



Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./Mrs. Patel Arya Niravkumar

of 3 CSE - 1 Class,

ID. No. 23DCS070 has satisfactorily completed

his/ her term work in CSE201 - Java Programming for

the ending in Nov. 2024 /2025

Date : 17/10/2024

Amaravati

Sign. of Faculty

G

Head of Department

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Department of Computer Science & Engineering

Subject Name: JAVA PROGRAMMING

Semester: 3

Subject Code: CSE201

Academic year: 2024-25

Part - 1

No .	Aim of the Practical
1.	<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>=>Steps:</p> <p>Step-1:Download JDK (Java Development Kit): Go to the Oracle JDK download page or the OpenJDK page. Choose the appropriate version for your operating system (Windows, macOS, or Linux). Download the installer.</p> <p>Step-2:Install JDK: Run the downloaded installer. Follow the installation instructions. The default settings are usually fine. During installation, note the installation path. You might need it later.</p> <p>Step-3:Set Environment Variables (Windows): Go to Control Panel > System and Security > System > Advanced system settings. Click on Environment Variables. Under System variables, click New and add a new variable named JAVA_HOME with the value of the JDK installation path. Find the Path variable in the System variables section, select it, and click Edit. Add a new entry with the path to the bin directory inside your JDK installation directory (e.g., C:\Program Files\Java\jdk-11\bin). Verify Installation: Open a command prompt (or terminal on macOS/Linux). Type java -version and javac -version to check if Java and the Java compiler are properly installed.</p>

=>Comparison:

Java	C++
1. Platform-independent, Java bytecode works on any operating system.	1. Platform dependent, should be compiled for different platforms.
2. Java is both Compiled and Interpreted Language.	2.C++ is a Compiled Language.
3.It has limited support for pointers.	3.It strongly supports pointers.
4.Java uses the (System class): System.in for input and System.out for output.	4.C++ uses cin for input and cout for an output operation.

JDK: JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JRE: JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

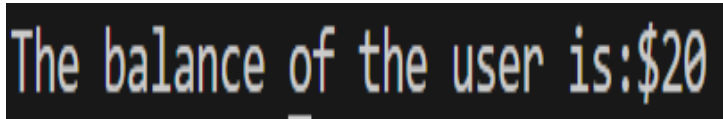
JVM: JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

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:

```
class prac12{  
    public static void main(String[] args)  
    {  
        String b="$20";  
        System.out.println("The balance of the user is:"+b);  
    }  
}
```

OUTPUT:



CONCLUSION:

I learnt about storing value in variable and print that value in java language. In short I learnt about output statement.

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

PROGRAM CODE:

```
import java.util.Scanner;
Class prac3
{
public static void main(String[] args)
{

float sp,spk,spm;
Scanner myObj = new Scanner(System.in);
DecimalFormat df=new DecimalFormat("#.##");

System.out.println("Enter distance in meters");
float distance= myObj.nextFloat();
System.out.println("Enter time in hours");
int th= myObj.nextInt();

System.out.println("Enter time in min");
int tm= myObj.nextInt();

System.out.println("Enter time in sec");
int ts= myObj.nextInt();

float km=distance/1000f;
System.out.println("Distance in km :"+km);

float ml=distance/1609f;
System.out.println("Distance in miles :"+df.format(ml));

sp=distance/ts;
System.out.println("Speed in m/s:" + df.format(sp));

spk=km/th;
System.out.println("Speed in km/h:" + df.format(spk));

spm=ml/th;
System.out.println("Speed in ml/h:" +df.format(spm));
}
}
```

OUTPUT:

```
java -cp /tmp/pazY4ucIzH/Main
Enter distance in meters
5000
Enter time in hours
8
Enter time in min
180
Enter time in sec
1600
Distance in km :5.0
Distance in miles :3.11
Speed in m/s:3.12
Speed in km/h:0.62
Speed in ml/h:0.39
```

CONCLUSION:

I learnt that how we can take input from the user in java and print the values in float rounded notation .also I learnt about decimalformat.

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

PROGRAM CODE:

```
import java.util.*;
class prac14 {
    public static void main(String[] args) {
        Scanner obj=new Scanner(System.in);
        System.out.println("Enter the number of days ");
        int n=obj.nextInt();
        float arr[]=new float[n];
        for(int i=0;i<n;i++)
        {
            System.out.print("Enter the daily expense of day:"+(i+1)+"
");
            arr[i]=obj.nextFloat();
        }
        float sum=0;
        for(int j=0;j<n;j++)
        {
            sum+=arr[j];
        }
        System.out.println("The sum of the daily expenses="+sum+"
rupees");
    }
}
```

OUTPUT:

```
java -cp /tmp/nh6WUxhFIL/Main
Enter the number of days
5
Enter the daily expense of day:1 80
Enter the daily expense of day:2 160
Enter the daily expense of day:3 45
Enter the daily expense of day:4 820
Enter the daily expense of day:5 456
The sum of the daily expenses=1561.0 rupees

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

CONCLUSION:

By performing this practical I learnt that how we can do various operations on array in java.

5.

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

PROGRAM CODE:

```
class prac15 {
    public static void main(String[] args) {
        int Code[]={ 1,2,3,4,5 };
        double tax=0,total=0,itembill;
        float Price[]={ 300,2000,40,10,200 };

        for(int i=0;i<Code.length;i++)
        {
            int code=Code[i];
            float price=Price[i];
            switch(code)
            {
                case 1:
                    tax=0.08*price;
                    break;
                case 2:
                    tax=0.12*price;
                    break;
                case 3:
                    tax=0.05*price;
                    break;
                case 4:
                    tax=0.075*price;
                    break;
                case 5:
                    tax=0.03*price;
                    break;
            }
            itembill=price+tax;
            System.out.println("Code:"+code+", price of item: "+itembill);
            total+=itembill;
        }
    }
}
```

```
System.out.println("The total amount of electronic products  
is:"+total);  
}  
}
```

OUTPUT:

```
Code:1, price of item: 324.0  
Code:2, price of item: 2240.0  
Code:3, price of item: 42.0  
Code:4, price of item: 10.75  
Code:5, price of item: 206.0  
The total amount of electronic products is:2822.75  
  
=== Code Execution Successful ===
```

CONCLUSION:

From this practical I learnt about switch case in java and how array can be executed inside looping structure.

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

PROGRAM CODE:

```
import java.util.*;
class days {
    public static void main(String[] args) {
        int n;
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter the number of days");
        n = obj.nextInt();
        days func = new days();
        func.fibo(n);
    }

    void fibo(int days) {

        int arr[] = new int[days];

        if (days >= 1) {
            arr[0]=0;
            arr[1] = 1;
        }
        for (int i = 2; i < days; i++) {
            arr[i] = arr[i - 1] + arr[i - 2];
        }
        for (int j = 0; j < days; j++) {
            System.out.println("Day:" + (j+1) + " routine time:" + arr[j]
+" minutes");
        }
    }
}
```

OUTPUT:

```
^ java -cp /tmp/DAwj6SuV91/days|
Enter the number of days
5
Day:1 routine time:0 minutes
Day:2 routine time:1 minutes
Day:3 routine time:1 minutes
Day:4 routine time:2 minutes
Day:5 routine time:3 minutes

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

CONCLUSION:

In Java, the Fibonacci series program in Java can be generated using an array of methods, including for loops, while loops, recursion, memoization, and iterative approaches.

No.	Aim of the Practical
7.	<p>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;</p> <p>front_times('Chocolate', 2) → 'ChoCho'</p> <p>front_times('Chocolate', 3) 'ChoChoCho'</p> <p>front_times('Abc', 3) → 'AbcAbcAbc'</p> <p>PROGRAM CODE:</p> <pre>import java.util.*; public class pra7 { public static void main(String[] args) { String a; Scanner sc = new Scanner(System.in); System.out.println("Enter the string :"); a = sc.nextLine(); System.out.println("Enter the number of time you want to repeat the first three character :"); int n = sc.nextInt(); String newstr = front_times(a, n); System.out.println("The new string is \n" + newstr); } static String front_times(String a, int b) { String newstr = ""; if (a.length() <= 3) { for (int i = 0; i < b; i++) { newstr += a; } } else { for (int i = 0; i < b; i++) { newstr += a.substring(0, 3); } } return newstr; } }</pre>

OUTPUT:

```
Enter the string :  
chocolate  
Enter the number of time you want to repeat the first three character :  
3  
The new string is  
chochocho
```

CONCLUSION:

From this experiment I learnt the substring function in java and how we can print the desired characters repeatedly.

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.util.*;

class pra8 {
    public static void main(String[] args) {

        int n;
        Scanner obj = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        n = obj.nextInt();
        int arr[] = new int[n];
        int count=0;

        for (int i = 0; i < n; i++) {
            System.out.println("Enter the element:" + (i + 1));

            arr[i] = obj.nextInt();
            if(arr[i]==9)
            {
                count++;
            }

        }

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

    }
}
```

OUTPUT:

```
Enter the size of the array:
4
Enter the element:1
9
Enter the element:2
3
Enter the element:3
5
Enter the element:4
9
Number of 9's in array: 2
```

CONCLUSION:

By performing this practical I understood the operations performed in loop.

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

`double_char('The') → 'TThhee'`

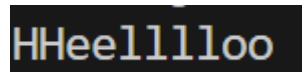
`double_char('AAbb') → 'AAAAbbbb'`

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

PROGRAM CODE:

```
public class pra9 {  
    public static void main(String[] args) {  
  
        String str = "Hello";  
        doublechar(str);  
    }  
  
    static void doublechar(String s) {  
        StringBuilder result = new StringBuilder();  
        for (int i = 0; i < s.length(); i++) {  
            result.append(s.charAt(i));  
            result.append(s.charAt(i));  
        }  
        System.out.println(result.toString());  
    }  
}
```

OUTPUT:



CONCLUSION:

From this practical I learnt the functions of stringbuilder class and the charAt and append functions.

10. Perform following functionalities of the string:

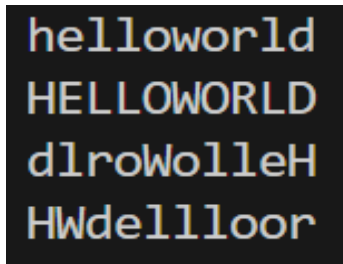
- Find Length of the String
- Lowercase of the String
- Uppercase of the String
- Reverse String
- Sort the String

PROGRAM CODE:

```
import java.util.*;
class pra10{
    public static void main(String[] args) {
        String s="HelloWorld";
        int n=s.length();
        StringBuffer obj=new StringBuffer(s);
        System.out.println("The length of the string: "+n);
        System.out.println(s.toLowerCase());
        System.out.println(s.toUpperCase());
        System.out.println(obj.reverse());
        char[] arr = s.toCharArray();
        Arrays.sort(arr);
        String sortarr = new String(arr);
        System.out.println(sortarr);

    }
}
```

OUTPUT:



```
helloworld
HELLOWORLD
dlrowolleH
Hwde11loor
```

CONCLUSION:

From this practical we learnt about various string functions in java and how to use stringBuffer class.

11.

Perform following Functionalities of the string:

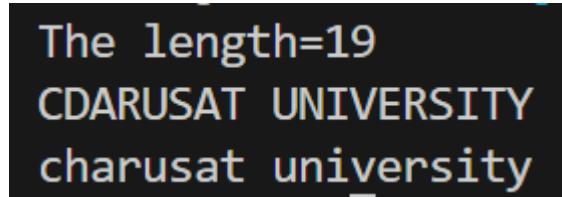
“CHARUSAT UNIVERSITY”

- Find length
- Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’
- Convert all character in lowercase.

PROGRAM CODE:

```
public class pra11 {  
    public static void main(String[] args) {  
  
        String a="CHARUSAT UNIVERSITY";  
        System.out.println("The length="+a.length());  
        StringBuffer obj=new StringBuffer(a);  
        obj.setCharAt( 1, 'D' );  
        System.out.println(obj.toString());  
        System.out.println(a.toLowerCase());  
    }  
}
```

OUTPUT:

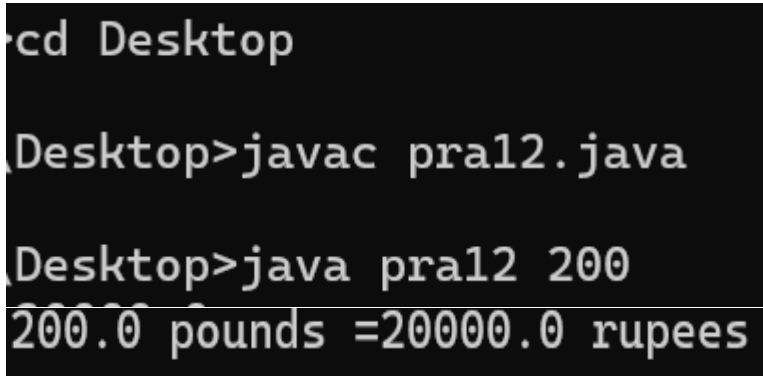


```
The length=19  
CDARUSAT UNIVERSITY  
charusat university
```

CONCLUSION:

By performing this experiment I learnt that how we can replace the desired character at specified index and also how to convert whole string into the lowercase.

Part - 3

No.	Aim of the Practical
12.	<p>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.</p> <p>PROGRAM CODE:</p> <pre>public class pra12 { public static void main(String[] args) { for(String str: args) { float argument = Float.parseFloat(str); float r=argument*100; System.out.println(argument+" pounds =" +r+" rupees"); } } }</pre> <p>OUTPUT:</p>  <pre>cd Desktop Desktop>javac pra12.java Desktop>java pra12 200 200.0 pounds =20000.0 rupees</pre> <p>CONCLUSION: From this practical I learnt about command-line arguments and with the help of it we can pass the value at a runtime as mentioned in above image.</p>

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:

```
class Employee{

    String fn,ln;
    double s=0;

    Employee(String a,String b,double c)
    {
        fn=a;
        ln=b;
        if(c<0)
        {
            s=0.0;
        }
        else
            s=c;
    }
    public void set(String fname,String lname,double sal)
    {
        fn=fname;
        ln=lname;
        if(s<0)
        {
            s=0.0;
        }
        else
            s=sal;
    }
    public void get()
    {
        System.out.println("First Name: "+fn);
        System.out.println("Last Name: "+ln);
        System.out.println("Yearly Salary: "+s*12);
    }
}
```

```

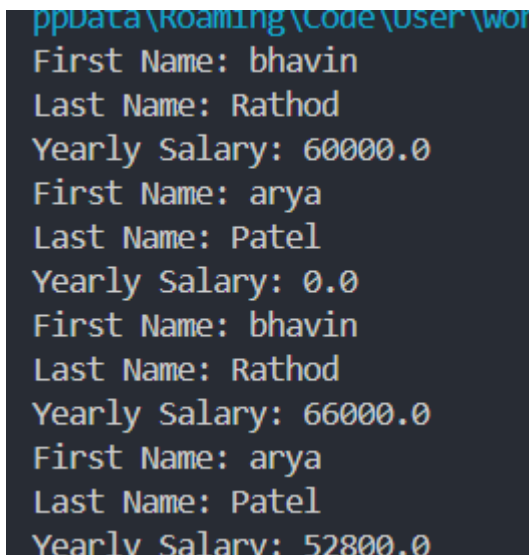
    }

    public void raise()
    {
        s+=s*(10.0/100);
    }
}

class pra13 {
    public static void main(String[] args)
    {
        Employee obj1 = new Employee("bhavin","Rathod",5000);
        Employee obj2 = new Employee("arya","Patel",-4000.0);
        obj1.get();
        obj2.get();
        obj2 = new Employee("arya","Patel",4000.0);
        obj1.raise();
        obj2.raise();
        obj1.get();
        obj2.get();
    }
}

```

OUTPUT:



```

ppData\Roaming\Code\User\work
First Name: bhavin
Last Name: Rathod
Yearly Salary: 60000.0
First Name: arya
Last Name: Patel
Yearly Salary: 0.0
First Name: bhavin
Last Name: Rathod
Yearly Salary: 66000.0
First Name: arya
Last Name: Patel
Yearly Salary: 52800.0

```

CONCLUSION:

From this practical I came to know about constructors in java and how we can use functions along with constructor in a program.

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.*;
public class pra14
{
    int month,day,year,d,m,y;

    pra14(int m,int d,int y)
    {
        month=m;
        day=d;
        year=y;
    }
    public void get()
    {
        System.out.println("Enter day:");
        Scanner sc=new Scanner(System.in);
        d=sc.nextInt();
        System.out.println("Enter month:");
        m=sc.nextInt();
        System.out.println("Enter year:");
        y=sc.nextInt();

    }
    public int getmonth()
    {
        return month;
    }
    public int getday()
    {
```

```

        return day;
    }
    public int getyear()
    {
        return year;
    }
    public void display()
    {
        System.out.println("The date using constructor is: ");
        System.out.println(day+"/"+month+"/"+year);

        System.out.println("The date using method is: ");
        System.out.println(d+"/"+m+"/"+y);

    }
    public static void main(String[] args) {
        pra14 obj=new pra14(11,21,2005);
        int p,q,r;
        int a=obj.getday();
        System.out.println("Day: "+a);
        int b=obj.getmonth();
        System.out.println("Month: "+b);
        int c=obj.getyear();
        System.out.println("Year: "+c);
        obj.get();
        obj.display();

    }
}

```


OUTPUT:

```
Day: 21
Month: 11
Year: 2005
Enter day:
1
Enter month:
5
Enter year:
2011
The date using constructor is:
21/11/2005
The date using method is:
1/5/2011
```

CONCLUSION:

From this experiment I learnt the concept of parameterized constructor and how we can pass the values through constructor to initialize the object.

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.*;
public class pra15 {
    int length,breadth;
    pra15(int l,int b)
    {
        length=l;
        breadth=b;
        System.out.println("Area using constructor="
"+(length*breadth));
    }

    public float returnArea(int l,int b)
    {
        return l*b;
    }
    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the length of the rectangle:");
        int a=sc.nextInt();
        System.out.println("Enter the breadth of the rectangle:");
        int b=sc.nextInt();
        pra15 obj=new pra15(a,b);
        float ans=obj.returnArea(a,b);
        System.out.println("Area= "+ans);

    }
}
```

OUTPUT:

```
Enetr the length of the rectangle:
3
Enetr the breadth of the rectangle:
4
Area using constructor= 12
Area= 12.0
```

CONCLUSION:

In this practical we created a parameterized constructor and the values with which we have initialized the object we returned the values.

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.*;
public class pra16 {
    double real;
    double img;

    // addition
    public void add(int r1,int im1,int r2,int im2) {
        int r,im;
        r= r1+r2;
        im= im1+im2;
        System.out.println("Sum= "+r+" "+im+"i");
    }

    // subtraction
    public void sub(int r1,int im1,int r2,int im2) {
        int r,im;
        r= r1-r2;
        im= im1-im2;
        if(im<0)
            System.out.println("Subtraction= "+r+im+"i");
        else
            System.out.println("Subtraction= "+r+" "+im+"i");
    }

    public void mul(int r1,int im1,int r2,int im2) {
        int r,im;
        r= r1*r2-im1*im2;
        im= r1*im2+im1*r2;
        System.out.println("Multiplication= "+r+" "+im+"i");
    }
}
```

```

public static void main(String[] args) {

    Scanner sc=new Scanner(System.in);
    System.out.println("Enter r1:");
    int r1=sc.nextInt();
    System.out.println("Enter im1:");
    int im1=sc.nextInt();
    System.out.println("Enter r2:");
    int r2=sc.nextInt();
    System.out.println("Enter im2:");
    int im2=sc.nextInt();
    pral6 obj1=new pral6();
    obj1.add(r1,im1,r2,im2);
    obj1.sub(r1,im1,r2,im2);
    obj1.mul(r1,im1,r2,im2);

}
}

```

OUTPUT:

```

Enter r1:
2
Enter im1:
-4
Enter r2:
3
Enter im2:
7
Sum= 5+3i
Subtraction= -1-11i
Multiplication= 34+2i

```

	CONCLUSION:
--	--------------------

	<p>In this practical I performed 3 various mathematical operations on complex numbers which are entered by the user.</p>
--	--

17

PART-4

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

PROGRAM CODE:

```
public class practical17{

    public static void main(String args[])

    {

        parent p = new parent();

        p.parentdisplay();

        children c = new children();

        c.childredisplay();

        System.out.println("Call from children class object.");

        c.parentdisplay();

    }

}

class parent

{

    void parentdisplay()

    {

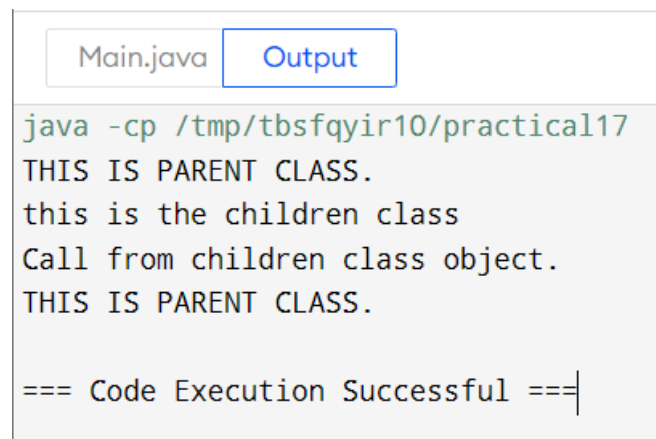
        System.out.println("THIS IS PARENT CLASS.");

    }

}
```

```
class children extends parent
{
    void childredisplay()
    {
        System.out.println("this is the children class");
    }
}
```

OUTPUT:



```
Main.java Output
java -cp /tmp/tbsfqyir10/practical17
THIS IS PARENT CLASS.
this is the children class
Call from children class object.
THIS IS PARENT CLASS.

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

CONCLUSION:

A parent class contains a method that prints "This is parent class," while a subclass has a method that prints "This is child class." By creating objects for both classes, we demonstrate inheritance—where the child class can access the parent's methods. This highlights the benefits of object-oriented programming, including code reuse and a clear hierarchy, allowing for organized and efficient code management.

18	<p>Create a class named 'Member' having the following members: Data members</p> <ul style="list-style-type: none"> 1 - Name 2 - Age 3 - Phone number 4 – Address 5 – Salary <p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre> class Member { private String name; private int age; private String phoneNumber; private String address; private double salary; public void setdetails(String name,int age,String phoneNumber,String address,double salary) { this.name = name; this.age = age; this.address = address; this.phoneNumber = phoneNumber; this.salary = salary; } </pre>
----	--

```

public void getdetails(){

    System.out.println("Name: " + this.name);

    System.out.println("Age: " + this.age);

    System.out.println("Phone Number: " + this.phoneNumber);

    System.out.println("Address: " + this.address);

}

public void printSalary() {

    System.out.println("Salary: " + salary);

}

}

class Employee extends Member {

    private String specialization;

    public void setSpecialization(String specialization) {

        this.specialization = specialization;

    }

    @ Override

    public void getdetails() {

        super.getdetails();

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

    }

}

class Manager extends Member {

```

```

private String department;

public void setDepartment(String department) {
    this.department = department;
}

@Override
public void getdetails() {
    super.getdetails();
    System.out.println("Department :" + department);
}
}

public class practical18 {
    public static void main(String[] args) {

        Employee employee = new Employee();
        employee.setdetails("Yaaaansh", 19, "6894575902", "Mumbai", 10000);
        employee.setSpecialization("Softy Maker");

        Manager manager = new Manager();
        manager.setdetails("Rahul", 19, "9106678890", "delhi", 200000);
        manager.setDepartment("HR");
    }
}

```

```
System.out.println("Employee Details:");
```

```
employee.getdetails();
```

```
employee.printSalary();
```

```
System.out.println("\nManager Details:");
```

```
manager.getdetails();
```

```
manager.printSalary();
```

```
}
```

```
}
```

OUTPUT:

	<div data-bbox="363 230 1189 1238"> <div> Main.java Output </div> <pre> java -cp /tmp/ANZW8zCyxl/practical18 Employee Details: Name: Yaaaansh Age: 19 Phone Number: 6894575902 Address: Mumbai Specialization :Softy Maker Salary: 10000.0 Manager Details: Name: Rahul Age: 19 Phone Number: 9106678890 Address: delhi Department :HR Salary: 200000.0 === Code Execution Successful === </pre> </div> <p><u>CONCLUSION:</u></p> <p>This example highlights how inheritance and encapsulation in object-oriented programming can lead to a more structured, efficient, and understandable codebase. By following these principles, developers can create systems that are easier to extend and maintain.</p>
19	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre>import java.util.Scanner;</pre>

```

public class practical19 {

    public static void main(String args[])

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

        Rectangle[] shape = new Rectangle[10];

        System.out.println("Enter the 1 for rectangle and 2 for Square :");

        int r = sc.nextInt();

        switch(r)

        {

            case 1:

                for(int i=0;i<5;i++){

                    System.out.println("Enter the length and breadth :");

                    int l = sc.nextInt();

                    int b = sc.nextInt();

                    shape[i] = new Rectangle(l, b);

                    shape[i].area();

                    shape[i].perimeter();

                }

                break;

            case 2:

                for(int m=5;m<10;m++){

                    System.out.println("Enter the side :");

```

```

        int s = sc.nextInt();

        shape[m] = new Square(s);

        shape[m].area();

        shape[m].perimeter();

    }

    break;

}

}

}

class Rectangle{

    int length;

    int breadth;

    Rectangle(int length,int breadth)

    {

        this.length = length;

        this.breadth = breadth;

    }

    void area()

    {

        System.out.println("Area :" + length*breadth);

    }

    void perimeter()

```

```

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

class Square extends Rectangle{

    Square(int side)
    {
        super(side,side);
    }

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

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

```

OUTPUT:

Main.java

Output

```
java -cp /tmp/I1cF8BzU5v/practical19
Enter the 1 for rectangle and 2 for Square :
1
Enter the length and breadth :
23
56
Area :1288
Perimeter :158
Enter the length and breadth :
2
5
Area :10
Perimeter :14
Enter the length and breadth :
|
```

CONCLUSION:

In this implementation, we defined a Rectangle class that includes methods to calculate and print the area and perimeter based on its length and breadth. The Square class inherits from Rectangle, passing the side length to the parent class constructor. We used an array of Rectangle objects to demonstrate polymorphism, allowing us to seamlessly handle both Rectangle and Square objects. This example emphasizes the principles of inheritance and method reuse in object-oriented programming.

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 displayS(){

        System.out.println("This is shape");

    }

}

class Rectangle extends Shape{

    void displayR()

    {

        System.out.println("This is rectangle shape");

    }

}

class Circle extends Shape{

    void displayC()

    {

        System.out.println("This is circular shape");

    }

}

class Square extends Rectangle{
```

```

void displaysq()
{
    System.out.println("Square is a rectangle");
}
}

public class practical20 {
    public static void main(String args[]){
        Square s = new Square();
        s.displayS();
        s.displayR();
        s.displaysq();
    }
}

```

OUTPUT:

Main.java

Output

```

java -cp /tmp/3z0gdTXRhU/practical20
This is shape
This is rectangle shape
Square is a rectangle

=== Code Execution Successful ===

```

CONCLUSION:

In this implementation, we created a base class Shape with a method that prints a generic shape message. The subclasses Rectangle and Circle provide specific messages for their shapes. The Square class, inheriting from Rectangle, showcases how subclasses can further specify behavior. By instantiating the Square class, we demonstrate method overriding and polymorphism, allowing us to emphasize the square's relationship with rectangles while still retaining the general shape characteristics.

21

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

PROGRAM CODE:

```
class Degree{

    void getDegree(){

        System.out.println("I got a degree");

    }

}

class UnderGraduate extends Degree

{   @Override

    void getDegree(){

        System.out.println("I am undergraduate");

    }

}

class PostGraduate extends Degree{

    @Override

    void getDegree(){

        System.out.println("I am postgraduate");

    }

}

public class practical21 {

    public static void main(String[] args) {

        Degree d = new Degree();

        PostGraduate p = new PostGraduate();
```

```
        UnderGraduate u = new UnderGraduate();

        d.getDegree();

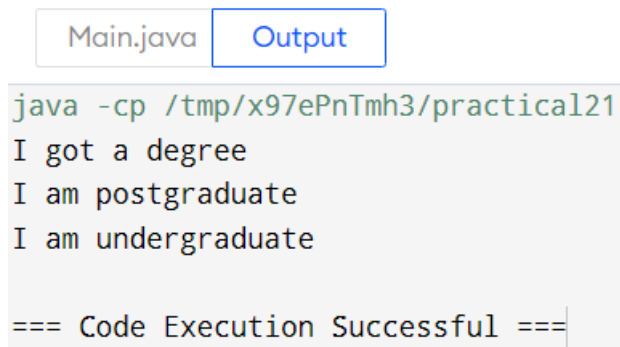
        p.getDegree();

        u.getDegree();

    }

}
```

OUTPUT:



```
Main.java Output
java -cp /tmp/x97ePnTmh3/practical21
I got a degree
I am postgraduate
I am undergraduate

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

CONCLUSION:

In this exercise, we implemented a base class Degree with a method getDegree, along with two subclasses Undergraduate and Postgraduate, each overriding the method to provide specialized output. This demonstrates the concepts of inheritance and method overriding in Java, showcasing how subclasses can extend the functionality of a parent class.

22

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

PROGRAM CODE:

```
interface Advancearithmetic{

    public int divisor_sum(int n);

}

class MyCalculater implements Advancearithmetic{

    int m;

    int sum = 0;

    @Override

    public int divisor_sum(int n) {

        for (int i = 1; i<=n; i++ ){

            if(n%i == 0)

            {

                sum += i;

            }

        }

        return sum;

    }

}

public class practical22 {
```

```

public static void main(String[] args) {

    MyCalcalater m1 = new MyCalcalater();

    int r = 6;

    System.out.println("the sum of the divisor : " + m1.divisor_sum(r));

    int o = 25;

    System.out.println("the sum of the divisor : " + m1.divisor_sum(o));

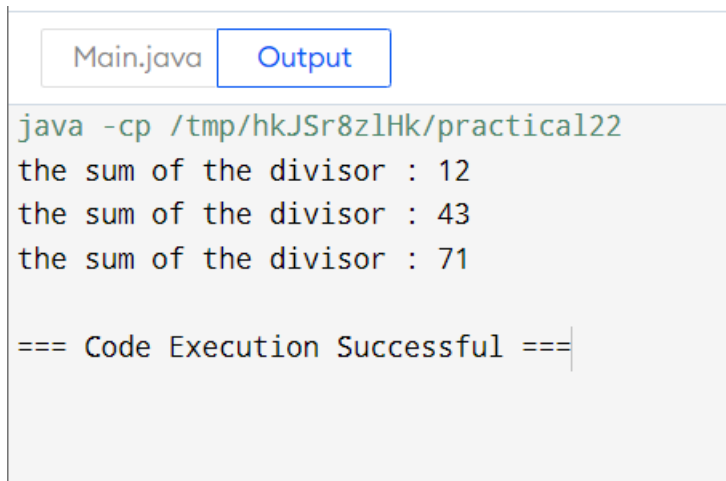
    int p = 12;

    System.out.println("the sum of the divisor : " + m1.divisor_sum(p));

}
}

```

OUTPUT:



The screenshot shows a code editor with two tabs: 'Main.java' and 'Output'. The 'Output' tab is active, displaying the following text:

```

java -cp /tmp/hkJSr8zlHk/practical22
the sum of the divisor : 12
the sum of the divisor : 43
the sum of the divisor : 71

=== Code Execution Successful ===

```

CONCLUSION:

This exercise demonstrates the use of interfaces in Java, specifically the AdvancedArithmetic interface, which mandates the implementation of the divisor_sum method. The MyCalcalator class provides the specific functionality to

	calculate the sum of divisors, showcasing object-oriented principles like abstraction and modular design. This approach enhances code reusability and maintainability.
--	--

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:

```
// Define the Shape interface

interface Shape {

    String getColor();

    double getArea();

    default String getDescription() {

        return "This is a shape.";

    }

}

class Circle implements Shape {

    private double radius;

    private String color;

    public Circle(double radius, String color) {

        this.radius = radius;

        this.color = color;

    }

}
```

```
@Override
```

```
public String getColor() {  
    return color;  
}
```

```
@Override
```

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

```
@Override
```

```
public String getDescription() {  
    return "This is a circle with color " + color + " and radius " + radius + ".";  
}  
}
```

```
class Rectangle implements Shape {
```

```
    private double length;
```

```
    private double width;
```

```
    private String color;
```

```
    public Rectangle(double length, double width, String color) {
```

```
        this.length = length;
```

```
        this.width = width;
```

```
        this.color = color;
```

```

    }

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

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

    @Override
    public String getDescription() {
        return "This is a rectangle with color " + color + ", length " + length + ", and
width " + 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);
        System.out.println(shape.getDescription());
        System.out.println("Background color: " + shape.getColor());
        System.out.println("Shape area: " + shape.getArea());
    }
}

public class practical23 {
    public static void main(String[] args) {

        Shape circle = new Circle(5.0, "Red");

        Shape rectangle = new Rectangle(4.0, 6.0, "Blue");

        Sign circleSign = new Sign(circle, "Welcome to the Campus Center");
        Sign rectangleSign = new Sign(rectangle, "Event Tonight!");

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

```

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

        rectangleSign.displaySign();

    }

}
```

OUTPUT:

Main.java

Output

```
java -cp /tmp/7mKFVvCukE/practical23
Circle Sign:
Sign Text: Welcome to the Campus Center
This is a circle with color Red and radius 5.0.
Background color: Red
Shape area: 78.53981633974483

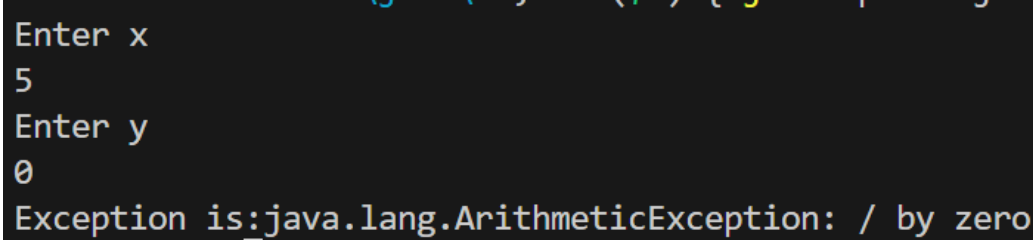
Rectangle Sign:
Sign Text: Event Tonight!
This is a rectangle with color Blue, length 4.0, and width 6.0.
Background color: Blue
Shape area: 24.0

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

	<u>CONCLUSION:</u>
--	---------------------------

	<p>We created a Shape interface and implemented Circle and Rectangle classes, demonstrating polymorphism in Java. The use of default methods allowed us to provide a common behavior (getting the shape's color) while retaining flexibility for specific implementations. This approach promotes cleaner and more maintainable code, effectively modeling real-world shapes and signs in a straightforward manner</p>
--	--

Part-5

No .	Aim of the Practical
24	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre>import java.util.*; public class pra24 { public static void main(String[] args) { int x,y; Scanner sc=new Scanner(System.in); System.out.println("Enter x"); x=sc.nextInt(); System.out.println("Enter x"); y=sc.nextInt(); try { int a=x/y; } catch (Exception e) { System.out.println("Exception is:"+e); } } }</pre> <p>OUTPUT:</p>  <p>CONCLUSION:</p> <p>From this experiment I learnt about exception handling in java using try-catch block.</p>

25

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

PROGRAM CODE:

```
public class pra25 {  
    public static void main(String[] args) {  
        try {  
  
            int result = divide(10, 0);  
            System.out.println("Result: " + result);  
        } catch (ArithmeticException e) {  
  
            System.out.println("Exception caught: Division by zero is not  
allowed.");  
        }  
    }  
  
    public static int divide(int a, int b) {  
        return a / b;  
    }  
}
```

OUTPUT:

```
Exception caught: Division by zero is not allowed.
```

CONCLUSION:

From this practical I learnt about throwing exception using try-catch block.

26

Write a java program to generate user defined exception using “throw” and “throws” keyword.

Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).

PROGRAM CODE:

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

public class pra26 {

    public static void validateAge(int age) throws
InvalidAgeException {
        if (age < 18) {
            throw new InvalidAgeException("Age is less than 18, not
eligible.");
        } else {
            System.out.println("Age is valid, eligible.");
        }
    }

    public static void main(String[] args) {
        try {
            validateAge(16);
        } catch (InvalidAgeException e) {
            System.out.println("Exception caught: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Exception caught: Age is less than 18, not eligible.
```

CONCLUSION:

By performing this practical I learnt about the throw and throws keyword in java and how it can be used to generate an user defined exception.

Part-6

No.	Aim of the Practical
27.	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre>import java.io.BufferedReader; import java.io.FileReader; import java.io.IOException; public class LineCounts { public static void main(String[] args) { 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("Error reading " + fileName + ": " + e.getMessage()); } } } }</pre> <p><u>OUTPUT:</u></p> 

CONCLUSION:

The Java program counts the lines in each text file given as command-line arguments, handling any read errors along the way. It outputs the results for each file, ensuring that it continues processing even if some files can't be read.

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

PROGRAM CODE:

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

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

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

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

OUTPUT:

```
C:\Users\BAPS\Documents>javac pra28.java  
C:\Users\BAPS\Documents>java pra28 D:\java\char.txt l  
Character 'l' appears 3 times in the file.
```

CONCLUSION:

The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access.

29.

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

PROGRAM CODE:

```
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
  
public class WordSearch {  
    public static void main(String[] args) {  
        if (args.length < 2) {  
            System.out.println("Usage: java WordSearch <word> <filename>");  
            return;  
        }  
  
        String searchWord = args[0];  
        String fileName = args[1];  
        Integer count = 0;  
  
        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  
            String line;  
            while ((line = reader.readLine()) != null) {  
                String[] words = line.split("\\W+");  
                for (String word : words) {  
                    if (word.equalsIgnoreCase(searchWord)) {  
                        count++;  
                    }  
                }  
            }  
        }  
    }  
}
```

```

    }
    }
    System.out.println("The word '" + searchWord + "' appears " + count + " times
in " + fileName);
    } catch (IOException e) {
        System.out.println("Error reading " + fileName + ": " + e.getMessage());
    }
}
}

```

OUTPUT:

```
C:\Users\BAPS\Desktop>javac pra29.java
```

```
C:\Users\BAPS\Desktop>java pra29 hello "C:\Users\BAPS\Desktop\hello.txt"
The word 'hello' appears 1 times in C:\Users\BAPS\Desktop\hello.txt
```

CONCLUSION:

This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java.

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

PROGRAM CODE:

```

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 fr = new FileReader(sourceFile);

```

```

        FileWriter fw = new FileWriter(destinationFile)) {

        int ch;
        while ((ch = fr.read()) != -1) {
            fw.write(ch);
        }
        System.out.println("Data copied from " + sourceFile + " to " + destinationFile);
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}

```

OUTPUT:

```

C:\Users\BAPS\Desktop>javac pra30.java

C:\Users\BAPS\Desktop>java pra30 hello.txt copy.txt
Data copied from hello.txt to copy.txt

```

CONCLUSION:

This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance.

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

PROGRAM CODE:

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

public class ConsoleToFile {
    public static void main(String[] args) {
        BufferedReader consoleReader = new BufferedReader(new
        InputStreamReader(System.in));
        String fileName = "output.txt";
    }
}

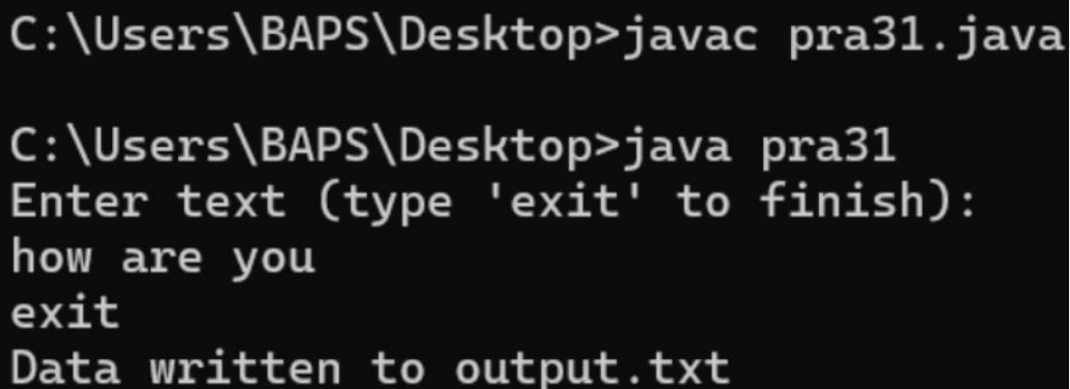
```

```
try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName)))
{
    System.out.println("Enter text (type 'exit' to finish):");

    String input;
    while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit")) {
        fileWriter.write(input);
        fileWriter.newLine();
    }

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

OUTPUT:



```
C:\Users\BAPS\Desktop>javac pra31.java

C:\Users\BAPS\Desktop>java pra31
Enter text (type 'exit' to finish):
how are you
exit
Data written to output.txt
```

CONCLUSION:

This program effectively demonstrates the use of character streams via `BufferedReader` and `BufferedWriter` for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with try-with-resources.

Part-7

No.	Aim of the practical
------------	-----------------------------

32

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

PROGRAM CODE:

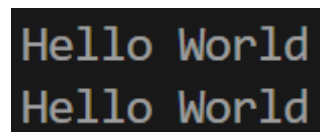
```
public class HelloWorld {
    static class HelloWorldThread extends Thread {
        public void run() {
            System.out.println("Hello World");
        }
    }

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

    public static void main(String[] args) {
        HelloWorldThread thread1 = new HelloWorldThread();
        thread1.start();

        Thread thread2 = new Thread(new HelloWorldRunnable());
        thread2.start();
    }
}
```

OUTPUT:



```
Hello World
Hello World
```

CONCLUSION:

This program demonstrates two approaches to creating threads in Java: extending the Thread class and implementing the Runnable interface. Both methods effectively print "Hello World," showcasing the flexibility of Java's concurrency model.

33

Write a program which takes N and number of threads as an argument. Program should distribute the task of

summation of N numbers amongst number of threads and final result to be displayed on the console.

PROGRAM CODE:

```
import java.util.Scanner;

class SumTask implements Runnable {
    private int start;
    private int end;
    private static int totalSum = 0;

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

    public void run() {
        int partialSum = 0;
        for (int i = start; i <= end; i++) {
            partialSum += i;
        }
        synchronized (SumTask.class) {
            totalSum += partialSum;
        }
    }

    public static int getTotalSum() {
        return totalSum;
    }
}

public class ThreadedSummation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter N: ");
        int N = scanner.nextInt();
        System.out.print("Enter number of threads: ");
        int numThreads = scanner.nextInt();

        Thread[] threads = new Thread[numThreads];
        int range = N / numThreads;
        int remainder = N % numThreads;
        int start = 1;

        for (int i = 0; i < numThreads; i++) {
            int end = start + range - 1;
```

```

        if (i == numThreads - 1) {
            end += remainder;
        }
        threads[i] = new Thread(new SumTask(start, end));
        threads[i].start();
        start = end + 1;
    }

    for (Thread thread : threads) {
        try {
            thread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    System.out.println("Total Sum: " + SumTask.getTotalSum());
}

```

OUTPUT:

```

Enter N: 46
Enter number of threads: 7
Total Sum: 1081

```

CONCLUSION:

This program effectively demonstrates how to utilize multiple threads in Java to perform a summation task concurrently. By distributing the workload among threads, it showcases improved efficiency in computation, making it a practical example of multithreading in action.

34

Write a java program that implements a multi-thread application that has three threads. First thread generates

random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.

PROGRAM CODE:

```
import java.util.Random;

class RandomNumberGenerator extends Thread {
    private final Object lock;

    public RandomNumberGenerator(Object lock) {
        this.lock = lock;
    }

    public void run() {
        Random random = new Random();
        while (true) {
            int number = random.nextInt(100);
            synchronized (lock) {
                MultiThreadApplication.lastNumber = number;
                lock.notifyAll();
                System.out.println("Generated: " + number);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}

class EvenNumberProcessor extends Thread {
    private final Object lock;

    public EvenNumberProcessor(Object lock) {
        this.lock = lock;
    }

    public void run() {
        while (true) {
```

```

        synchronized (lock) {
            try {
                lock.wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            if (MultiThreadApplication.lastNumber % 2 == 0) {
                int square = MultiThreadApplication.lastNumber *
MultiThreadApplication.lastNumber;
                System.out.println("Square: " + square);
            }
        }
    }
}

class OddNumberProcessor extends Thread {
    private final Object lock;

    public OddNumberProcessor(Object lock) {
        this.lock = lock;
    }

    public void run() {
        while (true) {
            synchronized (lock) {
                try {
                    lock.wait();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                if (MultiThreadApplication.lastNumber % 2 != 0) {
                    int cube = MultiThreadApplication.lastNumber *
MultiThreadApplication.lastNumber *
MultiThreadApplication.lastNumber;
                    System.out.println("Cube: " + cube);
                }
            }
        }
    }
}

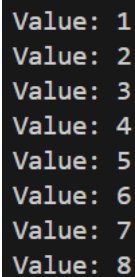
public class MultiThreadApplication {

```

35	<pre> public static int lastNumber; public static void main(String[] args) { Object lock = new Object(); RandomNumberGenerator generator = new RandomNumberGenerator(lock); EvenNumberProcessor evenProcessor = new EvenNumberProcessor(lock); OddNumberProcessor oddProcessor = new OddNumberProcessor(lock); generator.start(); evenProcessor.start(); oddProcessor.start(); } } </pre> <p>OUTPUT:</p> <pre> Generated: 43 Cube: 79507 Generated: 85 Cube: 614125 Generated: 8 Square: 64 Generated: 93 Cube: 804357 Generated: 11 Cube: 1331 Generated: 63 Cube: 250047 Generated: 80 Square: 6400 </pre> <p><u>CONCLUSION:</u> This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency can be leveraged for efficient task distribution.</p>
35	<p>Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.</p>

PROGRAM CODE:

```
public class IncrementVariable extends Thread {  
    private int value = 0;  
  
    public void run() {  
        while (true) {  
            value++;  
            System.out.println("Value: " + value);  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
  
    public static void main(String[] args) {  
        IncrementVariable incrementer = new IncrementVariable();  
        incrementer.start();  
    }  
}
```

OUTPUT:

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

CONCLUSION:

This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java.

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

‘THIRD’ thread to 7.

PROGRAM CODE:

```
class MyThread extends Thread {
    public MyThread(String name) {
        super(name);
    }

    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(getName() + ": " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ThreadPriorityExample {
    public static void main(String[] args) {
        MyThread firstThread = new MyThread("FIRST");
        MyThread secondThread = new MyThread("SECOND");
        MyThread thirdThread = new MyThread("THIRD");

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

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

OUTPUT:

```
THIRD: 1
SECOND: 1
FIRST: 1
THIRD: 2
SECOND: 2
FIRST: 2
THIRD: 3
FIRST: 3
SECOND: 3
THIRD: 4
SECOND: 4
FIRST: 4
THIRD: 5
SECOND: 5
FIRST: 5
```

CONCLUSION:

This program demonstrates thread creation and priority setting in Java. Each thread executes a simple loop, displaying its name and an iteration count, showcasing how thread priority can influence the execution order, although actual execution may vary due to the nature of thread scheduling.

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

PROGRAM CODE:

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

class ProducerConsumer {
    private final Queue<Integer> queue = new LinkedList<>();
    private final int capacity = 5;

    public void produce() throws InterruptedException {
        int value = 0;
        while (true) {
            synchronized (this) {
                while (queue.size() == capacity) {
                    wait();
                }
                queue.add(value);
                System.out.println("Produced: " + value);
                value++;
                notifyAll();
            }
            Thread.sleep(1000);
        }
    }

    public void consume() throws InterruptedException {
        while (true) {
            synchronized (this) {
                while (queue.isEmpty()) {
                    wait();
                }
                int value = queue.poll();
                System.out.println("Consumed: " + value);
                notifyAll();
            }
            Thread.sleep(1500);
        }
    }
}
```

```

class Producer extends Thread {
    private final ProducerConsumer pc;

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

    public void run() {
        try {
            pc.produce();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Consumer extends Thread {
    private final ProducerConsumer pc;

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

    public void run() {
        try {
            pc.consume();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class ProducerConsumerExample {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer();
        Producer producer = new Producer(pc);
        Consumer consumer = new Consumer(pc);

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

```

OUTPUT:

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

CONCLUSION:

The producer generates integers and adds them to a shared queue, while the consumer retrieves and consumes them. Synchronization ensures safe access to the shared resource, preventing data inconsistencies and race conditions.

Part-8

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

PROGRAM:-

```
import java.util.ArrayList;

public class MyStack {
    private ArrayList<Object> list;

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

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

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

    public Object peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return list.get(list.size() - 1);
    }

    public Object pop() {
```

```

        if (isEmpty()) {
            throw new IllegalStateException("Stack is empty");
        }
        return list.remove(list.size() - 1);
    }

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

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

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

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

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

```

OUTPUT:-

```

java -cp /tmp/xjdYt0g2ci/MyStack
Top element (peek): 30
Stack size: 3
Popped element: 30
New top element (peek): 20
Is stack empty? false
Is stack empty? true

```

CONCLUSION:-

The MyStack class effectively encapsulates stack functionalities using an ArrayList. It includes methods for checking if the stack is empty, getting its size, peeking at the top element, popping the top element, and pushing new elements onto the stack. This implementation provides an intuitive way to utilize stack operations while leveraging the dynamic nature of ArrayList for efficient storage and retrieval of elements.

39.

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

PROGRAM CODE:

```
public class SortUtil {
    public static <T extends Comparable<T>> void sort(T[] array) {
        for (int i = 0; i < array.length - 1; i++) {
            for (int j = 0; j < array.length - i - 1; j++) {
                if (array[j].compareTo(array[j + 1]) > 0) {
                    T temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product("Laptop", 1200),
            new Product("Phone", 800),
            new Product("Tablet", 600)
        };
        sort(products);

        for (Product product : products) {
            System.out.println(product.getName() + " - $" +
product.getPrice());
        }
    }
}

class Product implements Comparable<Product> {
```

```
private String name;
private int price;

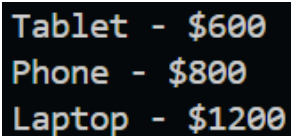
public Product(String name, int price) {
    this.name = name;
    this.price = price;
}

public String getName() {
    return name;
}

public int getPrice() {
    return price;
}

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

OUTPUT:



```
Tablet - $600
Phone - $800
Laptop - $1200
```

CONCLUSION:

This program demonstrates a generic bubble sort method that sorts an array of `Comparable` objects, specifically `Product` objects, based on their price. The `Product` class implements the `Comparable` interface, allowing the sorting to be based on the price attribute. After sorting, the products are displayed in ascending order of price.

40

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

PROGRAM CODE:

```
import java.util.*;

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

        Map<String, Integer> wordCountMap = new TreeMap<>();
        String[] words = text.split("\\s+");

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

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

OUTPUT:

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

CONCLUSION:

This program counts the occurrences of each word in a given text and displays them in alphabetical order using a `TreeMap`. It demonstrates basic string manipulation, word counting, and sorting capabilities.

41	<p>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.</p> <p><u>PROGRAM CODE:</u></p> <pre> import java.io.*; import java.util.*; public class P41 { private static final HashSet<String> keywords = new HashSet<>(); static { String[] keywordArray = { "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class", "const", "continue", "default", "do", "double", "else", "enum", "extends", "final", "finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int", "interface", "long", "native", "new", "package", "private", "protected", "public", "return", "short", "static", "strictfp", "super", "switch", "synchronized", "this", "throw", "throws", "transient", "try", "void", "volatile", "while" }; for (String keyword : keywordArray) { keywords.add(keyword); } } public static void main(String[] args) { Scanner scanner = new Scanner(System.in); System.out.print("Enter the path of the Java source file: "); String filePath = scanner.nextLine(); try { File file = new File(filePath); Scanner fileScanner = new Scanner(file); int keywordCount = 0; while (fileScanner.hasNext()) { String word = fileScanner.next(); if (keywords.contains(word)) { keywordCount++; } } System.out.println("Number of Java keywords in the file: " + keywordCount); </pre>
----	--

```
fileScanner.close();  
} catch (FileNotFoundException e) {  
System.out.println("File not found: " + filePath);  
} } }
```

OUTPUT:

```
Enter the path of the Java source file: prac27.java  
Number of Java keywords in the file: 15
```

CONCLUSION:

This program counts the number of Java keywords in a source file by reading the file and checking each word against a predefined set of keywords stored in a `HashSet`. It demonstrates keyword detection using file handling and basic string comparison.