single and multiple inheritances	In Java, single inheritance is possible while multiple inheritances can be achieved using Interfaces	Python provides both single and multiple inheritances
Thread Support	C++ does not have built-in support for threads; It depends on Libraries	Java has built-in thread support	Python supports multithreading
Execution Time	C++ is very fast. It's, in fact, the first choice of competitive programmers	Java is much faster than Python in terms of speed of execution but slower than C++.	Due to the interpreter, Python is slow in terms of execution

Program Handling	Functions and variables are used outside the class	Every bit of code(variables and functions) has to be inside the class itself.	Functions and variables can be declared and used outside the class
Library Support	C++ has limited library support	Java provides library support for many concepts like UI	Python has a huge set of libraries and modules.
Code Length	Code length is lesser than Java, around 1.5 times less.	Java code length is bigger than Python and C++.	Python has a smaller code length

Introduction to JDK: The Java Development Kit (JDK) is a cross-platformed software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets. It is a core package used in Java, along with the JVM (Java Virtual Machine) and the JRE (Java Runtime Environment).

Beginners often get confused with JRE and JDK, if you are only interested in running Java programs on your machine then you can easily do it using Java Runtime Environment.
However, if you would like to develop a Java-based software application then along with JRE you may need some additional necessary tools, which is called JDK.
The JDK has a private Java Virtual Machine (JVM) and a few other resources necessary for the development of a Java Application.
JDK contains:

- Java Runtime Environment (JRE),
- An interpreter/loader (Java),
- A compiler (javac),
- An archiver (jar) and many more.

The Java Runtime Environment in JDK is usually called Private Runtime because it is separated from the regular JRE and has extra content. The Private Runtime in JDK contains a JVM and all the class libraries present in the production environment, as well as additional libraries useful to developers, e.g, internationalization libraries and the IDL libraries.

Introduction to JRE: Java Runtime Environment (JRE) is an open-access software distribution that has a Java class library, specific tools, and a separate JVM. In Java, JRE is one of the interrelated components in the Java Development Kit (JDK). It is the most common environment available on devices for running Java programs. Java source code
is compiled and converted to Java bytecode. If you want to run this bytecode on any platform, you need JRE. The JRE loads classes check memory access and get

systemresources. JRE acts as a software layer on top of the operating system.

Components of Java JRE
The components of JRE are mentioned below:
- Integration libraries include Java Database Connectivity (JDBC)
- Java Naming, Interface Definition Language (IDL)
- Directory Interface (JNDI)
- Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP)
- Remote Method Invocation (RMI)
- Scripting

Java Virtual Machine (JVM) consists of Java HotSpot Client and Server VirtualMachine.
- User interface libraries include Swing, Java 2D, Abstract Window Toolkit (AWT), Accessibility, Image I/O, Print Service, Sound, drag, and Drop (DnD), and input methods.
- Lang and util base libraries, which include lang and util, zip, Collections, Concurrency Utilities, management, Java Archive (JAR), instrument, reflection, versioning, Preferences API, Ref Objects, Logging, and Regular Expressions.
- Other base libraries, including Java Management Extensions (JMX), Java Native Interface (JNI), Math, Networking, international support, input/output (I/O), Beans,
  Java Override Mechanism, Security, Serialization, extension mechanism, andJava for XML Processing (XML JAXP).
- Deployment technologies such as Java Web Start, deployment, and Java plug-in.

Introduction to JVM: JVM (Java Virtual Machine) is an abstract machine.
It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platformdependent).

It is:
- A specification where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
- An implementation Its implementation is known as JRE (Java Runtime Environment).
- Runtime Instance Whenever you write java command on the command prompt torun the java class, an instance of JVM is created.

The JVM performs following operation:
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:
- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Introduction to Javadoc: JavaDoc tool is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set ofsource file describing classes, methods, constructors, and fields.

Before using JavaDoc tool, you must include JavaDoc comments /**…
....
*/
providing information about classes, methods, and constructors, etc. For creating a goodand understandable document API for any java file you must write better comments for every class, method, constructor.

The JavaDoc comments is different from the normal comments because of the extra asterisk at the beginning of the comment. It may contain the HTML tags as well.

By writing a number of comments, it does not affect the performance of the Javaprogram as all the comments are removed at compile time.

Introduction to command line argument: Java command-line argument is an argument i.e. passed at the time of running the Java program. In Java, the command linearguments passed from the console can be received in the Java program and they can beused as input. The users can pass the arguments during the execution bypassing the command-line arguments inside the main() method.

Working command-line arguments
We need to pass the arguments as space-separated values. We can pass both strings andprimitive data types(int, double, float, char, etc) as command-line arguments. These arguments convert into a string array and are provided to the main() function as a stringarray argument. When command-line arguments are supplied to JVM, JVM wraps theseand supplies them to args[]. It can be confirmed that they are wrapped up in an args array by checking the length of args using args.length.

Internally, JVM wraps up these command-line arguments into the args[ ] array that we pass into the main() function. We can check these arguments using args.length method. JVM stores the first command-line argument at args[0], the second at args[1], the third at args[2], and so on.

2. **AIM :** Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20. Write a java program to store this balance in a variable and then display it to the user.

PROGRAM CODE :

```java
import java.util.Scanner;

public class prac2{
    public static void main(String[] args) {
        int balance=20,no;

        System.out.print("Enter your account number : ");
        Scanner s = new Scanner(System.in);
        no =s.nextInt();

        if(no==6)

    {
        System.out.print("Your curent balance is : "+ balance+"$");

    }
        else {System.out.print("Enter valid account number");}

        System.out.print("\nName : Mohit Barad \nID : 23DCS006 ");
    }

}
```

OUTPUT:

```
Enter your account number : 6
Your curent balance is : 20$
Name : Mohit Barad
ID : 23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```
OUTPUT: PRACTICAL-2

CONCLUSION:

This code demonstrates a simple Java program that prompts the user for an account number, checks if it matches a predefined value (1), and then displays the current balance if the match is successful. If the account number does not match, it prompts the user to enter a valid account number. The program uses basic control flow with an if-else statement to determine the output based on the user's input. It also illustrates the use of the Scanner class for reading user input from the console. Additionally, the program prints the name and ID of the author or a placeholder individual, showcasing basic string concatenation and output in Java. This code serves as a fundamental example of conditional logic, user input

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

PROGRAM CODE :

```
import java.util.Scanner;

public class prac3{
public static void main(String[] args){
float distance;
int hr,min,sec;

System.out.print("Enter The Distance(in meter) : ");
Scanner s = new Scanner(System.in);
distance =s.nextFloat();
float km=distance/1000;
float miles=distance/1609;

System.out.println("Enter The time(in hr,min,sec)");
hr =s.nextInt();
min = s.nextInt();
```
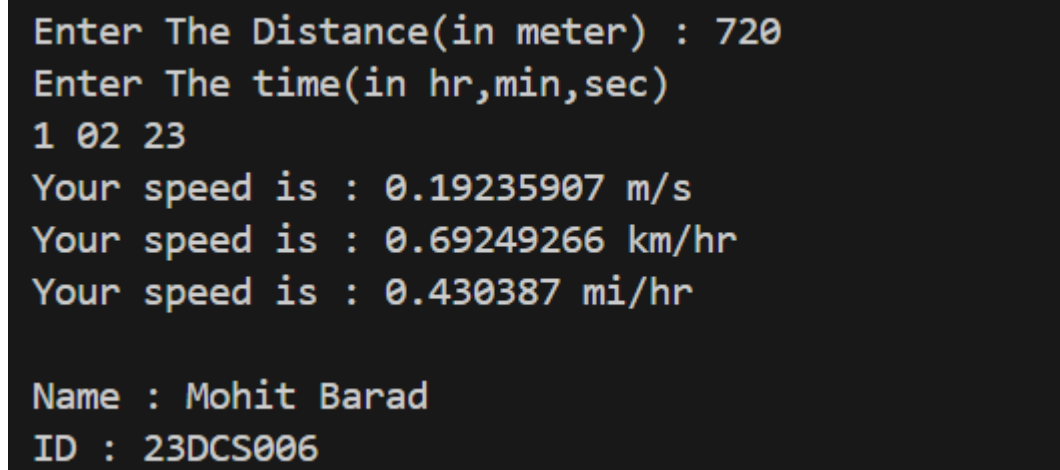
```
sec = s.nextInt();

float timeInSec = (hr*3600)+(min*60)+(sec);
float timeInHr = hr+((float) min /60)+((float) sec /3600);

float speed1 = distance/timeInSec;
float speed2 =km/timeInHr;
float speed3 =miles/timeInHr;

System.out.println("Your speed is : "+speed1+" m/s");
System.out.println("Your speed is : "+speed2+" km/hr");
System.out.println("Your speed is : "+speed3+" mi/hr");
System.out.print("\nName : Mohit Barad \nID : 23DCS006 ");
}
}
```

OUTPUT:

```
Enter The Distance(in meter) : 720
Enter The time(in hr,min,sec)
1 02 23
Your speed is : 0.19235907 m/s
Your speed is : 0.69249266 km/hr
Your speed is : 0.430387 mi/hr


Name : Mohit Barad
ID : 23DCS006
```

OUTPUT: PRACTICAL-3

CONCLUSION:

This program calculates and displays the speed of an object in meters per second, kilometers per hour, and miles per hour based on the user-provided distance (in meters) and time (in hours, minutes, and seconds). It utilizes the Scanner class for input, demonstrating basic Java input/output operations and arithmetic calculations. The conversion of distance to kilometers and miles, along with the conversion of time into seconds and hours, showcases practical applications of mathematical concepts in programming. The program outputs the calculated speeds in different units, providing a clear example of unit conversion and formatting output in Java. It concludes with the author's name and ID, personalizing the code output. This code snippet is a

straightforward example of applying mathematical formulas and user input in Java to solve real-world problems.

4. **<u>AIM :</u>** 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.

<u>PROGRAM CODE :</u>

```java
import java.util.Scanner;

public class prac4 {

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

        System.out.print("Enter the number of days you have expenses for: ");
        int numberOfDays = scanner.nextInt();

        double[] dailyExpenses = new double[numberOfDays];

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

        double totalExpenses = calculateTotalExpenses(dailyExpenses);

         System.out.println("Total Expenses for the Month: " + totalExpenses + "/-Rs.");

        System.out.print("\nName : Mohit Barad \nID : 23DCS006 ");
    }

    public static double calculateTotalExpenses(double[] expenses) {
        double total = 0;
        for (int i = 0; i < expenses.length; i++) {
            total += expenses[i];
        }
        return total;
    }
}
```

OUTPUT:

```
Enter the number of days you have expenses for: 4
Enter expense for day 1: 240
Enter expense for day 2: 350
Enter expense for day 3: 265
Enter expense for day 4: 179
Total Expenses for the Month: 1034.0/- Rs.

Name : Mohit Barad
ID : 23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-4

CONCLUSION:

This program is designed to calculate and display the total monthly expenses based on daily inputs from the user. It starts by asking the user for the number of days they have expense data for, then collects expense amounts for each day using a loop. These amounts are stored in an array of doubles. The program calculates the total expenses by passing this array to the calculateTotalExpenses method, which iterates over the array to sum the expenses. Finally, it displays the total expenses to the user. This program demonstrates basic Java concepts such as arrays, loops, methods, and user input handling with the Scanner class. It concludes with displaying the author's name and ID, personalizing the output.

5.

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

PROGRAM CODE :

import java.util.Scanner;

public class prac5 {

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

```java
        float[] price = {100, 100, 100, 100, 100};
        float totalPrice = 0;
        System.out.println("Welcome To Shop!");
        float tax = 0;
        int a;
        do {
            System.out.println("Enter product code (1 for motor, 2 for fan, 3 for tube, 4 for wires, 5 for others, 0 to finish):");
            a = ip.nextInt();
            switch (a) {
                case 1:
                    tax = (price[0] * 8) / 100;
                    break;
                case 2:
                    tax = (price[1] * 12) / 100;
                    break;
                case 3:
                    tax = (price[2] * 5) / 100;
                    break;
                case 4:
                    tax = (price[3] * 7.5f) / 100;
                    break;
                case 5:
                    tax = (price[4] * 3) / 100;
                    break;
                default:
                    tax = 0;
                    break;
            }
            if (a > 0 && a <= 5) {
                totalPrice += price[a - 1] + tax;
            } else {
                totalPrice += 0;
            }
        } while (a != 0);

        System.out.println("Total Bill: " + totalPrice);
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```

OUTPUT:

```
Welcome To Shop!
Enter product code (1 for motor, 2 for fan, 3 for tube, 4 for wires, 5 for others, 0 to finish):
1
Enter product code (1 for motor, 2 for fan, 3 for tube, 4 for wires, 5 for others, 0 to finish):
3
Enter product code (1 for motor, 2 for fan, 3 for tube, 4 for wires, 5 for others, 0 to finish):
2
Enter product code (1 for motor, 2 for fan, 3 for tube, 4 for wires, 5 for others, 0 to finish):
0
Total Bill: 325.0

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-5

CONCLUSION:

This program is a simple shopping cart calculator that computes the total bill including tax for a set of predefined products. It uses a do-while loop to continuously prompt the user for product codes until the user decides to finish by entering 0. Tax rates are applied differently based on the product selected, using a switch statement to determine the tax amount for each product. The total price, including tax, is accumulated and displayed at the end. This program demonstrates basic Java constructs such as arrays, loops, conditional statements, and user input handling with the Scanner class. It concludes by displaying the author's name and ID, adding a personal touch to the output.

6.

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

PROGRAM CODE :

```java
import java.util.Scanner;

public class prac6 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the number of days for your exercise routine: ");
        int n = scanner.nextInt();
```
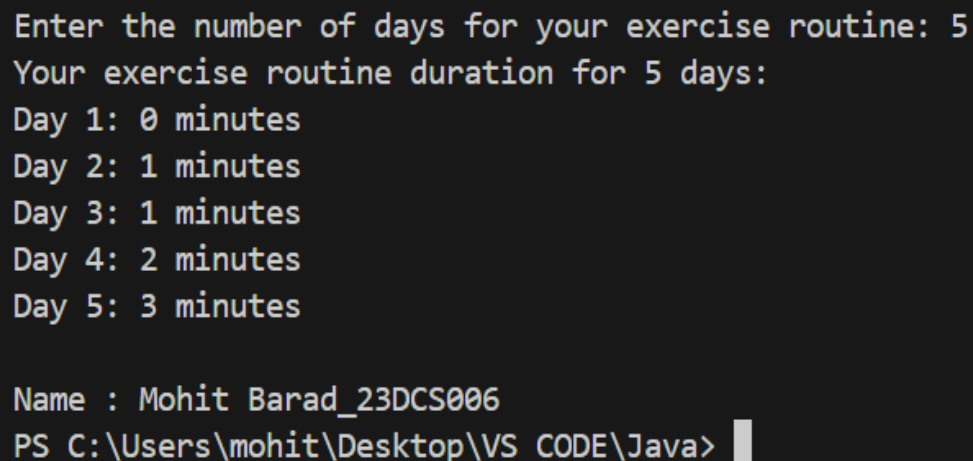
```java
        int firstTerm = 0, secondTerm = 1;

        System.out.println("Your exercise routine duration for " + n + " days:");

        for (int i = 1; i <= n; ++i) {
            System.out.println("Day " + i + ": " + firstTerm + " minutes");


            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
        System.out.print("\nName : Mohit Barad_23DCS006 ");
    }
}
```

OUTPUT:

```
Enter the number of days for your exercise routine: 5
Your exercise routine duration for 5 days:
Day 1: 0 minutes
Day 2: 1 minutes
Day 3: 1 minutes
Day 4: 2 minutes
Day 5: 3 minutes


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```
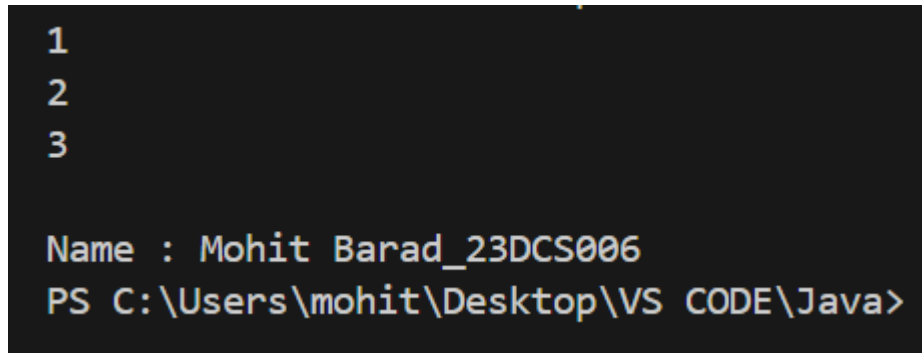
OUTPUT: PRACTICAL-6

CONCLUSION:

This program is designed to generate an exercise routine schedule based on the Fibonacci sequence, where the duration for each day is derived from this sequence. It prompts the user to enter the number of days they plan to follow the routine. Utilizing a simple for loop, it calculates the duration for each day, starting with 0 minutes on the first day and 1 minute on the second, then follows the Fibonacci pattern for subsequent days. The output is a daily exercise plan displaying the duration in minutes for the specified number of days. This program demonstrates the application of loops, basic arithmetic operations, and user input handling in Java. It concludes with displaying the author's name and ID, personalizing the output.

PART-II Strings

No.	Aim of the Practical

7.	**AIM :** Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front; front_times('Chocolate', 2) → 'ChoCho' front_times('Chocolate', 3) → 'ChoChoCho' front_times('Abc', 3) → 'AbcAbcAbc'
	
	 PROGRAM CODE :
	
	 import java.util.*;
	
	 public class prac7 {
	
	 static void front_times(String s, int a){
	 s=s.substring(0, 3);
	
	 for(int i=0;i<a;i++){
	 System.out.print(s);
	 }
	
	 }
	 public static void main(String[] args) {
	 Scanner ip = new Scanner(System.in);
	 String s;
	 System.out.print("Enter a String(Word containing more than 3 characters) : ");
	 s=ip.next();
	 System.out.print("How many times you want to repeat words? : ");
	 int n =ip.nextInt();
	 front_times(s,n);
	 System.out.print("\nName : Mohit Barad_23DCS006 ");
	
	 }
	 }
	 OUTPUT:
	
	 OUTPUT: PRACTICAL-7

CONCLUSION:

This Java program demonstrates the use of strings, loops, and user input handling. It specifically showcases substring manipulation by repeating the first three characters of a given string a specified number of times. The concepts of Java used include string manipulation, for loops, and the Scanner class for reading user input. Additionally, it employs static methods and basic input/output operations.

8. **AIM :** Given an array of ints, return the number of 9's in the array.

array_count9([1, 2, 9]) → 1

array_count9([1, 9, 9]) → 2

array_count9([1, 9, 9, 3, 9]) → 3

PROGRAM CODE :

```
import java.util.*;

    public class prac8 {
        public static int array_count9(int[] nums) {

            String arrayAsString = Arrays.toString(nums);

            int count = arrayAsString.length() - arrayAsString.replace("9", "").length();
            return count;
        }

        public static void main(String[] args) {
            System.out.println(array_count9(new int[]{1, 2, 9}));
            System.out.println(array_count9(new int[]{1, 9, 9}));
            System.out.println(array_count9(new int[]{1, 9, 9, 3, 9}));
            System.out.print("\nName : Mohit Barad_23DCS006 ");
        }
    }
```

OUTPUT:

```
1
2
3

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-8

CONCLUSION:

This Java program is designed to count the occurrences of the digit 9 within an array of integers. It utilizes the Arrays.toString() method for array-to-string conversion, string manipulation techniques to count occurrences, and the concept of array handling in Java. The program demonstrates the use of static methods, the main method for execution, and basic input/output operations. Key Java concepts include arrays, string manipulation, and the use of the replace method.

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

original, there are two chars.

double_char('The') → 'TThhee'

double_char('AAbb') → 'AAAAbbbb'

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

PROGRAM CODE :

```java
import java.util.*;

public class prac9 {

    static String double_char(String s){
        int a=s.length();
        String str3="";

        for(int i=0;i<a;i++){
            char result = s.charAt(i);
            str3= str3+result+result;
        }
        return str3;
```
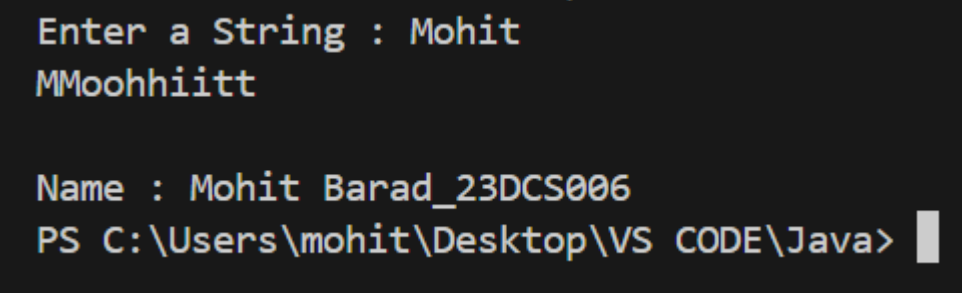
3

```
    }
    public static void main(String[] args) {
        Scanner ip = new Scanner(System.in);
        String s;
        System.out.print("Enter a String : ");
        s=ip.next();

        System.out.println(double_char(s));
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```

OUTPUT:

Enter a String : Mohit
MMoohhiitt

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>

OUTPUT: PRACTICAL-9

CONCLUSION:

This Java program demonstrates the use of strings, loops, and the Scanner class for input handling. It features a method double_char that duplicates each character in a given string, showcasing string manipulation and concatenation techniques. Key Java concepts utilized include methods, loops (for loop), character manipulation (charAt method), and basic input/output operations. The program also exemplifies the use of the Scanner class for reading user input from the console.

---

10. **AIM :** Perform following functionalities of the string:

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

PROGRAM CODE :

```java
import java.util.*;

  public class prac10 {

    public static String reverse(String s){
        String str3="";
        for(int i=s.length()-1;i>=0;i--){
            str3=str3+s.charAt(i);
        }
        return str3;
    }

    public static String sort(String s){
        char[] ch = s.toCharArray();
        Arrays.sort(ch);
        return new String(ch);
    }

    public static void main(String[] args) {
        Scanner ip = new Scanner(System.in);
        String s;
        System.out.print("Enter a String : ");
        s=ip.next();
        System.out.print("Length  : "+s.length()+"\n");
        System.out.print("Upper case : "+s.toUpperCase()+"\n");
        System.out.print("Lower case : "+s.toLowerCase()+"\n");
        System.out.print("Reverse : "+reverse(s)+"\n");
        System.out.print("Sort : "+sort(s)+"\n");
        System.out.print("\nName : Mohit Barad_23DCS006 ");


    }
  }
```

OUTPUT:

```
Enter a String : Charusat
Length  : 8
Upper case : CHARUSAT
Lower case : charusat
Reverse : tasurahC
Sort : Caahrstu

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-10

CONCLUSION:

This Java program demonstrates string manipulation techniques including reversing and sorting characters within a string. It utilizes loops for reversing the string, the Arrays.sort() method for sorting, and the Scanner class for reading user input. Key Java concepts employed are string manipulation, character arrays, and basic input/output operations. The program effectively showcases the use of control structures (loops) and array handling in Java.

11.	**AIM :** Perform following Functionalities of the string:

"CHARUSAT UNIVERSITY"

● Find length

● Replace 'H' by 'FIRST LATTER OF YOUR NAME'

● Convert all character in lowercase

PROGRAM CODE :

```java
import java.util.*;

    public class prac11 {

        public static void main(String[] args) {
```
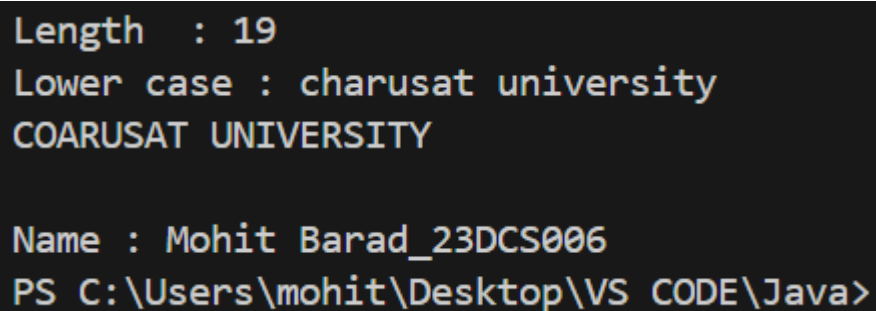
```java
        String s="CHARUSAT UNIVERSITY";
        System.out.print("Length  : "+s.length()+"\n");
        System.out.print("Lower case : "+s.toLowerCase()+"\n");
        System.out.println(s.replace('H', 'O'));
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```

OUTPUT:

```
Length  : 19
Lower case : charusat university
COARUSAT UNIVERSITY

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-11

CONCLUSION:

This Java program, demonstrates basic string manipulation operations such as calculating string length, converting to lowercase, and replacing characters within a string. It utilizes the String class methods length(), toLowerCase(), and replace(). The program showcases fundamental Java concepts including string handling and the use of built-in methods for string manipulation. It also illustrates basic output operations with System.out.print and System.out.println for displaying results.

PART-III Object Oriented Programming: Classes, Methods, Constructors

No.	Aim of the Practical

12.	**AIM :** Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.
	
	PROGRAM CODE :
	
	``java
	import java.util.*;
	class prac12
	{
	    public static void main(String[] args)
	    {
	    int a= Integer.parseInt(args[0]);
	    int c=a*100;
	
	    System.out.println("Currency in a rupees=" + c);
	
	    Scanner S1 = new Scanner(System.in);
	    int R=S1.nextInt();
	    int P=R*100;
	    System.out.println("Currency in Rupees=" + P);
	    System.out.print("\nName : Mohit Barad_23DCS006 ");
	
	    }
	}
	``
	
	OUTPUT:
	
	OUTPUT: PRACTICAL-12
	
	CONCLUSION:
	
	This Java program demonstrates basic arithmetic operations and user input handling. It converts a given amount to a different currency unit by multiplying it with a fixed conversion rate (100). The program utilizes command-line arguments for initial input and the Scanner class for runtime input from the user. Key Java concepts used include parsing command-line arguments, arithmetic operations, and handling user inputs with the Scanner class.

13. **AIM :** 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

PROGRAM CODE :

```java
import java.util.*;

class Employee {
    private String fname;
    private String lname;
    private Double Sal;

    Employee(String fs, String ls, double sl) {
        fname = fs;
        lname = ls;
        Sal = sl;
    }

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

    void putls() {
        System.out.println("" + lname);

    }

    void putsal() {
        System.out.println("" + Sal);
    }

    public void setFname(String fname) {
        this.fname = fname;
    }

    public void setLname(String lname) {
        this.lname = lname;
    }

    public void setSal(Double sal) {
        this.Sal = sal;
    }
```

```java
    public String getFname() {
      return fname;
    }

    public String getLname() {
      return lname;
    }

    public Double getSal() {
      return Sal;
    }
}

public class prac13 {
  public static void main(String[] args) {

      Employee emp1 = new Employee("John", "Doe", 3000.0);
      Employee emp2 = new Employee("Jane", "Doe", 4000.0);

      System.out.println(emp1.getFname() + " " + emp1.getLname() + "'s yearly salary:
" + (emp1.getSal() * 12));
      System.out.println(emp2.getFname() + " " + emp2.getLname() + "'s yearly salary:
" + (emp2.getSal() * 12));

      emp1.setSal(emp1.getSal() * 1.10);
      emp2.setSal(emp2.getSal() * 1.10);

      System.out.println(emp1.getFname() + " " + emp1.getLname() + "'s yearly salary
after 10% raise: " + (emp1.getSal() * 12));
      System.out.println(emp2.getFname() + " " + emp2.getLname() + "'s yearly salary
after 10% raise: " + (emp2.getSal() * 12));
      System.out.print("\nName : Mohit Barad_23DCS006 ");

  }
}
```

OUTPUT:

3

```
John Doe's yearly salary: 36000.0
Jane Doe's yearly salary: 48000.0
John Doe's yearly salary after 10% raise: 39600.00000000001
Jane Doe's yearly salary after 10% raise: 52800.0

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-13

CONCLUSION:

This code snippet defines an Employee class in Java, showcasing the use of encapsulation, one of the four fundamental OOP concepts. It provides a constructor for initializing objects with first name, last name, and salary. Methods for displaying these properties (putfs, putls, putsal) and setters (setFname, setLname) are included, demonstrating the use of class methods and data hiding by manipulating private data through public methods. This approach ensures controlled access to the class's fields, aligning with the encapsulation principle.

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

PROGRAM CODE :

```java
import java.util.*;

class Date {
    int year;
    int month;
    int day;
    Scanner S1 = new Scanner(System.in);

    void getDate() {
        day = S1.nextInt();
    }

    void getMonth() {
        month = S1.nextInt();
```

```java
        }

        void getYear() {
            year = S1.nextInt();
        }

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

        }

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

class prac14 {
    public static void main(String[] args) {
        Date d1 = new Date(28, 07, 2005);
        d1.displayDate();
 System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```
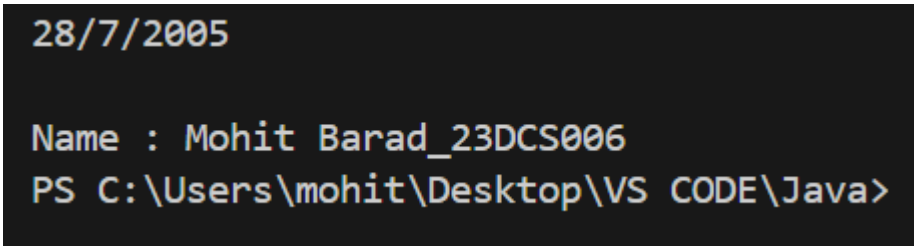
OUTPUT:

```
28/7/2005


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-14

CONCLUSION:

This Java code snippet demonstrates the use of classes and objects, fundamental concepts of Object-Oriented Programming (OOP). It defines a Date class with a

<table>
<tr><td></td><td>constructor to initialize the date and a method to display it. The main method in the class creates an instance of the Date class and calls its displayDate method to print the date. This example illustrates encapsulation by keeping the date fields private and accessing them through a public method, and constructor overloading by providing a specific constructor for date initialization.</td></tr>
<tr><td>15.</td><td>

AIM : Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.

PROGRAM CODE :

```java
import java.util.Scanner;

class Area {
    double length;
    double breadth;


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


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

class prac15 {
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter length: ");
        double length = scanner.nextDouble();
        System.out.print("Enter breadth: ");
        double breadth = scanner.nextDouble();

        Area A = new Area(length, breadth);
        System.out.println("The Area is : "+A.returnArea());
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```

OUTPUT:

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

```
Enter length: 5
Enter breadth: 6
The Area is : 30.0

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```
OUTPUT: PRACTICAL-15

CONCLUSION:

This Java code snippet demonstrates the use of classes, objects, and methods, which are key principles of Object-Oriented Programming (OOP). It includes user input handling with the Scanner class to dynamically receive length and breadth values. An Area class instance is created with these values, and its method returnArea() calculates and returns the area of a rectangle. The program showcases encapsulation by using class methods to operate on private class fields. Additionally, it illustrates the practical application of constructors for initializing object states.

16.

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

PROGRAM CODE :

```java
import java.util.*;

public class Complex_16 {

    private double real;
    private double imaginary;

    public Complex_16 () {
    }

    public Complex_16 (double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public static Complex_16  getInput() {
```

```java
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter real part: ");
        double real = scanner.nextDouble();
        System.out.print("Enter imaginary part: ");
        double imaginary = scanner.nextDouble();
        return new Complex_16 (real, imaginary);
    }

    public Complex_16 add(Complex_16 other) {
        return new Complex_16(real + other.real, imaginary + other.imaginary);
    }

    public Complex_16 subtract(Complex_16 other) {
        return new Complex_16(real - other.real, imaginary - other.imaginary);
    }

    public Complex_16 multiply(Complex_16 other) {
        double newReal = (real * other.real) - (imaginary * other.imaginary);
        double newImaginary = (real * other.imaginary) + (imaginary * other.real);
        return new Complex_16(newReal, newImaginary);
    }


    public String toString() {
        return String.format("%.2f + %.2fi", real, imaginary);
    }

    public static void main(String[] args) {
        System.out.println("Enter first complex number:");
        Complex_16 number1 = getInput();
        System.out.println("Enter second complex number:");
        Complex_16 number2 = getInput();

        Complex_16 sum = number1.add(number2);
        Complex_16 difference = number1.subtract(number2);
        Complex_16 product = number1.multiply(number2);

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
}
```

OUTPUT:

```
Enter first complex number:
Enter real part: 5
Enter imaginary part: 4
Enter second complex number:
Enter real part: 6
Enter imaginary part: 3
Sum: 11.00 + 7.00i
Difference: -1.00 + 1.00i
Product: 18.00 + 39.00i


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```
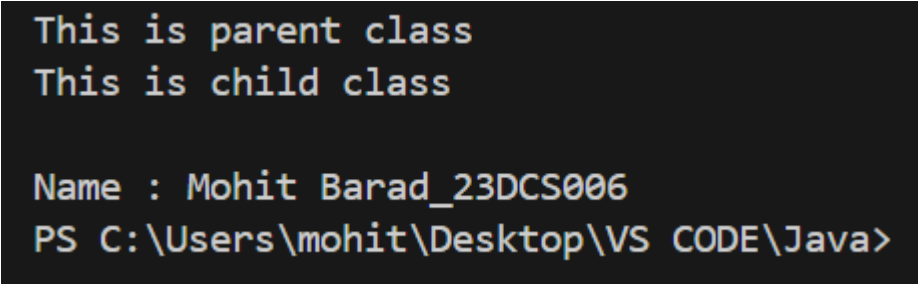
OUTPUT: PRACTICAL-16

CONCLUSION:

This Java code demonstrates the use of object-oriented programming (OOP) principles to perform arithmetic operations (addition, subtraction, multiplication) on complex numbers. It utilizes classes, objects, and methods to encapsulate the behavior and state of complex numbers. The toString method showcases polymorphism by overriding the default method in the Object class for custom string representation. The code also exemplifies the use of static methods and user input handling in Java.

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

PROGRAM CODE :

```
public class prac17 {

    void print1() {
        System.out.println("This is parent class");
    }

    public static void main(String[] args) {
        prac17 obj1 = new prac17();
        Child obj2 = new Child();
        obj1.print1();
        obj2.print1();
        System.out.print("\nName : Mohit Barad_23DCS006");
    }
}

class Child extends prac17 {
    void print1() {
        System.out.println("This is child class");
    }
}
```

OUTPUT:

```
This is parent class
This is child class

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-17

CONCLUSION:

This Java program demonstrates method overriding in inheritance. The Prac_17 class has a method print1 which is overridden by the Child class. When the print1 method is called on an instance of Child, it executes the overridden method in the Child class. The output shows the different messages from the parent and child classes, illustrating polymorphism.

18. **AIM :** Create a class named 'Member' having the following members: Data members
1 – Name

2 - Age

3 - Phone number

4 - Address

5 – Salary

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

PROGRAM CODE :

```java
class Member {
    String name;
    int age;
    String phoneNumber;
    String address;
    double salary;

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

class Employee extends Member {
    String specialization;
}

class Manager extends Member {
    String department;
}

public class prac18 {
    public static void main(String[] args) {
```

```java
        Employee employee = new Employee();
        employee.name = "Name 1";
        employee.age = 30;
        employee.phoneNumber = "123456789";
        employee.address = "India";
        employee.salary = 50000;
        employee.specialization = "Software Engineering";

        Manager manager = new Manager();
        manager.name = "name 2 ";
        manager.age = 40;
        manager.phoneNumber = "987654321";
        manager.address = "456 Elm St";
        manager.salary = 80000;
        manager.department = "IT";

        System.out.println("Employee Details:");
        System.out.println("Name: " + employee.name);
        System.out.println("Age: " + employee.age);
        System.out.println("Phone Number: " + employee.phoneNumber);
        System.out.println("Address: " + employee.address);
        employee.printSalary();
        System.out.println("Specialization: " + employee.specialization);

        System.out.println("\nManager Details:");
        System.out.println("Name: " + manager.name);
        System.out.println("Age: " + manager.age);
        System.out.println("Phone Number: " + manager.phoneNumber);
        System.out.println("Address: " + manager.address);
        manager.printSalary();
        System.out.println("Department: " + manager.department);

        System.out.print("\nName : Mohit Barad_23DCS006");
    }
}
```

OUTPUT:

```
Employee Details:
Name: Name 1
Age: 30
Phone Number: 123456789
Address: India
Salary: 50000.0
Specialization: Software Engineering

Manager Details:
Name: name 2
Age: 40
Phone Number: 987654321
Address: 456 Elm St
Salary: 80000.0
Department: IT


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-18

CONCLUSION:

This Java program demonstrates inheritance by defining a base class Member and two derived classes Employee and Manager. The Member class includes common attributes and a method to print the salary. The Employee and Manager classes inherit these attributes and add their own specific attributes. The main method creates an instance of Employee and sets its attributes, showcasing the use of inheritance.

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

PROGRAM CODE :

```java
public class prac19 {
    public static void main(String[] args) {
        Rectangle rect = new Rectangle(10, 5);
        Square sq = new Square(7);


        Rectangle[] shapes = new Rectangle[2];
        shapes[0] = rect;
        shapes[1] = sq;


        for (Rectangle shape : shapes) {
            shape.printArea();
            shape.printPerimeter();
            System.out.println();
        }
        System.out.print("\nName : Mohit Barad_23DCS006");

    }

}

class Rectangle {
    protected int length;
    protected int breadth;

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

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

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

class Square extends Rectangle {
    public Square(int side) {
```

```
        super(side, side);
    }
}
```

OUTPUT:

```
Area: 50
Perimeter: 30

Area: 49
Perimeter: 28



Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-19

CONCLUSION:

This Java program demonstrates polymorphism using inheritance. The Rectangle class is extended by the Square class, allowing both to be treated as Rectangle objects. An array of Rectangle objects is used to store both Rectangle and Square instances. The program iterates through the array, calling methods to print the area and perimeter of each shape.

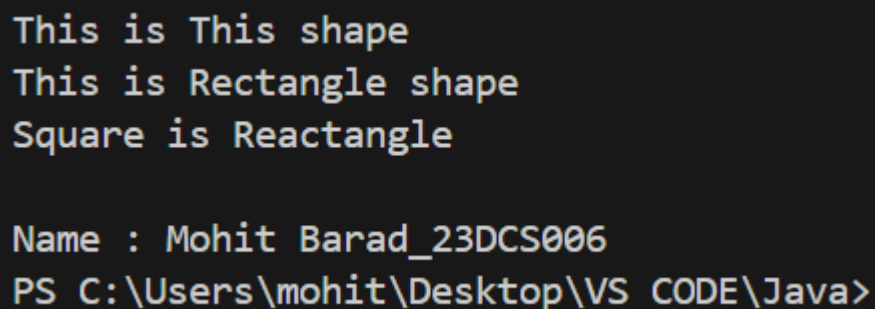|	

20.	**AIM :** Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.
	
	PROGRAM CODE :
	
	``java
	class Shape{
	    public void printS()
	    {
	        System.out.println("This is This shape");
	    }
	}
	class Rectangle extends Shape{
	    public void printR()
	    {
	        System.out.println("This is Rectangle shape");
	    }
	}
	``

```java
class Circle extends Shape{
    public void printC()
    {
        System.out.println("This is Circle shape");
    }
}
class Square extends Rectangle{
    public void printSq()
    {
        printS();
        printR();
        System.out.println("Square is Reactangle ");
    }
}
public class prac20{
    public static void main(String[] args)
    {
        Shape s = new Shape();
        Square A =new Square();
        A.printSq();
        System.out.print("\nName : Mohit Barad_23DCS006 ");
    }

}
```

OUTPUT:

OUTPUT: PRACTICAL-20

CONCLUSION:

This Java program demonstrates inheritance and method overriding. The Shape class is extended by Rectangle and Circle, each adding their own print methods. The Square class extends Rectangle and overrides its methods, showcasing multi-level inheritance. The program illustrates how derived classes can call methods from their parent classes.

| 21. | **AIM :** Create a class 'Degree' having a method 'getDegree' that prints "I got a degree". It has two subclasses namely 'Undergraduate' and 'Postgraduate' each having a method with the same name that prints "I am an Undergraduate" and "I am a Postgraduate" respectively. Call the method by creating an object of each of the three classes.

PROGRAM CODE :

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

    }

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

    }
}

public class prac21 {
    public static void main(String[] args) {
        Degree d = new Degree();
        Undergraduate u = new Undergraduate();
        Postgraduate p = new Postgraduate();
        d.getDegree();
        u.getDegree();
        p.getDegree();
        System.out.print("\nName : Mohit Barad_23DCS006 ");}

}
```
|

OUTPUT:


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


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-21

CONCLUSION:

This Java program demonstrates method overriding in inheritance. The Degree class has a method getDegree which is overridden by Undergraduate and Postgraduate classes. Each subclass provides its own implementation of the getDegree method. The main method creates instances of each class and calls their respective getDegree methods, showcasing polymorphism.
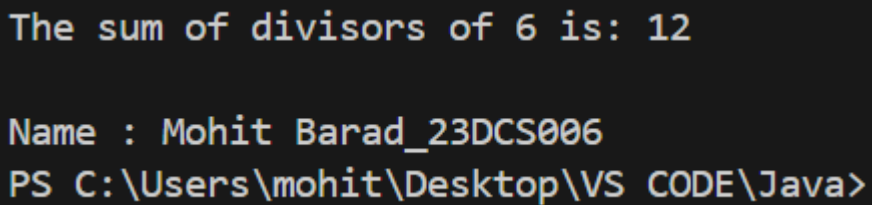
---

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

PROGRAM CODE :

```java
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;
    }
}
```

```java
public class prac22 {
    public static void main(String[] args) {
        MyCalculator myCalculator = new MyCalculator();
        int n = 6;
        int result = myCalculator.divisor_sum(n);
        System.out.println("The sum of divisors of " + n + " is: " + result);
        System.out.print("\nName : Mohit Barad_23DCS006 ");}

}
```

OUTPUT:

The sum of divisors of 6 is: 12

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>

OUTPUT: PRACTICAL-22

CONCLUSION:

This Java program demonstrates the implementation of an interface. The AdvancedArithmetic interface is implemented by the MyCalculator class, which provides the divisor_sum method. The main method creates an instance of MyCalculator and calculates the sum of divisors for a given number. The result is printed, showcasing the functionality of the implemented method.

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

PROGRAM CODE :

```java
interface Shape{
    void print();
}
class Circle implements Shape{
    int radius;
    String color;
```

```java
        Circle(int radius, String color){
            this.radius = radius;
            this.color = color;
        }
        public void print(){
            System.out.println("Radius : "+radius+" Color : "+color);
        }
}
class Rectangle implements Shape{
        int length;
        int width;
        String color;
        Rectangle(int length, int width, String color){
            this.length = length;
            this.width = width;
            this.color = color;
        }
        public void print(){
            System.out.println("Length : "+length+" Width : "+width+" Color : "+color);
        }
}
class Sign{
        Shape s;
        String text;
        Sign(Shape s, String text){
            this.s = s;
            this.text = text;
        }
        void print(){
            s.print();
            System.out.println("Text : "+text);
        }
}
public class prac23{
        public static void main(String[] args) {
            Circle c = new Circle(10, "Red");
            Rectangle r = new Rectangle(10, 20, "Blue");
            Sign s = new Sign(c, "Circle Sign");
            s.print();
            Sign s1 = new Sign(r, "Rectangle Sign");
            s1.print();
            System.out.print("\nName : Mohit Barad_23DCS006 ");

        }
}
```
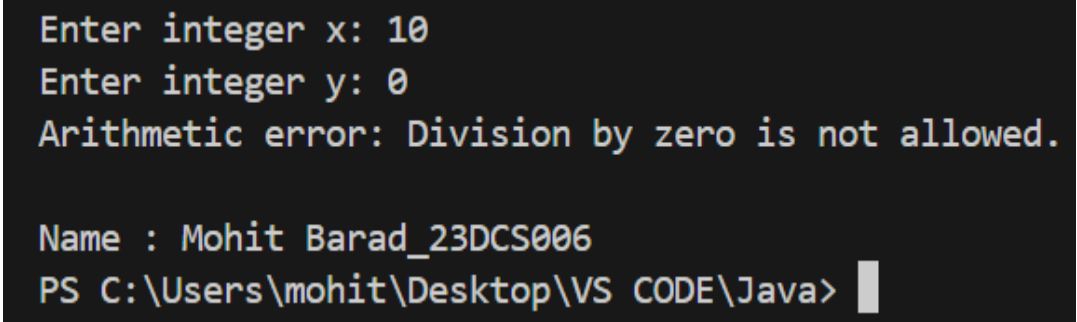
OUTPUT:

```
Radius : 10 Color : Red
Text : Circle Sign
Length : 10 Width : 20 Color : Blue
Text : Rectangle Sign

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```
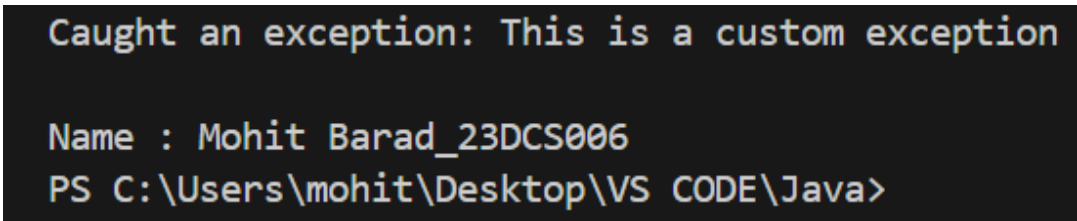
OUTPUT: PRACTICAL-23

CONCLUSION:

This Java program demonstrates the use of interfaces and polymorphism. The Shape interface is implemented by the Circle and Rectangle classes, each providing their own print method. Instances of Circle and Rectangle can be treated as Shape objects. This allows for flexible and interchangeable use of different shapes in the program.

PART-V Exception Handling

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

PROGRAM CODE :

```java
import java.util.Scanner;

public class prac24 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);

try {
System.out.print("Enter integer x: ");
int x = Integer.parseInt(scanner.nextLine());

System.out.print("Enter integer y: ");
int y = Integer.parseInt(scanner.nextLine());

int d = x / y;
System.out.println("Division " + d);
} catch (NumberFormatException e) {
System.out.println("Invalid input: Please enter valid integers.");
} catch (ArithmeticException e) {
System.out.println("Arithmetic error: Division by zero is not allowed.");
} finally {
scanner.close();
System.out.print("\nName : Mohit Barad_23DCS006");
}
}
}
```

OUTPUT:

```
Enter integer x: 10
Enter integer y: 0
Arithmetic error: Division by zero is not allowed.


Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-24

CONCLUSION:

This code is a simple console application that prompts the user to input two integers and performs division. It includes error handling for invalid input and division by zero. The Scanner object is used for input and is properly closed in the finally block. The program also prints a name and ID at the end. This ensures the program handles common input errors gracefully while demonstrating basic exception handling in Java.

---

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

PROGRAM CODE :

```java
public class prac25 {
    public static void main(String[] args) {
    try {
    throw new Exception("This is a custom exception");
    } catch (Exception e) {
    System.out.println("Caught an exception: " + e.getMessage());
    }
    System.out.print("\nName : Mohit Barad_23DCS006 ");

    }
    }
```

OUTPUT:

```
Caught an exception: This is a custom exception

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-25

CONCLUSION:

This java code demonstrates basic exception handling. It intentionally throws a custom exception and catches it, printing the exception message. The program then prints a name and ID. This example illustrates how to use try-catch blocks to manage exceptions in Java. It ensures that even when an exception occurs, the program continues to execute subsequent statements.

| 26. | **AIM :** Write a java program to generate user defined exception using "throw" and "throws" keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program). |

PROGRAM CODE :

```java
import java.io.IOException;
import java.sql.SQLException;

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

class CustomUncheckedException extends RuntimeException {
public CustomUncheckedException(String message) {
super(message);
}
}

public class prac26 {

public static void methodThrowingCheckedException() throws CustomCheckedException {
throw new CustomCheckedException("This is a custom checked exception");
}

public static void methodThrowingUncheckedException() {
throw new CustomUncheckedException("This is a custom unchecked exception");
}

public static void methodThrowingStandardCheckedExceptions() throws IOException, SQLException {
throw new IOException("This is an IOException");
}
public static void methodThrowingStandardUncheckedExceptions() {
throw new NullPointerException("This is a NullPointerException");
}

public static void main(String[] args) {
try {
methodThrowingCheckedException();
} catch (CustomCheckedException e) {
System.out.println("Caught checked exception: " + e.getMessage());
}

try {
methodThrowingStandardCheckedExceptions();
```
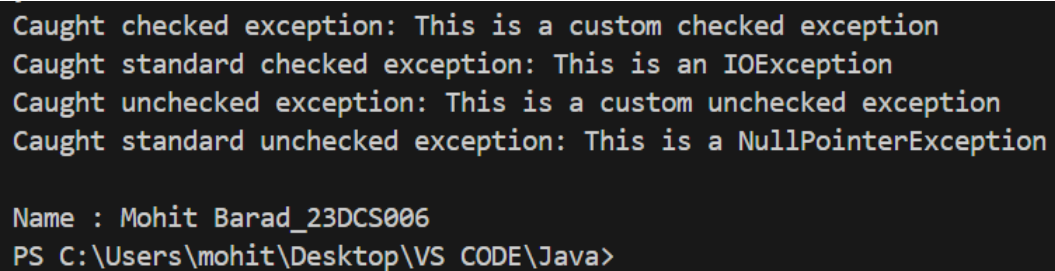
```
} catch (IOException | SQLException e) {
System.out.println("Caught standard checked exception: " + e.getMessage());
}

try {
methodThrowingUncheckedException();
} catch (CustomUncheckedException e) {
System.out.println("Caught unchecked exception: " + e.getMessage());
}

try {
methodThrowingStandardUncheckedExceptions();
} catch (NullPointerException | ArithmeticException e) {
System.out.println("Caught standard unchecked exception: " + e.getMessage());
}
System.out.print("\nName : Mohit Barad_23DCS006 ");
}
}
```

OUTPUT:

```
Caught checked exception: This is a custom checked exception
Caught standard checked exception: This is an IOException
Caught unchecked exception: This is a custom unchecked exception
Caught standard unchecked exception: This is a NullPointerException

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-26

CONCLUSION:

This Java code demonstrates handling both checked and unchecked exceptions. It uses multiple try-catch blocks to catch custom and standard exceptions, printing appropriate messages. This ensures robust error handling and program continuity. The program concludes by printing the author's name and ID. This example effectively illustrates exception management in Java.

PART-VI File Handling & Streams

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

PROGRAM CODE :

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

public class prac27 {
    public static void main(String[] args) {
        if (args.length == 0) {
            args = new String[]{"hello.txt"};
        }

        for (String fileName : args) {
            try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
                int lineCount = 0;
                while (reader.readLine() != null) {
                    lineCount++;
                }
                System.out.println(fileName + ": " + lineCount + " lines");
            } catch (IOException e) {
                System.err.println("Error reading file " + fileName + ": " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

```
hello.txt: 2 lines
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-27

CONCLUSION:

This program counts the number of lines in a file using Java. It reads each file specified in the command-line arguments or defaults to hello.txt if no arguments are provided. The program uses BufferedReader to read each line and increments a counter for each line read. It handles file reading errors gracefully using a try-with-resources block. The program prints the number of lines for each file processed. This showcases efficient file handling and error management in Java.

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

PROGRAM CODE :

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

public class prac28 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java CharCount <file> <character>");
            return;
        }

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

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
{
            int charCount = 0;
            int c;
            while ((c = reader.read()) != -1) {
                if (c == targetChar) {
                    charCount++;
                }
            }
            System.out.println("The character '" + targetChar + "' appears " + charCount
+ " times in the file " + fileName);
        } catch (IOException e) {
            System.err.println("Error reading file " + fileName + ": " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
C:\Users\mohit\Desktop\VS CODE\Java>java prac28 xanadu.txt e
The character 'e' appears 47 times in the file xanadu.txt

C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-28

CONCLUSION:

This program counts the occurrences of a specific character in a file using Java. It reads the file character by character with BufferedReader and compares each character to the target character. If they match, it increments a counter. The program handles file reading errors using a try-with-resources block to ensure the reader is closed properly. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient character processing and error management in Java.

---

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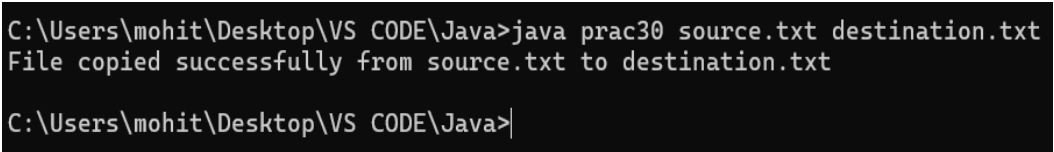
PROGRAM CODE :

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

public class prac29 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java prac29 <file> <word>");
            return;
        }

        String fileName = args[0];
        String targetWord = args[1];

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

            int wordCount = 0;
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\s+");
                for (String word : words) {
                    if (word.equals(targetWord)) {
                        wordCount++;
                    }
```

3

```
            }
          }
        System.out.println("The word '" + targetWord + "' appears " + wordCount + "
times in the file " + fileName);
      } catch (IOException e) {
        System.err.println("Error reading file " + fileName + ": " + e.getMessage());
      }

      // Wrapper Class Example
      Integer wrapperInt = Integer.valueOf(10); // Using Integer wrapper class
      int primitiveInt = wrapperInt.intValue(); // Converting back to primitive int
      System.out.println("Wrapper Class Example: Integer value is " + wrapperInt + "
and primitive int value is " + primitiveInt);
    }
}
```

OUTPUT:

```
C:\Users\mohit\Desktop\VS CODE\Java>java prac29 xanadu.txt is
The word 'is' appears 3 times in the file xanadu.txt
Wrapper Class Example: Integer value is 10 and primitive int value is 10

C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-29

CONCLUSION:

This program demonstrates how to count the occurrences of a specific word in a file
using Java. It reads the file line by line with BufferedReader and splits each line into
words. It then compares each word to the target word and increments a counter if they
match. The program handles file reading errors gracefully using a try-with-resources
block. It also provides usage instructions if the required command-line arguments are
not provided. This showcases efficient text processing and error management in Java.

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

PROGRAM CODE :

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class prac30 {
  public static void main(String[] args) {
    if (args.length != 2) {
      System.out.println("Usage: java prac30 <source file> <destination file>");
      return;
    }
```

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

        try (FileInputStream fis = new FileInputStream(sourceFile);
           FileOutputStream fos = new FileOutputStream(destinationFile)) {

           byte[] buffer = new byte[1024];
           int bytesRead;

           while ((bytesRead = fis.read(buffer)) != -1) {
              fos.write(buffer, 0, bytesRead);
           }

            System.out.println("File copied successfully from " + sourceFile + " to " +
destinationFile);
        } catch (IOException e) {
           System.err.println("Error copying file: " + e.getMessage());
        }
     }
}
```

OUTPUT:

```
C:\Users\mohit\Desktop\VS CODE\Java>java prac30 source.txt destination.txt
File copied successfully from source.txt to destination.txt

C:\Users\mohit\Desktop\VS CODE\Java>
```
OUTPUT: PRACTICAL-30

CONCLUSION:

This program demonstrates how to copy data from one file to another using byte streams in Java. It reads from a source file and writes to a destination file, creating the destination file if it does not exist. The program uses FileInputStream to read bytes and FileOutputStream to write bytes. It handles errors using a try-with-resources block to ensure streams are closed properly. The program also provides usage instructions if the required command-line arguments are not provided. This showcases efficient file handling and error management in Java.

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

PROGRAM CODE :

import java.io.*;

class prac31 {
    public static void main(String[] args) {

```java
    // Demonstrate character stream
    try (FileReader fr = new FileReader("input.txt");
      FileWriter fw = new FileWriter("output_char.txt")) {
      int c;
      while ((c = fr.read()) != -1) {
        fw.write(c);
      }
      System.out.println("Character stream copy completed.");
    } catch (IOException e) {
      System.err.println("Error with character stream: " + e.getMessage());
    }

    // Demonstrate byte stream
    try (FileInputStream fis = new FileInputStream("input.txt");
      FileOutputStream fos = new FileOutputStream("output_byte.txt")) {
      byte[] buffer = new byte[1024];
      int bytesRead;
      while ((bytesRead = fis.read(buffer)) != -1) {
        fos.write(buffer, 0, bytesRead);
      }
      System.out.println("Byte stream copy completed.");
    } catch (IOException e) {
      System.err.println("Error with byte stream: " + e.getMessage());
    }

    // Use BufferedReader and BufferedWriter to read from console and write to a
file
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
             BufferedWriter bw = new BufferedWriter(new
FileWriter("console_output.txt"))) {
      System.out.println("Enter text (type 'exit' to finish):");
      String line;
      while (!(line = br.readLine()).equalsIgnoreCase("exit")) {
        bw.write(line);
        bw.newLine();
      }
      System.out.println("Console input written to file.");
    } catch (IOException e) {
            System.err.println("Error with BufferedReader/BufferedWriter: " +
e.getMessage());
    }
  }
}
```
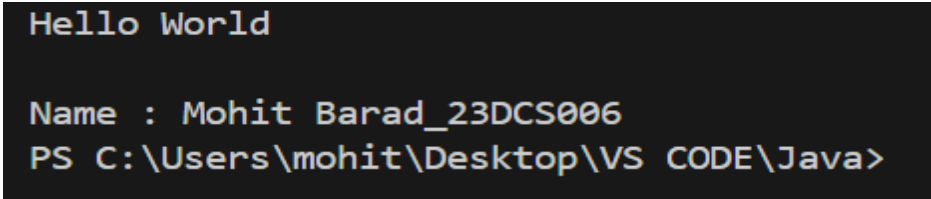
OUTPUT:

```
Character stream copy completed.
Byte stream copy completed.
Enter text (type 'exit' to finish):
mohit
exit
Console input written to file.
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-31

CONCLUSION:

This program demonstrates the use of character and byte streams in Java. It reads from input.txt and writes to output_char.txt using character streams, and to output_byte.txt using byte streams. Additionally, it uses BufferedReader to read console input and BufferedWriter to write the input to console_output.txt. The program continues to read from the console until the user types "exit". This showcases efficient file handling and console interaction in Java.

PART-VII Multithreading

No.	Aim of the Practical
32.	**AIM :** Write a program to create thread which display "Hello World" message. A. by extending Thread class B. by using Runnable interface.

PROGRAM CODE :

```
// using Thread class
class prac32 extends Thread {
    public void run() {
        System.out.println("Hello World");
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }

    public static void main(String[] args) {
        prac32 thread = new prac32();
        thread.start();

    }
}

// using Runnable interface
class prac32 implements Runnable {
    public void run() {
        System.out.println("Hello World");
        System.out.print("\nName : Mohit Barad_23DCS006 ");

    }

    public static void main(String[] args) {
        Thread thread = new Thread(new prac32());
        thread.start();
    }
}
```

OUTPUT:

```
Hello World

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-32

This code demonstrates two methods of creating threads in Java: by extending the Thread class and by implementing the Runnable interface. The run method prints a "Hello World" message along with the author's name and ID. The first approach is implemented and executed, while the second approach is commented out. Both methods achieve the same functionality but offer different ways to manage thread behavior and inheritance.

33.

AIM : Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.

PROGRAM CODE :

```java
class SumThread extends Thread {
    int start;
    int end;
    int[] numbers;
    int partialSum;

    public SumThread(int start, int end, int[] numbers) {
        this.start = start;
        this.end = end;
        this.numbers = numbers;
    }

    public void run() {
        partialSum = 0;
        for (int i = start; i < end; i++) {
            partialSum += numbers[i];
        }
    }

    public int getPartialSum() {
        return partialSum;
    }
}
public class prac33 {
    public static void main(String[] args) {
        int N = 4;
        int n = 2;

        if (N < n) {
            System.out.println("N should be greater than or equal to n.");
            return;
        }

        int[] numbers = new int[N];
```

```java
        for (int i = 0; i < N; i++) {
            numbers[i] = i + 1;
        }

        SumThread[] threads = new SumThread[n];
        int chunkSize = N / n;
        int remainder = N % n;

        int start = 0;
        for (int i = 0; i < n; i++) {
            int end = start + chunkSize + (i < remainder ? 1 : 0);
            threads[i] = new SumThread(start, end, numbers);
            threads[i].start();
            start = end;
        }

        int totalSum = 0;
        try {
            for (SumThread thread : threads) {
                thread.join();
                totalSum += thread.getPartialSum();
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("The total sum is: " + totalSum);
        System.out.print("\nName : Mohit Barad_23DCS006 ");
    }
}
```
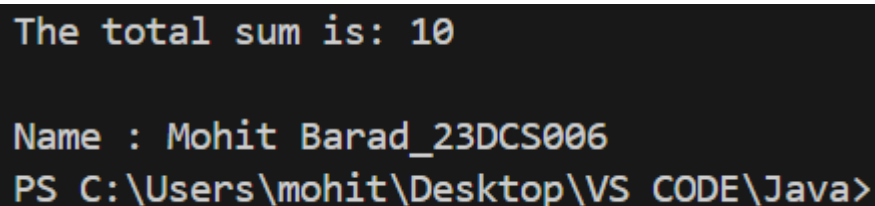
OUTPUT:

```
The total sum is: 10

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-33

CONCLUSION:

This code divides an array into chunks and processes each chunk in parallel using multiple threads to calculate the sum. Each thread computes a partial sum, which is then combined to get the total sum. The join() method ensures that the main thread

waits for all threads to complete before calculating the final result. The program concludes by printing the total sum along with the author's name and ID.

34.	**AIM :** 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 thevalue of cube of the number.

PROGRAM CODE :

```java
class Square extends Thread {
   int number;

   public void setNumber(int number) {
     this.number = number;
   }

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

class Cube extends Thread {
  int number;

   public void setNumber(int number) {
     this.number = number;
   }

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

public class prac34{
   public static void main(String[] args) {
     Square square = new Square();
     Cube cube = new Cube();

     for (int i = 1; i <= 10; i++) {
       System.out.println("Checking Number: " + i);
       if (i % 2 == 0) {
         square.setNumber(i);
         square.run();
       } else {
         cube.setNumber(i);
```

```
        cube.run();
      }
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
  System.out.print("\nName : Mohit Barad_23DCS006 ");
 }
}
```

OUTPUT:

```
Checking Number: 1
Cube of 1 is 1
Checking Number: 2
Square of 2 is 4
Checking Number: 3
Cube of 3 is 27
Checking Number: 4
Square of 4 is 16
Checking Number: 5
Cube of 5 is 125
Checking Number: 6
Square of 6 is 36
Checking Number: 7
Cube of 7 is 343
Checking Number: 8
Square of 8 is 64
Checking Number: 9
Cube of 9 is 729
Checking Number: 10
Square of 10 is 100

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS
```

OUTPUT: PRACTICAL-34

CONCLUSION:

This code alternates between calculating the square and cube of numbers from 1 to 10 using two separate classes, Square and Cube. It runs the appropriate calculation based on whether the number is even or odd, with a one-second delay between each calculation. The program demonstrates basic thread usage and control flow, although it directly calls the run() method instead of starting a new thread. It concludes by printing the author's name and ID.

35.

AIM : Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.

PROGRAM CODE :

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

class IncrementThread extends Thread {
   private int value = 0;

   public void run() {
      try {
         System.out.println("Value before increment: " + value);
         Thread.sleep(1000);
         value++;
         System.out.println("Value after increment: " + value);
         System.out.print("\nName : Mohit Barad_23DCS006 ");
      } catch (InterruptedException e) {
         System.out.println("Thread interrupted: " + e.getMessage());
      }
   }
}
```

OUTPUT:

```
Value before increment: 0
Value after increment: 1

Name : Mohit Barad_23DCS006
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-35

CONCLUSION:

This code demonstrates creating a thread by extending the Thread class to increment a variable's value after a one-second delay. The IncrementThread class overrides the run() method to perform the increment operation and print the value before and after the increment. The main method starts the thread, showcasing basic thread usage and synchronization with Thread.sleep(). The program concludes by printing the author's name and ID.

---

36. **AIM :** Write a program to create three threads 'FIRST', 'SECOND', 'THIRD'. Set the priority of the 'FIRST' thread to 3, the 'SECOND' thread to 5(default) and the 'THIRD' thread to 7.

PROGRAM CODE :

```java
class FirstThread extends Thread {
    public FirstThread() {
        super("FIRST");
    }

    public void run() {
        System.out.println("Thread FirstThread with priority " + getPriority() + " is running.");
    }
}
class SecondThread extends Thread {
    public SecondThread() {
        super("SECOND");
    }
    public void run() {
        System.out.println("Thread SecondThread with priority " + getPriority() + " is running.");
    }
}
class ThirdThread extends Thread {
```

```java
   public ThirdThread() {
      super("THIRD");
   }

   public void run() {
       System.out.println("Thread ThirdThread with priority " + getPriority() + " is
running.");
   }
}

public class prac36 {
   public static void main(String[] args) {
      System.out.print("\nName : Mohit Barad_23DCD006\n");
      FirstThread first = new FirstThread();
      SecondThread second = new SecondThread();
      ThirdThread third = new ThirdThread();

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

      first.start();
      second.start();
      third.start();

   }
}
```

OUTPUT:


```
Name : Mohit Barad_23DCD006
Thread ThirdThread with priority 7 is running.
Thread FirstThread with priority 3 is running.
Thread SecondThread with priority 5 is running.
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-36

CONCLUSION:

This code demonstrates creating and running three threads with different priorities.
Each thread prints a message indicating its priority when it runs. The Prac_36 class
sets the priorities for the threads and starts them. The program concludes by printing
the author's name and ID.

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

PROGRAM CODE :

```
class prac37 {
    public static void main(String[] args) {
        System.out.print("\nName : Mohit Barad_23DCS006\n");
        Buffer buffer = new Buffer();
        Thread producerThread = new Thread(new Producer(buffer));
        Thread consumerThread = new Thread(new Consumer(buffer));

        producerThread.start();
        consumerThread.start();
    }
}

class Buffer {
    private int data;
    private boolean isEmpty = true;

    public synchronized void produce(int value) throws InterruptedException {
        while (!isEmpty) {
            wait();
        }
        data = value;
        isEmpty = false;
        System.out.println("Produced: " + data);
        notify();
    }

    public synchronized int consume() throws InterruptedException {
        while (isEmpty) {
            wait();
        }
        isEmpty = true;
        System.out.println("Consumed: " + data);
        notify();
        return data;
    }
}

class Producer implements Runnable {
    private Buffer buffer;

    public Producer(Buffer buffer) {
        this.buffer = buffer;
    }
```
|

```java
      public void run() {
         for (int i = 0; i < 10; i++) {
            try {
               buffer.produce(i);
               Thread.sleep(500);
            } catch (InterruptedException e) {
               Thread.currentThread().interrupt();
            }
         }
      }
}

class Consumer implements Runnable {
   private Buffer buffer;

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

   public void run() {
      for (int i = 0; i < 10; i++) {
         try {
            buffer.consume();
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
         }
      }
   }
}
```

OUTPUT:

```
Name : Mohit Barad_23DCS006
Produced: 0
Consumed: 0
Produced: 1
Consumed: 1
Produced: 2
Consumed: 2
Produced: 3
Consumed: 3
Produced: 4
Consumed: 4
Produced: 5
Consumed: 5
Produced: 6
Consumed: 6
Produced: 7
Consumed: 7
Produced: 8
Consumed: 8
Produced: 9
Consumed: 9
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-37

CONCLUSION:

This code implements the producer-consumer problem using thread synchronization with a shared buffer. The Buffer class uses wait() and notify() to manage access between the Producer and Consumer threads. The Producer thread adds data to the buffer, while the Consumer thread removes data from it. The program demonstrates effective use of thread synchronization to coordinate the actions of multiple threads.
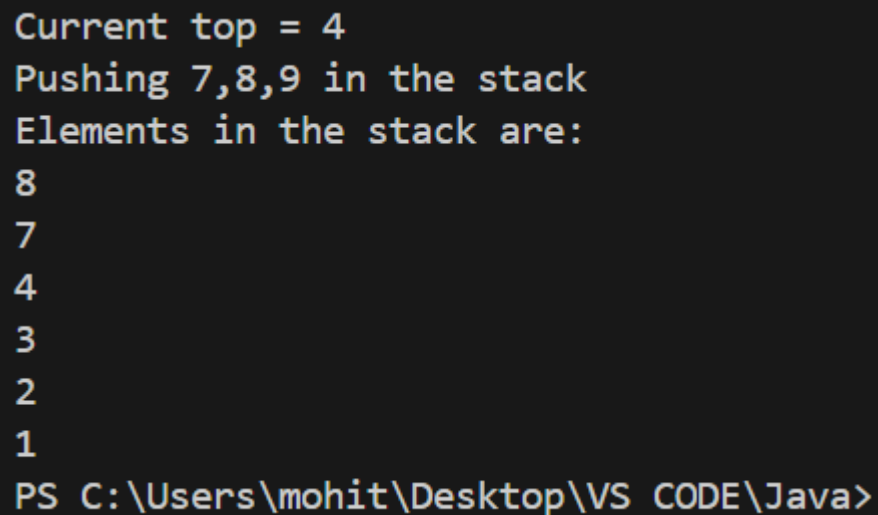
PART-VIII Collection Framework and Generic

No.	Aim of the Practical
38.	**AIM** : Design a Custom Stack using ArrayList class, which implements following functionalities of stack. My Stack -list ArrayList<Object>: A list to store elements. +isEmpty: boolean: Returns true if this stack is empty.

+getSize(): int: Returns number of elements in this stack.

+peek(): Object: Returns top element in this stack without removing it.

+pop(): Object: Returns and Removes the top elements in this stack.

+push(o: object): Adds new element to the top of this stack.

PROGRAM CODE :

```java
import java.util.*;

class MyStack {
    ArrayList<Object> list;

    MyStack(Object elements[]) {
        list = new ArrayList<Object>();
        for (int i = 0; i < elements.length; i++) {
            list.add(elements[i]);
        }
    }

    MyStack() {
        list = new ArrayList<Object>();
    }

    boolean isEmpty() {
        return (list.size() == 0);
    }

    Object peek() {
        return list.get(list.size() - 1);
    }

    Object pop() {
        Object ob = list.get(list.size() - 1);
        list.remove(list.size() - 1);
        return ob;
```

```
    }

    void push(Object o) {
      list.add(o);
    }
}

public class prac38 {
  public static void main(String[] args) {
    Integer arr[] = new Integer[] { 1, 2, 3, 4 };
    MyStack s = new MyStack(arr);
    System.out.println("Current top = " + s.peek());
    System.out.println("Pushing 7,8,9 in the stack");
    s.push(7);
    s.push(8);
    s.push(9);
    s.pop();
    System.out.println("Elements in the stack are: ");
    while (!s.isEmpty()) {
      System.out.println(s.pop());
    }
  }
}
```

OUTPUT:

```
Current top = 4
Pushing 7,8,9 in the stack
Elements in the stack are:
8
7
4
3
2
1
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-38

2

CONCLUSION:

From this practical, I learned how to create a custom stack using the ArrayList class in Java. I implemented basic stack functionalities like checking if the stack is empty, getting the size, viewing the top element, and performing push and pop operations. This exercise helped me understand how to use an ArrayList to dynamically store elements and simulate a stack structure.

39.	**AIM :** Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a generic method that can sort any array of Comparable objects. Create a generic method in Java that sorts an array of Comparable objects. This method should be versatile enough to sort arrays of different types of objects (such as products, customers, or orders) as long as they implement the Comparable interface.

PROGRAM CODE :

```java
public class prac39 {

    public static <T extends Comparable<T>> void sortArray(T[] array) {
        int n = array.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    T temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swapped = true;
                }
            }
            if (!swapped) {
                break;
            }
        }
    }

    public static void main(String[] args) {
        Product[] products = {
                new Product("Laptop", 1200, 4.5),
                new Product("Phone", 800, 4.3),
                new Product("Headphones", 150, 4.7),
                new Product("Monitor", 300, 4.4)
```

```java
        };

        sortArray(products);

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

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

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

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

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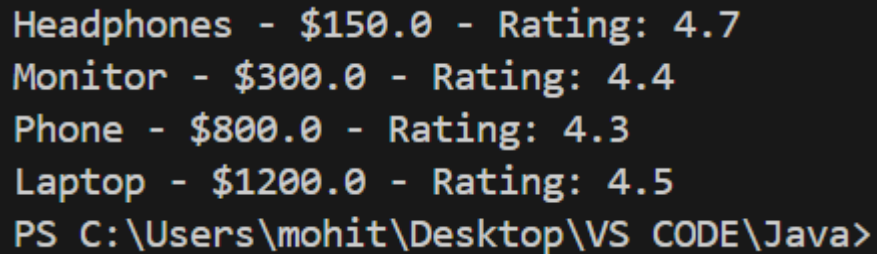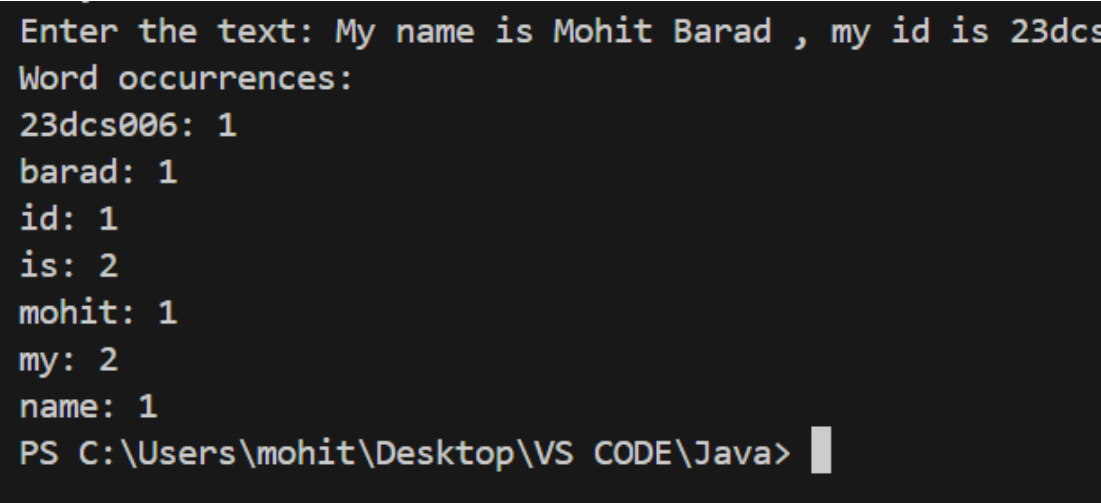
OUTPUT:

```
Headphones - $150.0 - Rating: 4.7
Monitor - $300.0 - Rating: 4.4
Phone - $800.0 - Rating: 4.3
Laptop - $1200.0 - Rating: 4.5
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-39

CONCLUSION:

Through this practical, I gained insights into implementing a generic method in Java to sort arrays of objects that implement the Comparable interface. I learned how to ensure flexibility and reusability by enabling the method to sort various types of objects, such as products, customers, and orders, based on their natural ordering.

40. **AIM :** Write a program that counts the occurrences of words in a text and displays the words and their occurrences in alphabetical order of the words. Using Map and Set Classes.

PROGRAM CODE :

```java
import java.util.*;

public class prac40 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the text: ");
        String inputText = sc.nextLine();

        inputText = inputText.toLowerCase();
        HashMap<String, Integer> wordCountMap = new HashMap<>();

        StringBuilder currentWord = new StringBuilder();

        for (int i = 0; i < inputText.length(); i++) {
            char c = inputText.charAt(i);

            if (Character.isLetter(c) || Character.isDigit(c)) {
                currentWord.append(c);
            } else {
                if (currentWord.length() > 0) {
                    String word = currentWord.toString();
                    wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
                    currentWord.setLength(0);
                }
            }
        }

        if (currentWord.length() > 0) {
            String word = currentWord.toString();
            wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
        }

        TreeSet<String> sortedWords = new TreeSet<>(wordCountMap.keySet());
```

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

        sc.close();
    }
}
```

OUTPUT:

```
Enter the text: My name is Mohit Barad , my id is 23dcs
Word occurrences:
23dcs006: 1
barad: 1
id: 1
is: 2
mohit: 1
my: 2
name: 1
PS C:\Users\mohit\Desktop\VS CODE\Java> █
```

OUTPUT: PRACTICAL-40

CONCLUSION:

In this practical, We learned how to use Java's Map and Set classes to count and display the occurrences of words in a given text. We implemented a method that not only counts the occurrences but also sorts the words in alphabetical order. This exercise enhanced my understanding of utilizing collections to efficiently manage and manipulate data.

41.

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

PROGRAM CODE :

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

public class prac41 {
    public static void main(String[] args) throws IOException {
```

6

```java
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the file name you want to scan : ");
        String f = sc.nextLine();
        File file = new File(f);
        FileReader br = new FileReader(file);
        BufferedReader fr = new BufferedReader(br);
         String keywords[] = new String[] { "abstract", "assert ", "boolean", "break",
"byte", "case", "catch", "char",
            "class",
             "continue", "default", "do", "double", "else", "enum ", "extends", "final",
"finally",
             "float", "for", "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" };
      HashSet<String> set = new HashSet<String>();
      for (int i = 0; i < keywords.length; ++i) {
         set.add(keywords[i]);
      }
      String st;
      int count = 0;
      while ((st = fr.readLine()) != null) {
         StringTokenizer str = new StringTokenizer(st, " +-/*%<>;:=&|!~()");

         while (str.hasMoreTokens()) {
            String swre = str.nextToken();
            if (set.contains(swre)) {
               count++;
            }
         }
      }
      System.out.println("Total keywords are : " + count);
      fr.close();
      sc.close();
   }
}
```

OUTPUT:

```
Enter the file name you want to scan : prac40.java
Total keywords are : 18
PS C:\Users\mohit\Desktop\VS CODE\Java>
```

OUTPUT: PRACTICAL-41

CONCLUSION:

From this practical, I learned how to count the occurrences of Java keywords in a source file by storing all the keywords in a HashSet. By using the contains() method, I was able to check whether a word is a keyword or not. This practical improved my skills in working with Java's collection framework, particularly using sets for fast lookups.