

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

Department of Computer Science & Engineering

Subject Name: JAVA PROGRAMMING**Semester: III****Subject Code: CSE201****Academic year: 2024****Aim of the Practical**

No.	PART-I Data Types, Variables, String, Control Statements, Operators, Arrays
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> <p><u>PROGRAM CODE:</u></p> <p>JDK (Java Development Kit): The JDK is a software development environment used for developing Java applications. It includes the JRE (Java Runtime Environment), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed for Java development.</p> <p>JRE (Java Runtime Environment): The JRE provides the libraries, Java Virtual Machine (JVM), and other components to run applications written in Java. It includes the JVM, core classes, and supporting files.</p> <p>JVM (Java Virtual Machine): The JVM is an abstract machine that enables your computer to run a Java program. It converts bytecode into machine language. It provides a platform-independent way of executing Java bytecode, ensuring that Java programs can run on any device equipped with a JVM.</p> <p>Javadoc: Javadoc is a documentation generator tool included in the JDK. It generates API documentation in HTML format from Java source code.</p> <p>Command Line Arguments: Command line arguments are parameters passed to a program</p>

at the time it is invoked. They allow users to specify configurations or input data when running a program.

Eclipse IDE: Eclipse is a popular open-source Integrated Development Environment (IDE) for Java development. Developers use Eclipse for writing, testing, and debugging Java code efficiently.

Console Programming: Console programming involves writing and running programs that interact with the user through a text-based interface.

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:

```
//Practical-2
```

```
class A
```

```
{
```

```
public static void main(String [] args)
```

```
{
```

```
int a=20;
```

```
System.out.print("Total Bank balance is="+a);
```

```
}
```

```
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.22631.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA>javac A.java

C:\Users\TISHA RANA>java A
Total Bank balance is=$20
C:\Users\TISHA RANA>
```

CONCLUSION:

From the above practical, we learned about the syntax in java to print a value on console screen.

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:

```
//Practical-3(user input)
import java.util.Scanner;
class Speed
{
    public static void main(String [] args)
    {
        Scanner obj=new Scanner(System.in);
        System.out.println("Enter distance(in m):" );

        double dist=obj.nextDouble();
        System.out.println("Enter time(in hr):");
        int hr=obj.nextInt();

        System.out.println("Enter time(in min):");
        int min=obj.nextInt();

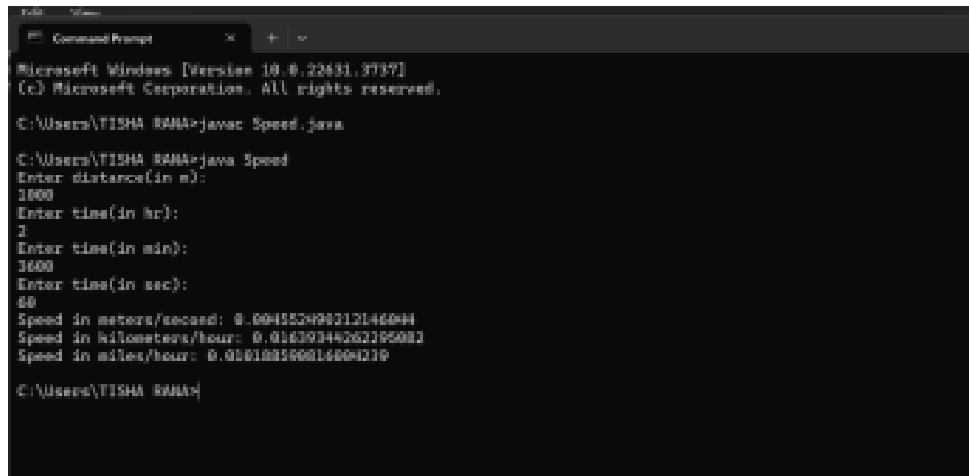
        System.out.println("Enter time(in sec):");
```

```

        int sec=obj.nextInt();
        int tsec=sec+(60*sec)+(3600*sec);
        double s1=dist/tsec;
        double s2=(dist*0.001)/(tsec/3600);
        double s3=(dist/1609)/(tsec/3600);
        System.out.println("Speed in meters/second: "+s1 );
        System.out.println("Speed in kilometers/hour: "+s2);
        System.out.println("Speed in miles/hour: "+s3);
    }
}

```

OUTPUT:



```

Microsoft Windows [Version 10.0.22431.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA >java Speed.java

C:\Users\TISHA >java Speed
Enter distance(in m):
1000
Enter time(in hr):
2
Enter time(in min):
3600
Enter time(in sec):
60
Speed in meters/second: 0.00453498012146044
Speed in kilometers/hour: 0.01632344262165003
Speed in miles/hour: 0.010100360814604219

C:\Users\TISHA >

```

CONCLUSION:

From the above practical, we learned how to take user input and print on the console screen using Scanner class's object. Also how to store user input into a variable and accessing it.

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:

//Practical-4

```
import java.util.Scanner;
```

```
class Expense
```

```
{
```

```
    public static void main(String [] args)
```

```
    {
```

```
        Scanner obj=new Scanner(System.in);
```

```
        System.out.print("Enter no. of days in month:");
```

```
        int days=obj.nextInt();
```

```
        double [] expense=new double[days];
```

```
        for(int i=0;i<days;i++)
```

```
        {
```

```
            System.out.print("Enter daily expense:");
```

```
            expense[i]=obj.nextDouble();
```

```
        }
```

```
        double totalexp=0;
```

```
        for(int i=0;i<days;i++)
```

```
        {
```

```
            totalexp=totalexp+expense[i];
```

```
        }
```

```
        System.out.println("Total expense of month="+totalexp);
```

```
    }
```

```
}
```

OUTPUT:

```

(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA\OneDrive\Desktop>javac Expense.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Expense
Enter no. of days in month:4
Enter daily expense:1000
Enter daily expense:1050
Enter daily expense:2000
Enter daily expense:4500
Total expense of month=8550.0

C:\Users\TISHA RANA\OneDrive\Desktop>

```

CONCLUSION:

In the above practical we are using control statement for loop for calculating the daily expenses of the individual with the help of Array.

*

Supplementary

You are creating a library management system. The library has two separate lists of books for fiction and non-fiction. The system should merge these lists into a single list for inventory purposes. Write a Java program to merge two arrays.

PROGRAM CODE:

```

public class Library {

    public static void main(String[] args) {

        String[] fiction = {"The Great Gatsby", "Harry Potter", "1984"};
        String[] nonFiction = {"Sapiens", "Educated"};

        String[] Merge = mergeArrays(fiction, nonFiction);

        System.out.println("Library Inventory:");
        for (String book : Merge)

```

```

        {
            System.out.println(book);
        }
    }

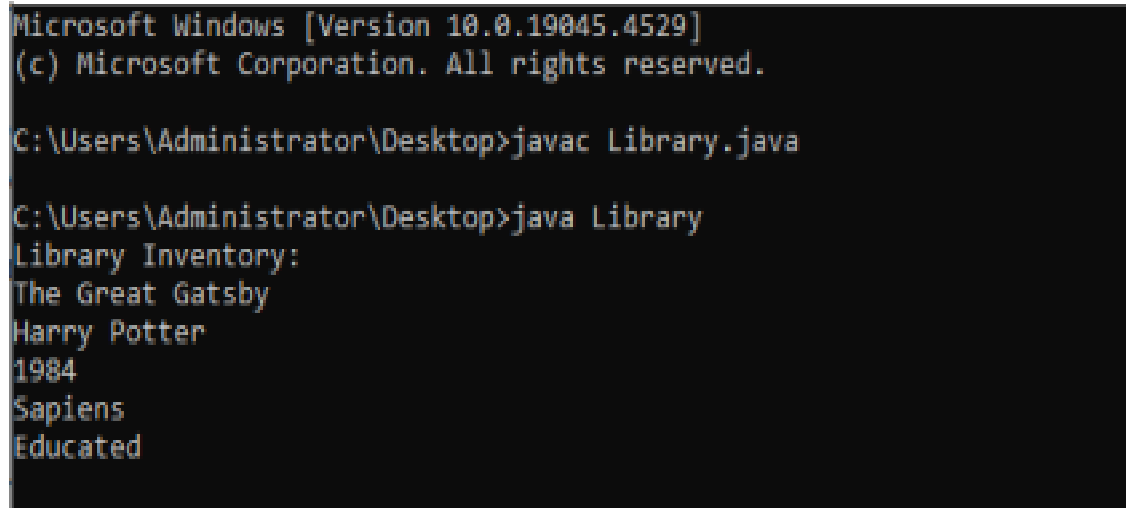
    static String[] mergeArrays(String[] array1, String[] array2) {
        int l1 = array1.length;
        int l2 = array2.length;

        String[] mergedArray = new String[l1 + l2];
        System.arraycopy(array1, 0, mergedArray, 0, l1);
        System.arraycopy(array2, 0, mergedArray, l1, l2);

        return mergedArray;
    }
}

```

OUTPUT:



```

Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>javac Library.java

C:\Users\Administrator\Desktop>java Library
Library Inventory:
The Great Gatsby
Harry Potter
1984
Sapiens
Educated

```

CONCLUSION:

From the above practical, we learned about how to merge two arrays and print it on console screen with the help of example of Library Management System where we combined two arrays Fiction and Non fiction books.

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

PROGRAM CODE:

//Practical-5

import java.util.Scanner;

class Electrical

```
{  
    public static void main(String [] args)  
    {  
        Scanner o=new Scanner(System.in);  
        System.out.print("Enter the bill amount:");  
        double bill=o.nextDouble();  
        System.out.println("Enter the choice: 1.)Motor 2.)Fan 3.)Tube 4.)Wire-");  
        int choice=o.nextInt();  
        double total=0;  
        switch(choice)  
        {  
            case 1:  
            {  
                total=(bill*0.08)+bill;  
            }  
            break;  
            case 2:  
            {  
                total=(bill*1.2)+bill;  
            }  
            break;  
            case 3:  
            {  
                total=(bill*0.05)+bill;  
            }  
            break;  
            case 4:  
            {  
                total=(bill*0.075)+bill;  
            }  
            break;  
            default:  
            {  
                total=(bill*0.03)+bill;  
            }  
        }  
        System.out.println("Total bill amount is: "+total);  
    }  
}
```

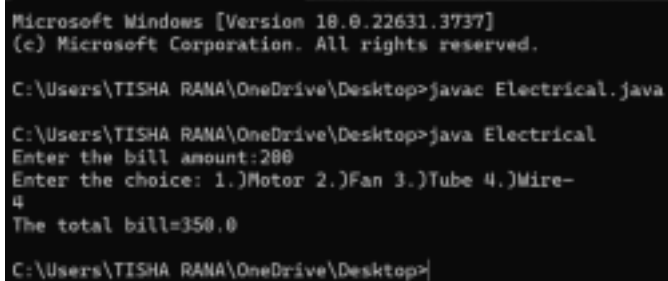
```

break;
case 4:
{
total=(bill*0.75)+bill;
}
break;
default:
{
total=(bill*0.03)+bill;
}
break;

}
System.out.println("The total bill="+total);
}
}

```

OUTPUT:



```

Microsoft Windows [Version 10.0.22631.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA\OneDrive\Desktop>javac Electrical.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Electrical
Enter the bill amount:200
Enter the choice: 1.)Motor 2.)Fan 3.)Tube 4.)Wire-
4
The total bill=350.0

C:\Users\TISHA RANA\OneDrive\Desktop>

```

CONCLUSION:

In the above practical we prepared an electrical shop bill with help of Switch case to calculate total bill along with tax on different appliances.

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.Scanner;
class Fibo
{
    static int Sum(int n)
    {
        if (n <= 0)
            { return 0;}

        int fibo[]=new int[n+1];
        fibo[0] = 0; fibo[1] = 1;
        int sum = fibo[0] + fibo[1];
        for (int i=2; i<=n; i++)
        {
            fibo[i] = fibo[i-1]+fibo[i-2];
            sum += fibo[i];
        }

        return sum;
    }

    public static void main(String args[])
    {
        Scanner obj=new Scanner(System.in);
        System.out.print("enter n :");
        int n=obj.nextInt();
        System.out.println("Sum of Fibonacci" + " numbers is : "+ Sum(n));
    }
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>javac Fibo.java

C:\Users\Administrator\Desktop>java Fibo
enter n :5
Sum of Fibonacci numbers is : 12

C:\Users\Administrator\Desktop>_
```

CONCLUSION:

In the above practical we performed the summation of Fibonacci series with help of array and for loop.

Supplementary

Imagine you are developing a classroom management system. You need to keep track of the grades of students in a class. After collecting the grades, you want to display each student's grade along with a message indicating if they have passed or failed. Let's assume the passing grade is 50.

PROGRAM CODE:

```
public class Classroom {

    public static void main(String[] args)
    {

        String[] students = {"Sita", "Geeta", "Meena"};
        int[] grades = {85, 45, 60};
        int passing = 50;

        displayGrades(students, grades, passing);
    }

    static void displayGrades(String[] students, int[] grades, int passingGrade) {
        for (int i = 0; i < students.length; i++)
        {
            String student = students[i];
            int grade = grades[i];
            String status = (grade >= passingGrade) ? "Passed" : "Failed";

            System.out.println(student + ": " + grade + " - " + status);
        }
    }
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>javac Classroom.java

C:\Users\Administrator\Desktop>java Classroom
Sita: 85 - Passed
Geeta: 45 - Failed
Meena: 60 - Passed

C:\Users\Administrator\Desktop>
```

CONCLUSION:

In the above practical we made a Classroom Management System to keep track on the students grade. According to the grades we indicated whether a student have passed or failed.

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:

```
//1.)Substring
import java.lang.String;
class ABC
{
    public static void main(String[] args)
    {
        System.out.println(frontTimes("Chocolate", 2));
        System.out.println(frontTimes("Chocolate", 3));
        System.out.println(frontTimes("Abc", 3));
    }

    static String frontTimes(String str, int n)
    {
        String front = str.length() < 3 ? str :

        str.substring(0, 3); String result = "";

        for (int i = 0; i < n; i++)
        {
            result = result + front;
        }

        return result;
    }
}
```

OUTPUT:

```
(c) Microsoft Corporation. All rights reserved.  
C:\Users\Administrator\Desktop>javac ABC.java  
C:\Users\Administrator\Desktop>java ABC  
ChoCho  
ChoChoCho  
AbcAbcAbc  
C:\Users\Administrator\Desktop>
```

CONCLUSION:

Using the string class we performed some functions like string length, substring etc... to get the required output on console screen.

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:

```
import java.lang.String;
class Num9
{
    public static void main(String [] args)
    {
        System.out.println(count(new int[] { 1,2,9 }));
        System.out.println(count(new int[] { 1,9,9 }));
        System.out.println(count(new int[] { 1,9,9,3,9 }));
    }
    static int count(int[] num)
    {
        int c=0;
        for(int i=0;i<num.length;i++)
        {
            if(num[i]==9)
            {
                c++;
            }
        }
        return c;
    }
}
```

OUTPUT:

```
C:\Users\Administrator\Desktop>javac Num9.java  
C:\Users\Administrator\Desktop>java Num9  
1  
2  
3  
C:\Users\Administrator\Desktop>_
```

CONCLUSION:

In the above practical we calculated the number of times 9 is appearing in the given arrays with help of loop and count variable.

Supplementary Experiment:

1. Write a Java program to replace each substring of a given string that matches the given regular expression with the given replacement.

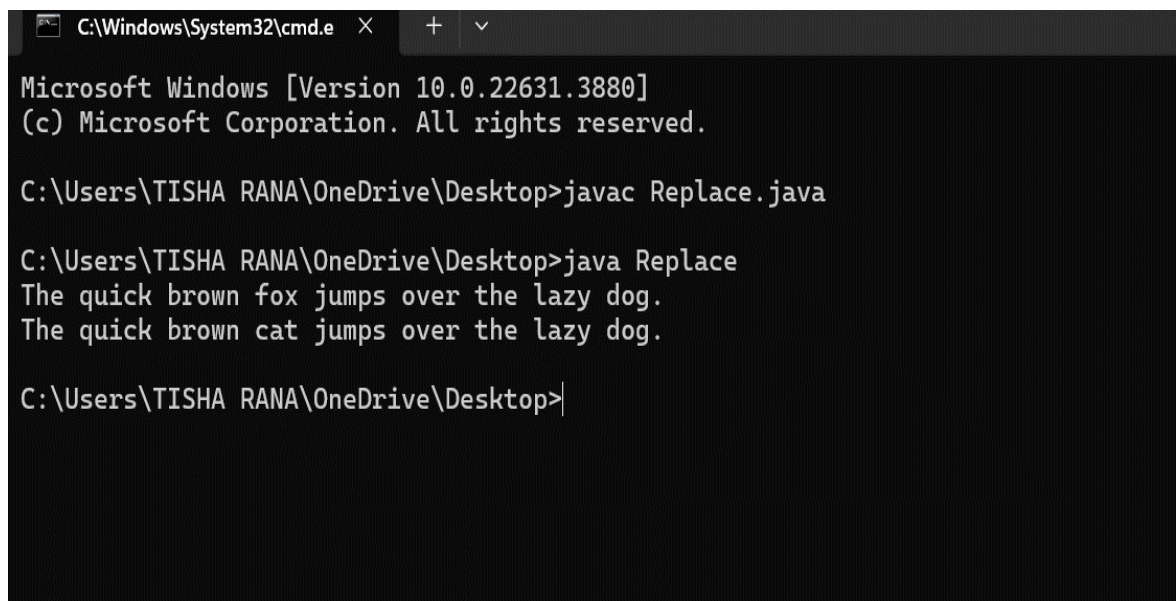
Sample string : "The quick brown fox jumps over the lazy dog."

In the above string replace all the fox with cat.

PROGRAM CODE:

```
class Replace {  
    public static void main(String[] args)  
    {  
        String sample= "The quick brown fox jumps over the lazy dog.";  
        String find = "fox";  
        String replacement = "cat";  
        String result = sample.replaceAll(find, replacement);  
        System.out.println(sample);  
        System.out.println(result);  
    }  
}
```

OUTPUT:



```
C:\Windows\System32\cmd.e  X  +  v  
Microsoft Windows [Version 10.0.22631.3880]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>javac Replace.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Replace  
The quick brown fox jumps over the lazy dog.  
The quick brown cat jumps over the lazy dog.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

We used `replaceAll()` string method to replace all fox words with cat in order to get required output.

9. Given a string, return a string where for every char in the original, there are two chars.

`double_char('The') → 'TThhee'`

`double_char('AAbb') → 'AAAAbbbb'`

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

PROGRAM CODE:

```
import java.lang.String;
class Class
{

    public static void main(String [] args)
    {
        System.out.println(doublechar("The"));
        System.out.println(doublechar("AAbb"));
        System.out.println(doublechar("Hi-there"));
    }
    static String doublechar(String str)
    {
        String result="";
        for(int i=0;i<str.length();i++)
        {
            Char c=str.charAt(i);
            result=result+c+c;
        }
        return result;
    }
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>javac Class.java

C:\Users\Administrator\Desktop>java Class
Thhee
AAAbbbb
Hii--tthheerree

C:\Users\Administrator\Desktop>
```

CONCLUSION:

In the above practical, we doubled each character of the string using charAt() function of fetch that particular character and then doubled it and print it on console screen.

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:

```
import java.util.Arrays;
public class Str {

    static int getLength(String str)
    {
        return str.length();
    }

    static String toLowerCase(String str)
    {
        return str.toLowerCase();
    }

    static String toUpperCase(String str)

    {
        return str.toUpperCase();
    }
}
```

```
static String reverseString(String str)
{
    return new StringBuilder(str).reverse().toString();
}
```

```
static String sortString(String str)
{
    char[] temp = str.toCharArray();
    Arrays.sort(temp);
    return new String(temp);
}
```

```
public static void main(String[] args)
{
    String str = "Hello World";

    System.out.println("Original String: " + str);

    System.out.println("Length of the String: " + getLength(str));

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

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

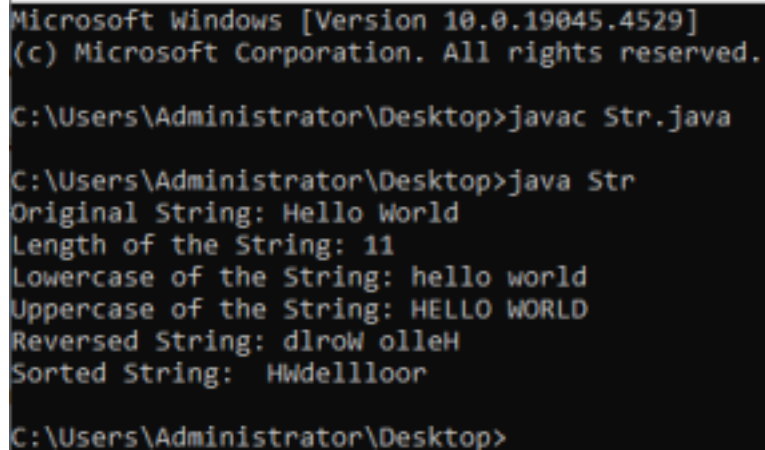
    System.out.println("Reversed String: " + reverseString(str));
}
```



```
System.out.println("Sorted String: " + sortString(str));
```

```
}  
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.19045.4529]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\Administrator\Desktop>javac Str.java  
  
C:\Users\Administrator\Desktop>java Str  
Original String: Hello World  
Length of the String: 11  
Lowercase of the String: hello world  
Uppercase of the String: HELLO WORLD  
Reversed String: dlroW olleH  
Sorted String: Hwdellloor  
  
C:\Users\Administrator\Desktop>
```

CONCLUSION:

In the above practical, we used functions of arrays class along with String Builder class to find length of string, to convert into lower case, to convert into upper case, to reverse string and to sort the string according to the ASCII value.

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:

//P11

class Str

{

 public static void main(String []args)

 {

 String S1="CHARUSAT UNIVERSITY";

 System.out.println("String is:"+S1);

 int len=S1.length();

 System.out.println("Length of String is:"+len);

 String S2=S1.replace('H','T');

 String S3=S1.toLowerCase();

 System.out.println(" String 2 is:"+S2);

 System.out.println(" String 3 is:"+S3);

 }

}

OUTPUT:

```
C:\Windows\System32\cmd.e × + ∨  
Microsoft Windows [Version 10.0.22631.3880]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>javac Str.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Str  
String is:CHARUSAT UNIVERSITY  
Length of String is:19  
String 2 is:CTARUSAT UNIVERSITY  
String 3 is:charusat university  
  
C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical we used length(), replace() and toLowerCase() functions to get required output on the console screen.

Supplementary Experiment:

1. Write a Java program to count and print all duplicates in

the input string.

Sample Output:

The given string is: resource

The duplicate characters and counts are:

e appears 2 times

r appears 2 times

PROGRAM CODE:

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

```
public class Main {
```

```
    public static void Duplicate(String str)
```

```
    {
```

```
        int len = str.length();
```

```
        char[] chars = str.toCharArray();
```

```
        Arrays.sort(chars);
```

```
        String sortedStr = new String(chars);
```

```
        for (int i = 0; i < len; i++)
```

```
        {
```

```
            int count = 1;
```

```

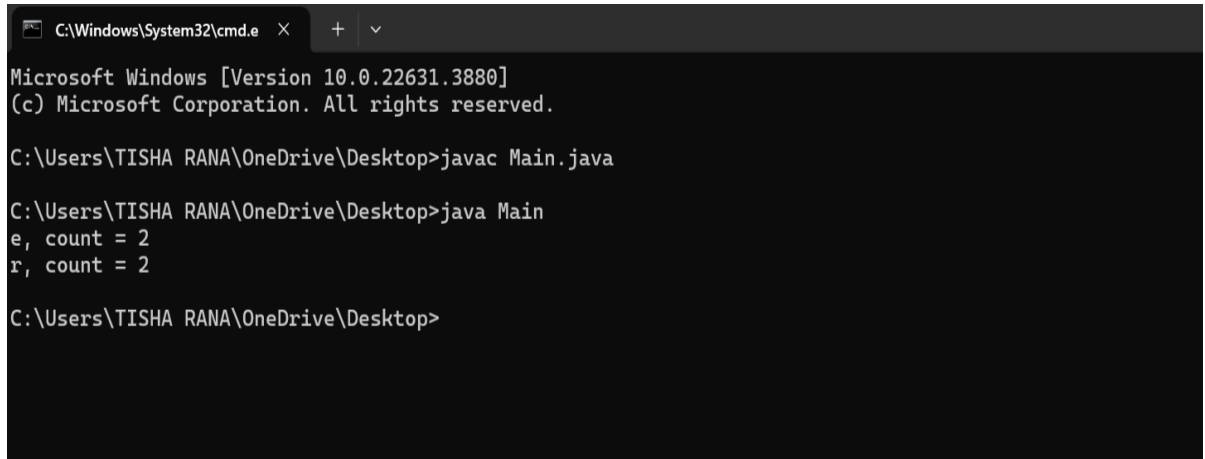
while (i < len - 1
    && sortedStr.charAt(i)
        == sortedStr.charAt(i + 1))
{
    count++;
    i++;
}

if (count > 1)
{
    System.out.println(sortedStr.charAt(i)+ ", count = " + count);
}
}

public static void main(String[] args)
{
    String str = "resource";
    Duplicate(str);
}
}

```

OUTPUT:



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA\OneDrive\Desktop>javac Main.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Main
e, count = 2
r, count = 2

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical, we learned how to find the duplicate character in given String with help of string functions in java.

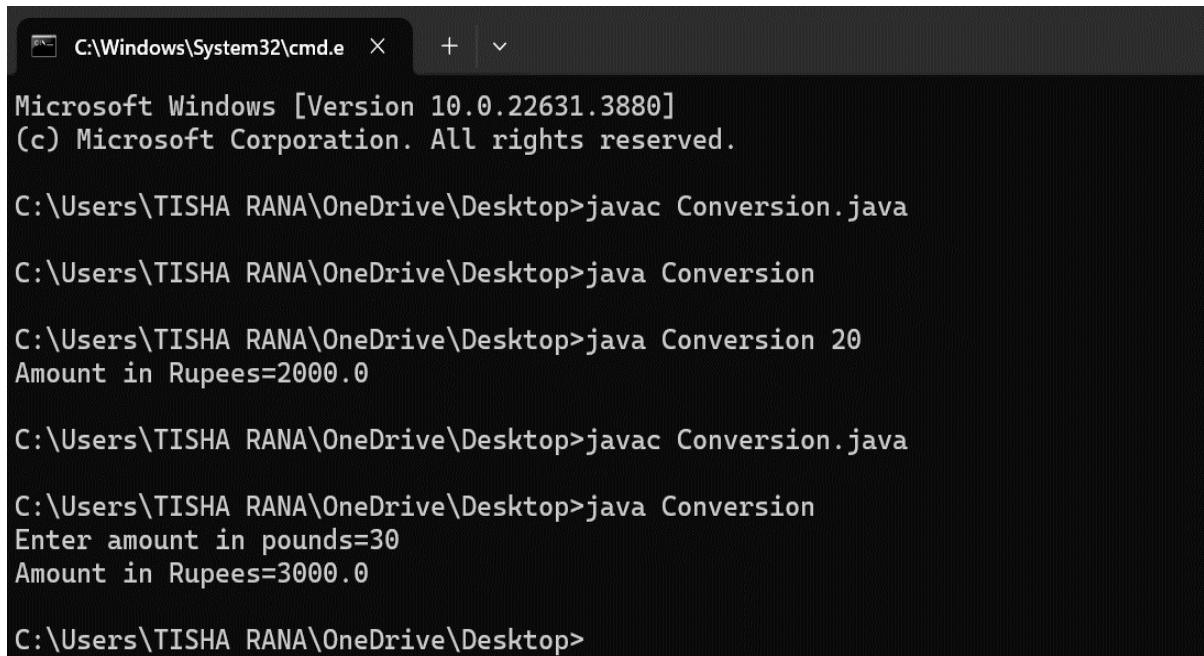
PART-III Object Oriented Programming: Classes, Methods, Constructors

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:

```
//12.)User interactive and command-line
import java.util.*;
class Conversion
{
    static double conversion(double p)
    {
        double rupee=100*p;
        return rupee;
    }
    public static void main(String [] args)
    {
        if(args.length>0)
        {
            double pound=Double.parseDouble(args[0]);
            System.out.println("Amount in Rupees="+conversion(pound));
        }
        /*Scanner obj=new Scanner(System.in);
        System.out.print("Enter amount in pounds=");
        double pound=obj.nextDouble();
        System.out.println("Amount in Rupees="+conversion(pound));*/
    }
}
```

OUTPUT:



```
C:\Windows\System32\cmd.e × + ∨  
Microsoft Windows [Version 10.0.22631.3880]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>javac Conversion.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion 20  
Amount in Rupees=2000.0  
  
C:\Users\TISHA RANA\OneDrive\Desktop>javac Conversion.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion  
Enter amount in pounds=30  
Amount in Rupees=3000.0  
  
C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical we developed a method named conversion which converts the amount from pounds to rupees. We took input from user using Command-line argument and using scanner class. In command-line argument intake we used parse method that reads string from command-line argument and provides respected type of output.

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:

```
//13.)Employee details
class Employee
{
    String firstName;
    String lastName;
    double monthlySalary;

    Employee(String firstName, String lastName, double monthlySalary)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        setMonthlySalary(monthlySalary);
    }

    void setFirstName(String firstName)
    {
        this.firstName = firstName;
    }

    void setLastName(String lastName)
    {
        this.lastName = lastName;
    }

    void setMonthlySalary(double monthlySalary)
    {
        if (monthlySalary > 0)
        {
            this.monthlySalary = monthlySalary;
        }
    }
}
```

```

        else
        {
            this.monthlySalary = 0.0;
        }
    }

    String getFirstName()
    {
        return firstName;
    }

    String getLastName()
    {
        return lastName;
    }

    double getMonthlySalary()
    {
        return monthlySalary;
    }

    double getYearlySalary()
    {
        return monthlySalary * 12;
    }

    void giveRaise()
    {
        this.monthlySalary = this.monthlySalary * 1.10;
    }
}

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

        Employee emp1 = new Employee("Tisha", "Rana", 30000.00);
        Employee emp2 = new Employee("Janvi", "Patel", 40000.00);
    }
}

```

```

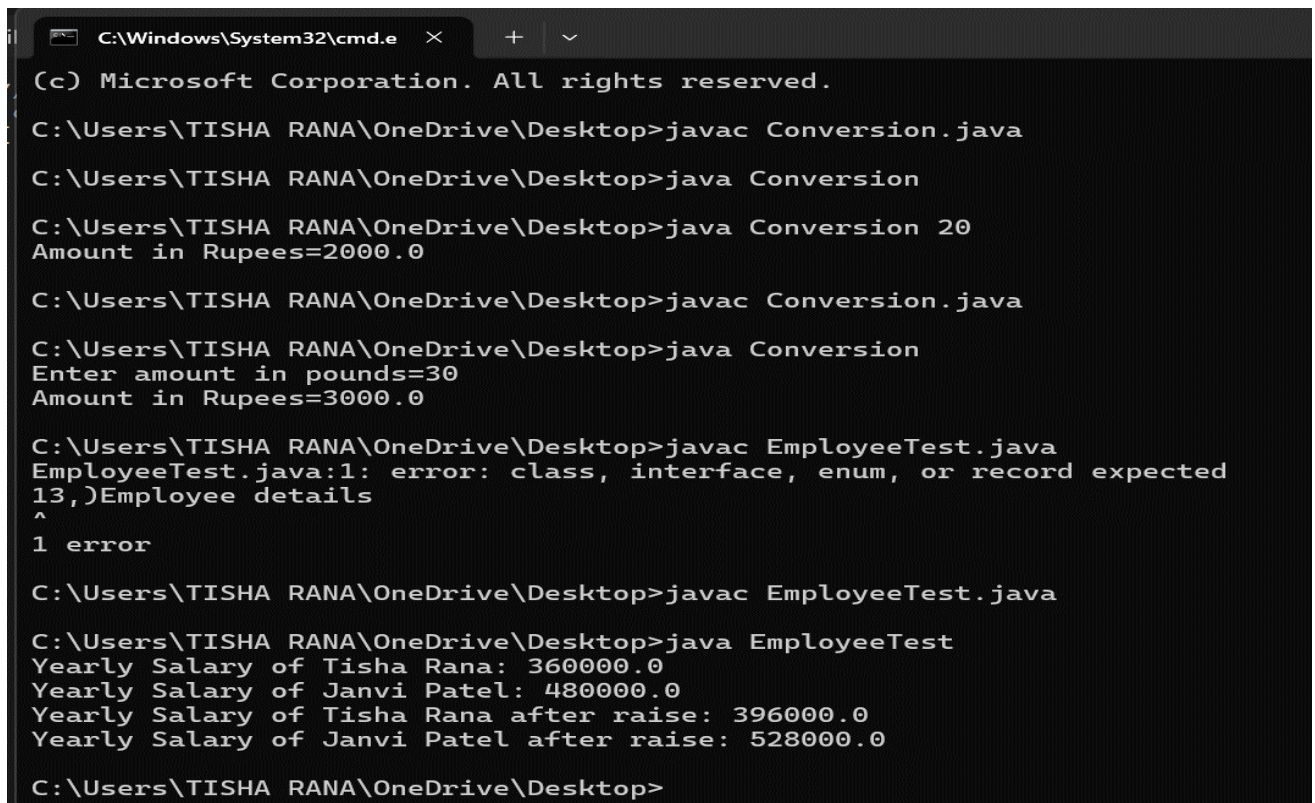
        System.out.println("Yearly Salary of " + emp1.getFirstName() + " " +
emp1.getLastName() + ": " + emp1.getYearlySalary());
        System.out.println("Yearly Salary of " + emp2.getFirstName() + " " +
emp2.getLastName() + ": " + emp2.getYearlySalary());

        emp1.giveRaise();
        emp2.giveRaise();

        System.out.println("Yearly Salary of " + emp1.getFirstName() + " " +
emp1.getLastName() + " after raise: " + emp1.getYearlySalary());
        System.out.println("Yearly Salary of " + emp2.getFirstName() + " " +
emp2.getLastName() + " after raise: " + emp2.getYearlySalary());
    }
}

```

OUTPUT:



```

C:\Windows\System32\cmd.e  x  +  v
(c) Microsoft Corporation. All rights reserved.
C:\Users\TISHA RANA\OneDrive\Desktop>javac Conversion.java
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion 20
Amount in Rupees=2000.0
C:\Users\TISHA RANA\OneDrive\Desktop>javac Conversion.java
C:\Users\TISHA RANA\OneDrive\Desktop>java Conversion
Enter amount in pounds=30
Amount in Rupees=3000.0
C:\Users\TISHA RANA\OneDrive\Desktop>javac EmployeeTest.java
EmployeeTest.java:1: error: class, interface, enum, or record expected
13,)Employee details
^
1 error
C:\Users\TISHA RANA\OneDrive\Desktop>javac EmployeeTest.java
C:\Users\TISHA RANA\OneDrive\Desktop>java EmployeeTest
Yearly Salary of Tisha Rana: 360000.0
Yearly Salary of Janvi Patel: 480000.0
Yearly Salary of Tisha Rana after raise: 396000.0
Yearly Salary of Janvi Patel after raise: 528000.0
C:\Users\TISHA RANA\OneDrive\Desktop>

```

CONCLUSION:

In the above practical we understood the concept of Constructors by developing Parameterized constructor to enter employee details and with the help of methods we calculated yearly salary and raise of each employee and displayed the employee details.

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

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

    void setYear(int year)
    {
        this.year = year;
    }
    int getMonth()
    {
        return month;
    }

    int getDay()
    {
```

```

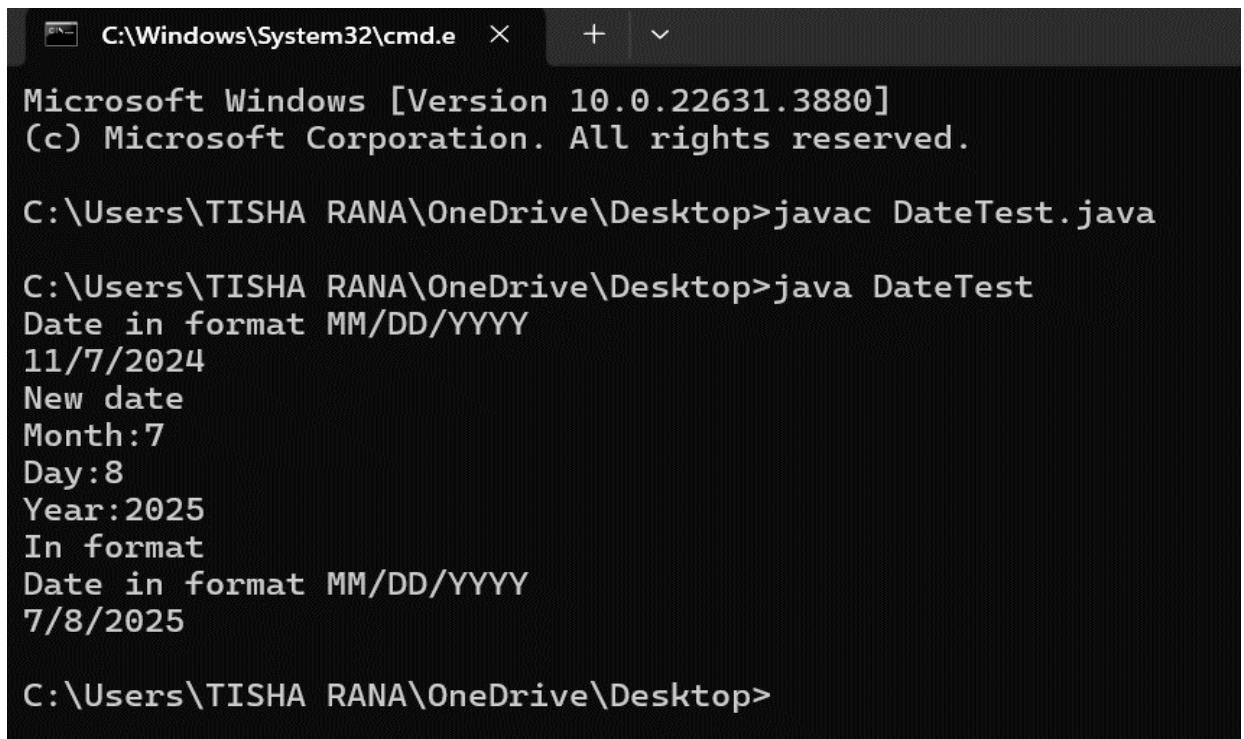
        return day;
    }

    int getYear()
    {
        return year;
    }

    void DisplayDate()
    {
        System.out.println("Date in format MM/DD/YYYY");
        System.out.println(month+ "/" +day+ "/" + year);
    }
}
class DateTest
{
    public static void main(String [] args)
    {
        Date d1=new Date(07,11,2024);
        d1.DisplayDate();
        System.out.println("New date");
        d1.setMonth(07);
        d1.setDay(8);
        d1.setYear(2025);
        System.out.println("Month:"+d1.getMonth());
        System.out.println("Day:"+d1.getDay());
        System.out.println("Year:"+d1.getYear());
        System.out.println("In format");
        d1.DisplayDate();
    }
}

```

OUTPUT

A screenshot of a Windows Command Prompt window. The title bar shows the path 'C:\Windows\System32\cmd.e' and standard window controls. The command prompt displays the following text:

```
Microsoft Windows [Version 10.0.22631.3880]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA\OneDrive\Desktop>javac DateTest.java

C:\Users\TISHA RANA\OneDrive\Desktop>java DateTest
Date in format MM/DD/YYYY
11/7/2024
New date
Month:7
Day:8
Year:2025
In format
Date in format MM/DD/YYYY
7/8/2025

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical we initialized the Date class variables using various constructors and using display method we displayed the date in required format.

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 length;
    int breadth;
    Area(int l,int b)
    {
        length=l;
        breadth=b;
    }
    int returnArea()
    {
        return length*breadth;
    }
    public static void main(String []args)
    {
        System.out.println("Enter length:");
        Scanner obj=new Scanner(System.in);
        int l=obj.nextInt();
        System.out.println("Enter breadth:");

        int b=obj.nextInt();
        Area rectangle= new Area(l,b);
        int a= rectangle.returnArea();
        System.out.println("Area of rectangle is:"+a);

    }
}
```

OUTPUT


```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Area.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Area
Enter length:
12
Enter breadth:
13
Area of rectangle is:156

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical we made parameterized constructor that takes user input for length and breadth of rectangle and a method that returns the calculated area of rectangle.

Supplementary Experiment:

1. Write a Java program to create a class called "Airplane" with a flight number, destination, and departure time attributes, and methods to check flight status and delay.
[L:M]

PROGRAM CODE:

```
//15.)Supplementary
import java.util.*;
class Airplane
{
    String Flightno;
    String Destination;
    String departure;
    Airplane(String Flightno,String Destination,String departure)
    {
        this.Flightno=Flightno;
        this.Destination=Destination;
        this.departure=departure;
    }
}
```

```

    }
    void delay(String s)
    {
        if(s==departure)
        {
            System.out.println("Flight is on time.");
        }
        else
        {
            System.out.println("Flight is not on time.");
        }
    }
    void Display()
    {
        System.out.println("Flight number="+Flightno);
        System.out.println("Flight destination="+Destination);
        System.out.println("Flight time="+departure);
    }
    public static void main(String [] args)
    {
        Airplane a1=new Airplane("CND207","PARIS","8:30 PM");
        Scanner obj=new Scanner(System.in);
        a1.Display();
        System.out.println("Enter flight time:");
        String s=obj.nextLine();
        a1.delay(s);
    }
}

```

OUTPUT

```
Microsoft Windows [Version 10.0.22631.3958]  
(c) Microsoft Corporation. All rights reserved.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>javac Airplane.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Airplane  
Flight number=CND207  
Flight destination=PARIS  
Flight time=8:30 PM  
Enter flight time:  
9:00  
Flight is not on time.  
  
C:\Users\TISHA RANA\OneDrive\Desktop>
```

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.*;

class Complex {
    int real;
    int img;

    // Default constructor
    Complex() {
    }

    // Parameterized constructor
    Complex(int real, int img) {
        this.real = real;
        this.img = img;
    }

    // Method to add two complex numbers
    Complex add(Complex c) {
        Complex t = new Complex();
        t.real = this.real + c.real;
        t.img = this.img + c.img;
        return t;
    }

    // Method to subtract two complex numbers
    Complex sub(Complex c) {
        Complex t = new Complex();
        t.real = this.real - c.real;
        t.img = this.img - c.img;
        return t;
    }

    // Method to multiply two complex numbers
    Complex mul(Complex c) {
        Complex t = new Complex();
        t.real = this.real * c.real - this.img * c.img;
        t.img = this.real * c.img + this.img * c.real;
        return t;
    }
}
```

```

    }

    // Method to display the complex number
    void display() {
        System.out.println(this.real + "+" + this.img + "i");
    }

    public static void main(String[] args) {
        Scanner obj = new Scanner(System.in);
        System.out.println("1st Complex No.");
        System.out.println("Enter complex no. real part:");
        int r = obj.nextInt();
        System.out.println("Enter complex no. img part:");
        int i = obj.nextInt();

        System.out.println("2nd Complex No.");
        System.out.println("Enter complex no. real part:");
        int r2 = obj.nextInt();
        System.out.println("Enter complex no. img part:");
        int i2 = obj.nextInt();

        Complex c1 = new Complex(r, i);
        c1.display();
        Complex c2 = new Complex(r2, i2);
        c2.display();

        Complex c3 = c1.add(c2);
        System.out.print("Sum: ");
        c3.display();

        Complex c4 = c1.sub(c2);
        System.out.print("Difference: ");
        c4.display();

        Complex c5 = c1.mul(c2);
        System.out.print("Product: ");
        c5.display();
    }
}

```

OUTPUT

```
C:\Users\TISHA RANA\OneDrive\Desktop>java Complex
1st Complex No.
Enter complex no. real part:
0
Enter complex no. img part:
0
2nd Complex No.
Enter complex no. real part:
1
Enter complex no. img part:
1
0+0i
1+1i
Sum: 1+1i
Difference: -1+-1i
Product: 0+0i
```

CONCLUSION:

In the above practical we performed mathematical calculation for two Complex Numbers given by user with the help of constructors and methods.

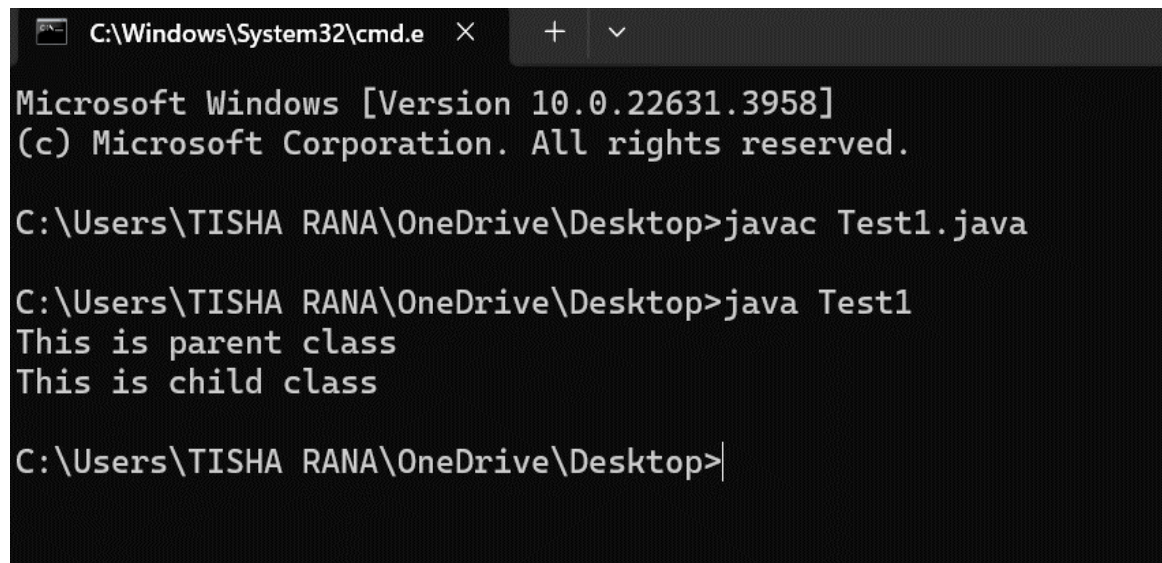
PART-IV Inheritance, Interface, Package

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:

```
//17.)Parent-child class
class Test1
{
public static void main(String []args)
{
Child c1=new Child();
c1.mssg1();
c1.mssg2();
}
}
class Parent
{
void mssg1()
{
System.out.println("This is parent class");
}
}
class Child extends Parent
{
void mssg2()
{
System.out.println("This is child class");
}
}
```

OUTPUT



```
C:\Windows\System32\cmd.e  ×  +  v
Microsoft Windows [Version 10.0.22631.3958]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TISHA RANA\OneDrive\Desktop>javac Test1.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Test1
This is parent class
This is child class

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical, we understood the concept of Single inheritance. And by making object of each class we called the method of parent class using these objects.

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;

Employee(String name, int age, String phoneNumber, String address, double salary, String
specialization) {
this.name = name;
this.age = age;
this.phoneNumber = phoneNumber;
this.address = address;
this.salary = salary;
this.specialization = specialization;
}

void display() {
System.out.println("Employee Details:");
System.out.println("Name: " + name);
```

```

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

class Manager extends Member {
String department;

Manager(String name, int age, String phoneNumber, String address, double salary, String
department) {
this.name = name;
this.age = age;
this.phoneNumber = phoneNumber;
this.address = address;
this.salary = salary;
this.department = department;
}

void display() {
System.out.println("Manager Details:");
System.out.println("Name: " + name);
System.out.println("Age: " + age);

System.out.println("Phone Number: " + phoneNumber);
System.out.println("Address: " + address);
System.out.println("Department: " + department);
printSalary();
}
}

public class Main {
public static void main(String[] args) {
Employee e1 = new Employee("Tisha Rana", 30, "1234567890", "123 Main St", 50000,
"IOT");

Manager m1 = new Manager("Hetvi Shah", 40, "0987654321", "456 Elm St", 75000, "IT");

e1.display();
System.out.println();
m1.display();
}
}

```

```
}  
}
```

OUTPUT

```
1 C:\Users\TISHA RANA\OneDrive\Desktop>javac Main.java  
t  
t  
t C:\Users\TISHA RANA\OneDrive\Desktop>java Main  
t Employee Details:  
t Name: Tisha Rana  
t Age: 30  
t Phone Number: 1234567890  
r Address: 123 Main St  
r Specialization: IOT  
r Salary: 50000.0  
  
3 Manager Details:  
3 Name: Hetvi Shah  
3 Age: 40  
3 Phone Number: 0987654321  
1 Address: 456 Elm St  
1 Department: IT  
1 Salary: 75000.0  
1
```

CONCLUSION:

In the above practical, we understood concept of Hierarchical Inheritance. We made a parent class “Member” which extends two child classes “Manager” and “Employee”. By Objects of these class we called methods of parent class also.

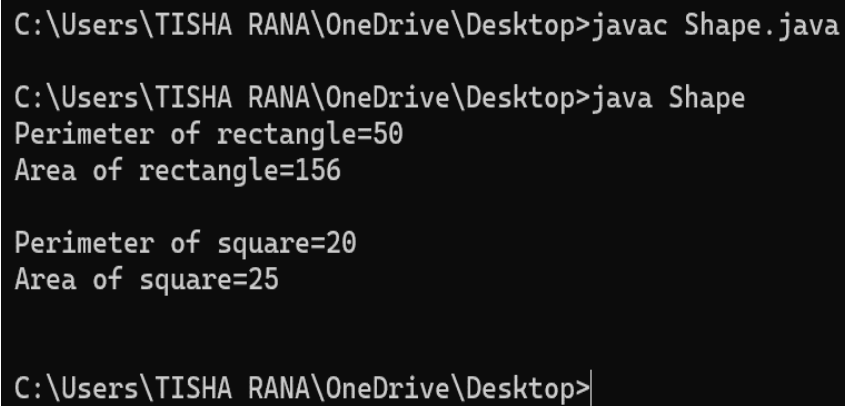
19. Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit 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 Shape
{
public static void main(String []args)
{
Rectangle []shape=new Rectangle[2];
shape[0]=new Rectangle(12,13);
shape[1]=new Square(5);
for(int i=0;i<2;i++)
{
shape[i].perimeter();
shape[i].area();
System.out.println();
}
}
}
class Rectangle
{
int length;
int width;
void area()
{
System.out.println("Area of rectangle="+length*width);
}
void perimeter()
{
System.out.println("Perimeter of rectangle="+2*(length+width));
}
Rectangle(int l,int w)
{
length=l;
width=w;
}
}
class Square extends Rectangle
{
```

```
Square(int side)
{
super(side,side);
}
void perimeter()
{
System.out.println("Perimeter of square="+4*length));
}
void area()
{
System.out.println("Area of square="+length*length));
}
}
```

OUTPUT



```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Shape.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Shape
Perimeter of rectangle=50
Area of rectangle=156

Perimeter of square=20
Area of square=25

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

In the above practical, we understood the concept of inheriting the constructors of Parent class to child class. Here the constructors of class Rectangle are extended in class Square also and respective methods are called by their objects in Class Shape.

Supplementary Experiment:

1.) Write a Java program to create a vehicle class hierarchy.

The base class should be Vehicle, with subclasses Truck, Car and Motorcycle. Each subclass should have properties such as make, model, year, and fuel type. Implement methods for calculating fuel efficiency, distance traveled, and maximum speed. [L:A]

PROGRAM CODE:

```
abstract class Vehicle {
String make;
String model;
int year;
String fuelType;
Vehicle(String make, String model, int year, String fuelType)
{
this.make = make;
this.model = model;
this.year = year;
this.fuelType = fuelType;
}

void displayDetails()
{
System.out.println("Make: " + make);
System.out.println("Model: " + model);
System.out.println("Year: " + year);
System.out.println("Fuel Type: " + fuelType);
}
}

class Truck extends Vehicle
{
double fuelEfficiency;
double maxSpeed;

Truck(String make, String model, int year, String fuelType, double fuelEfficiency, double
maxSpeed)
{
super(make, model, year, fuelType);
this.fuelEfficiency = fuelEfficiency;
this.maxSpeed = maxSpeed;
}
```

```

double calculateFuelEfficiency()
{
return fuelEfficiency;
}

double calculateDistanceTraveled(double fuelConsumed) {
return fuelEfficiency * fuelConsumed;
}

double getMaxSpeed() {
return maxSpeed;
}
}

class Car extends Vehicle
{
double fuelEfficiency;
double maxSpeed;

Car(String make, String model, int year, String fuelType, double fuelEfficiency, double
maxSpeed) {
super(make, model, year, fuelType);
this.fuelEfficiency = fuelEfficiency;
this.maxSpeed = maxSpeed;
}

double calculateFuelEfficiency()
{
return fuelEfficiency;
}

double calculateDistanceTraveled(double fuelConsumed)
{
return fuelEfficiency * fuelConsumed;
}

double getMaxSpeed()
{
return maxSpeed;
}
}

```

```

class Motorcycle extends Vehicle {
double fuelEfficiency;
double maxSpeed;

Motorcycle(String make, String model, int year, String fuelType, double fuelEfficiency,
double
maxSpeed)
{
super(make, model, year, fuelType);
this.fuelEfficiency = fuelEfficiency;
this.maxSpeed = maxSpeed;
}
double calculateFuelEfficiency()
{
return fuelEfficiency;
}

double calculateDistanceTraveled(double fuelConsumed)
{
return fuelEfficiency * fuelConsumed;
}

double getMaxSpeed()
{
return maxSpeed;
}
}

public class Motors {
public static void main(String[] args) {
Truck truck = new Truck("Ashok-Leyland", "F-150", 2020, "Diesel", 15, 120);
Car car = new Car("Hundyai", "Creta", 2022, "Petrol", 30, 130);
Motorcycle motorcycle = new Motorcycle("Harley-Davidson", "Sportster", 2019, "Petrol",
50, 110);

System.out.println("Truck Details:");
truck.displayDetails();
System.out.println("Fuel Efficiency: " + truck.calculateFuelEfficiency() + " miles per
gallon");
System.out.println("Distance Traveled with 10 gallons: " +
truck.calculateDistanceTraveled(10) + " miles");
System.out.println("Maximum Speed: " + truck.getMaxSpeed() + " mph");
System.out.println("*****");
}
}

```



```

System.out.println("Car Details:");
car.displayDetails();
System.out.println("Fuel Efficiency: " + car.calculateFuelEfficiency() + " miles per
gallon");
System.out.println("Distance Traveled with 10 gallons: " +
car.calculateDistanceTraveled(10) + " miles");
System.out.println("Maximum Speed: " + car.getMaxSpeed() + " mph");
System.out.println("*****");

System.out.println("Motorcycle Details:");
motorcycle.displayDetails();
System.out.println("Fuel Efficiency: " + motorcycle.calculateFuelEfficiency() + " miles per
gallon");
System.out.println("Distance Traveled with 10 gallons: " +
motorcycle.calculateDistanceTraveled(10) + " miles");
System.out.println("Maximum Speed: " + motorcycle.getMaxSpeed() + " mph");
System.out.println("*****");

}
}

```

OUTPUT

```

C:\Users\TISHA RANA\OneDrive\Desktop>java Motors
Truck Details:
Make: Ashok-Leyland
Model: F-150
Year: 2020
Fuel Type: Diesel
Fuel Efficiency: 15.0 miles per gallon
Distance Traveled with 10 gallons: 150.0 miles
Maximum Speed: 120.0 mph
*****
Car Details:
Make: Hundayi
Model: Creta
Year: 2022
Fuel Type: Petrol
Fuel Efficiency: 30.0 miles per gallon
Distance Traveled with 10 gallons: 300.0 miles
Maximum Speed: 130.0 mph
*****
Motorcycle Details:
Make: Harley-Davidson
Model: Sportster
Year: 2019
Fuel Type: Petrol
Fuel Efficiency: 50.0 miles per gallon
Distance Traveled with 10 gallons: 500.0 miles
Maximum Speed: 110.0 mph
*****

```

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

        Square square = new Square();

        square.printShape();
        square.printRectangle();
    }
}
```

```
}  
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Main.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Main  
This is shape  
This is rectangular shape
```

CONCLUSION:

From the above given practical we are implementing compile time polymorphism with help of inheritance. This demonstrates how inheritance works in Java, allowing a subclass to access methods from its superclass and providing additional functionality specific to the subclass.

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:

```
//21.)
class Degree {
    void getDegree() {
        System.out.println("I got a degree");
    }
}

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

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

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

        Degree d = new Degree();
        Undergraduate ug = new Undergraduate();
        Postgraduate pg = new Postgraduate();

        d.getDegree();
        ug.getDegree();
        pg.getDegree();
    }
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Graduate.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Graduate
I got a degree
I am an Undergraduate
I am a Postgraduate
```

CONCLUSION:

This shows the concept of polymorphism in inheritance, where a subclass can override a parent class method to provide its specific implementation. When a method is defined in a parent class and overridden in the subclasses, the method called is determined by the object type, not the reference type

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:

```
//22.)
interface AdvancedArithmetic
{
    int divisor_sum(int n);
}
class MyCalculator implements AdvancedArithmetic
{
    int s=0;
    public int divisor_sum(int n)
    {
        for(int i=1;i<=n;i++)
        {
            if(n%i==0)
            {
                s+=i;
            }
        }
        return s;
    }
}
class Divisor
{
    public static void main(String [] args)
    {
        MyCalculator a=new MyCalculator();
        int sum=a.divisor_sum(1000);
        System.out.println("Divisor sum="+sum);
    }
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Divisor.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java Divisor  
Divisor sum of 1000=2340
```

CONCLUSION:

We create an interface AdvancedArithmetic with a method divisor_sum(int n). Then, we create a class MyCalculator that implements this interface, providing the logic for summing divisors of a given number.

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();

    default void describe()
    {
        System.out.println("This is a " + getColor() + " shape.");
    }

    double calculateArea();
}

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 calculateArea()
{
    return Math.PI * radius * radius;
}

public double getRadius()
{
    return 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 calculateArea()
    {
        return length * width;
    }

    public double getLength()

```

```

    {
        return length;
    }

    public double getWidth()
    {
        return width;
    }
}

class Sign
{
    private Shape shape;
    private String text;

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

    public void displaySign()
    {
        System.out.println("Sign text: " + text);
        shape.describe();
        System.out.println("Shape area: " + shape.calculateArea());
    }
}

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

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

        Shape rectangle = new Rectangle(4, 6, "Blue");
    }
}

```

```
Sign circleSign = new Sign(circle, "Welcome to the Campus Center");  
Sign rectangleSign = new Sign(rectangle, "Library Hours");
```

```
circleSign.displaySign();  
System.out.println();  
rectangleSign.displaySign();  
}  
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac TestShape.java
```

```
C:\Users\TISHA RANA\OneDrive\Desktop>java TestShape
```

```
Sign text: Welcome to the Campus Center
```

```
This is a Red shape.
```

```
Shape area: 78.53981633974483
```

```
Sign text: Library Hours
```

```
This is a Blue shape.
```

```
Shape area: 24.0
```

CONCLUSION:

In this program, we create a hierarchy of shapes (Circle and Rectangle) using interfaces and demonstrate the significance of interface default methods. The interface provides a default method for calculating area or other functionalities that subclasses can either use as-is or override.

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:

//24.)

```
import java.util.InputMismatchException;
```

```
import java.util.Scanner;
```

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

```
class Exception
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
    Scanner sc = new Scanner(System.in);
```

```
    try {
```

```
        System.out.print("Enter x: ");
```

```
        int x = sc.nextInt();
```

```
        System.out.print("Enter y: ");
```

```
        int y = sc.nextInt();
```

```
        int result = x / y;
```

```
        System.out.println("Result of x / y is= " + result);
```

```
    }
```

```
    catch (InputMismatchException e)
```

```
    {
```

```
        System.out.println("Error! Enter valid integer.");
```

```
    }
```

```
    catch (ArithmeticException e)
```

```
    {
```

```
        System.out.println("Error! Division by zero is not allowed.");
```

```
    }
```

```
}
```

```
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Exception.java
```

```
C:\Users\TISHA RANA\OneDrive\Desktop>java Exception
```

```
Enter x: 2
```

```
Enter y: 0
```

```
Error! Division by zero is not allowed.
```

```
C:\Users\TISHA RANA\OneDrive\Desktop>java Exception
```

```
Enter x: x
```

```
Error! Enter valid integers.
```

CONCLUSION:

Given Java program that takes two integers x and y as input, and handles exceptions for invalid inputs (non-integer values or division by zero). This approach ensures the program doesn't crash on invalid input and provides clear feedback to the user.

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

PROGRAM:

```
//25.)
import java.io.*;
import java.util.*;

class TryCatch
{
    public static void main(String[] args)
    {
        int[] a = {1, 2, 3, 4, 5};
        try
        {
            System.out.println("a[1] = " + a[1]);
            System.out.println("a[6] = " + a[6]);
            System.out.println("2/0 = " + (2/0));
        }
        catch(ArithmeticException e)
        {
            System.out.println("Error! Integer division by zero.");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Error! Array index out of bounds.");
        }
        finally
        {
            System.out.println("Code executed.");
        }
    }
}
```

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>java TryCatch
a[1] = 2
Error! Array index out of bounds.
Code executed.
```

CONCLUSION:

The program demonstrates the use of multiple catch blocks to handle specific exceptions:
i.) `ArrayIndexOutOfBoundsException` is caught when attempting to access an invalid array index.

ii.) `ArithmeticException` is caught when attempting to divide by zero.

The finally block executes after the exception handling, ensuring that the message "Code executed." is always printed, whether an exception occurs or not.

26.

Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).

PROGRAM:

//26.)

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

```
class InvalidAgeException extends Exception
```

```
{
```

```
    public InvalidAgeException(String message)
```

```
    {
```

```
        super(message);
```

```
    }
```

```
}
```

```
public class ThrowThrows {
```

```
    public static void validateAge(int age) throws InvalidAgeException
```

```
{
```

```
    if (age < 18)
```

```
    {
```

```
        throw new InvalidAgeException("Age must be at least 18.");
```

```
    }
```

```
    System.out.println("Age is valid.");
```

```
}
```

```
    public static void main(String[] args)
```

```
{
```

```
        Scanner sc = new Scanner(System.in);
```

```
        try
```

```
        {
```

```
            System.out.println("Enter Age:");
```

```
            int x = sc.nextInt();
```

```
            validateAge(x);
```

```
        }
```

```
        catch (InvalidAgeException e)
```

```
        {
```



```
        System.out.println("Caught Exception: " + e.getMessage());
    }
    catch (InputMismatchException e)
    {

        System.out.println("Invalid input! Please enter a valid integer.");
    }

    finally
    {
        sc.close();
    }
}
```

OUTPUT:

```
java -cp /tmp/AEoqqFITFb/ThrowThrows
Enter Age:
x
Invalid input! Please enter a valid integer.

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

CONCLUSION:

From the above given code we understand the concept of Custom Exception handling where we create a custom exception and handle it using throw and throws keywords. By these concepts, you can write more reliable and maintainable Java code, ensuring that exceptional conditions are properly handled without disrupting the program's normal flow.

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:

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

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

        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.out.println(fileName + ": " + e.getMessage());
            }
        }
    }
}
```

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

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>java LineCounts File1.txt File2 File3.txt
File1.txt: 4 lines
Error reading file: File2 - File2 (The system cannot find the file specified)
File3.txt: 5 lines
```

CONCLUSION:

The provided Java program effectively counts the number of lines in each specified text file passed as command-line arguments. It reads through each file, utilizing a `BufferedReader` to handle potentially large files efficiently. If any errors occur (such as a file not being found), the program captures these exceptions and prints an error message while continuing to process the remaining files. This robust error handling ensures that users receive feedback on all specified files, promoting better usability and reliability in scenarios where multiple files are processed simultaneously. Such a program is particularly useful for batch file processing and can be extended for further functionalities like counting words or characters, making it a versatile tool for text file analysis.

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:

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

public class CharacterCount
{
    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Usage: java CharacterCounter <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 currentChar;

            while ((currentChar = reader.read()) != -1)
            {
                if (currentChar == targetChar)
                {
                    charCount++;
                }
            }

            System.out.println("The character '" + targetChar + "' appears " + charCount + "
times in " + fileName);
        }
        catch (IOException e)
        {

```

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

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>java CharacterCount File1.txt a
The character 'a' appears 5 times in File1.txt

C:\Users\TISHA RANA\OneDrive\Desktop>
```

CONCLUSION:

The Java program designed to count the occurrences of a specified character in a given text file effectively demonstrates how to handle file input and command-line arguments. By reading through the file line by line and checking each character, it provides an accurate count of the specified character's appearances. This approach not only showcases the ability to process text files but also emphasizes the flexibility of command-line arguments, allowing users to customize their queries easily. Such functionality can be particularly useful in text analysis, helping users gain insights into character frequency, which may be important for applications like data processing, linguistic analysis, or even cryptography.

4o mini

29.

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

PROGRAM:

//29.)

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

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

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

```
public class WordSearch
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
    if (args.length < 2)
```

```
    {
```

```
        System.out.println("Mention java WordSearch <filename> <word>");
```

```
        return;
```

```
    }
```

```
    String fileName = args[0];
```

```
    String searchWord = args[1];
```

```
    int wordCount = 0;
```

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

```
    {
```

```
        String line;
```

```
        while ((line = reader.readLine()) != null)
```

```
        {
```

```
            if (line.contains(searchWord))
```

```
            {
```

```
                wordCount++;
```

```
            }
```

```
        }
```

```
        Integer count = Integer.valueOf(wordCount);
```

```
        System.out.println("The word " + searchWord + " appears " + count + " time(s) in " + fileName);
```

```
    }
```

```
    catch (IOException e)
```

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

OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>java WordSearch File1.txt java  
The word 'java' appears 5 time(s) in File1.txt
```

```
C:\Users\TISHA RANA\OneDrive\Desktop>>java WordSearch  
'WordSearch' is not recognized as an internal or external command,  
operable program or batch file.
```

```
C:\Users\TISHA RANA\OneDrive\Desktop>java WordSearch File1.txt  
Usage: java WordSearch <file> <word>
```

CONCLUSION:

The Java program for searching a given word in a file demonstrates effective file handling and string manipulation techniques. By utilizing a `BufferedReader`, the program efficiently reads through the file line by line, allowing for quick searches while minimizing memory usage. The implementation of a wrapper class is illustrated through the use of `Integer`, showcasing how primitive data types can be treated as objects, which is essential for various operations like handling null values and utilizing collection frameworks.

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:

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

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

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

        try (FileReader fileReader = new FileReader(sourceFile);
            FileWriter fileWriter = new FileWriter(destinationFile))
        {

            int ch;

            while ((ch = fileReader.read()) != -1)
            {
                fileWriter.write(ch);
            }

            System.out.println("Data has been copied from " + sourceFile + " to " +
destinationFile);

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


OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac FileCopy.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java FileCopy File1.txt DestinationFile.txt  
Data has been copied from File1.txt to DestinationFile.txt
```

CONCLUSION:

The Java program designed to copy data from one file to another effectively demonstrates file handling capabilities within the language. By utilizing `FileReader` and `FileWriter` in conjunction with `BufferedReader` and `BufferedWriter`, the program efficiently reads data from the source file and writes it to the destination file. Importantly, it includes functionality to automatically create the destination file if it does not already exist, enhancing usability and flexibility. This program not only showcases basic file I/O operations but also highlights Java's robust error handling, ensuring that the process remains smooth even in cases where files may not be readily available.

31.

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

PROGRAM:

//P-31.)

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

```
public class StreamClass
```

```
{
```

```
    public static void main(String[] args)
```

```
    {
```

```
        System.out.println("==== Character Stream =====");
```

```
        copyCharacterStream("File1.txt", "destinationChar.txt"); // Reads char from File1 and
writes in destination file
```

```
        System.out.println("==== Byte Stream =====");
```

```
        copyByteStream("File1.txt", "destinationByte.txt"); // Reads byte from File1 and
writes in destination file
```

```
        System.out.println("==== BufferedReader and BufferedWriter =====");
```

```
        writeConsoleToFile("consoleOutput.txt"); // Reads from Console screen and writes in
File
```

```
    }
```

```
    public static void copyCharacterStream(String File1, String destinationFile)
```

```
    {
```

```
        try (FileReader reader = new FileReader(File1);
```

```
            FileWriter writer = new FileWriter(destinationFile))
```

```
        {
```

```
            int character;
```

```
            while ((character = reader.read()) != -1)
```

```
            {
```

```
                writer.write(character);
```

```
            }
```

```
            System.out.println("Character stream copy completed from " + File1 + " to " +
destinationFile);
```

```
        } catch (IOException e)
```

```
        {
```

```

        System.err.println("Error in character stream: " + e.getMessage());
    }
}

public static void copyByteStream(String File1, String destinationFile)
{
    try (FileInputStream inputStream = new FileInputStream(File1);
        FileOutputStream outputStream = new FileOutputStream(destinationFile))
    {

        int byteData;
        while ((byteData = inputStream.read()) != -1)
        {
            outputStream.write(byteData);
        }
        System.out.println("Byte stream copy completed from " + File1 + " to " +
destinationFile);

    }
    catch (IOException e)
    {
        System.err.println("Error in byte stream: " + e.getMessage());
    }
}

public static void writeConsoleToFile(String outputFile)
{
    try (BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(outputFile)))
    {

        System.out.println("Enter text (type 'exit' to finish):");

        String line;
        while (!(line = bufferedReader.readLine()).equalsIgnoreCase("exit"))
        {
            bufferedWriter.write(line);
            bufferedWriter.newLine();
        }
        System.out.println("Text from console has been written to " + outputFile);
    }
}

```

```

    }
    catch (IOException e)
    {
        System.err.println("Error in buffered I/O: " + e.getMessage());
    }
}
}

```

OUTPUT:

```

C:\Users\TISHA RANA\OneDrive\Desktop>javac StreamClass.java

C:\Users\TISHA RANA\OneDrive\Desktop>java StreamClass
===== Character Stream =====
Character stream copy completed from File1.txt to destinationChar.txt
===== Byte Stream =====
Byte stream copy completed from File1.txt to destinationByte.txt
===== BufferedReader and BufferedWriter =====
Enter text (type 'exit' to finish):
Hello!
Welcome to world of Java
exit
Text from console has been written to consoleOutput.txt

```

CONCLUSION:

The Java program demonstrating the use of character and byte streams effectively illustrates the differences between these two types of data handling. Character streams, such as `BufferedReader` and `BufferedWriter`, are utilized for reading and writing text data, providing a convenient way to handle character encoding and ensure efficient input and output operations. The program reads user input from the console and writes it to a file, showcasing how buffered streams can enhance performance by reducing the number of I/O operations.

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:

//P-31.)

//A.)

class A extends Thread

```
{
    public void run()
    {
        System.out.println("Hello World");
    }
    public static void main(String[] args)
    {
        System.out.println("Message:");
        A thread=new A();
        thread.start();
    }
}
```

//B.)

//P-40.)

class B implements Runnable

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

    public static void main(String[] args)
    {
        System.out.println("Message:");

        B mythread = new B();

        Thread thread = new Thread(mythread);
```

```
        thread.start();  
    }  
}
```

OUTPUT:

A.)

```
C:\Users\TISHA RANA\OneDrive\Desktop>java A  
Message:  
Hello World
```

B.)

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac B.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java B  
Message:  
Hello World
```

CONCLUSION:

A. By Extending the Thread Class: The program that displays "Hello World" by extending the Thread class showcases the simplicity of creating threads in Java. By overriding the run() method, it directly utilizes the thread's built-in functionalities. This approach is straightforward and suitable for tasks where the thread itself is the main entity, making it easy to understand for beginners.

B. By Using the Runnable Interface: The program that implements the Runnable interface demonstrates a more flexible way to create threads. This approach allows for better separation of thread logic from the thread execution, enabling multiple threads to share the same task or logic. By implementing Runnable, it also facilitates the use of thread pools and enhances reusability, making it a preferred choice for more complex applications.

33.

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

PROGRAM:

//p-32.)

class SumTask implements Runnable

```
{
    private int start;
    private int end;
    private long result;

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

    public long getResult()
    {
        return result;
    }

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

public class ThreadSum

```
{
    public static void main(String[] args)
    {
        if (args.length < 2)
        {
            System.out.println("Usage: java ThreadedSum <N> <number of threads>");
            return;
        }
    }
}
```

```

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

if (N <= 0 || numThreads <= 0)
{
    System.out.println("N and number of threads must be positive integers.");
    return;
}

long totalSum = 0;
int range = N / numThreads;
int remainder = N % numThreads;

Thread[] threads = new Thread[numThreads];

for (int i = 0; i < numThreads; i++)
{
    int start = i * range + 1;
    int end = (i == numThreads - 1) ? start + range + remainder - 1 : start + range - 1;

    SumTask task = new SumTask(start, end);
    threads[i] = new Thread(task);
    threads[i].start();

    try
    {
        threads[i].join();
        totalSum += task.getResult();
    }
    catch (InterruptedException e)
    {
        e.printStackTrace();
    }
}

System.out.println("The sum of the first " + N + " natural numbers is: " + totalSum);
}

```


OUTPUT:

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac ThreadSum.java  
  
C:\Users\TISHA RANA\OneDrive\Desktop>java ThreadSum 10 5  
The sum of the first 10 natural numbers is: 55
```

CONCLUSION:

The program that takes an integer NNN and the number of threads as arguments effectively illustrates how to distribute tasks among multiple threads in Java. By dividing the summation task, it optimizes performance and demonstrates the power of multi-threading for parallel computation. This program highlights the efficiency gained by breaking down larger tasks, enabling faster execution and better resource utilization in computational applications.

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:

```
import java.util.Random;
```

```
// Thread to generate random numbers
```

```
class NumberGenerator extends Thread {
```

```
    public void run() {
```

```
        Random random = new Random();
```

```
        while (true) {
```

```
            int number = random.nextInt(100); // Generate a random integer between 0 and 99
```

```
            System.out.println("Generated Number: " + number);
```

```
            if (number % 2 == 0) {
```

```
                new SquareCalculator(number).start();
```

```
            } else {
```

```
                new CubeCalculator(number).start();
```

```
            }
```

```
            try {
```

```
                Thread.sleep(1000); // Wait for 1 second
```

```
            } catch (InterruptedException e) {
```

```
                System.out.println("Number generation interrupted");
```

```
            }
```

```
        }
```

```
    }
```

```
}
```

```
// Thread to calculate the square of a number
```

```
class SquareCalculator extends Thread {
```

```
    private int number;
```

```
    public SquareCalculator(int number) {
```

```
        this.number = number;
```

```
    }
```

```
    public void run() {
```

```
        int square = number * number;
```

```
        System.out.println("Square of " + number + " is: " + square);
```

```
    }
```

```
}
```

```
// Thread to calculate the cube of a number
class CubeCalculator extends Thread {
    private int number;

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

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

// Main class to start the threads
public class MultiThreadApp {
    public static void main(String[] args) {
        NumberGenerator generator = new NumberGenerator();
        generator.start();
    }
}
```

OUTPUT:

```
Generated Number: 23
Cube of 23 is: 12167
Generated Number: 42
Square of 42 is: 1764
Generated Number: 17
Cube of 17 is: 4913
Generated Number: 68
Square of 68 is: 4624
Generated Number: 9
Cube of 9 is: 729
Generated Number: 12
Square of 12 is: 144
```

CONCLUSION:

The Java program that implements a multi-threaded application with three threads showcases the dynamic capabilities of threads in processing varying tasks. The first thread generates random integers, while the second and third threads handle even and odd values, respectively, performing distinct calculations (square and cube). This program emphasizes the importance of inter-thread communication and coordination, demonstrating how threads can work together to perform related but distinct tasks. It serves as an excellent example of how multi-threading can be effectively utilized to manage concurrent operations, enhancing application responsiveness and efficiency.

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:

```
// Thread class to increment the variable
class IncrementThread extends Thread {
    private int number = 0; // Variable to increment

    public void run() {
        while (true) {
            number++; // Increment the variable by 1
            System.out.println("Value of number: " + number);
            try {
                Thread.sleep(1000); // Pause for 1 second
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
}

// Main class to start the thread
public class IncrementValueApp {
    public static void main(String[] args) {
        IncrementThread incrementThread = new IncrementThread();
        incrementThread.start(); // Start the thread
    }
}
```

OUTPUT:

```
Value of number: 1  
Value of number: 2  
Value of number: 3  
Value of number: 4  
Value of number: 5  
Value of number: 6  
Value of number: 7  
Value of number: 8
```

CONCLUSION:

This program effectively demonstrates how to use threads in Java to perform a task asynchronously. By creating a thread that sleeps for one second before incrementing a variable, it showcases the use of the `sleep()` method to introduce delays in execution. This approach can be particularly useful in scenarios where timing and scheduling of tasks are critical. The separation of the increment logic into a thread enhances application responsiveness and illustrates basic thread management principles, making it an excellent introduction to multi-threading concepts.

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:

// Thread class to display thread information

```
class MyThread extends Thread {
    public MyThread(String name) {
        super(name); // Set the name of the thread
    }

    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println(getName() + " is running with priority " + getPriority());
            try {
                Thread.sleep(500); // Pause for half a second
            } catch (InterruptedException e) {
                System.out.println(getName() + " interrupted");
            }
        }
    }
}
```

// Main class to create and set thread priorities

```
public class ThreadPriorityExample {
    public static void main(String[] args) {
        // Creating threads
        MyThread thread1 = new MyThread("FIRST");
        MyThread thread2 = new MyThread("SECOND");
        MyThread thread3 = new MyThread("THIRD");

        // Setting thread priorities
        thread1.setPriority(3); // Priority of FIRST thread set to 3
        thread2.setPriority(5); // Priority of SECOND thread remains 5 (default)
        thread3.setPriority(7); // Priority of THIRD thread set to 7

        // Starting threads
        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

OUTPUT:

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

CONCLUSION:

This program illustrates how to set and manage thread priorities in Java, highlighting the ability to influence the execution order of threads. By creating three threads with different priorities, it showcases how Java allows developers to customize the scheduling of threads based on their importance. Although thread scheduling is ultimately managed by the Java Virtual Machine (JVM), this program helps to understand the concept of priorities in multi-threading and can be useful in scenarios where certain tasks need to be prioritized over others, promoting efficient resource management.

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

PROGRAM:

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

// Producer class
class Producer implements Runnable {
    private final Queue<Integer> buffer;
    private final int maxSize;

    public Producer(Queue<Integer> buffer, int maxSize) {
        this.buffer = buffer;
        this.maxSize = maxSize;
    }

    @Override
    public void run() {
        int value = 0;
        while (true) {
            synchronized (buffer) {
                while (buffer.size() == maxSize) {
                    try {
                        System.out.println("Buffer is full, Producer is waiting...");
                        buffer.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                System.out.println("Producer produced: " + value);
                buffer.add(value++);
                buffer.notifyAll(); // Notify the Consumer that an item is available
            }
            try {
                Thread.sleep(500); // Simulate time taken to produce an item
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
// Consumer class
class Consumer implements Runnable {
    private final Queue<Integer> buffer;

    public Consumer(Queue<Integer> buffer) {
        this.buffer = buffer;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (buffer) {
                while (buffer.isEmpty()) {
                    try {
                        System.out.println("Buffer is empty, Consumer is waiting...");
                        buffer.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                int value = buffer.remove();
                System.out.println("Consumer consumed: " + value);
                buffer.notifyAll(); // Notify the Producer that there's space available
            }
            try {
                Thread.sleep(1000); // Simulate time taken to consume an item
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

// Main class to start Producer and Consumer threads
public class ProducerConsumerProblem {
    public static void main(String[] args) {
        Queue<Integer> buffer = new LinkedList<>();
        int maxSize = 5; // Maximum size of the buffer

        Thread producerThread = new Thread(new Producer(buffer, maxSize), "Producer");
        Thread consumerThread = new Thread(new Consumer(buffer), "Consumer");

        producerThread.start();
    }
}
```

```
        consumerThread.start();  
    }  
}
```

OUTPUT:

```
Producer produced: 0  
Producer produced: 1  
Producer produced: 2  
Producer produced: 3  
Producer produced: 4  
Buffer is full, Producer is waiting...  
Consumer consumed: 0  
Consumer consumed: 1  
Producer produced: 5  
Producer produced: 6  
Buffer is full, Producer is waiting...  
Consumer consumed: 2  
Consumer consumed: 3  
Producer produced: 7
```

CONCLUSION:

This program provides a robust solution to the producer-consumer problem, demonstrating the importance of thread synchronization in multi-threaded applications. By using synchronized methods and wait-notify mechanisms, it ensures that producers and consumers operate safely on a shared resource (the queue) without causing race conditions or deadlocks. This example is crucial for understanding how to manage shared data between threads effectively, and it showcases practical applications of Java's concurrency utilities. The implementation emphasizes the balance between producing and consuming resources, a common challenge in concurrent programming.

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:

```
import java.util.ArrayList;

public class MyStack {
    // List to store the stack elements
    private ArrayList<Object> list = new ArrayList<>();

    // Check if the stack is empty
    public boolean isEmpty() {
        return list.isEmpty();
    }

    // Get the size of the stack
    public int getSize() {
        return list.size();
    }

    // Peek at the top element without removing it
    public Object peek() {
        if (!isEmpty()) {
            return list.get(list.size() - 1);
        } else {
            return "Stack is empty";
        }
    }

    // Pop the top element and remove it from the stack
    public Object pop() {
        if (!isEmpty()) {
            return list.remove(list.size() - 1);
        }
    }
}
```

```

    } else {
        return "Stack is empty";
    }
}

// Push a new element onto the stack
public void push(Object o) {
    list.add(o);
    System.out.println("Pushed " + o + " onto the stack");
}

// Main method to demonstrate stack operations
public static void main(String[] args) {
    MyStack stack = new MyStack();

    // Check if stack is empty
    System.out.println("Is stack empty? " + stack.isEmpty());

    // Push elements onto the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);

    // Get the size of the stack
    System.out.println("Size of stack: " + stack.getSize());

    // Peek at the top element
    System.out.println("Top element (peek): " + stack.peek());

    // Pop an element from the stack
    System.out.println("Popped element: " + stack.pop());

    // Check the size of the stack after popping
    System.out.println("Size of stack after pop: " + stack.getSize());

    // Peek at the top element again
    System.out.println("Top element after pop (peek): " + stack.peek());

    // Check if stack is empty
    System.out.println("Is stack empty? " + stack.isEmpty());
}
}

```

OUTPUT:

```
Is stack empty? true
Pushed 10 onto the stack
Pushed 20 onto the stack
Pushed 30 onto the stack
Size of stack: 3
Top element (peek): 30
Popped element: 30
Size of stack after pop: 2
Top element after pop (peek): 20
Is stack empty? false
```

CONCLUSION:

The custom stack implementation using the ArrayList class provides a flexible and dynamic way to store elements in a last-in-first-out (LIFO) manner. The methods isEmpty, getSize, peek, pop, and push encapsulate the stack's core functionalities, allowing easy management of the stack's state. This implementation is particularly useful in scenarios where managing temporary data is necessary, such as during algorithm execution or expression evaluation. The use of ArrayList also ensures that the stack can grow dynamically as needed, making it a robust choice for various applications.

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:

```
import java.util.Arrays;

// Product class that implements the Comparable interface
class Product implements Comparable<Product> {
    private String name;
    private double price;
    private double rating;

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

    // Getter methods
    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public double getRating() {
        return rating;
    }

    // Compare products based on price (natural ordering)
    @Override
    public int compareTo(Product other) {
        return Double.compare(this.price, other.price);
    }

    @Override
    public String toString() {
```

```

        return "Product{name='" + name + "', price='" + price + "', rating='" + rating + "'}";
    }
}

// Generic method to sort an array of Comparable objects
public class GenericSortExample {

    public static <T extends Comparable<T>> void sortArray(T[] array) {
        Arrays.sort(array); // Sort using natural ordering
    }

    public static void main(String[] args) {
        // Create an array of Product objects
        Product[] products = {
            new Product("Laptop", 1200.99, 4.5),
            new Product("Smartphone", 800.50, 4.7),
            new Product("Tablet", 300.75, 4.3),
            new Product("Headphones", 150.30, 4.6)
        };

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

        // Sort the products array using the generic method
        sortArray(products);

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

```


OUTPUT:

```
Products before sorting:
Product{name='Laptop', price=1200.99, rating=4.5}
Product{name='Smartphone', price=800.5, rating=4.7}
Product{name='Tablet', price=300.75, rating=4.3}
Product{name='Headphones', price=150.3, rating=4.6}

Products after sorting by price (natural order):
Product{name='Headphones', price=150.3, rating=4.6}
Product{name='Tablet', price=300.75, rating=4.3}
Product{name='Smartphone', price=800.5, rating=4.7}
Product{name='Laptop', price=1200.99, rating=4.5}
```

CONCLUSION:

The generic sorting method implemented in the GenericSorter class leverages Java's type system to create a versatile utility that can sort any array of objects implementing the Comparable interface. This approach not only promotes code reusability and flexibility but also allows for easy integration with various data types, such as integers, strings, or custom objects like products or orders. By using Java's built-in sorting capabilities, the method ensures efficient and reliable sorting, making it an essential tool in applications requiring dynamic data management and sorting functionality.

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

public class Per_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:**

The word count program effectively demonstrates how to utilize Java's Map and Set classes to count and organize word occurrences in a text file. By using a HashMap to store the words and their counts, the program provides an efficient way to manage data, allowing for quick lookups and updates. The subsequent sorting of the words using a TreeMap ensures that the output is presented in alphabetical order, enhancing readability. This approach is invaluable for applications involving text analysis, providing a foundation for more complex data processing tasks.

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

**PROGRAM CODE:**

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

public class Per_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:**

```
C:\Users\TISHA RANA\OneDrive\Desktop>javac Per_41.java

C:\Users\TISHA RANA\OneDrive\Desktop>java Per_41
Number of keywords: 57

C:\Users\TISHA RANA\OneDrive\Desktop>|
```

### **CONCLUSION:**

The keyword count program demonstrates an effective way to analyze a Java source file for its keywords using a HashSet. By storing the Java keywords in the set, the program enables fast membership checking using the contains() method, allowing for efficient counting of keywords. The implementation highlights the importance of collections in managing groups of data, especially in programming contexts where performance and accuracy are critical. This program is particularly useful for developers looking to analyze and understand code quality or conduct static code analysis in their projects.