

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY****DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH****Department of Computer Science & Engineering****Subject Name: JAVA PROGRAMMING****Semester: 3****Subject Code: CSE201****Academic year:2024-25****PART-I Data Types, Variables, String, Control Statements, Operators, Arrays**

No.	Aim of the Practical
1.	<p>Demonstration of installation steps of Java,Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.</p> <div data-bbox="337 989 1032 1440"> <pre> graph TD     subgraph JDK [Java Development Kit]         subgraph JRE [Java Runtime Environment JRE]             subgraph JVM_Box [ ]                 direction TB                 JVM[Java Virtual Machine JVM]                 Lib[Library classes]             end             JVM_Box --- Plus1[+]             Lib --- Plus2[+]         end         Dev[Dev tools]     end </pre> <p>The diagram illustrates the components of the Java Development Kit (JDK). The JDK is represented by a large orange rectangle. Inside it is the Java Runtime Environment (JRE), shown as a green rectangle. Within the JRE, there is a pink rectangle containing the Java Virtual Machine (JVM) and Library classes, separated by a plus sign. To the right of the JRE, within the JDK, are Dev tools, also separated by a plus sign.</p> </div> <p>1.JDK (Java Development Kit) is a Kit that provides the environment to develop and execute(run) the Java program. JDK is a kit (or package) that includes two things</p> <ul style="list-style-type: none"> <li>• Development Tools (to provide an environment to develop your java programs)</li> <li>• JRE (to execute your java program).</li> </ul> <p>2. JRE (Java Runtime Environment) is an installation package that provides an environment to only run (not develop) the java program (or application) onto your</p>

machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

3. JVM (Java Virtual Machine) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an interpreter.

JVM (Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE (Java Runtime Environment).

Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

When we compile a *.java* file, *.class* files (contains byte-code) with the same class names present in *.java* file are generated by the Java compiler.

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 and documentation in a set of source 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 good and understandable document API for any java file you must write better comments for every class, method, constructor.

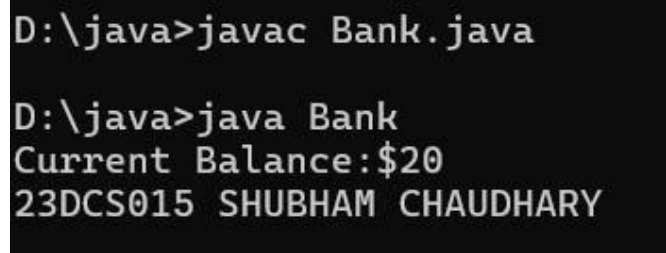
**Command Line Argument:** Java command-line argument is an argument i.e. passed at the time of running the Java program. In the command line, the arguments passed from the console can be received in the java program and they can be used as input. The users can pass the arguments during the execution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

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

**PROGRAM CODE:**

```
import java.util.*;
class Bank{
public static void main(String args[]){
int a=20;
System.out.println("Current Balance:$"+a);
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}
```

**OUTPUT:**



```
D:\java>javac Bank.java
D:\java>java Bank
Current Balance:$20
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

This program demonstrates the basic concept of storing and displaying data in Java, showcasing the use of variables and output statements.

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

**PROGRAM CODE:**

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

```
System.out.print("enter Distance in meter:");
int d=sc.nextInt();
System.out.println("enter the time:");
System.out.print("Enter Hour:");
int hr=sc.nextInt();
System.out.print("Enter minutes:");
int min=sc.nextInt();
System.out.print("Enter seconds:");
int sec=sc.nextInt();
int time=(hr*3600)+(min*60)+sec;
float v=(float)d/time;
System.out.println("Speed in m/s:"+v);
time=hr+(min/60)+(sec/3600);
v=(float)(d*0.001)/time;
System.out.println("Speed in km/h:"+v);
v=(float)(d/1609)/time;
System.out.println("speed in mil/h:"+v);
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}
```

**OUTPUT:**

```
D:\java>javac Distance.java

D:\java>java Distance
enter Distance in meter:100
enter the time:
Enter Hour:1
Enter minutes:2
Enter seconds:60
Speed in m/s:0.026455026
Speed in km/h:0.1
speed in mil/h:0.0
23DCS015 SHUBHAM CHAUDHARY
```

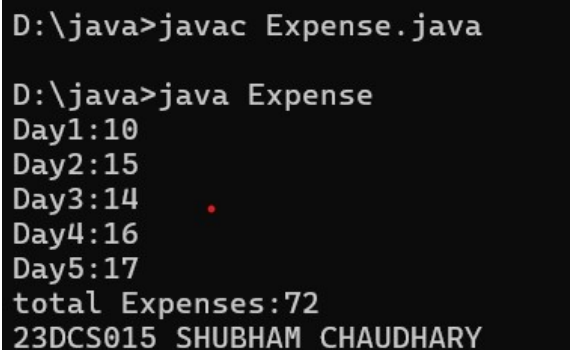
**CONCLUSION:**

This program demonstrates the ability to take user input, perform calculations, and display results in different units, showcasing fundamental programming concepts and unit conversions.

4. Imagine you are developing a budget tracking application. You need to calculate the total expenses for the month. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to calculate the sum of elements in an array representing daily expenses.

**PROGRAM CODE:**

```
import java.util.*;
class Expense{
public static void main(String args[]){
int day[]=new int[30];
Scanner sc=new Scanner(System.in);
int sum=0;
for(int i=0;i<5;i++){
System.out.print("Day"+(i+1)+":");
int a=sc.nextInt();
sum+=a;
}
System.out.println("total Expenses:"+sum);
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}
```

**OUTPUT:**A screenshot of a terminal window showing the compilation and execution of a Java program. The first line shows the compilation command: 'D:\java>javac Expense.java'. The second line shows the execution command: 'D:\java>java Expense'. The output consists of five prompts for daily expenses: 'Day1:10', 'Day2:15', 'Day3:14', 'Day4:16', and 'Day5:17'. Each prompt is followed by a user input. After the fifth input, the program outputs 'total Expenses:72' and '23DCS015 SHUBHAM CHAUDHARY' on separate lines. A small red cursor is visible after the input '14' for Day3.

```
D:\java>javac Expense.java
D:\java>java Expense
Day1:10
Day2:15
Day3:14
Day4:16
Day5:17
total Expenses:72
23DCS015 SHUBHAM CHAUDHARY
```

	<p><b><u>CONCLUSION:</u></b></p> <p>This program demonstrates the use of arrays and loops to collect and calculate the sum of daily expenses, providing a simple yet effective way to track monthly expenditures.</p>
5.	<p>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.</p> <p><b><u>PROGRAM CODE:</u></b></p> <pre> class Bill{     public static void main(String[] args) {          int[] codes = {1, 2, 3, 4, 5};         double[] prices = {1000, 500, 200, 150, 800};          double totalBill = 0;         for (int i = 0; i &lt; codes.length; i++) {             totalBill += calculateBill(codes[i], prices[i]);         }          System.out.println("Total Bill: " + totalBill);         System.out.println("23DCS015 SHUBHAM CHAUDHARY");      }      public static double calculateBill(int code, double price) {         double bill = 0;         switch (code) {             case 1:                 bill = price + (price * 0.08); // 8% tax for motor                 break;             case 2:                 bill = price + (price * 0.12); // 12% tax for fan                 break; </pre>

```

        case 3:
            bill = price + (price * 0.05); // 5% tax for tube light
            break;
        case 4:
            bill = price + (price * 0.075); // 7.5% tax for wires
            break;
        default:
            bill = price + (price * 0.03); // 3% tax for all other items
            break;
    }
    return bill;

}
}

```

**OUTPUT:**

```

D:\java>javac Bill.java

D:\java>java Bill
Total Bill: 2835.25
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

This program demonstrates the use of a switch statement to apply different tax rates based on product codes, calculating the total bill by iterating through the arrays of codes and prices. It showcases a practical application of conditional statements in Java programming.

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

**PROGRAM CODE:**

```

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

```

```

int n;
System.out.print("enter the number of day:");
n=sc.nextInt();
int n1=0,n2=1,n3;
System.out.print(n1+" "+n2);
for(int i=2;i<n;++i){
    n3=n1+n2;
    System.out.print(" "+n3);
    n1=n2;
    n2=n3;
}
System.out.println(" ");
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}

```

**OUTPUT:**

```

D:\java>javac Exercise.java

D:\java>java Exercise
enter the number of day:5
0 1 1 2 3
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

This program demonstrates the use of a loop to generate the Fibonacci series, calculating the exercise duration for each day based on the user-input number of days. It showcases a practical application of mathematical concepts in Java programming, providing a fun and interactive way to generate an exercise routine.

**PART-II Strings**

- 7 Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;
- front\_times('Chocolate', 2) → 'ChoCho'
  - front\_times('Chocolate', 3) → 'ChoChoCho'
  - front\_times('Abc', 3) → 'AbcAbcAbc'

**PROGRAM CODE:**



```
public class String1 {  
    public static void main(String[] args) {  
        System.out.println(frontTimes("Chocolate", 2));  
        System.out.println(frontTimes("Chocolate", 3));  
        System.out.println(frontTimes("Abc", 3));  
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");  
    }  
    public static String frontTimes(String str, int n) {  
        String front;  
        if (str.length() >= 3) {  
            front = str.substring(0, 3);  
        } else {  
            front = str;  
        }  
        String result = "";  
        for (int i = 0; i < n; i++) {  
            result += front;  
        }  
        return result;  
    }  
}
```

**OUTPUT:**

```
D:\java>javac String1.java  
  
D:\java>java String1  
ChoCho  
ChoChoCho  
AbcAbcAbc  
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

To solve the problem of generating n copies of the front part of a string

in Java, use the substring function to extract the first three characters (or the entire string if it's shorter). Then, repeat this segment n times. This method ensures correct handling even if the string is shorter than three characters. The solution is efficient, using simple string slicing and repetition.

- 8 Given an array of ints, return the number of 9's in the
- array. array\_count9([1, 2, 9]) → 1  
array\_count9([1, 9, 9]) → 2  
array\_count9([1, 9, 9, 3, 9]) → 3

**PROGRAM CODE:**

```
public class String2 {  
    public static int count9(int[] arr){  
        int c=0;  
        for(int i=0;i<arr.length;i++)  
        {  
            if(arr[i]==9) {  
                c++;  
            }  
        }  
        return c;  
    }  
    public static void main(String[] args)  
    {  
        int[] arr1 = {1,9,2,5};  
        int p = count9(arr1);  
        System.out.println(p);  
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");  
    }  
}
```

**OUTPUT:**

```
D:\java>javac String2.java

D:\java>java String2
1
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

In this practical we learn how to use function argument, use the count variable And print number of int in the array is same.

- 9 Given a string, return a string where for every char in the original, there are two chars.  
double\_char('The') → 'TThhee'  
double\_char('AAAbb') → 'AAAAbbbb'  
double\_char('Hi-There') → 'HHii--TThheerree'

**PROGRAM CODE:**

```
public class String3
{
    static String double_char(String s)
    {
        String s2 = new String();
        int n = s.length();
        for(int i = 0; i < n; i++)
        {
            s2 += s.substring(i, i+1) + s.substring(i, i+1);
        }
        return s2;
    }
    public static void main(String[] args)
    {
        String s = "There";
        System.out.println(s);
        String s2 = new String();
        s2 = double_char(s);
        System.out.println(s2);
    }
}
```

```
System.out.println("23DCS015 SHUBHAM CHAUDHARY");  
}  
}
```

**OUTPUT:**

```
D:\java>javac String3.java  
  
D:\java>java String3  
There  
TThheerree  
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

To double every character in a string in Java, iterate through each character, appending it twice to a new string. This method ensures each character is duplicated, creating a new string with repeated characters. It provides an efficient and straightforward solution for string manipulation tasks.

10 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:**

```
public class StringMethod4 {  
    public static void main(String[] args) {  
        String str = "charusat university";  
        int length = str.length();  
        System.out.println("Length of the string: " + length);  
  
        String lowerCaseStr = str.toLowerCase();  
        System.out.println("Lowercase of the string: " + lowerCaseStr);  
    }  
}
```

```

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

String reverseStr = new StringBuilder(str).reverse().toString();
System.out.println("Reversed string: " + reverseStr);
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}

```

**OUTPUT:**

```

D:\java>javac StringMethod4.java

D:\java>java StringMethod4
Length of the string: 19
Lowercase of the string: charusat university
Uppercase of the string: CHARUSAT UNIVERSITY
Reversed string: ytisrevinu tasurahc
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

In this practical we learn how to use different type of function use like, how to find length using length(), toLowerCase(), toUpperCase(), reversed(), and short the String and display the output.

11. Perform following Functionalities of the string:  
 “CHARUSAT UNIVERSITY”
- Find length
  - Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’
  - Convert all character in lowercase

**PROGRAM CODE:**

```

public class String5 {
    public static void main(String[] args) {
        String str = "CHARUSAT UNIVERSITY";

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

```

```
char firstLetterOfYourName = 'S';
String replacedStr = str.replace('H', firstLetterOfYourName);
System.out.println("String after replacing 'H' with '" +
firstLetterOfYourName + "': " + replacedStr);

String lowerCaseStr = str.toLowerCase();
System.out.println("Lowercase of the string: " + lowerCaseStr);
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}
```

**OUTPUT:**

```
D:\java>javac String5.java

D:\java>java String5
Length of the string: 19
String after replacing 'H' with 'S': CSARUSAT UNIVERSITY
Lowercase of the string: charusat university
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

This code finds the length of the string, replaces 'H' with 'your name first latter', and converts all characters to lowercase, printing the results.

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

- 12.** Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.

**PROGRAM CODE:**

```
import java.util.*;

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

```

        if (args.length == 1) {
            // Command-line argument
            double amount = Double.parseDouble(args[0]);
            double rupees = amount*100;
            System.out.println(amount + " Pounds = " + rupees + " Rupees");
        }
        else {
            // Interactive mode
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter amount in Pounds: ");
            double amount = scanner.nextDouble();
            double rupees = amount*100;
            System.out.println(amount + " Pounds = " + rupees + " Rupees");
        }
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

**OUTPUT:**

```

D:\java>javac Pra.java

D:\java>java Pra
Enter amount in Pounds: 15
15.0 Pounds = 1500.0 Rupees
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

The Java program converts Pounds to Rupees at a fixed rate of 1 Pound = 100 Rupees, handling both command-line arguments and interactive user inputs effectively, ensuring user-friendly currency conversion.

- 13.** Create a class called Employee that includes three pieces of Information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise and display each Employee's yearly salary again.

**PROGRAM CODE:**

```
import java.util.*;
class Employee{
    Scanner sc=new Scanner(System.in);

    String fs=" ";
    String ls=" ";
    double sal;
    Employee(String f,String l,double s){
        fs=f;
        ls=l;
        sal=s;
    }
    void setfs(){
        fs=sc.nextLine();
    }

    void setls(){
        ls=sc.nextLine();
    }

    void setsal(){
        sal=sc.nextDouble();
        if(sal<0){
            sal=0.0;
        }
        else{
            sal=sal+(sal*0.10);
        }
    }

    String getfs(){
        return fs;
    }
    String getls(){
        return ls;
    }
    double getsal(){
        return sal;
    }
}
class EmpT{
    public static void main(String args[]){
        Employee E1=new Employee("", "",0.0);
        System.out.println("Enter First name:");
        E1.setfs();
        System.out.println("enter last name:");
        E1.setls();
        System.out.println("enter salary:");
        E1.setsal();
    }
}
```

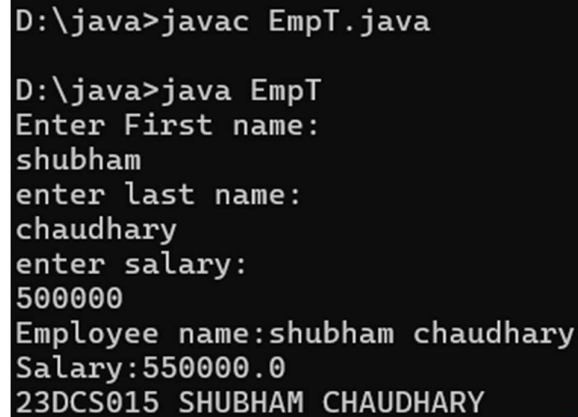


```

System.out.println("Employee name:"+E1.getfs()+" "+E1.getls());
System.out.println("Salary:"+E1.getsal());
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}

```

### **OUTPUT:**



```

D:\java>javac EmpT.java

D:\java>java EmpT
Enter First name:
shubham
enter last name:
chaudhary
enter salary:
500000
Employee name:shubham chaudhary
Salary:550000.0
23DCS015 SHUBHAM CHAUDHARY

```

### **CONCLUSION:**

The Employee class initializes and updates employee data through setters and getters. The EmpT class demonstrates capturing employee details and adjusting salaries, showcasing the class's functionality effectively.

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

### **PROGRAM CODE:**

```

import java.util.*;
class Date{
    Scanner sc=new Scanner(System.in);

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

```

```
m=month;
y=year;
}
void setd(){
    d=sc.nextInt();
    if(d>31){
        System.out.println("enter right day");
    }
}

void setm(){
    m=sc.nextInt();
    if(m>12){
        System.out.println("enter right month");
    }
}

void sety(){
    y=sc.nextInt();
}

int getd(){
    return d;
}
int getlm(){
    return m;
}
int gety(){
    return y;
}
void displayDate(){

    System.out.println("\nDate:"+d+"/"+m+"/"+y);
}
}

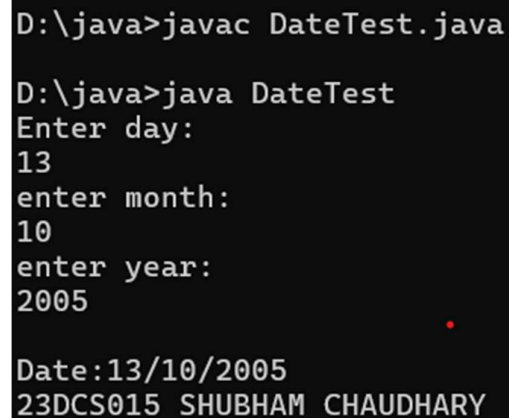
class DateTest{
    public static void main(String args[]){
        Date D=new Date(0,0,0);
        System.out.println("Enter day:");
        D.setd();
        System.out.println("enter month:");
        D.setm();
```

```

System.out.println("enter year:");
D.sety();
D.displayDate();
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
}

```

### **OUTPUT:**



```

D:\java>javac DateTest.java

D:\java>java DateTest
Enter day:
13
enter month:
10
enter year:
2005

Date:13/10/2005
23DCS015 SHUBHAM CHAUDHARY

```

### **CONCLUSION:**

The Date class successfully manages date information with proper setters, getters, and a display method. The DateTest application effectively demonstrates the class's ability to handle and display date information.

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

### **PROGRAM CODE:**

```

import java.util.*;
class Area{
int l, b;
Scanner sc=new Scanner(System.in);
Area(){
}
Area(int length, int breadth){
b=breadth;
l=length;
}
void setl(){
l=sc.nextInt();
}
}

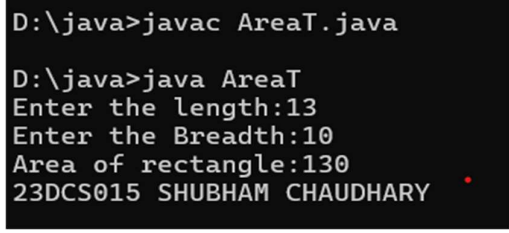
```

```

void setb() {
    b=sc.nextInt();
}
void returnArea() {
    System.out.println(l*b);
}
}
class AreaT {
    public static void main(String args[]) {
        Area a=new Area(0,0);
        System.out.print("Enter the length:");
        a.setl();
        System.out.print("Enter the Breadth:");
        a.setb();
        System.out.print("Area of rectangle:");
        a.returnArea();
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

### **OUTPUT:**



```

D:\java>javac AreaT.java

D:\java>java AreaT
Enter the length:13
Enter the Breadth:10
Area of rectangle:130
23DCS015 SHUBHAM CHAUDHARY

```

### **CONCLUSION:**

The program effectively calculates and prints the area of a rectangle using the Area class, which initializes length and breadth via constructor parameters and provides a method to return the calculated area.

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

### **PROGRAM CODE:**

```

import java.util.Scanner;

class Complex {
    private double real;
    private double imag;

    public Complex(double real, double imag) {

```

```
this.real = real;
this.imag = imag;
}
public Complex add(Complex other) {
    double real = this.real + other.real;
    double imag = this.imag + other.imag;
    return new Complex(real, imag);
}
public Complex subtract(Complex other) {
    double real = this.real - other.real;
    double imag = this.imag - other.imag;
    return new Complex(real, imag);
}
public Complex multiply(Complex other) {
    double real = this.real * other.real - this.imag * other.imag;
    double imag = this.real * other.imag + this.imag * other.real;
    return new Complex(real, imag);
}
public String toString() {
    return real + " + " + imag + "i";
}
}
public class Prac_16 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter real part of first complex number: ");
        double real1 = scanner.nextDouble();
        System.out.print("Enter imaginary part of first complex number: ");
        double imag1 = scanner.nextDouble();

        System.out.print("Enter real part of second complex number: ");
        double real2 = scanner.nextDouble();
        System.out.print("Enter imaginary part of second complex number: ");
        double imag2 = scanner.nextDouble();

        Complex num1 = new Complex(real1, imag1);
        Complex num2 = new Complex(real2, imag2);

        Complex sum = num1.add(num2);
        Complex difference = num1.subtract(num2);
        Complex product = num1.multiply(num2);
```

```

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

**OUTPUT:**

```

D:\java>javac Prac_16.java

D:\java>java Prac_16
Enter real part of first complex number: 4
Enter imaginary part of first complex number: 5
Enter real part of second complex number: 7
Enter imaginary part of second complex number: 3
Sum: 11.0 + 8.0i
Difference: -3.0 + 2.0i
Product: 13.0 + 47.0i
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

The Complex class correctly performs addition, subtraction, and multiplication of two complex numbers, with methods for each operation. User input for real and imaginary parts demonstrates the class's functionality.

**PART-IV Inheritance, Interface, Package**

17. Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent.

**PROGRAM CODE:**

```

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

class Child extends Parent {
    void printChildMessage() {

```

```

        System.out.println("This is child class");
    }
}

public class Prac17 {
    public static void main(String[] args) {

        Parent parentObj = new Parent();
        parentObj.printMessage();

        Child childObj = new Child();
        childObj.printChildMessage();
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

**OUTPUT:**

```

D:\java>javac Prac17.java

D:\java>java Prac17
This is parent class
This is child class
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

The program demonstrates inheritance with a 'child' class extending the 'P17' class. It creates an object of the 'child' class and calls its specific method while also showing the parent class message via the constructor, illustrating basic class inheritance.

- 18.** Create a class named 'Member' having the following members: Data members
- 1 - Name
  - 2 - Age
  - 3 - Phone number
  - 4 - Address
  - 5 - Salary
- It also has a method named 'printSalary' which prints the salary of the members. Two classes 'Employee' and 'Manager' inherits the 'Member' class.

The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

**PROGRAM CODE:**

```
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) {

        Employee employee = new Employee();
        employee.name = "shubham chaudhary";
        employee.age = 18;
        employee.phoneNumber = "6356859545";
        employee.address = "modasa";
        employee.salary = 500000;
        employee.specialization = "farmer";
    }
}
```



```
Manager manager = new Manager();
manager.name = "rajubhai patel";
manager.age = 46;
manager.phoneNumber = "9265487512";
manager.address = "navi shinol";
manager.salary = 800000;
manager.department = "farming";

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);
System.out.println("Specialization: " + employee.specialization);
employee.printSalary();

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);
System.out.println("Department: " + manager.department);
manager.printSalary();
System.out.println("23DCS015 SHUBHAM CHAUDHARY");
}
```

**OUTPUT:**

```

D:\java>javac Prac18.java

D:\java>java Prac18
Employee Details:
Name: shubham chaudhary
Age: 18
Phone Number: 6356859545
Address: modasa
Specialization: farmer
Salary: 500000.0

Manager Details:
Name: rajubhai patel
Age: 46
Phone Number: 9265487512
Address: navi shinol
Department: farming
Salary: 800000.0
23DCS015 SHUBHAM CHAUDHARY

```

### **CONCLUSION:**

The program showcases inheritance by having 'Employee' and 'Manager' classes extend the 'P18' class. It collects and prints details using polymorphism, with specific attributes for each subclass, demonstrating data handling and method overriding in Java.

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

### **PROGRAM CODE:**

```

class Rectangle {

    protected double length;
    protected double breadth;

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

```

```
        this.breadth = breadth;
    }

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

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

class Square extends Rectangle {

    public Square(double side) {
        super(side, side);
    }
}

public class Prac19 {
    public static void main(String[] args) {

        Rectangle[] shapes = new Rectangle[2];
        shapes[0] = new Rectangle(1, 3);
        shapes[1] = new Square(4);

        for (Rectangle shape : shapes) {
            shape.printArea();
            shape.printPerimeter();
            System.out.println();
            System.out.println("23DCS015 SHUBHAM CHAUDHARY");
        }
    }
}
```

**OUTPUT:**

```
D:\java>javac Prac19.java

D:\java>java Prac19
Area of Rectangle: 3.0
Perimeter of Rectangle: 8.0

23DCS015 SHUBHAM CHAUDHARY
Area of Rectangle: 16.0
Perimeter of Rectangle: 16.0

23DCS015 SHUBHAM CHAUDHARY
```

### **CONCLUSION:**

The program demonstrates inheritance, where the 'Square' class inherits from the 'Rectangle' class. It also uses constructors to initialize values and an array of objects to print the area and perimeter, showcasing polymorphism and object-oriented principles.

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

### **PROGRAM CODE:**

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

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

class Circle extends Shape {
    void printCircle() {
```

```

        System.out.println("This is circular shape");
    }
}
class Square extends Rectangle {
    void printSquare() {
        System.out.println("Square is a rectangle");
    }
}
public class Prac20{
    public static void main(String[] args) {
        Square square = new Square();
        square.printShape();
        square.printRectangle();
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

**OUTPUT:**

```

D:\java>javac Prac20.java

D:\java>java Prac20
This is shape
This is rectangular shape
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

The program demonstrates inheritance in Java, where the 'Square' class inherits methods from both 'Shape' and 'Rectangle'. The 'Square' object can call methods from its superclass, showcasing method inheritance and polymorphism in object-oriented programming.

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

**PROGRAM CODE:**

```
class Degree {
    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 degree = new Degree();
        Undergraduate undergraduate = new Undergraduate();
        Postgraduate postgraduate = new Postgraduate();
        degree.getDegree();
        undergraduate.getDegree();
        postgraduate.getDegree();
        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}
```

**OUTPUT:**

```
D:\java>javac Prac21.java
D:\java>java Prac21
I got a degree
I am an Undergraduate
I am a Postgraduate
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

This program demonstrates method overriding in Java, where the subclasses 'Undergraduate' and 'Postgraduate' provide their own implementations of the 'getDegree()' method. Each object calls its respective method, showcasing polymorphism and inheritance in object-oriented programming.

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

**PROGRAM CODE:**

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

class MyCalculator implements AdvancedArithmetic {

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

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

public class Prac22 {
    public static void main(String[] args) {
        MyCalculator calculator = new MyCalculator();
        int number = 5;
        int result = calculator.divisor_sum(number);
        System.out.println("The sum of divisors of " + number + " is: " + result);
    }
}
```

```

        System.out.println("23DCS015 SHUBHAM CHAUDHARY");
    }
}

```

**OUTPUT:**

```

D:\java>javac Prac22.java

D:\java>java Prac22
The sum of divisors of 5 is: 6
23DCS015 SHUBHAM CHAUDHARY

```

**CONCLUSION:**

The program demonstrates implementing the 'AdvancedArithmetic' interface with the 'calledMyCalculator' class. It efficiently calculates and returns the sum of divisors for a given integer using a loop up to its square root, optimizing performance.

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

**PROGRAM CODE:**

```

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

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

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

```



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

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

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

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

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

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

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

class Sign {
    private Shape backgroundShape;
    private String text;
```

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

public void display() {
    System.out.println("Sign Text: " + text);
    System.out.println("Background Shape Details:");
    backgroundShape.printDetails();
}
}

public class P23 {
    public static void main(String[] args) {
        Shape circle = new Circle(5, "Red");
        Shape rectangle = new Rectangle(4, 6, "Blue");

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

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

        System.out.println("\nSign 2:");
        sign2.display();

    }
}
```

**OUTPUT:**

```
D:\java>javac Prac23.java

D:\java>java Prac23
Sign 1:
Sign Text: Welcome to the Campus!
Background Shape Details:
Color: Red
Area: 78.53981633974483

Sign 2:
Sign Text: Library Entrance
Background Shape Details:
Color: Blue
Area: 24.0
23DCS015 SHUBHAM CHAUDHARY
```

**CONCLUSION:**

The program highlights the use of an interface default method, allowing shared functionality like `printDetails()` across shape classes (`Circle` and `Rectangle`). This reduces code duplication, promotes code reuse, and demonstrates backward compatibility while simplifying object design in Java.

**PART-V Exception Handling**

24. Write a java program which takes two integers x & y as input, you have to compute x/y. If x and y are not integers or if y is zero, exception will occur and you have to report it.

**PROGRAM CODE:**

```
import java.util.Scanner;

public class Prac24 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {

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

```

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

        System.out.println("The result of " + x + " / " + y + " = " + result);
    }
    catch (NumberFormatException e) {
        System.out.println("Input is not a valid integer.");
    }
    catch (ArithmeticException e) {
        System.out.println("Cannot divide by zero.");
    }
    finally {
        scanner.close();
    }
}
}

```

**OUTPUT:**

```

D:\java>javac  Prac24.java

D:\java>java  Prac24
Enter the value of x: 4
Enter the value of y: 0
Cannot divide by zero.

```

**CONCLUSION:**

The program prompts the user to input two integers and performs division. It handles invalid input by catching exceptions if the input is not a valid integer or if there's an attempt to divide by zero. The program ensures user-friendly error messages are displayed in these cases. Finally, it closes the scanner resource after execution.

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

**PROGRAM CODE:**

```

class BankAccount {
    private double balance;

    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }
}

```

```
public void withdraw(double amount) {
    if (amount > balance) {
        throw new RuntimeException("Insufficient funds for this withdrawal.");
    }
    balance -= amount;
    System.out.println("Successfully withdrew: $" + amount);
}

public double getBalance() {
    return balance;
}
}

public class P25{
    public static void main(String[] args) {
        try {

            BankAccount account = new BankAccount(500);

            System.out.println("Initial Balance: $" + account.getBalance());
            System.out.println("withdraw amount: $600");

            account.withdraw(600);
            System.out.println("Current Balance: $" + account.getBalance());

        } catch (RuntimeException e) {
            System.out.println("Error: " + e.getMessage());
        } System.out.println("Program continues after exception handling.");
    }
}
```

**OUTPUT:**

```
D:\java>javac P25.java

D:\java>java P25
Initial Balance: $500.0
withdraw amount: $600
Error: Insufficient funds for this withdrawal.
Program continues after exception handling.
```

**CONCLUSION:**

	<p>The program intentionally throws an exception using the throw keyword. The exception is caught by the catch block, which prevents the program from crashing. The caught exception's message is displayed to the user. This demonstrates how Java's try-catch mechanism handles exceptions. The program ends gracefully even when an exception occurs.</p>
26.	<p>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).</p> <p><b><u>PROGRAM CODE:</u></b></p> <pre> class InvalidAgeException extends Exception {     public InvalidAgeException(String message) {         super(message);     } }  public class Prac26 {     public static void checkAge(int age) throws InvalidAgeException {         if (age &lt; 18) {             throw new InvalidAgeException("Age is less than 18. Not eligible to vote.");         } else {             System.out.println("You are eligible to vote.");         }     }      public static void main(String[] args) {         try {              checkAge(16);         }         catch (InvalidAgeException e) {             System.out.println("Exception caught: " + e.getMessage());         }     } } </pre> <p><b><u>OUTPUT:</u></b></p> <pre> Exception caught: Age is less than 18. Not eligible to vote. PS D:\JAVA FILES\out&gt; </pre> <p><b><u>CONCLUSION:</u></b></p>

The program demonstrates a user-defined exception called `InvalidAgeException`, which is thrown when a person's age is less than 18. The `checkAge` method checks the age and either throws the exception or confirms eligibility to vote. The thrown exception is caught in the main method using a try-catch block. This shows how custom exceptions can be created and handled in Java.

## PART-VI File Handling & Streams

27. Write a program that will count the number of lines in each file that is specified on the command line. Assume that the files are text files. Note that multiple files can be specified, as in "java Line Counts file1.txt file2.txt file3.txt". Write each file name, along with the number of lines in that file, to standard output. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.

### **PROGRAM CODE:**

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

public class PRAC27 {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java LineCounter <file1> <file2> ...");
            return;
        }

        for (String filename : args) {
            try {
                int lineCount = countLines(filename);
                System.out.println(filename + ": " + lineCount + " lines");
            } catch (IOException e) {
                System.err.println("Error reading file: " + filename);
            }
        }

        private static int countLines(String filename) throws IOException {
            try (BufferedReader reader = new BufferedReader(new FileReader(filename)))
            {
                int lines = 0;
                while (reader.readLine() != null) {
```

```

        lines++;
    }
    return lines;
}
}
}

```

### **OUTPUT:**

```

D:\java>javac PRAC27.java

D:\java>java PRAC27
Usage: java LineCounter <file1> <file2> ...

```

### **CONCLUSION:**

This Java program, PRAC27, reads multiple file names from command-line arguments and counts the number of lines in each file. It uses a `BufferedReader` to read each file and handle any `IOException` that may occur. The results are printed to the console, displaying the line count for each specified file.

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

### **PROGRAM CODE:**

```

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

public class PRAC28 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage:    java    CharacterCounter    <filename>
<character>");
            return;
        }

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

        try {
            int charCount = countCharacterOccurrences(filename, targetChar);

```



```

        System.out.println("Occurrences of '" + targetChar + "' in " + filename + ": "
+ charCount);
    } catch (IOException e) {
        System.err.println("Error reading file: " + filename);
    }
}

private static int countCharacterOccurrences(String filename, char targetChar)
throws IOException {
    try (BufferedReader reader = new BufferedReader(new FileReader(filename)))
    {
        int count = 0;
        int currentChar;
        while ((currentChar = reader.read()) != -1) {
            if (currentChar == targetChar) {
                count++;
            }
        }
        return count;
    }
}
}

```

**OUTPUT:**

```

D:\java>javac PRAC28.java

D:\java>java PRAC28
Usage: java CharacterCounter <filename> <character>

```

**CONCLUSION:**

The PRAC28 Java program counts the occurrences of a specified character in a given file, using command-line arguments for the file name and target character. It reads the file character-by-character with a `BufferedReader` and handles any `IOException` that might occur. The result displays the total count of the target character in the specified file.

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

**PROGRAM CODE:**

```
import java.io.BufferedReader;
```

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

public class PRAC29 {
    public static void main(String[] args) {
        if (args.length != 2) {
            System.out.println("Usage: java WordSearch <filename> <word>");
            return;
        }

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

        try {
            boolean found = searchWord(filename, targetWord);
            if (found) {
                System.out.println("Word " + targetWord + " found in " + filename);
            } else {
                System.out.println("Word " + targetWord + " not found in " + filename);
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + filename);
        }
    }

    private static boolean searchWord(String filename, String targetWord) throws
    IOException {
        try (BufferedReader reader = new BufferedReader(new FileReader(filename)))
        {
            String line;
            while ((line = reader.readLine()) != null) {
                if (line.contains(targetWord)) {
                    return true;
                }
            }
            return false;
        }
    }
}
```

**OUTPUT:**

```
D:\java>javac PRAC29.java
```

```
D:\java>java PRAC29
```

```
Usage: java WordSearch <filename> <word>
```

**CONCLUSION:**

The PRAC29 Java program searches for a specified word in a given file using command-line arguments for the file name and target word. It reads the file line-by-line with a `BufferedReader` and checks if any line contains the word. The result indicates whether the word was found or not in the specified file.

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

**PROGRAM CODE:**

```
import java.io.*;

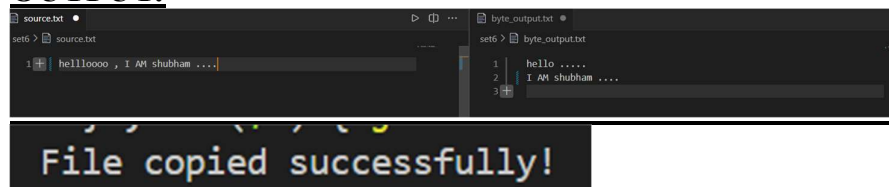
public class Prac_30{
    public static void main(String[] args) {
        String sourceFile = "source.txt";
        String destinationFile = "destination.txt";

        try (BufferedReader reader = new BufferedReader(new
        FileReader(sourceFile));
            BufferedWriter writer = new BufferedWriter(new
        FileWriter(destinationFile))) {

            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }

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

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

**OUTPUT:****CONCLUSION:**

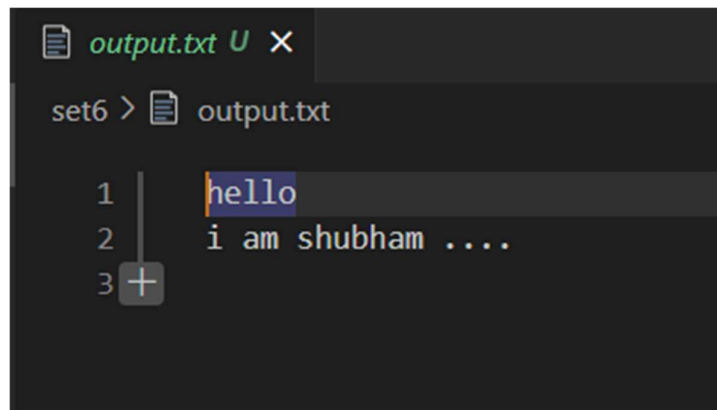
- `BufferedReader`: Reads the data from the source file in an efficient manner by buffering characters.

	<ul style="list-style-type: none"> <li>• <b>BufferedWriter:</b> Writes the data to the destination file and buffers it for efficient output.</li> <li>• If the destination file (destination.txt) does not exist, it is automatically created by <b>FileWriter</b>.</li> </ul>
<b>31.</b>	<p>Write a program to show use of character and byte stream. Also show use of <b>BufferedReader/BufferedWriter</b> to read console input and write them into a file.</p> <p><b><u>PROGRAM CODE:</u></b></p> <pre>import java.io.*;  public class PRAC31 {     public static void main(String[] args) {         String filePath = "output.txt";          try (BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));             BufferedWriter fileWriter = new BufferedWriter(new FileWriter(filePath)))         {              System.out.println("Enter text (type 'exit' to quit):");              String line;             while (!(line = consoleReader.readLine()).equalsIgnoreCase("exit")) {                 fileWriter.write(line);                 fileWriter.newLine();             }              System.out.println("Data written to file successfully!");          } catch (IOException e) {             System.out.println("An error occurred: " + e.getMessage());         }          try (FileInputStream fis = new FileInputStream(filePath);             FileOutputStream fos = new FileOutputStream("byte_output.txt")) {              int byteData;             while ((byteData = fis.read()) != -1) {                 fos.write(byteData);             }              System.out.println("File copied using byte stream successfully!");         } catch (IOException e) {             System.out.println("An error occurred: " + e.getMessage());         }     } }</pre>

```
}  
}  
}
```

**OUTPUT:**

```
Enter text (type 'exit' to quit):  
hello  
i am shubham ....  
exit  
Data written to file successfully!  
File copied using byte stream successfully!
```

**CONCLUSION:**

This program reads lines of text input from the console and writes them to a file named output.txt. It continues to read input until the user types "exit."

**PART-VII Multithreading**

32. Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.

**PROGRAM CODE:**

```
class MyThread extends Thread {  
    public void run() {
```

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

public class prec32_a {
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

**OUTPUT:**

```
D:\java>javac prec32_a.java

D:\java>java prec32_a
Hello World
```

**PROGRAM CODE:**

```
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Hello World");
    }
}

public class prec32_b {
    public static void main(String[] args) {
        MyRunnable m1 = new MyRunnable();
        Thread t1 = new Thread(m1);
        t1.start();
    }
}
```

**OUTPUT:**

```
D:\java>javac prec32_b.java

D:\java>java prec32_b
Hello World
```

**CONCLUSION:**

In both examples, threads are used to execute the run() method. In the first case, MyThread directly extends Thread and overrides its run() method, allowing the

	<p>thread to be started using <code>t1.start()</code>. In the second example, <code>MyRunnable</code> implements the <code>Runnable</code> interface, and its <code>run()</code> method is passed to a <code>Thread</code> object. Both approaches are valid for multithreading in Java, but implementing <code>Runnable</code> is preferred in most cases since it allows the class to extend other classes as well.</p>
33.	<p>Write a program which takes N and number of threads as an argument. Program should distribute the task of summation of N numbers amongst number of threads and final result to be displayed on the console.</p> <p><b><u>PROGRAM CODE:</u></b></p> <pre>import java.util.Scanner;  class MyThread extends Thread {     int start, end;     static int sum = 0;      MyThread(int start, int end) {         this.start = start;         this.end = end;     }      static void addSum(int partialSum) {         sum += partialSum;     }      public void run() {         int partialSum = 0;         for (int i = start; i &lt;= end; i++) {             partialSum += i;         }         addSum(partialSum);     } }  public class prec33 {     public static void main(String[] args) {         int N, numThreads;         Scanner s = new Scanner(System.in);          System.out.print("Enter the number 'N': ");         N = s.nextInt();</pre>

```

        System.out.print("Enter the number of threads to be used (should be less than or equal to N): ");
        numThreads = s.nextInt();

        MyThread[] threads = new MyThread[numThreads];

        int range = N / numThreads;
        int start = 1, end;

        for (int i = 0; i < numThreads; i++) {
            end = (i == numThreads - 1) ? N : start + range - 1; the remainder
            threads[i] = new MyThread(start, end);
            threads[i].start(); // Start the thread
            start = end + 1;
        }

        try {
            for (int i = 0; i < numThreads; i++) {
                threads[i].join();
            }
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }

        System.out.println("The sum of numbers from 1 to " + N + " is: " + MyThread.sum);
    }
}

```

### **OUTPUT:**

```

D:\java>javac prec33.java

D:\java>java prec33
Enter the number 'N': 3
Enter the number of threads to be used (should be less than or equal to N): 2
The sum of numbers from 1 to 3 is: 6

```

### **CONCLUSION:**

In this program, multiple threads are used to calculate the sum of numbers from 1 to N. Each thread is responsible for a portion of the range, and the results are combined using a synchronized method to ensure thread safety. The use of `Thread.join()` ensures that the main program waits for all threads to complete before displaying the final sum. This approach speeds up the summation by distributing the task across multiple threads.

- 34.** Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.



**PROGRAM CODE:**

```
class even extends Thread {
    int n;

    public even(int n) {
        this.n = n;
    }

    public void run() {
        System.out.println(n * n);
    }
}

class odd extends Thread {
    int n;

    public odd(int n) {
        this.n = n;
    }

    public void run() {
        System.out.println(n * n * n);
    }
}

public class prec34 {
    public static void main(String[] args) {
        even e;
        odd o;

        for (int i = 1; i <= 10; i++) {
            e = new even(i);
            o = new odd(i);
            try {
                if (i % 2 == 0) {
                    e.start();
                } else {
                    o.start();
                }
                Thread.sleep(1000);
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

```

    }
}
}
}

```

**OUTPUT:**

```
D:\java>javac Prec34.java
```

```
D:\java>java Prec34
```

```
Odd cube of 1: 1
```

```
Even square of 2: 4
```

```
Odd cube of 3: 27
```

```
Even square of 4: 16
```

```
Odd cube of 5: 125
```

```
Even square of 6: 36
```

```
Odd cube of 7: 343
```

```
Even square of 8: 64
```

```
Odd cube of 9: 729
```

```
Even square of 10: 100
```

**CONCLUSION:**

In this program, separate threads calculate the square of even numbers and the cube of odd numbers from 1 to 10. The synchronized block ensures that the output is printed without interruption, and a delay of 1 second between thread executions is added for clarity. This demonstrates multithreading for handling different tasks based on even or odd numbers.

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

**PROGRAM CODE:**

```

class IncrementThread extends Thread {
    private int number;

    public IncrementThread(int number) {
        this.number = number;
    }

    @Override
    public void run() {
        try {

```

```

        Thread.sleep(1000);
        number++;
        System.out.println("Incremented value after 1 second: " + number);
    } catch (InterruptedException e) {
        System.out.println("Thread was interrupted");
    }
}
}

public class prec35 {
    public static void main(String[] args) {
        int initialValue = 0;
        System.out.println("Initial value: " + initialValue);
        IncrementThread incrementThread = new IncrementThread(initialValue);
        incrementThread.start();
    }
}

```

**OUTPUT:**

```

D:\java>javac prec35.java

D:\java>java prec35
Initial value: 0
Incremented value after 1 second: 1

```

**CONCLUSION:**

In this program, a thread is created to increment a number by 1 after a 1-second delay. The initial value is displayed first, and the thread then increments the value and prints it. This demonstrates basic thread creation and delayed execution using `Thread.sleep()`.

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

**PROGRAM CODE:**

```

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

class Second extends Thread {

```

```
public void run() {
    System.out.println("Second");
}

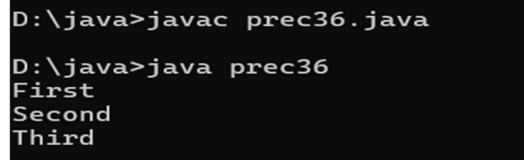
class Third extends Thread {
    public void run() {
        System.out.println("Third");
    }
}

public class prec36 {
    public static void main(String[] args) {
        first f = new first();

        Second s = new Second();
        Third t = new Third();

        f.setPriority(8);
        s.setPriority(4);
        t.setPriority(6);

        f.start();
        s.start();
        t.start();
    }
}
```

**OUTPUT:**

```
D:\java>javac prec36.java
D:\java>java prec36
First
Second
Third
```

**CONCLUSION:**

In this program, three threads (first, Second, and Third) are created and assigned different priorities. The thread priorities influence the likelihood of execution but do not guarantee the order in which they will run. This demonstrates how thread priorities can be set, though actual execution order depends on the system's thread scheduling.

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

**PROGRAM CODE:**

```
class Produce extends Thread {
    int n;
    boolean produced = false;

    Produce(int n) {
        this.n = n;
    }

    public synchronized void run() {
        for (int i = 1; i <= n; i++) {
            while (produced) { // Wait if something is already produced
                try {
                    wait();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
            System.out.println("Produced: " + i);
            produced = true;
            notify(); // Notify the consumer that production is done
        }
    }
}

class Consume extends Thread {
    int n;
    Produce producer;

    Consume(int n, Produce producer) {
        this.n = n;
        this.producer = producer;
    }

    public synchronized void run() {
        for (int i = 1; i <= n; i++) {
            synchronized (producer) {
                while (!producer.produced) { // Wait until something is produced
                    try {
                        producer.wait();
                    } catch (InterruptedException e) {
```



**PART-VIII Collection Framework and Generic**

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

**PROGRAM CODE:**

```
import java.util.ArrayList;

public class Prac_38 {
    private ArrayList<Object> list = new ArrayList<>();

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

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

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

    public Object pop() {
        if (!isEmpty()) {
            return list.remove(list.size() - 1);
        }
        return null;
    }

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

    public static void main(String[] args) {
        MyStack stack = new MyStack();
        stack.push(1);
        stack.push(2);
        stack.push(3);
    }
}
```

```

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

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

    }
}

```

**OUTPUT:**

```

Top element is: 3
Stack size is: 3
Popped element: 3
Stack size after pop: 2

```

**CONCLUSION:**

The program defines a simple stack implementation using ArrayList. It allows basic stack operations such as push(), pop(), and peek(), while also checking if the stack is empty and retrieving its size. The main method demonstrates these operations by adding and removing elements, as well as showing the top element and the size of the stack before and after popping an element.

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

**PROGRAM CODE:**

```

import java.util.Arrays;

public class Prac_39 {

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

```



```
public static void main(String[] args) {
    Integer[] intArray = {3, 1, 4, 2, 5};
    System.out.println("Before sorting: " + Arrays.toString(intArray));
    sort(intArray);
    System.out.println("After sorting: " + Arrays.toString(intArray));

    String[] stringArray = {"Apple", "Banana", "Orange", "Mango"};
    System.out.println("Before sorting: " + Arrays.toString(stringArray));
    sort(stringArray);
    System.out.println("After sorting: " + Arrays.toString(stringArray));

    Product[] productArray = {
        new Product("Laptop", 1200),
        new Product("Phone", 800),
        new Product("Tablet", 600)
    };
    System.out.println("Before sorting products by price: " + Arrays.toString(productArray));
    sort(productArray);
    System.out.println("After sorting products by price: " + Arrays.toString(productArray));
}

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

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

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

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

**OUTPUT:**

```

Before sorting: [3, 1, 4, 2, 5]
After sorting: [1, 2, 3, 4, 5]
Before sorting: [Apple, Banana, Orange, Mango]
After sorting: [Apple, Banana, Mango, Orange]
Before sorting products by price: [Laptop ($1200), Phone ($800), Tablet ($600)]
After sorting products by price: [Tablet ($600), Phone ($800), Laptop ($1200)]

```

### **CONCLUSION:**

This program demonstrates a generic sort method that can sort arrays of any type that implements the Comparable interface, such as Integer, String, and custom Product objects. It sorts an integer array, a string array, and an array of Product objects by price using the Arrays.sort() method. The program effectively highlights how different types can be sorted in a consistent manner.

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

### **PROGRAM CODE:**

```

import java.util.*;

public class Prac_40 {

    public static void main(String[] args) {
        String text = "This is a sample text. This text is for counting words. Counting words is fun.";
        text = text.replaceAll("[^a-zA-Z]", "").toLowerCase();
        String[] words = text.split("\\s+");
        Map<String, Integer> wordCountMap = new HashMap<>();

        for (String word : words) {
            if (!word.isEmpty()) {
                wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);
            }
        }

        Set<String> uniqueWords = new TreeSet<>(wordCountMap.keySet());
        System.out.println("Word Count:");
        for (String word : uniqueWords) {
            System.out.println(word + ": " + wordCountMap.get(word));
        }
    }
}

```

**OUTPUT:**

```

Word Count:
a: 1
counting: 2
for: 1
fun: 1
is: 3
sample: 1
text: 2
this: 2
words: 2

```

**CONCLUSION:**

This program processes a text to count the frequency of each word. It removes punctuation, converts the text to lowercase, and uses a HashMap to store the word counts. The unique words are then displayed in sorted order, along with their respective frequencies, using a TreeSet for sorted output.

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

**PROGRAM CODE:**

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
public class Prac_41
{
    public static void main(String[] args) {
        String filePath = "YourJavaFile.java";
        HashSet<String> keywords = new HashSet<>();
        String[] javaKeywords = {
            "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char",
            "class", "const", "continue", "default", "do", "double", "else", "enum",
            "extends", "final", "finally", "float", "for", "goto", "if", "implements",
            "import", "instanceof", "int", "interface", "long", "native", "new",
            "null", "package", "private", "protected", "public", "return", "short",
            "static", "strictfp", "super", "switch", "synchronized", "this",
            "throw", "throws", "transient", "try", "void", "volatile", "while"
        };
    }
}

```

```

    };

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

    int keywordCount = 0;

    try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] words = line.split("\\W+");
            for (String word : words) {
                if (keywords.contains(word)) {
                    keywordCount++;
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

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

```

## OUTPUT:

```

Comment Code |
public class YourJavaFile {
    Run | Debug
    public static void main(String[] args) {
        int count = 0;
        for (int i = 0; i < 10; i++) {
            if (i % 2 == 0) {
                count++;
            }
        }
        System.out.println("Count of even numbers: " + count);

        if (count > 5) {
            System.out.println(x:"More than 5 even numbers found.");
        } else {
            System.out.println(x:"5 or fewer even numbers found.");
        }

        // Using a method
        YourJavaFile example = new YourJavaFile();
        example.printMessage();
    }

    public void printMessage() {
        System.out.println(x:"Hello, this is an example of Java keywords.");
    }
}

h1 patel\OneDrive\Desktop\java\set8\ ; if ($?) { java
($?) { java Per 41 }
Number of keywords: 15
PS C:\Users\maharshi patel\OneDrive\Desktop\java\set8>

```

**CONCLUSION:**

This program reads a Java source file, identifies and counts the occurrences of Java keywords using a predefined list stored in a HashSet for fast lookup. It processes the file line by line, splitting it into words, and increments the keyword count whenever a match is found. The total number of Java keywords is then displayed at the end.