

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: CSE-201****Academic year:2024-2025****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><u>Java Installation Steps:</u></p> <ol style="list-style-type: none">1. Download JDK: Visit Oracle's JDK download page and select the version for your OS.2. Install JDK: Run the installer and follow instructions.3. Set Environment Variables: Add JDK's bin directory to your system PATH.4. Verify Installation: Use java -version and javac -version in Command Prompt. <p><u>Object-Oriented Concepts:</u></p> <p>- Inheritance, Polymorphism, Encapsulation, Abstraction: Key principles for structuring code.</p> <p><u>Comparison with Other Languages:</u></p> <ul style="list-style-type: none">- C++: Manual memory management.- Python: Slower, dynamically typed.- C#: Tied to Microsoft ecosystem. |

Java Components:

- JDK: Development kit.
- JRE: Runtime environment.
- JVM: Executes Java bytecode.
- Javadoc: Generates documentation.
- Command Line Arguments: Pass configuration info to main(String[] args).

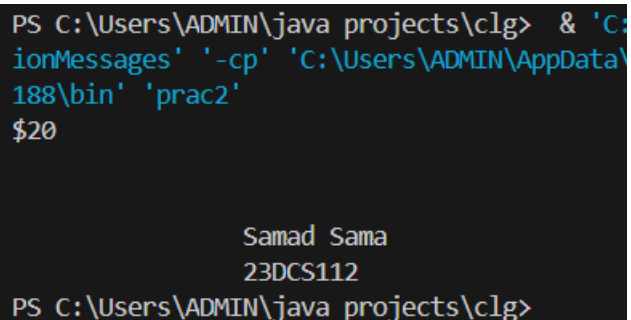
IDEs and Console Programming:

- Eclipse, NetBeans, BlueJ: Popular IDEs for Java development.
- Console Programming: Compile with javac and run with java.

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:

```
public class prac2 {  
  
    public static void main(String[] args) {  
        int money = 20;  
        System.out.println("$"+money);  
    }  
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> & 'C:\Users\ADMIN\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\188\bin' 'prac2'  
$20  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

In this example, we created a basic Java program that simulates a banking application by storing a user's account balance in a variable and then displaying it. This demonstrates the fundamental concept of variable storage and output in Java, which is essential for developing more complex banking systems.

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;
public class prac3 {

    public static void main(String[] args) {
        System.out.println("Enter the Distace in meters");
        Scanner sc = new Scanner(System.in);

        float distance = sc.nextFloat();
        char ch=sc.next();
        System.out.println("Enter the time in hour");
        float hour = sc.nextFloat();

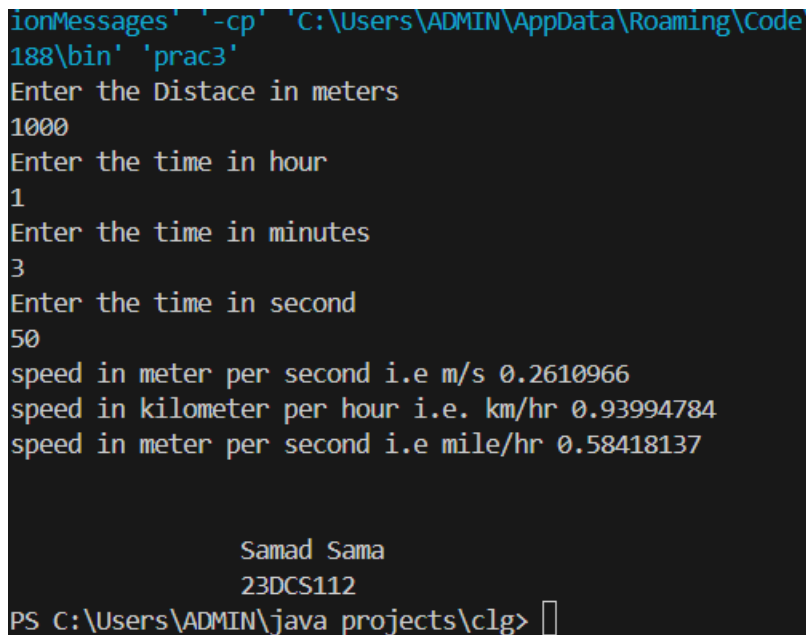
        System.out.println("Enter the time in minutes");
        float minutes = sc.nextFloat();

        System.out.println("Enter the time in second");
        float seconds = sc.nextFloat();

        System.out.println("speed in meter per second i.e m/s " + distance / (hour * 3600 +
minutes * 60 + seconds));
        System.out.println("speed in kilometer per hour i.e. km/hr " + distance / (1000 *
(hour + minutes / 60 + seconds / 3600)));
```

```
System.out.println("speed in meter per second i.e mile/hr " + distance / (1609 * (hour  
+ minutes / 60 + seconds / 3600)));  
  
}  
}
```

OUTPUT:



```
ionMessages' '-cp' 'C:\Users\ADMIN\AppData\Roaming\Code  
188\bin' 'prac3'  
Enter the Distace in meters  
1000  
Enter the time in hour  
1  
Enter the time in minutes  
3  
Enter the time in second  
50  
speed in meter per second i.e m/s 0.2610966  
speed in kilometer per hour i.e. km/hr 0.93994784  
speed in meter per second i.e mile/hr 0.58418137  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg> |
```

CONCLUSION:

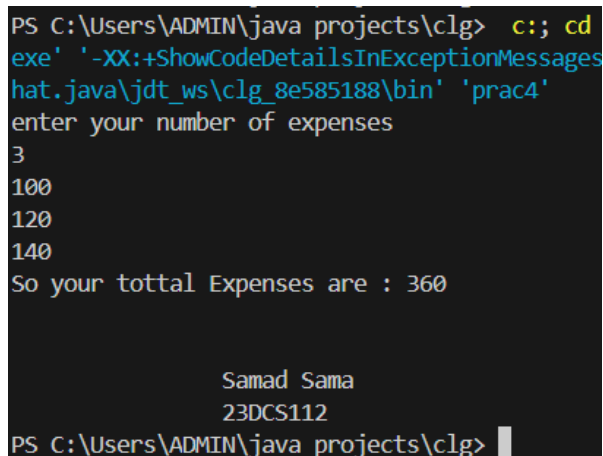
In this program, we used Java's Scanner class to take user input for distance in meters and time in hours, minutes, and seconds. We then converted the time to total seconds to simplify the calculations. The program calculates and displays the speed in three different units: meters per second, kilometers per hour, and miles per hour. This example demonstrates how to handle user input, perform unit conversions, and output results in Java.

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.*;
public class prac4 {
    public static void main(String[] args) {
        int total=0;
        System.out.println("enter your number of expenses");
        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        int [] arr= new int[n];//you have to dynamically initialise it just like this
        for(int i=0;i<n;i++){
            arr[i]=sc.nextInt();
        }
        for(int i=0;i<n;i++){
            total+=arr[i];
        }
        System.out.println("So your tottal Expenses are : " + total);
    }
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> c:: cd
exe' '-XX:+ShowCodeDetailsInExceptionMessages
hat.java\jdt_ws\clg_8e585188\bin' 'prac4'
enter your number of expenses
3
100
120
140
So your tottal Expenses are : 360

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> |
```

CONCLUSION:

This Java program provides a simple yet effective way to track and calculate monthly expenses based on daily inputs from the user. It demonstrates how to use arrays to store data, take user input through a loop, and perform a summation of array elements.

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:

```
import java.util.Scanner;
public class prac5 {
    public static void main(String[] args) {
        int[] arr={ 1,2,3,4,5};
        float[] arr2={ 100,200,300,400,500};
        Scanner sc = new Scanner(System.in);
        int ch=sc.nextInt();
        switch(ch){
            case 1:
                System.out.println("The price with tax are "+ (arr2[0]+arr2[0]*0.08f));
                break;

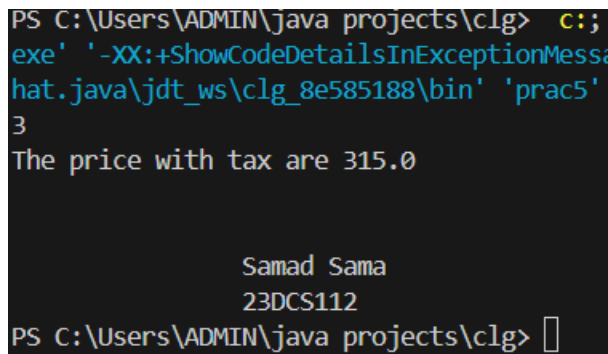
            case 2:
                System.out.println("The price with tax are "+ (arr2[1]+arr2[1]*0.12f));
                break;

            case 3:
                System.out.println("The price with tax are "+ (arr2[2]+arr2[2]*0.05f));
                break;
```

```
        case 4:
            System.out.println("The price with tax are "+ (arr2[3]+arr2[3]*0.075f));
            break;

        case 5:
            System.out.println("The price with tax are "+ (arr2[4]+arr2[4]*0.03f));
            break;
        default:
            System.out.println("SORRY!");

    }
    sc.close();
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> c:\Program Files\Java\jdk-11.0.10\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\ADMIN\java projects\clg_8e585188\bin' 'prac5'
3
The price with tax are 315.0

                Samad Sama
                23DCS112
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

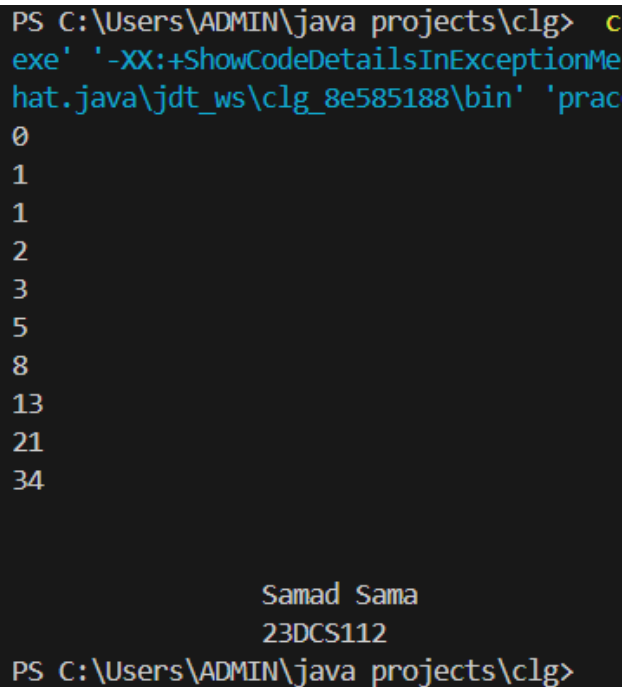
This practical exercise reinforced the fundamentals of array handling, control structures, and basic arithmetic operations in Java

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:

```
public class prac6 {  
    public static void main(String[] args) {  
        int sum = 0, j = 1, k = 0, temp = 0;  
        for (int i = 0; temp <= 50; i++) {  
            System.out.println(temp);  
            k = j;  
            j = temp;  
            temp = j + k;  
            sum += temp;  
        }  
        System.out.println(sum);  
    }  
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> java -XX:+ShowCodeDetailsInExceptionMessages -jar hat.java\jdt_ws\clg_8e585188\bin\prac6.exe  
0  
1  
1  
2  
3  
5  
8  
13  
21  
34  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg>
```


CONCLUSION:

This program provides an easy way to generate an exercise routine based on the Fibonacci series, which is often used to progressively increase workout durations. By following this routine, you can ensure a gradual increase in your exercise time, which can help improve endurance and overall fitness. The Fibonacci series starts with 0 and 1, and each subsequent term is the sum of the previous two terms. This pattern creates a naturally progressive routine, making it a useful tool for designing exercise plans.

Part - 2

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:

```
import java.util.*;

class p7 {
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            System.out.println("Enter a word: ");
            String s = sc.nextLine();

            System.out.println("Enter a number: ");
            int a = sc.nextInt();
            front_times(s, a);
        }
    }

    static void front_times(String s, int a)
    {
        for (int i = 0; i < a; i++) {
            System.out.print(s.substring(0, 3));
        }
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> c::; cd 'c:\
bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMe
8338561aa0db3b\redhat.java\jdt_ws\clg_8e585188\t
Samad Sama
bir day
da
10
a
Hello

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> █
```

CONCLUSION:

From this program we can conclude how to take input from the use using Scanner class and also we use some functions from the String class and use its function substring to print upto three characters.

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

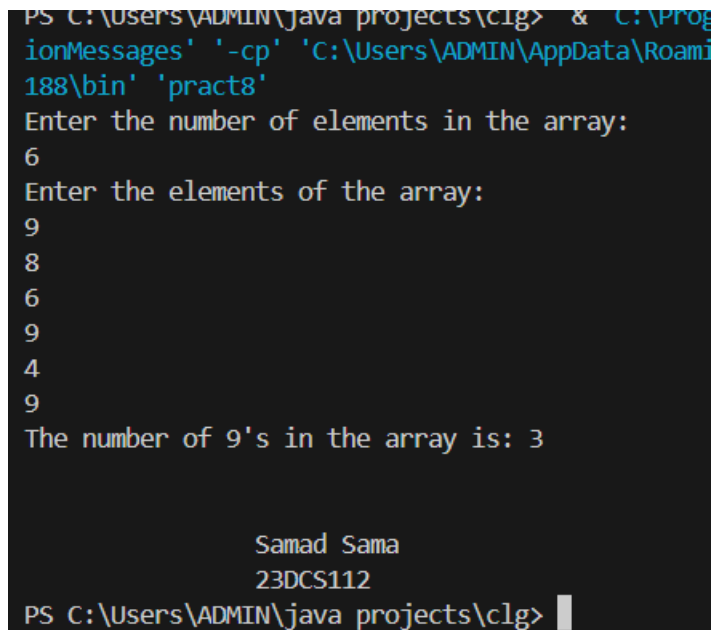
array_count9([1, 9, 9, 3, 9]) → 3

PROGRAM CODE:

```
import java.util.*;
public class pract8 {
    public static void main(String[] args) {
        System.out.println("Enter the number of elements in the array:");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
    }
}
```

```
    }  
    System.out.println("The number of 9's in the array is: " + array_count9(arr));  
    sc.close();  
}  
public static int array_count9(int[] arr) {  
    int count = 0;  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i] == 9) {  
            count++;  
        }  
    }  
    return count;  
}  
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> & C:\Program Files\Java\jdk-11.0.10\bin\java.exe -jar C:\Users\ADMIN\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Java\jre-11.0.10\bin\pract8.jar  
Enter the number of elements in the array:  
6  
Enter the elements of the array:  
9  
8  
6  
9  
4  
9  
The number of 9's in the array is: 3  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg> |
```

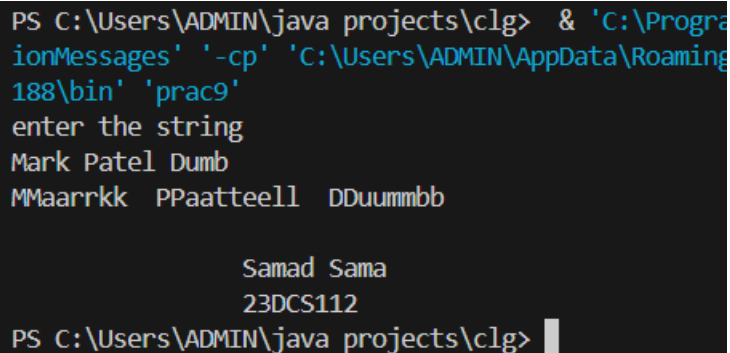
CONCLUSION:

From this java program we can conclude that how to insert an integer value from the use using nextInt and to find the occurrence of any number using for 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('AAAbb') → 'AAAAbbbb'
double_char('Hi-There') → 'HHii--TThheerree'

PROGRAM CODE:

```
import java.util.Scanner;
public class prac9 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter the string");
        String st = sc.nextLine();
        double_char(st);
        System.out.println("\n\n\t\tSamad Sama\n\t\t\t23DCS112");
    }
    public static void double_char(String st){
        for(int i=0;i<st.length();i++){
            System.out.print(st.charAt(i));
            System.out.print(st.charAt(i));
        }
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> & 'C:\Program Files\Java\jdk-11.0.10\bin\java.exe' -cp 'C:\Users\ADMIN\AppData\Roaming\Microsoft\Windows\CurrentVersion\Temp\188\bin' 'prac9'
enter the string
Mark Patel Dumb
MMAarrkk PPaatteell DDuummbb

        Samad Sama
        23DCS112
PS C:\Users\ADMIN\java projects\clg> █
```

CONCLUSION:

This Java program takes a user-inputted string and calculate the length of the string, duplicates each character using function in that string, and then prints the double char of that string to the output using for loop.

10.

Perform following functionalities of the string:

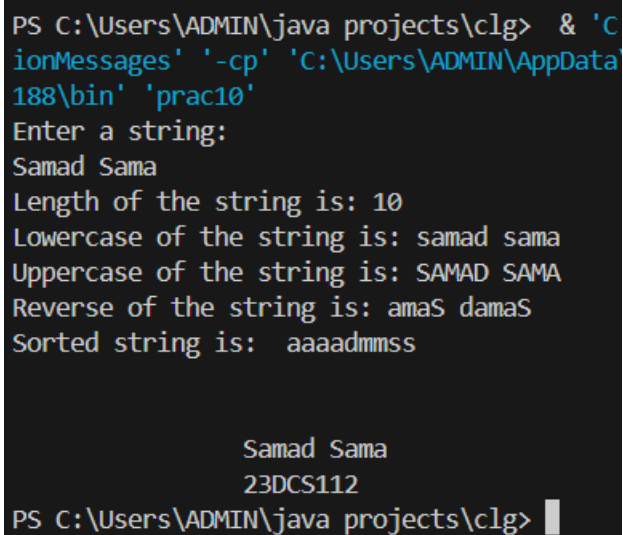
- Find Length of the String
- Lowercase of the String
- Uppercase of the String
- Reverse String
- Sort the string

PROGRAM CODE:

```
import java.util.*;
public class prac10 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a string:");
        String str= sc.nextLine();
        System.out.println("Length of the string is: " + str.length());
        System.out.println("Lowercase of the string is: " + str.toLowerCase());
        System.out.println("Uppercase of the string is: " + str.toUpperCase());
        System.out.println("Reverse of the string is: " + reverse(str));
        System.out.println("Sorted string is: " + sort(str));
        System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");
    }
    public static String reverse(String str) {
        String rev = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            rev += str.charAt(i);
        }
        return rev;
    }
    public static String sort(String str) {
        char[] arr =str.toLowerCase().toCharArray();
```

```
for (int i = 0; i < arr.length; i++) {  
    for (int j = i + 1; j < arr.length; j++) {  
        if (arr[i] > arr[j]) {  
            char temp = arr[i];  
            arr[i] = arr[j];  
            arr[j] = temp;  
        }  
    }  
    return new String(arr);  
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> & 'C:  
ionMessages' '-cp' 'C:\Users\ADMIN\AppData  
188\bin' 'prac10'  
Enter a string:  
Samad Sama  
Length of the string is: 10  
Lowercase of the string is: samad sama  
Uppercase of the string is: SAMAD SAMA  
Reverse of the string is: amaS damaS  
Sorted string is: aaaadmmss  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg>
```

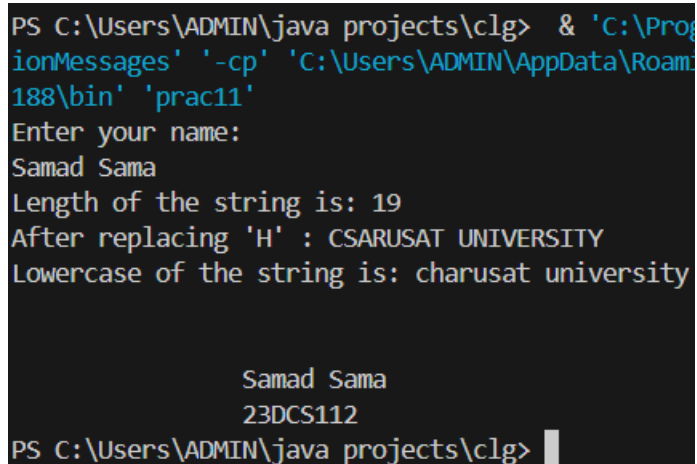
CONCLUSION:

In this java program we learn and different types of String methods like for finding the length of string, to convert it to lower or uppercase and converting into array and then sorting it and again converting into string using various methods.

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:

```
import java.util.*;
public class prac11 {
    public static void main(String[] args) {
        String str = "CHARUSAT UNIVERSITY";
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter your name:");
        String input = sc.next();
        if (input.length() > 0) {
            char n = input.charAt(0);
            System.out.println("Length of the string is: " + str.length());
            System.out.println("After replacing 'H' : " + str.replace('H', n));
        } else {
            System.out.println("No character entered.");
        }
        System.out.println("Lowercase of the string is: " + str.toLowerCase());
        sc.close();
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> & 'C:\Program Files\Java\jdk-11.0.10\bin\java.exe' -cp 'C:\Users\ADMIN\AppData\Roaming\Microsoft\Windows\CurrentVersion\Shell\188\bin' 'prac11'
Enter your name:
Samad Sama
Length of the string is: 19
After replacing 'H' : CSARUSAT UNIVERSITY
Lowercase of the string is: charusat university

                Samad Sama
                23DCS112
PS C:\Users\ADMIN\java projects\clg>
```


CONCLUSION:

In this java program we learn how to find the length of the given String and then replacing its character with another character using setcharat method and then in last we use tolowercase to convert it into lower case.

Part – 3

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

PROGRAM CODE:

prac14.java

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

```
public class prac13 {
    public static void main(String[] args) {
        String c;
        while(true)
        {

            System.out.println("Enter First name: ");
            try (Scanner sc = new Scanner(System.in)) {
                String fname = sc.nextLine();
                System.out.println("Enter Last name: ");
                String lname = sc.nextLine();
                System.out.println("Enter Salary: ");
                double salary = sc.nextInt();
                Employee emp1 = new Employee(fname, lname, salary);
                System.out.println("Yearly Salary for " + emp1.getFirstName() + " " +
emp1.getLastName() + ": $" + emp1.yearlySalary());
                emp1.giveRaise(emp1.getMonthlySalary() * 0.10);
                System.out.println("\nAfter 10% Raise:");
                System.out.println("Yearly Salary for " + emp1.getFirstName() + " " +
emp1.getLastName() + ": $" + emp1.yearlySalary());
                System.out.println("Do you want to continue: (Y/N)");
                c = sc.next();
            }
            if (!"y".equals(c))
                break;
        }
    }
}
```

```
        System.out.println(c);
    }
}
}
```

Employee.java

```
public class Employee {
    private String firstName;
    private String lastName;
    private double monthlySalary;

    public Employee(String firstName, String lastName, double monthlySalary) {
        this.firstName = firstName;
        this.lastName = lastName;
        if (monthlySalary > 0) {
            this.monthlySalary = monthlySalary;
        } else {
            this.monthlySalary = 0.0;
        }
    }

    public String getFirstName() {
        return firstName;
    }

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

    public String getLastName() {
        return lastName;
    }

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

    public double getMonthlySalary() {
```

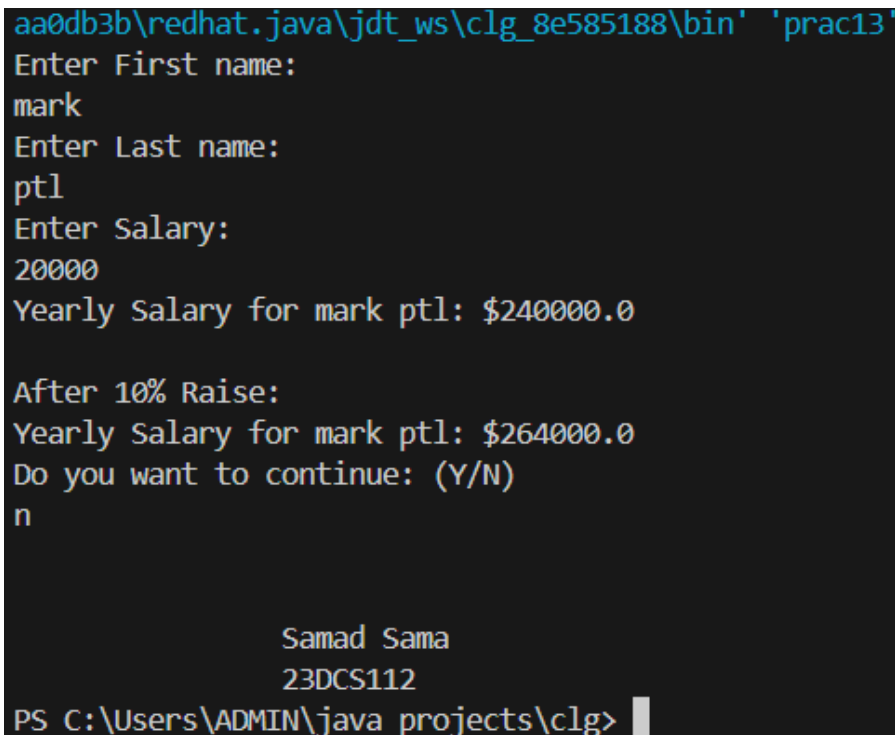
```
        return monthlySalary;
    }

    public void setMonthlySalary(double monthlySalary) {
        if (monthlySalary > 0) {
            this.monthlySalary = monthlySalary;
        } else {
            this.monthlySalary = 0.0;
        }
    }

    public double yearlySalary() {
        return monthlySalary * 12;
    }

    public void giveRaise(double raiseAmount) {
        monthlySalary += raiseAmount;
    }
}
```

OUTPUT:



```
aa0db3b\redhat.java\jdt_ws\clg_8e585188\bin' 'prac13'
Enter First name:
mark
Enter Last name:
ptl
Enter Salary:
20000
Yearly Salary for mark ptl: $240000.0

After 10% Raise:
Yearly Salary for mark ptl: $264000.0
Do you want to continue: (Y/N)
n

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

The provided code consists of a Java application that prompts the user for an employee's first name, last name, and monthly salary, using the Employee class to manage employee details and calculate the yearly salary. The prac13 class continuously asks for employee details, calculates the yearly salary, gives a 10% raise, and displays the updated yearly salary. The loop continues until the user chooses to stop by entering a response other than 'y'. The Employee class ensures that the monthly salary is positive, provides getter and setter methods for employee details, and includes methods to compute the yearly salary and apply a salary raise.

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:**prac14.java**

```
public class prac14 {
    public static void main(String[] args) {
        Date myDate = new Date(7, 22, 2024);

        System.out.print("Initial date: ");
        myDate.displayDate();

        myDate.setDay(25);
        myDate.setMonth(12);
        myDate.setYear(2024);

        System.out.print("Modified date: ");
        myDate.displayDate();
        System.out.println("Day: " + myDate.getDay());
        System.out.println("Month: " + myDate.getMonth());
        System.out.println("Year: " + myDate.getYear());
        System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");
    }
}
```

Date.java

```
public class Date {  
    private int month;  
    private int day;  
    private int year;  
  
    public Date(int day, int month, int year) {  
        this.month = month;  
        this.day = day;  
        this.year = year;  
    }  
  
    public void setMonth(int month) {  
        this.month = month;  
    }  
  
    public void setDay(int day) {  
        this.day = day;  
    }  
  
    public void setYear(int year) {  
        this.year = year;  
    }  
  
    public int getMonth() {  
        return month;  
    }  
  
    public int getDay() {  
        return day;  
    }  
  
    public int getYear() {  
        return year;  
    }  
  
    public void displayDate() {  
        System.out.println(day + "/" + month + "/" + year);  
    }  
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> & 'C:
ages' '-cp' 'C:\Users\ADMIN\AppData\Roaming
' 'prac14'
Initial date: 7/22/2024
Modified date: 25/12/2024
Day: 25
Month: 12
Year: 2024

                Samad Sama
                23DCS112
PS C:\Users\ADMIN\java projects\clg> |
```

CONCLUSION:

The Date class effectively handles date information with instance variables for month, day, and year, initialized via a constructor. Through setter and getter methods, it ensures flexible manipulation and retrieval of date components. The DateTest application demonstrates the class's functionality by displaying dates formatted with forward slashes, showcasing the class's practical utility.

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:

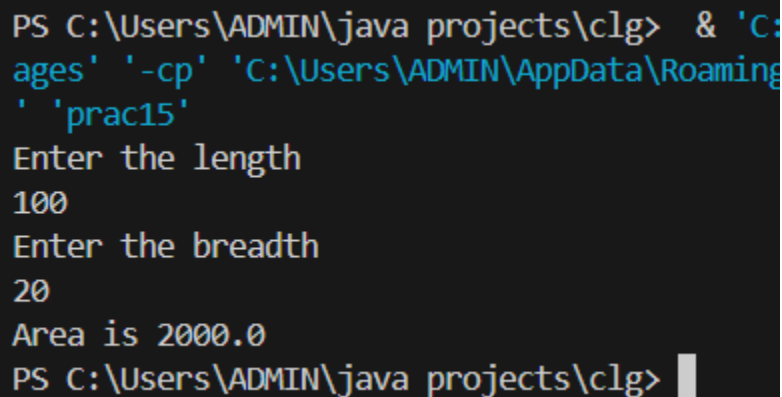
```
import java.util.*;
class Area{

    double length;
    double breadth;
    Area(double x,double y){
        length=x;
        breadth=y;
    }
}
```

```
double return_area(){
    return length*breadth;
}
}
public class prac15 {
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            double a,b;
            System.out.println("Enter the length");
            a=sc.nextDouble();
            System.out.println("Enter the breadth");
            b=sc.nextDouble();

            Area obj = new Area(a,b);
            System.out.println("Area is "+obj.return_area());
        }
    }
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> java -cp 'C:\Users\ADMIN\AppData\Roaming\java\path\to\classes' 'C:\Users\ADMIN\AppData\Roaming\java\path\to\prac15'
Enter the length
100
Enter the breadth
20
Area is 2000.0
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

The Area class efficiently calculates the area of a rectangle using length and breadth provided via its constructor. With the returnArea method, it simplifies area computation. The program's keyboard input functionality and method demonstration underscore its practical application in geometry calculations.

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

PROGRAM CODE:

```
import java.util.*;
class Complex{
    int realpart;
    int imaginarypart;

    Complex(){
        realpart=2;
        imaginarypart=3;
    }
    Complex(int x,int y){
        realpart=x;
        imaginarypart=y;
    }
    Complex(Complex obj){
        realpart=obj.realpart;
        imaginarypart=obj.imaginarypart;
    }
    Complex add(Complex o1,Complex o2){
        Complex demo = new Complex();
        demo.realpart=o1.realpart+o2.realpart;
        demo.imaginarypart=o1.imaginarypart+o2.imaginarypart;
        return demo;
    }
    Complex sub(Complex o1,Complex o2){
        Complex demo = new Complex();
        demo.realpart=o1.realpart-o2.realpart;
        demo.imaginarypart=o1.imaginarypart-o2.imaginarypart;
        return demo;
    }
    Complex multi(Complex o1,Complex o2){
        Complex demo = new Complex();
        demo.realpart=(o1.realpart*o2.realpart-o1.imaginarypart*o2.imaginarypart);
        demo.imaginarypart=(o1.realpart*o2.imaginarypart+o2.realpart*o1.imaginarypart);
        return demo;
    }
}
```

```
}
void display(){
    System.out.println("The complex number is "+realpart+" + "+imaginarypart+"i");
}

};
public class prac16 {
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            Complex obj1 = new Complex();
            int r,i;
            System.out.println("Enter the real part ");
            r=sc.nextInt();
            System.out.println("Enter the imaginary part ");
            i=sc.nextInt();
            Complex obj2 = new Complex(r,i);
            Complex copy = new Complex(obj2);
            System.out.println("Enter the opertaion you want to perform +,-,*");
            char choice = sc.next().charAt(0);//char at string index 0
            switch (choice) {
                case '+' -> {
                    Complex temp = new Complex();
                    Complex obj3=temp.add(obj1, obj2);
                    obj3.display();
                }
                case '-' -> {
                    Complex temp1 = new Complex();
                    Complex obj4=temp1.sub(obj1, copy);
                    obj4.display();
                }
                case '*' -> {
                    Complex temp2 = new Complex();
                    Complex obj5=temp2.multi(obj1, obj2);
                    obj5.display();
                }
                default -> {
                }
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> c:; cd
XX:+ShowCodeDetailsInExceptionMessages' '-cp'
a\jdt_ws\clg_8e585188\bin' 'prac16'
Enter the real part
12 10
Enter the imaginary part
Enter the operation you want to perform +,-,*
+
The complex number is 14 + 13i

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> |
```

CONCLUSION:

The Complex class effectively handles arithmetic operations on complex numbers with separate methods for sum, difference, and product. By allowing user input for the real and imaginary parts, the class demonstrates its practicality and versatility in performing complex number calculations.

Part – 4

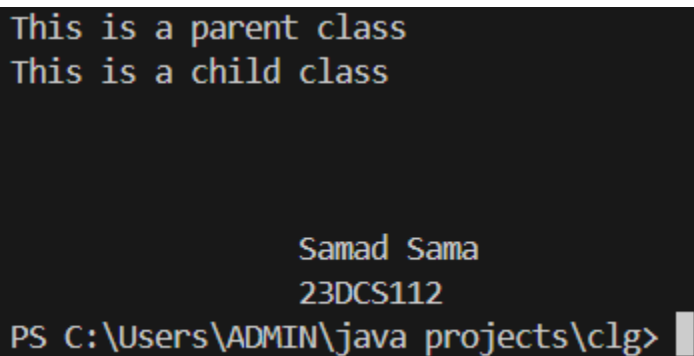
17. Create a class with a method that prints "This is parent class" and its subclass with another method that prints "This is child class". Now, create an object for each of the class and call 1 - method of parent class by object of parent

PROGRAM CODE:

```
class parent{
    void display(){
        System.out.println("This is a parent class");
    }
}
class child1 extends parent{
    void display1(){
        System.out.println("This is a child class");
    }
}
public class prac17 {
    public static void main(String[] args) {
        child1 obj = new child1();
        obj.display();
        obj.display1();
        System.out.println();
        System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");

    }
}
```

OUTPUT:



```
This is a parent class
This is a child class

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

The provided Java code demonstrates the concept of inheritance, where the `child1` class extends the `parent` class, inheriting its methods. The `parent` class contains a `display()` method that prints a message indicating it is a parent class. The `child1` class adds its own method, `display1()`, which prints a message indicating it is a child class. In the `main` method, an instance of `child1` is created, and both the `display()` and `display1()` methods are called, demonstrating that `child1` can use methods from both itself and its parent class. The output of the program confirms the successful invocation of these methods, followed by a printed name and student ID.

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:

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

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

    void printSalary() {
        System.out.println("Your salary is: " + salary);
    }
}
```

```
class Employee extends Member {
    String specialization;
```

```
void setData() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter your name:");
    name = sc.nextLine();
    System.out.println("Enter your age:");
    age = sc.nextInt();
    sc.nextLine(); // Consume newline left-over
    System.out.println("Enter your phone number:");
    phoneNumber = sc.nextLong();
    sc.nextLine(); // Consume newline left-over
    System.out.println("Enter your address:");
    address = sc.nextLine();
    System.out.println("Enter your salary:");
    salary = sc.nextDouble();
    sc.nextLine(); // Consume newline left-over
    System.out.println("Enter your specialization:");
    specialization = sc.nextLine();
}

void display() {
    System.out.println("Name      : " + name);
    System.out.println("Age      : " + age);
    System.out.println("Phone    : " + phoneNumber);
    System.out.println("Address   : " + address);
    System.out.println("Specialization : " + specialization);
    printSalary();
}
}

class Manager extends Member {
    String department;

    void setData() {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter your name:");
        name = sc.nextLine();
        System.out.println("Enter your age:");
        age = sc.nextInt();
        sc.nextLine(); // Consume newline left-over
        System.out.println("Enter your phone number:");
        phoneNumber = sc.nextLong();
        sc.nextLine(); // Consume newline left-over
```

```
        System.out.println("Enter your address:");
        address = sc.nextLine();
        System.out.println("Enter your salary:");
        salary = sc.nextDouble();
        sc.nextLine(); // Consume newline left-over
        System.out.println("Enter your department:");
        department = sc.nextLine();
    }

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

public class prac18 {
    public static void main(String[] args) {
        Employee employee = new Employee();
        employee.setData();
        employee.display();

        Manager manager = new Manager();
        manager.setData();
        manager.display();

        System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> java prac18
Enter your name:
Sam
Enter your age:
19
Enter your phone number:
907967687
Enter your address:
vdchsvsbovs
Enter your salary:
12321
Enter your specialization:
bsdkcsdc
Name           : Sam
Age            : 19
Phone          : 907967687
Address        : vdchsvsbovs
Specialization : bsdkcsdc
Your salary is: 12321.0
Enter your name:
mark
Enter your age:
18
Enter your phone number:
6986124
Enter your address:
hjhaf
Enter your salary:
987
Enter your department:
igsdfhg
Name           : mark
Age            : 18
Phone          : 6986124
Address        : hjhaf
Department     : igsdfhg
Your salary is: 987.0

Samad Sama
23DCS112
```


CONCLUSION:

The provided Java code demonstrates inheritance and method overriding with a `Member` class, and its subclasses `Employee` and `Manager`. Both subclasses inherit properties and methods from `Member`, and each has additional properties (`specialization` for `Employee` and `department` for `Manager`). The `setData` methods in both subclasses use the `Scanner` class to gather user input for their respective properties. The `display` methods in `Employee` and `Manager` override the base class method to print all relevant details, including the salary. This approach shows how inheritance allows for code reuse and extension in object-oriented programming. The program concludes with printing the inputted details and a student signature.

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.

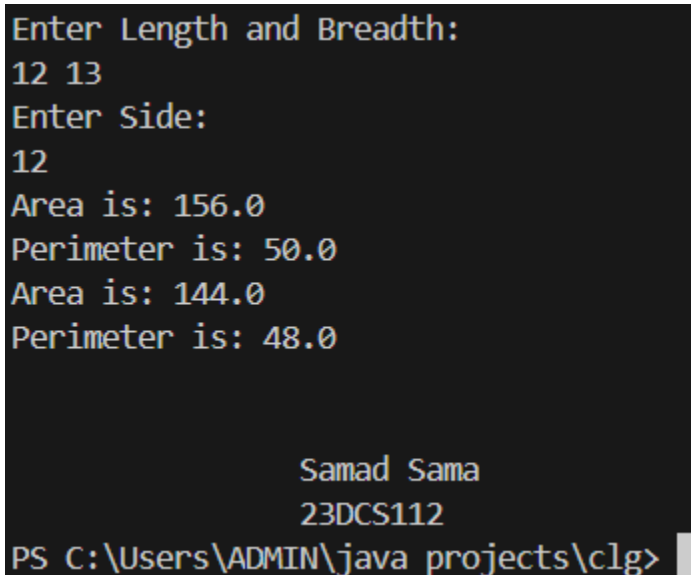
PROGRAM CODE:

```
import java.util.*;
class rectangle
{
    protected double l,b;
    public rectangle(double l,double b)
    {
        this.l=l;
        this.b=b;
    }
    public void printArea()
    {
        System.out.println("Area is: "+(l*b));
    }
    public void printPerimeter()
    {
        System.out.println("Perimeter is: "+(2*(l+b)));
    }
}

class square extends rectangle
{
```

```
protected double s;  
public square(double s) {  
    super(s,s);  
}  
}  
public class prac19 {  
    public static void main(String[] args) {  
        double lenght,breadth,side;  
        try(Scanner sc = new Scanner(System.in)){  
            System.out.println("Enter Length and Breadth: ");  
            lenght = sc.nextDouble();  
            breadth = sc.nextDouble();  
            System.out.println("Enter Side: ");  
            side = sc.nextDouble();  
            sc.close();  
        }  
        rectangle rect = new rectangle(lenght,breadth);  
        square sqr = new square(side);  
        rectangle[] shapes = {rect,sqr};  
        for(rectangle shape : shapes) {  
            shape.printArea();  
            shape.printPerimeter();  
        }  
        System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");  
    }  
}
```

OUTPUT:



```
Enter Length and Breadth:  
12 13  
Enter Side:  
12  
Area is: 156.0  
Perimeter is: 50.0  
Area is: 144.0  
Perimeter is: 48.0  
  
Samad Sama  
23DCS112  
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

The provided Java code demonstrates inheritance and polymorphism with a `rectangle` class and a `square` subclass. The `square` class extends `rectangle`, utilizing the parent class's properties and methods to calculate the area and perimeter. The program uses a `Scanner` to gather user input for the dimensions of the rectangle and square, then creates instances of each. These instances are stored in an array of type `rectangle`, showcasing polymorphism as the `printArea` and `printPerimeter` methods are called on each shape. This approach highlights how common methods can be reused and extended for different shapes through inheritance. The program concludes with printing the calculated areas and perimeters and a student signature.

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{
    public void print1(){
        System.out.println("This is shape");
    }
}
class Rectangle extends Shape{
    public void print2(){
        System.out.println("This is rectagle shape");
    }
}
class Circle extends Shape{
    public void print3(){
        System.out.println("This is circular shape");
    }
}

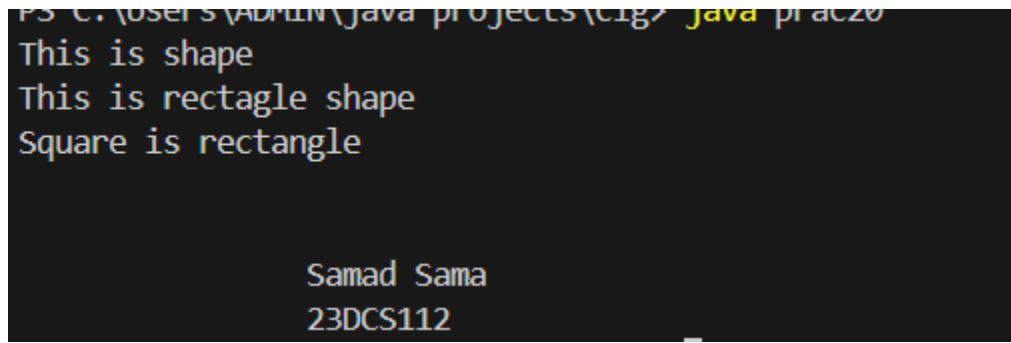
class Square extends Rectangle{
    public void print4(){
        System.out.println("Square is rectangle");
    }
}
public class prac20 {
```

```

public static void main(String[] args) {
    Square obj = new Square();
    obj.print1();
    obj.print2();
    obj.print4();
    System.out.println("\n\n\t\tSamad Sama\n\t\t23DCS112");
}
}

```

OUTPUT:



```

PS C:\Users\ADMIN\java projects\cig> java prac20
This is shape
This is rectagle shape
Square is rectangle

Samad Sama
23DCS112

```

CONCLUSION:

The provided Java code demonstrates multi-level inheritance in object-oriented programming. The `Shape` class serves as the base class with a method `print1()`. The `Rectangle` and `Circle` classes extend `Shape`, each adding their own methods `print2()` and `print3()` respectively. The `Square` class further extends `Rectangle`, adding the `print4()` method. In the `main` method, an instance of `Square` is created, and methods from all levels of the inheritance hierarchy are called, showcasing inheritance and method overriding. This illustrates how derived classes can inherit and extend functionality from their parent classes.

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

PROGRAM CODE:

```

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

```

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

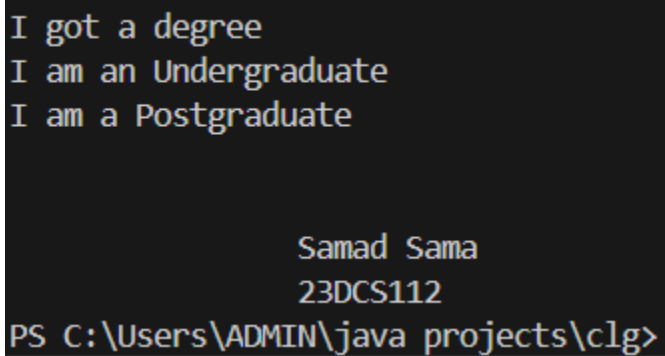
class Postgraduate extends Degree {
    @Override
    public void getDegree() {
        System.out.println("I am a Postgraduate");
    }
}

public class prac21 {
    public static void main(String[] args) {
        Degree degree = new Degree();
        degree.getDegree();

        Undergraduate undergraduate = new Undergraduate();
        undergraduate.getDegree();

        Postgraduate postgraduate = new Postgraduate();
        postgraduate.getDegree();
    }
}
```

OUTPUT:

A screenshot of a terminal window with a black background and white text. The first three lines show the output of the program: "I got a degree", "I am an Undergraduate", and "I am a Postgraduate". The next two lines show the user's name and enrolment number: "Samad Sama" and "23DCS112". The last line shows the command prompt path: "PS C:\Users\ADMIN\java projects\clg>".

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

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

The provided Java code demonstrates method overriding in an inheritance hierarchy. The `Degree` class defines a method `getDegree()` that prints a general message. The `Undergraduate` and `Postgraduate` classes extend `Degree` and override the `getDegree()` method to provide specific messages for each type of degree. In the `main` method, instances of `Degree`, `Undergraduate`, and `Postgraduate` are created, and their respective `getDegree()` methods are called. This illustrates how method overriding allows subclasses to provide specific implementations of a method inherited from a superclass. The output reflects the specialized behavior of each subclass, showcasing polymorphism.

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:

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

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

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

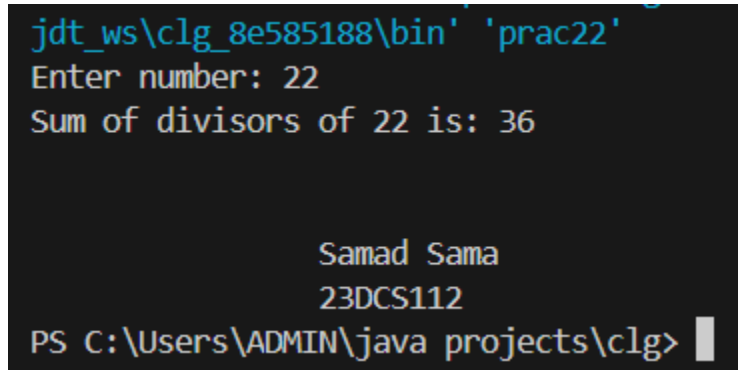
```
public class prac22 {
    public static void main(String[] args) {
        MyCalculator myCalculator = new MyCalculator();
        try (Scanner sc = new Scanner(System.in)) {
```

```

        System.out.print("Enter number: ");
        int n = sc.nextInt();
        System.out.println("Sum of divisors of " + n + " is: " +
myCalculator.divisor_sum(n));
    }
}
}

```

OUTPUT:



```

jdt_ws\clg_8e585188\bin' 'prac22'
Enter number: 22
Sum of divisors of 22 is: 36

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg>

```

CONCLUSION:

This program implements an interface `AdvancedArithmetic` which has a method `divisor_sum(int n)` to compute the sum of divisors of a given number. The `MyCalculator` class implements this interface by overriding the `divisor_sum` method and calculates the sum of all divisors of the input number. In the `main` method, the program takes a number as input from the user and outputs the sum of its divisors. The use of the `Scanner` class allows for user input, and the result is displayed on the console. This demonstrates interface implementation and basic divisor computation in Java.

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:

```

import java.util.Scanner;
interface Shape
{

```

```
String getColor();
default double getArea()
{
    return 0;
}
}
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 this.color;
    }
    @Override
    public double getArea()
    {
        return Math.PI * radius * radius;
    }
}
class Rectangle implements Shape
{
    private double length;
    private double width;
    private String color;
    public Rectangle(double length, double width, String color)
    {
        this.length = length;
        this.width = width;
        this.color = color;
    }
    @Override
    public String getColor()
    {
        return this.color;
    }
    @Override
```



```
        public double getArea()
        {
            return length * width;
        }
    }
}
class Sign
{
    private Shape backgroundShape;
    private String text;
    public Sign(Shape backgroundShape, String text)
    {
        this.backgroundShape = backgroundShape;
        this.text = text;
    }
    public void displaySign()
    {
        System.out.println("Sign:");
        System.out.println("Background Shape Color: " + backgroundShape.getColor());
        System.out.println("Background Shape Area: " + backgroundShape.getArea());
        System.out.println("Text: " + text);
    }
}
public class prac23
{
    public static void main(String[] args)
    {
        try (Scanner sc = new Scanner(System.in))
        {
            System.out.println("Enter radius and color for the circle: ");
            double cr = sc.nextDouble();
            sc.nextLine();
            String cc = sc.nextLine();
            Circle circle = new Circle(cr, cc);
            System.out.println("Enter length, width, and color for the rectangle: ");
            double rl = sc.nextDouble();
            double rw = sc.nextDouble();
            sc.nextLine();
            String rc = sc.nextLine();
            Rectangle rectangle = new Rectangle(rl, rw, rc);
            System.out.println("Enter text for the circle sign: ");
            String cs = sc.nextLine();
            Sign circleSign = new Sign(circle, cs);
            System.out.println("Enter text for the rectangle sign: ");
```

```
String rs = sc.nextLine();
Sign rectangleSign = new Sign(rectangle, rs);
circleSign.displaySign();
rectangleSign.displaySign();
    }
}
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> java prac23
Enter radius and color for the circle:
22
red
Enter length, width, and color for the rectangle:
34 56
white
Enter text for the circle sign:
hello
Enter text for the rectangle sign:
heyy
Sign:
Background Shape Color: red
Background Shape Area: 1520.5308443374597
Text: hello
Sign:
Background Shape Color: white
Background Shape Area: 1904.0
Text: heyy

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> |
```

CONCLUSION:

In this program, the `Shape` interface defines a contract for obtaining the color and area of shapes, with default behavior provided for `getArea()`. The `Circle` and `Rectangle` classes implement this interface, each providing specific details for calculating their areas and retrieving their colors. The `Sign` class uses these shapes to create a sign that displays the shape's color, area, and associated text. The `main` method allows user input to create and display signs for both shapes. This setup demonstrates the use of interfaces, class inheritance, and object composition in Java to model and handle different shapes and their properties effectively.

Part - 5

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

PROGRAM CODE:

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

public class prac24 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int x = 0, y = 0;

        try {
            System.out.print("Enter integer x: ");
            x = scanner.nextInt();
            System.out.print("Enter integer y: ");
            y = scanner.nextInt();
            int result = x / y;
            System.out.println("Result of x / y is: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero is not allowed.");
        } catch (InputMismatchException e) {
            System.out.println("Error: Invalid input. Please enter integers.");
        } finally {
            scanner.close();
        }
    }
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> & 'C:\P
\Users\ADMIN\AppData\Roaming\Code\User\worksp
Enter integer x: 10
Enter integer y: 2
Result of x / y is: 5

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> & 'C:\P
\Users\ADMIN\AppData\Roaming\Code\User\worksp
Enter integer x: 10
Enter integer y: 0
Error: Division by zero is not allowed.

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> & 'C:\P
\Users\ADMIN\AppData\Roaming\Code\User\worksp
Enter integer x: 10
Enter integer y: 2.1
Error: Invalid input. Please enter integers.

Samad Sama
23DCS112
PS C:\Users\ADMIN\java projects\clg> █
```

CONCLUSION:

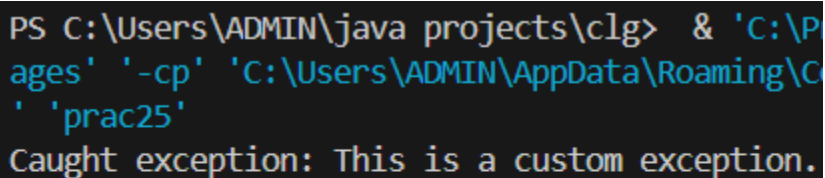
This program demonstrates handling exceptions in Java with a focus on user input and arithmetic operations. It prompts the user to enter two integers, then attempts to divide the first by the second. If a division by zero occurs, it catches the `ArithmeticException` and displays an appropriate error message. It also catches `InputMismatchException` to handle cases where the user inputs non-integer values, providing feedback to enter valid integers. The `finally` block ensures that the `Scanner` resource is properly closed, regardless of whether an exception was thrown. This approach ensures robust handling of common input and arithmetic errors.

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

PROGRAM CODE:

```
class CustomException extends Exception
{
    public CustomException(String message)
    {
        super(message);
    }
}
public class prac25
{
    public static void main(String[] args)
    {
        try
        {
            throw new CustomException("This is a custom exception.");
        } catch (CustomException e)
        {
            System.out.println("Caught exception: " + e.getMessage());
        }
    }
}
```

OUTPUT:



```
PS C:\Users\ADMIN\java projects\clg> & 'C:\Program Files\Java\jdk-11.0.10\bin\java.exe' -cp 'C:\Users\ADMIN\AppData\Roaming\Code\workspace\prag25' 'prac25'
Caught exception: This is a custom exception.
```

Samad Sama
23DCS112

```
PS C:\Users\ADMIN\java projects\clg>
```

CONCLUSION:

This program defines a custom exception class `CustomException` that extends the `Exception` class. It includes a constructor to initialize the exception with a specific error message. In the `main` method, the program intentionally throws an instance of `CustomException` with a custom message. This exception is then caught in the `catch` block, where the message is retrieved and printed. This example demonstrates how to create and handle user-defined exceptions in Java, allowing for more specific error handling in applications.

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:

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

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

public class prac26 {

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

    public static void readFile() throws java.io.FileNotFoundException {
        try (java.io.FileReader file = new java.io.FileReader("non_existent_file.txt")) {
        } catch (FileNotFoundException e) {
            throw e;
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
public static void divideByZero() {
    int result = 10 / 0;
}

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

    try {
        readFile();
    } catch (java.io.FileNotFoundException e) {
        System.out.println("Checked Exception Caught (FileNotFoundException): " +
e.getMessage());
    }

    try {
        divideByZero();
    } catch (ArithmeticException e) {
        System.out.println("Unchecked Exception Caught (ArithmeticException): " +
e.getMessage());
    }

    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        System.out.println("Checked Exception Caught (InterruptedException): " +
e.getMessage());
    }

    try {
        int[] array = new int[2];
        int num = array[3];
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("Unchecked Exception Caught
(ArrayIndexOutOfBoundsException): " + e.getMessage());
    }
    System.out.println("Program continues after exception handling.");
}
}
```

OUTPUT:

```
PS C:\Users\ADMIN\java projects\clg> java prac26
Custom Exception Caught: Age is less than 18, not eligible for voting.
Checked Exception Caught (FileNotFoundException): non_existent_file.txt (The system cannot find the file specified)
Unchecked Exception Caught (ArithmeticException): / by zero
Unchecked Exception Caught (ArrayIndexOutOfBoundsException): Index 3 out of bounds for length 2
Program continues after exception handling.

        Samad Sama
        23DCS112
PS C:\Users\ADMIN\java projects\clg> █
```

CONCLUSION:

This program demonstrates handling various exceptions in Java. It includes a custom exception `InvalidAgeException` for validating ages and throws it if the age is below 18. The `readFile` method demonstrates handling the checked `FileNotFoundException`, rethrowing it after catching it. The `divideByZero` method shows handling of the unchecked `ArithmeticException`. Additionally, the program handles `InterruptedException` and `ArrayIndexOutOfBoundsException`, which are also caught and reported. This example illustrates how to manage both checked and unchecked exceptions, ensuring robust error handling and graceful program continuation.

Part - 6

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

PROGRAM CODE:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class prac27{
    public static void main(String[] args)
    {
        if (args.length == 0)
        {
            System.out.println("Please specify a filename.");
            return;
        }
        for (String fileName : args)
        {
            int linecount = 0;

            try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
            {
                while (reader.readLine() != null)
                {
                    linecount++;
                }
                System.out.println(fileName + ": " + linecount + " lines");
            } catch (IOException e)
            {
                System.out.println("Error reading line "+ fileName + " - " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\samas\java projects\clg> java prac27 file.txt
file.txt: 2 lines
```

```
Samad Sama
23DCS112
```

CONCLUSION:

The provided Java program efficiently counts the number of lines in each text file specified on the command line. It handles multiple files, processes each file sequentially, and provides useful feedback in the event of an error. By using the try-with-resources feature, it ensures proper resource management and file handling. This program demonstrates how to handle command-line arguments, manage file I/O operations, and handle exceptions in Java, making it a robust solution for counting lines across multiple files.

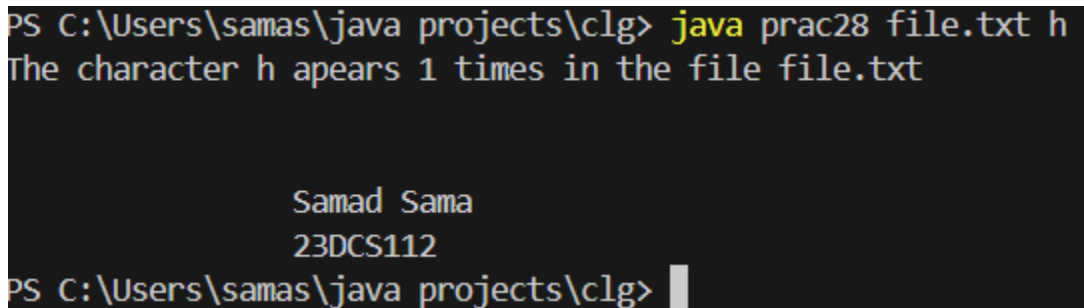
28. Write an example that counts the number of times a particular character, such as e, appears in a file. The character can be specified at the command line. You can use xanadu.txt as the input file.

PROGRAM CODE:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class prac28{
    public static void main(String[] args) {
        if (args.length < 2)
        {
            System.out.println("Run : java prac28 <filename> <character>");
            return;
        }
        String filename = args[0];
        char Char = args[1].charAt(0);
        try(BufferedReader reader = new BufferedReader(new FileReader(filename)))
        {
            int count = 0;
            int curr;
```

```
        while ((curr = reader.read()) != -1)
        {
            if(curr == Char)
            {
                count++;
            }
        }
        System.out.println("The character " + Char + " appears " + count + " times in the
file " + filename);
    }
    catch(IOException e)
    {
        System.out.println("Error reading file: " + filename + " - " + e.getMessage());
    }
}
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac28 file.txt h
The character h appears 1 times in the file file.txt

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg>
```

CONCLUSION:

The program successfully reads a text file and counts how many times a specified character appears in it. It accepts the filename and the character to search for as command-line arguments. By using `BufferedReader` for file reading and a loop to check each character in the file, it efficiently counts occurrences. The program also handles file-related errors, such as the file not being found or issues with reading, using a try-catch block, ensuring that the program exits gracefully while providing informative error messages when something goes wrong. This makes the program both robust and easy to use.

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;
import java.util.Scanner;
public class prac29
{
    public static void main(String[] args)
    {
        try (Scanner sc = new Scanner(System.in))
        {
            System.out.print("Enter the filename: ");
            String fileName = sc.nextLine();
            System.out.print("Enter the word to search for: ");
            String targetWord = sc.nextLine();
            Integer wordCount = 0;
            try (BufferedReader reader = new BufferedReader(new FileReader(fileName)))
            {
                String line;
                while ((line = reader.readLine()) != null)
                {
                    String[] words = line.split("\\s+");
                    for (String word : words) {
                        if (word.equals(targetWord))
                        {
                            wordCount++;
                        }
                    }
                }
                System.out.println("The word '" + targetWord + "' appears " + wordCount + "
times in the file " + fileName);
            }
            catch (IOException e)
            {
                System.out.println("Error reading file: " + fileName + " - " + e.getMessage());
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\samas\java projects\clg> java prac29
Enter the filename: file.txt
Enter the word to search for: Hello
The word 'Hello' appears 1 times in the file file.txt

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg> |
```

CONCLUSION:

The program efficiently reads a specified file and counts how many times a user-specified word appears in it. It utilizes a Scanner to get the filename and the target word from the user, and then a BufferedReader to read the file line by line. The words in each line are split and compared to the target word, incrementing a counter for each match. The program handles file-related errors gracefully with a try-catch block, ensuring that if the file cannot be read, an appropriate error message is displayed. This program demonstrates effective use of file handling, string manipulation, and user input 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.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class prac30 {

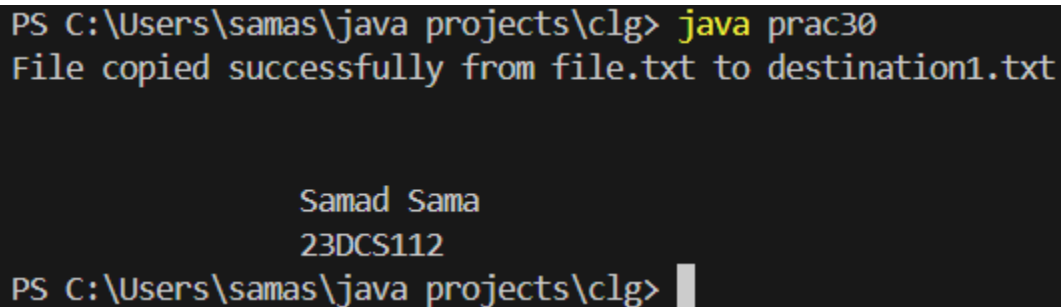
    public static void main(String[] args) {
        String sourceFile = "txt3.txt";
        String destinationFile = "destination1.txt";
        try (
            FileInputStream inputStream = new FileInputStream(sourceFile);
            FileOutputStream outputStream = new FileOutputStream(destinationFile)) {
```

```
byte[] buffer = new byte[1024];
int bytesRead;
while ((bytesRead = inputStream.read(buffer)) != -1) {
    outputStream.write(buffer, 0, bytesRead);
}

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

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

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac30
File copied successfully from file.txt to destination1.txt

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg>
```

CONCLUSION:

This program demonstrates how to copy the contents of one file to another using Java's `FileInputStream` and `FileOutputStream`. It reads data from the source file in chunks (using a byte buffer) and writes it to the destination file. This efficient approach allows handling large files without consuming excessive memory. The program handles potential `IOException` errors, such as file not found or permission issues, and provides informative feedback. Overall, it showcases how to perform basic file I/O operations in Java for copying binary or text files between locations.

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.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

public class prac31 {

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

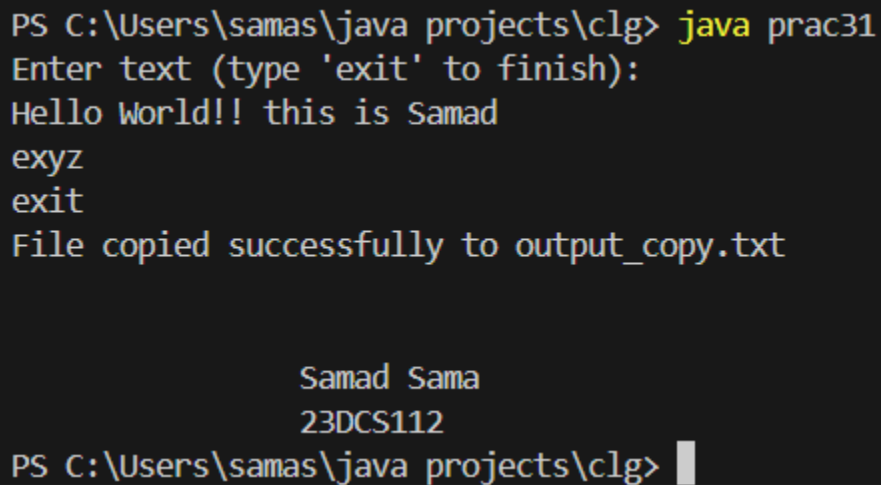
            System.out.println("Enter text (type 'exit' to finish):");
            String line;
            while (!(line = consoleReader.readLine()).equalsIgnoreCase("exit")) {
                fileWriter.write(line);
                fileWriter.newLine();
            }

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
        try (
            FileInputStream fileInputStream = new FileInputStream(fileName);
            FileOutputStream fileOutputStream = new
FileOutputStream("output_copy.txt")) {

            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = fileInputStream.read(buffer)) != -1) {
                fileOutputStream.write(buffer, 0, bytesRead);
            }
        }
    }
}
```

```
        System.out.println("File copied successfully to output_copy.txt");
    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac31
Enter text (type 'exit' to finish):
Hello World!! this is Samad
exyz
exit
File copied successfully to output_copy.txt

                Samad Sama
                23DCS112
PS C:\Users\samas\java projects\clg> |
```

CONCLUSION:

This program efficiently handles two tasks: first, it captures user input from the console and writes it to a file (output.txt) until the user types "exit". Second, it reads this file and copies its contents to a new file (output_copy.txt). The program uses `BufferedReader` and `BufferedWriter` for efficient character stream handling, and `FileInputStream` and `FileOutputStream` to perform the file copy operation. It also includes error handling to manage potential I/O issues gracefully. This demonstrates basic file writing and copying operations in Java.

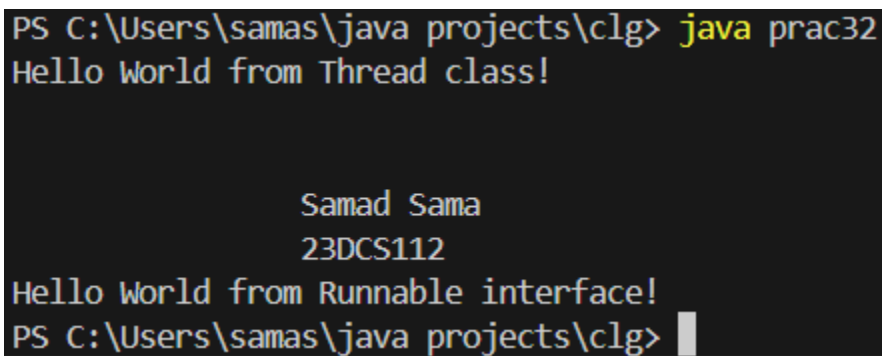
Part - 7

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

PROGRAM CODE:

```
class HelloWorldThread extends Thread {
    @Override
    public void run() {
        System.out.println("Hello World from Thread class!");
    }
}
class HelloWorldRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Hello World from Runnable interface!");
    }
}
public class prac32 {
    public static void main(String[] args) {
        HelloWorldThread thread1 = new HelloWorldThread();
        thread1.start();
        HelloWorldRunnable runnable = new HelloWorldRunnable();
        Thread thread2 = new Thread(runnable);
        thread2.start();
    }
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac32
Hello world from Thread class!

Samad Sama
23DCS112
Hello world from Runnable interface!
PS C:\Users\samas\java projects\clg> █
```

CONCLUSION:

We demonstrated two approaches to creating threads in Java: one by extending the Thread class and another by implementing the Runnable interface. Both approaches are effective for multi-threading, but the Runnable interface is more flexible as it allows the class to extend other classes as well. The program illustrates how to use these methods, with each thread executing independently, displaying the "Hello World" message from different contexts. This provides a solid foundation for understanding thread creation and execution in Java.

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 int[] result;

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

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

```
}

public class prac33 {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Usage: java prac33 <N> <number_of_threads>");
            return;
        }

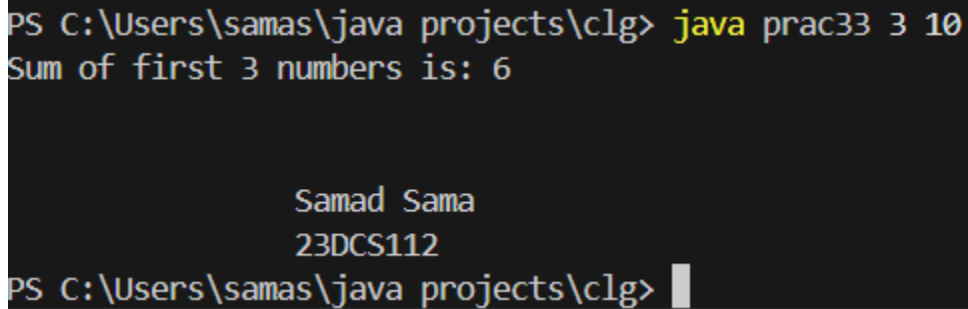
        int N = Integer.parseInt(args[0]);
        int numberOfThreads = Integer.parseInt(args[1]);

        int[] result = new int[1];
        Thread[] threads = new Thread[numberOfThreads];
        int numbersPerThread = N / numberOfThreads;

        for (int i = 0; i < numberOfThreads; i++) {
            int start = i * numbersPerThread + 1;
            int end = (i == numberOfThreads - 1) ? N : start + numbersPerThread - 1;
            threads[i] = new Thread(new SumTask(start, end, result));
            threads[i].start();
        }

        for (int i = 0; i < numberOfThreads; i++) {
            try {
                threads[i].join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Sum of first " + N + " numbers is: " + result[0]);
    }
}
```

OUTPUT:


```
PS C:\Users\samas\java projects\clg> java prac33 3 10
Sum of first 3 numbers is: 6

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg> |
```

CONCLUSION:

This program effectively demonstrates how to distribute the task of summing the first NNN numbers among a specified number of threads in Java. By dividing the workload, each thread calculates a portion of the total sum, which promotes parallel processing and improves efficiency. The program uses the Runnable interface to define the summation task and manages thread execution through the Thread class. After all threads complete their execution, the program aggregates the results and displays the final sum on the console. This approach showcases fundamental concepts in multi-threading, such as task distribution, thread management, and synchronization, making it a practical example for understanding concurrent programming in Java.

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 NumberProcessor processor;

    public RandomNumberGenerator(NumberProcessor processor) {
        this.processor = processor;
    }

    @Override
    public void run() {
        Random random = new Random();
```

```
while (true) {  
    int number = random.nextInt(100);  
    System.out.println("Generated: " + number);  
    processor.process(number);  
    try {  
        Thread.sleep(1000);  
    } catch (InterruptedException e) {  
        break;  
    }  
}  
}  
}
```

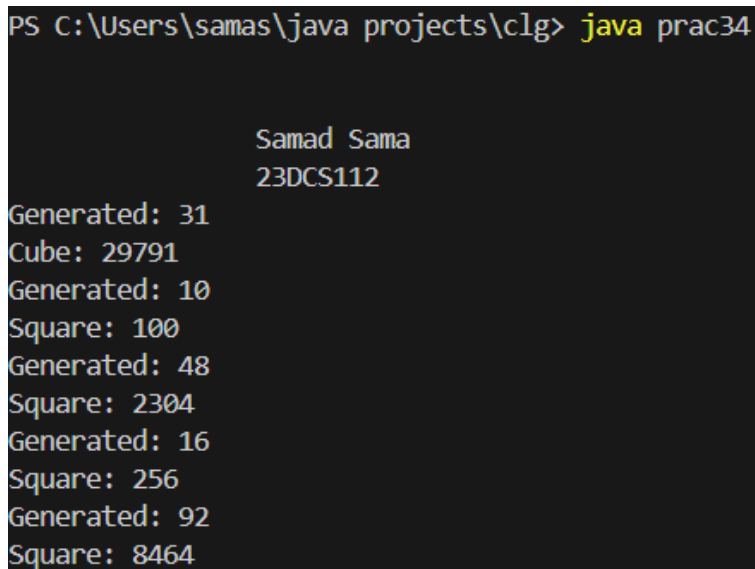
```
class NumberProcessor {  
    private final Object lock = new Object();  
    private int number;  
  
    public void process(int number) {  
        synchronized (lock) {  
            this.number = number;  
            lock.notifyAll();  
        }  
    }  
  
    public void computeSquare() {  
        while (true) {  
            synchronized (lock) {  
                try {  
                    lock.wait();  
                } catch (InterruptedException e) {  
                    break;  
                }  
                if (number % 2 == 0) {  
                    System.out.println("Square: " + (number * number));  
                }  
            }  
        }  
    }  
  
    public void computeCube() {  
        while (true) {  
            synchronized (lock) {  
                try {
```

```
        lock.wait();
    } catch (InterruptedException e) {
        break;
    }
    if (number % 2 != 0) {
        System.out.println("Cube: " + (number * number * number));
    }
}
}
}

public class prac34 {
    public static void main(String[] args) {
        NumberProcessor processor = new NumberProcessor();
        RandomNumberGenerator generator = new RandomNumberGenerator(processor);
        Thread squareThread = new Thread(processor::computeSquare);
        Thread cubeThread = new Thread(processor::computeCube);

        generator.start();
        squareThread.start();
        cubeThread.start();
    }
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac34

        Samad Sama
        23DCS112
Generated: 31
Cube: 29791
Generated: 10
Square: 100
Generated: 48
Square: 2304
Generated: 16
Square: 256
Generated: 92
Square: 8464
```

CONCLUSION:

This program effectively showcases multi-threading in Java by creating a system that generates random integers and processes them based on their parity. By utilizing synchronization mechanisms, the program ensures that the generated numbers are processed safely and concurrently, demonstrating the power of threading for managing tasks that require coordination.

4o mini

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

PROGRAM CODE:

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

class Incrementer implements Runnable {
    private int value = 0;

    @Override
    public void run() {
        while (true) {
            value++;
            System.out.println("Current value: " + value);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
                break;
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\samas\java projects\clg> java prac35

        Samad Sama
        23DCS112
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
Current value: 6
```

CONCLUSION:

This program demonstrates a simple use of threads in Java by incrementing a variable and displaying its value every second. The use of the sleep() method illustrates how to control the execution flow of a thread effectively, allowing for timed operations in multi-threaded applications.

4o mini

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 FirstThread extends Thread {
    @Override
    public void run() {
        System.out.println("FIRST thread is running with priority: " + getPriority());
    }
}

class SecondThread extends Thread {
    @Override
    public void run() {
        System.out.println("SECOND thread is running with priority: " + getPriority());
    }
}
```



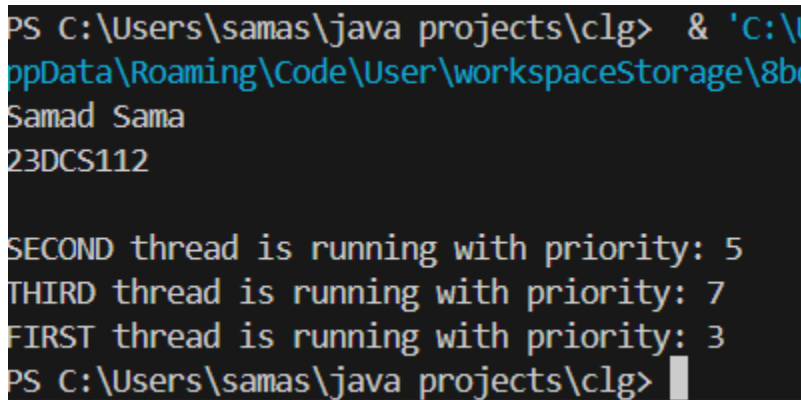
```
class ThirdThread extends Thread {
    @Override
    public void run() {
        System.out.println("THIRD thread is running with priority: " + getPriority());
    }
}

public class prac36 {
    public static void main(String[] args) {
        FirstThread firstThread = new FirstThread();
        SecondThread secondThread = new SecondThread();
        ThirdThread thirdThread = new ThirdThread();

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

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

OUTPUT:



```
PS C:\Users\samas\java projects\clg> & 'C:\V
ppData\Roaming\Code\User\workspaceStorage\8b
Samad Sama
23DCS112

SECOND thread is running with priority: 5
THIRD thread is running with priority: 7
FIRST thread is running with priority: 3
PS C:\Users\samas\java projects\clg> █
```

CONCLUSION:

This program demonstrates how to create and manage multiple threads in Java with different priorities. By setting specific priorities for each thread, the program allows the Java runtime to handle the execution order based on these priorities, showcasing basic concepts of thread management in concurrent programming.

37. 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> buffer = new LinkedList<>();
    private final int capacity;

    public ProducerConsumer(int capacity) {
        this.capacity = capacity;
    }

    public void produce(int item) throws InterruptedException {
        synchronized (this) {
            while (buffer.size() == capacity) {
                wait();
            }
            buffer.add(item);
            System.out.println("Produced: " + item);
            notifyAll();
        }
    }

    public int consume() throws InterruptedException {
        synchronized (this) {
            while (buffer.isEmpty()) {
                wait();
            }
            int item = buffer.poll();
            System.out.println("Consumed: " + item);
            notifyAll();
            return item;
        }
    }
}

class Producer extends Thread {
    private final ProducerConsumer pc;

    public Producer(ProducerConsumer pc) {
```

```
        this.pc = pc;
    }

    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                pc.produce(i);
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Consumer extends Thread {
    private final ProducerConsumer pc;

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

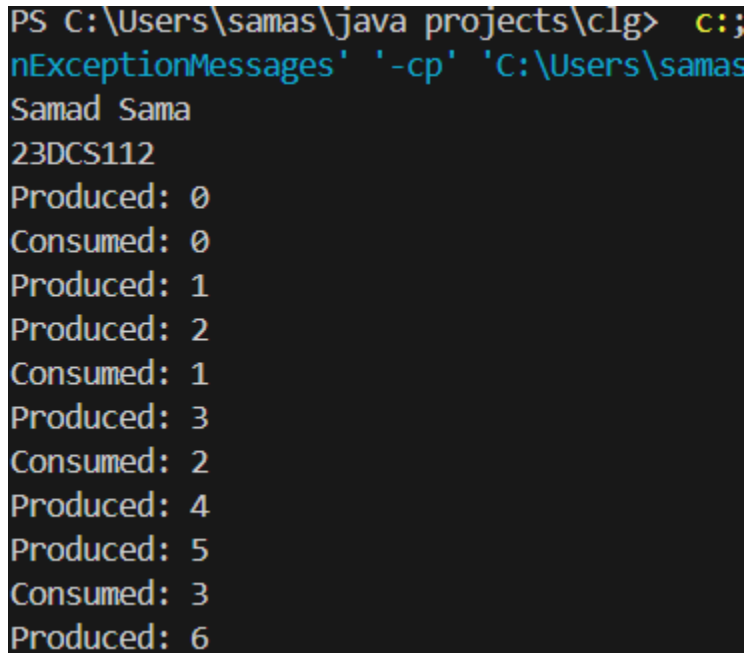
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
            try {
                pc.consume();
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class prac37 {
    public static void main(String[] args) {
        ProducerConsumer pc = new ProducerConsumer(5);
        Producer producer = new Producer(pc);
        Consumer consumer = new Consumer(pc);

        producer.start();
```

```
        consumer.start();
    }
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> c:;
nExceptionMessages' '-cp' 'C:\Users\samas
Samad Sama
23DCS112
Produced: 0
Consumed: 0
Produced: 1
Produced: 2
Consumed: 1
Produced: 3
Consumed: 2
Produced: 4
Produced: 5
Consumed: 3
Produced: 6
```

CONCLUSION:

This program effectively demonstrates the Producer-Consumer problem using thread synchronization in Java. It ensures safe communication between producer and consumer threads through proper handling of shared resources, utilizing the `wait()` and `notifyAll()` methods to synchronize access to the buffer. This implementation serves as a foundational example of handling concurrency issues in multi-threaded applications.

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 CODE:

```
import java.util.ArrayList;

class CustomStack {
    private ArrayList<Object> stackList;

    public CustomStack() {
        stackList = new ArrayList<>();
    }

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

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

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

    public Object pop() {
        if (isEmpty()) {
            return null;
        }
    }
}
```

```
    }
    return stackList.remove(stackList.size() - 1);
}

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

public void display() {
    System.out.println("Stack: " + stackList);
}
}

public class prac38 {
    public static void main(String[] args) {
        CustomStack stack = new CustomStack();

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

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

        stack.display();

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

        stack.display();

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

OUTPUT:

```

PS C:\Users\samas\java projects\clg> & 'C:\
ppData\Roaming\Code\User\workspaceStorage\8b
Stack size: 3
Top element (peek): 30
Stack: [10, 20, 30]
Popped element: 30
Stack size after pop: 2
Stack: [10, 20]
Is stack empty? false
Is stack empty after popping all? true

                Samad Sama
                23DCS112
PS C:\Users\samas\java projects\clg>

```

CONCLUSION:

This program provides a functional implementation of a custom stack using the ArrayList class in Java. It encapsulates the core functionalities of a stack, such as push, pop, peek, and checking for emptiness, making it a suitable structure for managing a collection of elements with Last In, First Out (LIFO) behavior.

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

PROGRAM CODE:

```
import java.util.Arrays;
```

```
public class prac39 {
```

```
    public static <T extends Comparable<T>> void sortArray(T[] array) {
```

```
        Arrays.sort(array);
    }

    public static void main(String[] args) {
        Product[] products = {
            new Product("Laptop", 1500, 4.5),
            new Product("Smartphone", 800, 4.7),
            new Product("Tablet", 400, 4.2)
        };

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

        sortArray(products);

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

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;
    }

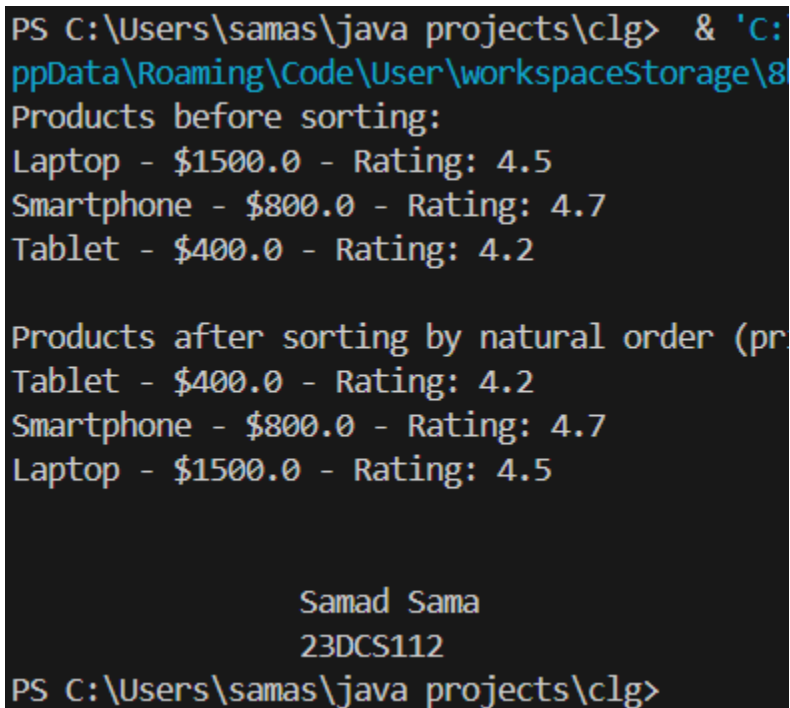
    @Override
    public int compareTo(Product other) {
```



```
        return Double.compare(this.price, other.price);
    }

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

OUTPUT:



```
PS C:\Users\samas\java projects\clg> & 'C:\ProgramData\Roaming\Code\User\workspaceStorage\8
Products before sorting:
Laptop - $1500.0 - Rating: 4.5
Smartphone - $800.0 - Rating: 4.7
Tablet - $400.0 - Rating: 4.2

Products after sorting by natural order (pr
Tablet - $400.0 - Rating: 4.2
Smartphone - $800.0 - Rating: 4.7
Laptop - $1500.0 - Rating: 4.5

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg>
```

CONCLUSION:

This program implements a generic method to sort arrays of objects that implement the Comparable interface. It defines a Product class that represents a product with attributes for name, price, and rating. The program demonstrates sorting an array of Product objects based on price using the generic sorting method. This design allows for flexibility and reusability, enabling the sorting of various object types in a type-safe manner, which is essential in developing scalable applications like e-commerce platforms.

4o mini

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 prac40 {
    public static void main(String[] args) {
        String text = "apple banana apple orange banana apple grape orange orange";
        countWords(text);
    }

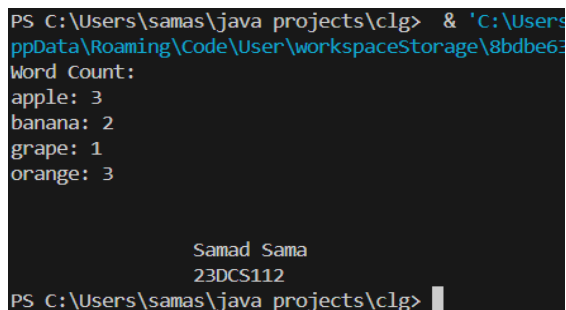
    public static void countWords(String text) {
        String[] words = text.split("\\s+");
        Map<String, Integer> wordCountMap = new HashMap<>();

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

        List<String> sortedWords = new ArrayList<>(wordCountMap.keySet());
        Collections.sort(sortedWords);

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

OUTPUT:



```
PS C:\Users\samas\java projects\clg> & 'C:\Users\
ppData\Roaming\Code\User\workspaceStorage\8bde63
Word Count:
apple: 3
banana: 2
grape: 1
orange: 3

Samad Sama
23DCS112
PS C:\Users\samas\java projects\clg> █
```

CONCLUSION:

This program efficiently counts and displays the occurrences of words in a given text. By utilizing a Map to store word counts and a List to sort the keys, it ensures that the output is presented in alphabetical order. This method demonstrates how to handle word occurrences using collections, making it suitable for text 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 prac41 {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Usage: java prac41 <source-file.java>");
            return;
        }

        String fileName = args[0];
        HashSet<String> keywords = initializeKeywords();
        int keywordCount = countKeywords(fileName, keywords);

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

    private static HashSet<String> initializeKeywords() {
        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);
}

return keywords;
}

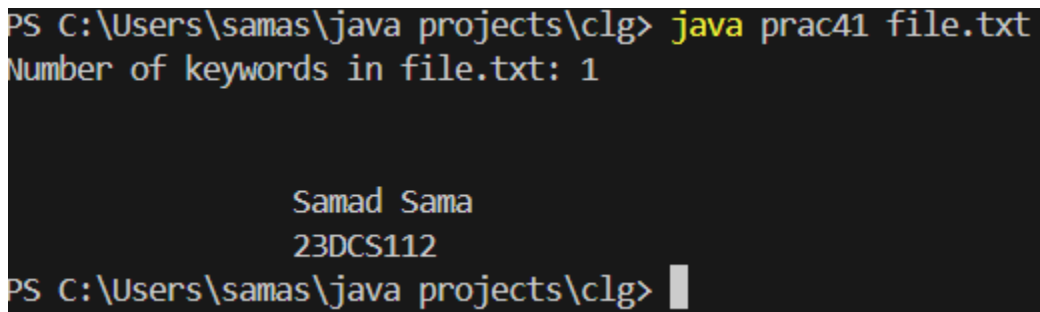
private static int countKeywords(String fileName, HashSet<String> keywords) {
    int count = 0;

    try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
        String line;
        while ((line = reader.readLine()) != null) {
            String[] words = line.split("\\W+"); // Split on non-word characters
            for (String word : words) {
                if (keywords.contains(word)) {
                    count++;
                }
            }
        }
    } catch (IOException e) {
        System.out.println("Error reading file: " + fileName + " - " + e.getMessage());
    }

    return count;
}
```

```
}  
}
```

OUTPUT:



```
PS C:\Users\samas\java projects\clg> java prac41 file.txt  
Number of keywords in file.txt: 1  
  
Samad Sama  
23DCS112  
PS C:\Users\samas\java projects\clg> |
```

CONCLUSION:

This program effectively counts the number of Java keywords present in a specified source file. By leveraging a HashSet to store Java keywords, the program allows for efficient keyword lookups using the `contains()` method. The implementation reads the file line by line, splits the content into words, and checks each word against the keyword set, ultimately providing a count of how many keywords were found. This approach not only demonstrates the practical use of collections in Java but also serves as a useful tool for code analysis, aiding developers in understanding the structure and usage of keywords within Java source code.