JSPM University Pune

Faculty of Science and Technology

School of Computational Sciences

Foundation of Java Programming

(F.Y./S.Y./T.Y, 2023 Course)

LABORATORY MANUAL

(Effective from AY: 2024-25)

JSPM University Pune

S.Y. BCA “BACHELOR OF COMPUTER APPLICATION”

SEMESTER-III

Course Type: BCA	**Lab Course Title:** Foundation of Java Programming Lab	
Course Code: 230GCAB0_03	**Teaching Scheme:** (Hrs./Week)	**Examination Scheme:**
Credits: 2	**Lecture (L): 2** **Tutorial (T):** 0 **Practical (P):** 4 **Experiential Learning (EL):** 0	**Practical (PR):** 40 Marks

List of Laboratory Experiments

Group A

1.	Write a JAVA program to create a simple calculator.
2.	Write a JAVA program to determine if a given number is even or odd.
3.	Write a JAVA program to print the day of the week based on a given number (1-7) using switch statement.
4.	Write a program to check if a given number is an Armstrong number.
5.	Write a JAVA program for String reversal.
6.	Write a JAVA program to display Prime numbers between 1 to 50.
7.	Develop a JAVA program to find the sum of all even numbers between 1 and 100.
8.	Write a JAVA program to **Print a right-angled triangle:** * ** *** ****.
9.	Write a JAVA program to **Print a pyramid:** * *** ***** ******* *********

10	Write a JAVA program to find the sum of all elements of 2D array
Group B	
11	Write a JAVA program to find the minimum and maximum from 1D array.
12	Write a JAVA program to perform the linear search on 1D array.
13	Write a JAVA program to Concatenate two strings.
14	Write a JAVA program to Convert a string to uppercase or lowercase:
15	Write a Java program to perform a binary search on a sorted array.
16	Create a class named Person with attributes name, age, and address. Create objects of this class and assign values to their attributes. Print the details of these objects.
17	Define a class Animal with attributes name and sound. Create objects of different animal classes (e.g., Dog, Cat, Bird) that inherit from Animal, and demonstrate polymorphism by calling their specific makeSound() methods.
18	Create two classes: Shape (superclass) with a method draw(), and Circle (subclass) that overrides the draw() method to display "Drawing Circle". Create objects of both classes and call the draw() method.
19	Create an interface named Animal with the following methods: • void makeSound() • void eat() Create two classes, Dog and Cat, that implement the Animal interface. Implement the makeSound() and eat() methods for each class to represent the specific behaviors of dogs and cats
20	Create a package named shapes and define two classes within it: • **Circle:** This class should have an attribute radius and methods like calculateArea(), calculatePerimeter(), and displayShape().

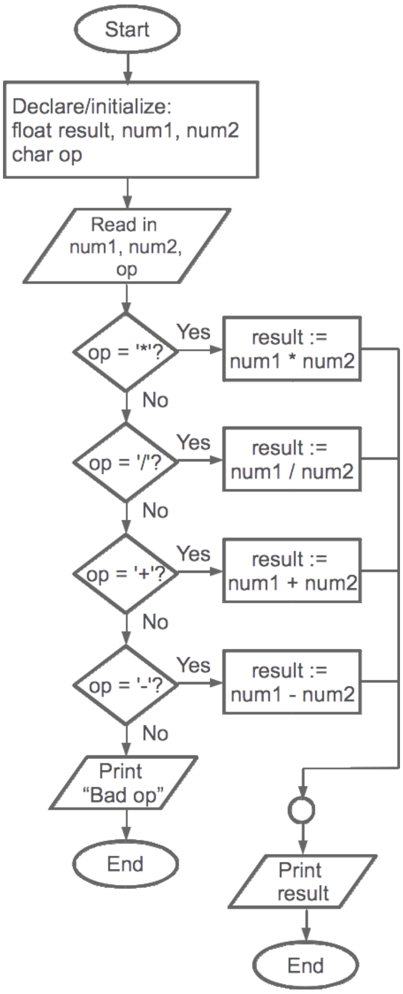
	• **Rectangle:** This class should have attributes length and breadth and methods like calculateArea(), calculatePerimeter(), and displayShape(). Create a separate class named ShapeTester in the default package (or another package if desired) to demonstrate the usage of the shapes package. Create instances of Circle and Rectangle objects and call their methods.
Group C	
21	Create a package named mypackage. Inside this package, define a class MyClass with a method displayMessage () that prints "Hello from MyClass!". Write another class outside the package to import and use this class..
22	Write a JAVA program to display the use of multithreading.
23	Write a JAVA program to demonstrate the use for exception handling.
24	Write a JAVA program to read the contents of a text file and print them to the console.
25	Write a Java program to read from a file, write to a file, and copy content from one file to another.
26	Write a JAVA program to Use a FileReader to read characters from a file one at a time.
27	Write a JAVA program to Create a simple applet that displays a message and an image.
28	Write a JAVA program to Use the drawLine() method of the Graphics object to draw lines.
29	Write a JAVA program to Use the drawOval() method of the Graphics object to draw lines.
30	Write a Demo program in JAVA on Event handling.

Problem Statement: 1. Write a Java program to create a simple calculator.

Aim: To create a simple calculator in Java that performs basic arithmetic operations (addition, subtraction, multiplication, and division) based on user input without using any applet or graphical user interface (GUI).

Objective: To demonstrate the use of basic input/output in Java using the Scanner class. To use the switch-case statement to perform operations based on user input.

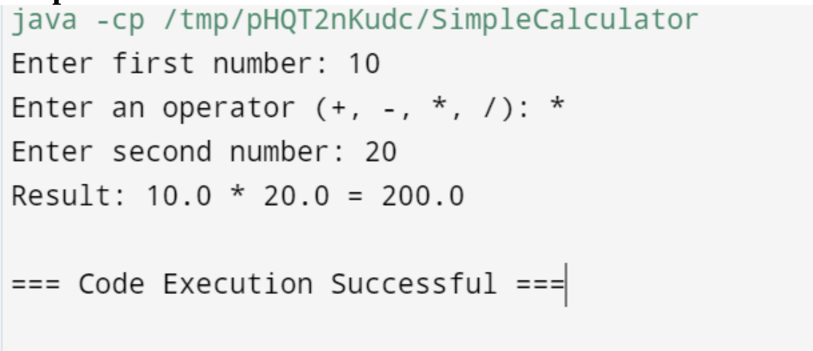
Flowchart:



Source code :

```
import java.util.Scanner;

public class SimpleCalculator {

    public static void main(String[] args) {
        // Create a scanner object to take input from the user
        Scanner scanner = new Scanner(System.in);

        // Variables to store numbers and the result
        double num1, num2, result;
        char operator;

        // Take input from the user
        System.out.print("Enter first number: ");
        num1 = scanner.nextDouble();

        System.out.print("Enter an operator (+, -, *, /): ");
        operator = scanner.next().charAt(0);

        System.out.print("Enter second number: ");
        num2 = scanner.nextDouble();

        // Perform the calculation based on the operator
        switch (operator) {
            case '+':
                result = num1 + num2;
                System.out.println("Result: " + num1 + " + " + num2 + " = " + result);
                break;

            case '-':
                result = num1 - num2;
                System.out.println("Result: " + num1 + " - " + num2 + " = " + result);
                break;

            case '*':
                result = num1 * num2;
                System.out.println("Result: " + num1 + " * " + num2 + " = " + result);
                break;

            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                    System.out.println("Result: " + num1 + " / " + num2 + " = " + result);
                } else {
                    System.out.println("Error: Division by zero is not allowed.");
                }
                break;

            default:
                System.out.println("Error: Invalid operator.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Output :



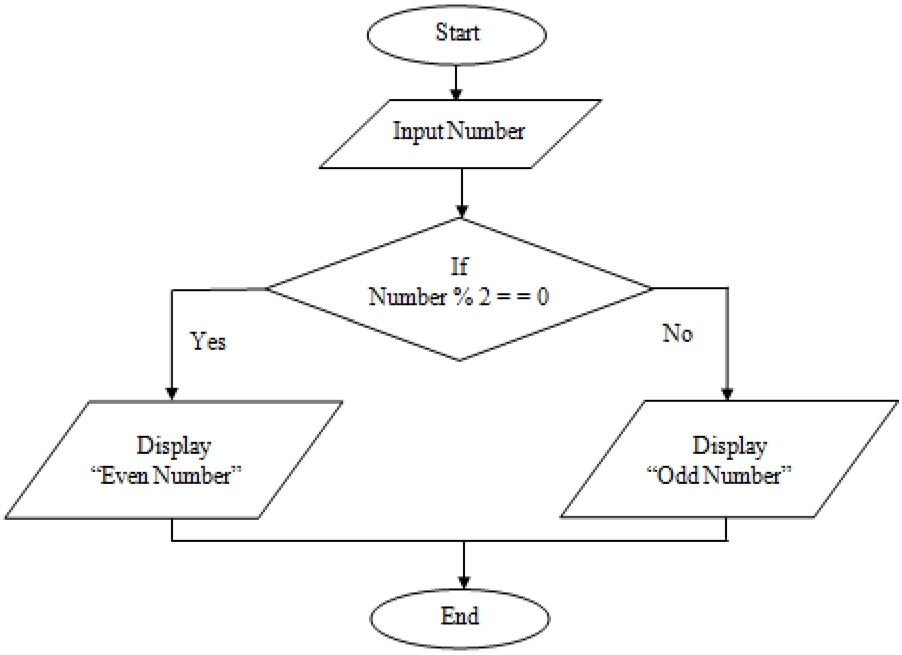
```
java -cp /tmp/pHQT2nKudc/SimpleCalculator
Enter first number: 10
Enter an operator (+, -, *, /): *
Enter second number: 20
Result: 10.0 * 20.0 = 200.0

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

Problem Statement: 2. Write a Java program to determine if given number is even or odd.

Aim: To write a simple Java program to determine whether a given number is even or odd.

Objective: To use basic input/output operations in Java. To demonstrate the use of the modulus operator (%) for determining the remainder. To apply conditional statements (if-else) for checking whether the number is even or odd.

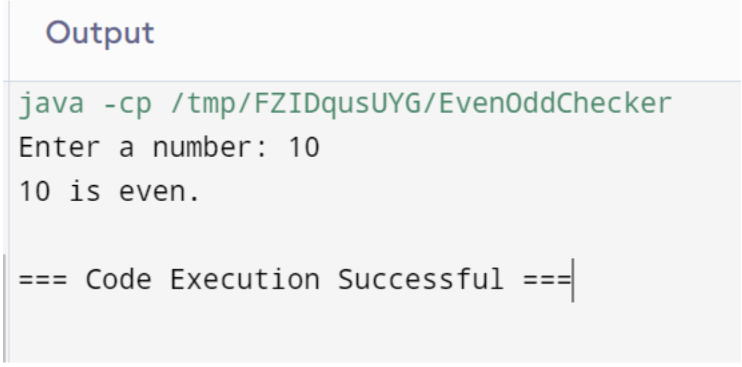
Flowchart:



Source Code:

```
import java.util.Scanner;

public class EvenOddChecker {

    public static void main(String[] args) {
        // Create a Scanner object for input
        Scanner scanner = new Scanner(System.in);

        // Input number
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Check if the number is even or odd
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Output:



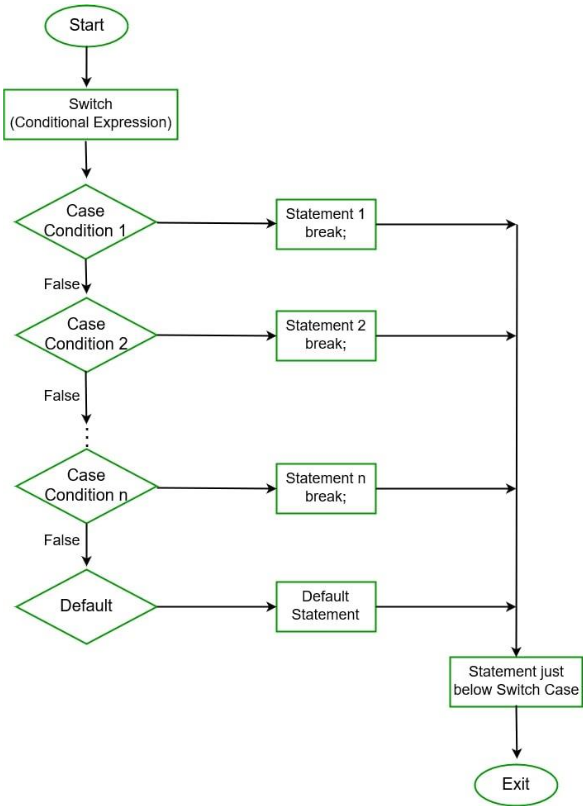
```
java -cp /tmp/FZIDqusUYG/EvenOddChecker
Enter a number: 10
10 is even.

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

Problem Statement: 3. Write Java program to print the day of week based on given number (1-7) using switch statement,

Aim: To write a Java program that prints the corresponding day of the week based on a given number (1-7) using a switch statement.

Objective: To use the switch statement to handle multiple cases for determining the day of the week. To take a number input from the user and print the corresponding day. To demonstrate how the switch control structure works for multiple condition checking.

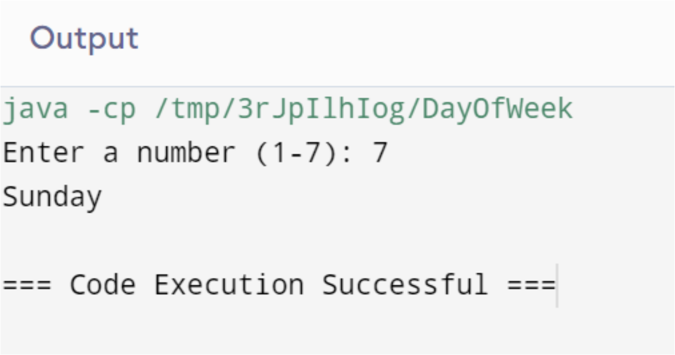
Flowchart:



Code:

```
import java.util.Scanner;

public class DayOfWeek {

    public static void main(String[] args) {
        // Create Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Input: Number representing the day of the week
        System.out.print("Enter a number (1-7): ");
        int day = scanner.nextInt();

        // Switch statement to determine the day of the week
        switch (day) {
            case 1:
                System.out.println("Monday");
                break;
            case 2:
                System.out.println("Tuesday");
                break;
            case 3:
                System.out.println("Wednesday");
                break;
            case 4:
                System.out.println("Thursday");
                break;
            case 5:
                System.out.println("Friday");
                break;
            case 6:
                System.out.println("Saturday");
                break;
            case 7:
                System.out.println("Sunday");
                break;
            default:
                System.out.println("Invalid input! Please enter a number between 1 and 7.");
        }
// Close the scanner
        scanner.close();
    }
}
```

Output:



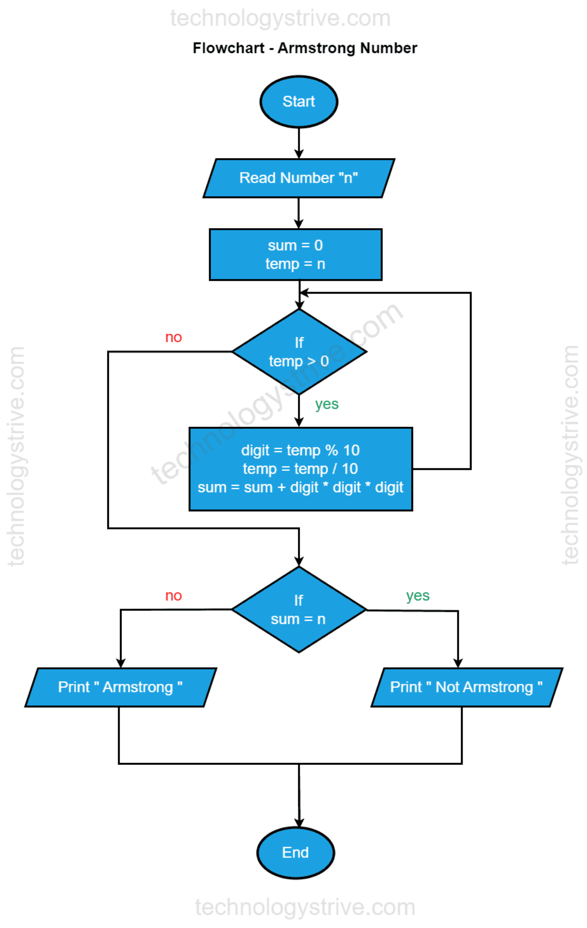
```
java -cp /tmp/3rJpIlhIog/DayOfWeek
Enter a number (1-7): 7
Sunday

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

Problem Statement: 4. Write a Java Program to check if given number is Armstrong or not.

Aim: To write a Java program to check whether a given number is an Armstrong number or not.

Objective: To understand how to break down a number into its digits.To compute the sum of cubes (or power) of the digits and compare it with the original number.
To demonstrate the use of loops and conditional statements in Java to check if a number is Armstrong.

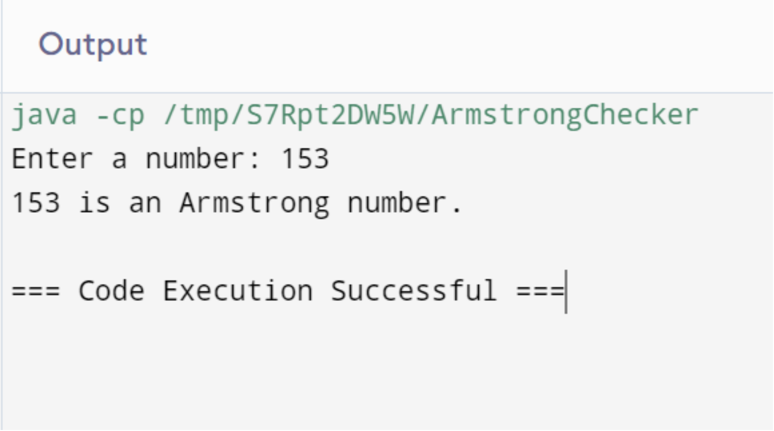
Flowchart:



Code:

```
import java.util.Scanner;

public class ArmstrongChecker {

    public static void main(String[] args) {
        // Create Scanner object to take input from user
        Scanner scanner = new Scanner(System.in);

        // Input: number to be checked
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Variable to store original number
        int originalNumber = number;
        int remainder, result = 0;

        // Calculate the sum of cubes of its digits
        while (originalNumber != 0) {
            remainder = originalNumber % 10;           // Get last digit
            result += Math.pow(remainder, 3);          // Add cube of last digit to result
            originalNumber /= 10;                      // Remove last digit
        }

        // Check if the original number is equal to the result
        if (result == number) {
            System.out.println(number + " is an Armstrong number.");
        } else {
            System.out.println(number + " is not an Armstrong number.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Output:



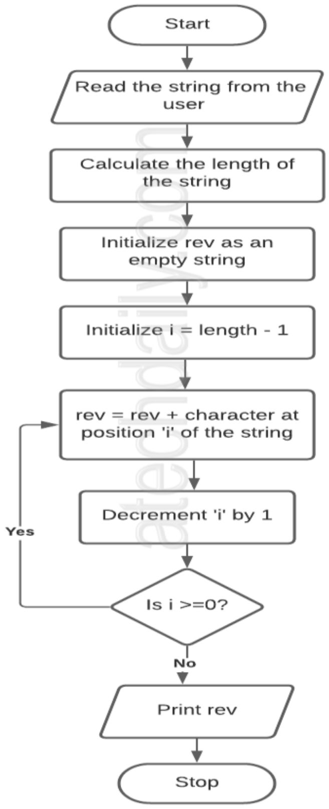
```
java -cp /tmp/S7Rpt2DW5W/ArmstrongChecker
Enter a number: 153
153 is an Armstrong number.

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

Problem Statement: 5.Write Java Program for String reversal.

Aim: To write a Java program to reverse a given string.

Objective: To understand how to manipulate strings in Java.
To demonstrate the use of loops for reversing the order of characters in a string.
To apply string manipulation methods provided by the Java API.

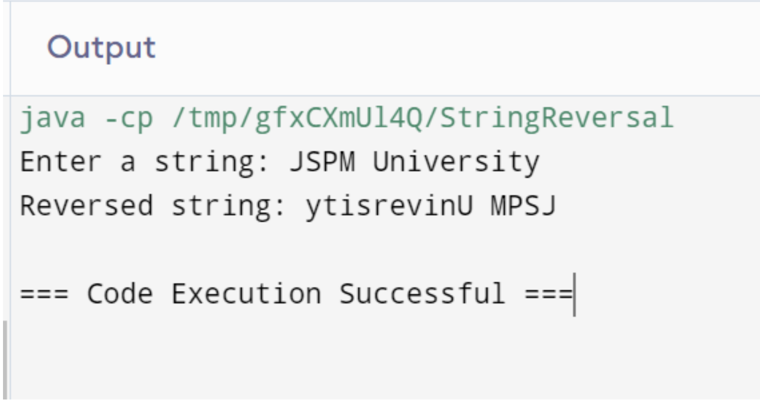
Flowchart:


Start
Read the string from the user
Calculate the length of the string
Initialize rev as an empty string
Initialize i = length - 1
rev = rev + character at position 'i' of the string
Decrement 'i' by 1
Yes
Is i >=0?
No
Print rev
Stop

Code:

```java
import java.util.Scanner;

public class StringReversal {

    public static void main(String[] args) {
        // Create a Scanner object for input
        Scanner scanner = new Scanner(System.in);

        // Input: Read the string from the user
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        // Reverse the string using a loop
        String reversed = "";
        for (int i = input.length() - 1; i >= 0; i--) {
            reversed += input.charAt(i);
        }

        // Output: Display the reversed string
        System.out.println("Reversed string: " + reversed);

        // Close the scanner
        scanner.close();
    }
}
```

Output:



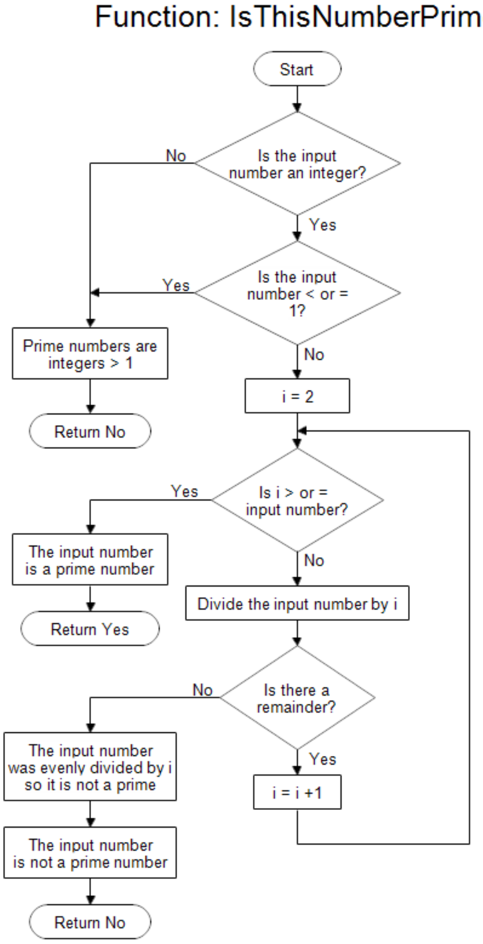
```
java -cp /tmp/gfxCXmUl4Q/StringReversal
Enter a string: JSPM University
Reversed string: ytisrevinU MPSJ

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

Problem Statement: 6.Write a Java program to print prime numbers between 1 to 50.

Aim: To write a Java program that prints all prime numbers between 1 and 50.

Objective:
To understand the concept of prime numbers.
To use loops and conditional statements to check if a number is prime.
To display all prime numbers within the specified range of 1 to 50.

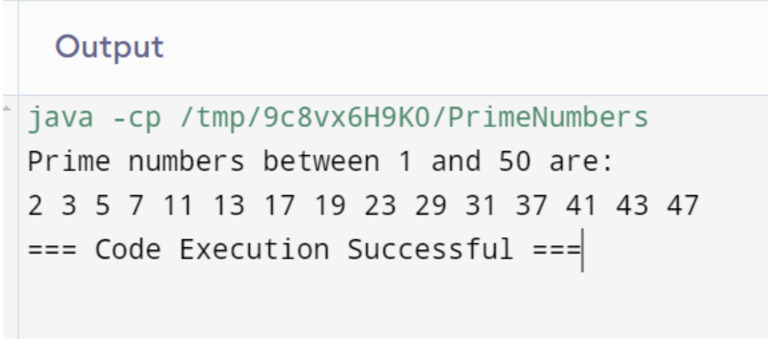
Flowchart:


Function: IsThisNumberPrime
Start
Is the input number an integer?
No
Yes
Is the input number < or = 1?
Yes
No
Prime numbers are integers > 1
Return No
i = 2
Is i > or = input number?
Yes
No
The input number is a prime number
Return Yes
Divide the input number by i
Is there a remainder?
No
Yes
The input number was evenly divided by i so it is not a prime
i = i +1
The input number is not a prime number
Return No

Code:

```
public class PrimeNumbers {

    public static void main(String[] args) {
        System.out.println("Prime numbers between 1 and 50 are: ");

        // Loop through numbers from 1 to 50
        for (int num = 2; num <= 50; num++) {
            boolean isPrime = true;

            // Check if num is prime by dividing it with numbers less than itself
            for (int i = 2; i <= num / 2; i++) {
                if (num % i == 0) {
                    isPrime = false;
                    break;
                }
            }

            // If number is prime, print it
            if (isPrime) {
                System.out.print(num + " ");
            }
        }
    }
}
```

Output:



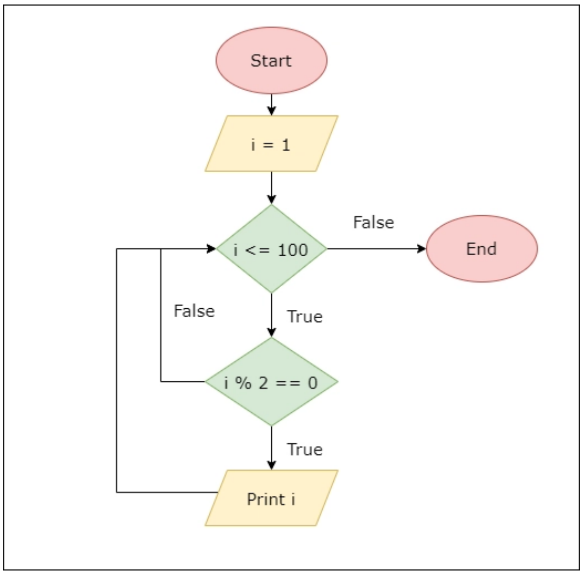
```
java -cp /tmp/9c8vx6H9KO/PrimeNumbers
Prime numbers between 1 and 50 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
=== Code Execution Successful ===
```

Problem Statement:7 Develop a Java Program to find sum of all even numbers between 1 to 100.

Aim: To develop a Java program to find the sum of all even numbers between 1 and 100.

Objective: To use loops to iterate through numbers from 1 to 100.To check if a number is even using the modulus operator (%). To calculate the sum of all even numbers between 1 and 100 and display the result.

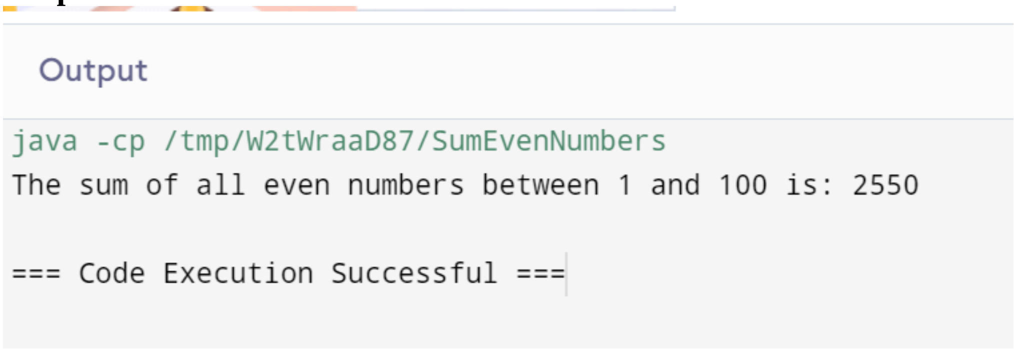
Flowchart:



Code:

```
public class SumEvenNumbers {

    public static void main(String[] args) {
        int sum = 0;

        // Loop through numbers from 1 to 100
        for (int i = 1; i <= 100; i++) {
            // Check if the number is even
            if (i % 2 == 0) {
                sum += i;  // Add the even number to the sum
            }
        }

        // Print the sum of all even numbers
        System.out.println("The sum of all even numbers between 1 and 100 is: " + sum);
    }
}
```

Output:



```
java -cp /tmp/W2tWraaD87/SumEvenNumbers
The sum of all even numbers between 1 and 100 is: 2550

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

Problem Statement: 8.Write a Java program to Print a right-angled triangle:

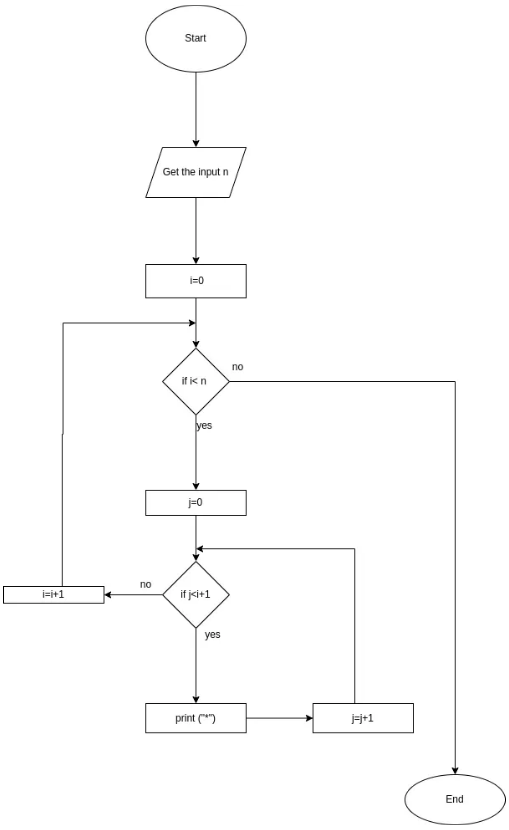
```
*
**
***
****.
```

Aim: To write a Java program to print a right-angled triangle pattern of asterisks (*).

Objective:

To understand how to use nested loops in Java.
To learn how to control the number of characters printed in each line using loops.
To print a right-angled triangle of stars, where the number of stars in each row increases progressively.

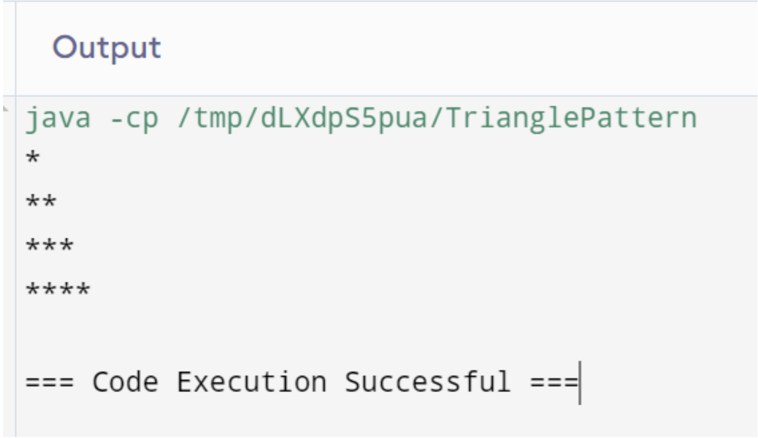
Flowchart:



Code:

```
public class TrianglePattern {

    public static void main(String[] args) {
        int rows = 4;  // Number of rows for the triangle

        // Outer loop to handle the number of rows
        for (int i = 1; i <= rows; i++) {
            // Inner loop to print stars in each row
            for (int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            // Print a new line after each row
            System.out.println();
        }
    }
}
```

Output:



Output

```
java -cp /tmp/dLXdpS5pua/TrianglePattern
*
**
***
****


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

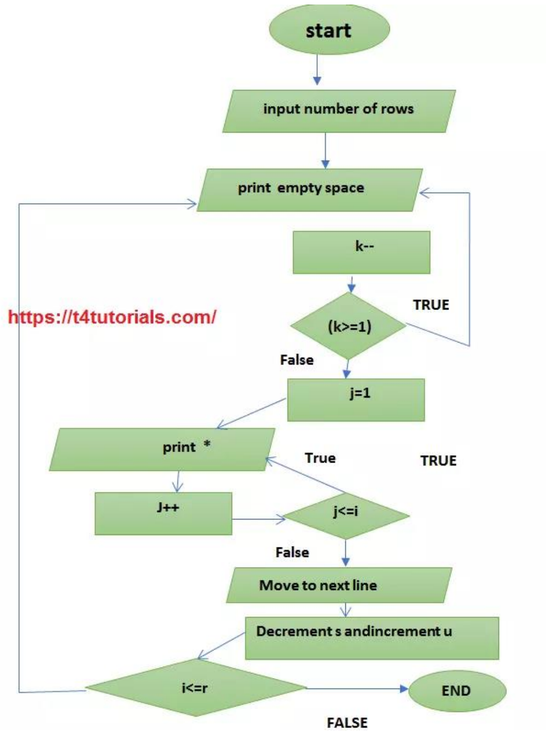
Problem Statement: 9. Write a JAVA program to Print a pyramid:

```
    *
   ***
  *****
 *******
*********
```

Aim: To write a Java program that prints a pyramid pattern of asterisks (*).

Objective:
To understand the use of nested loops for printing patterns.
To learn how to manage spaces and stars (*) to create a pyramid shape.
To print a pyramid where the number of stars increases progressively while being centered.

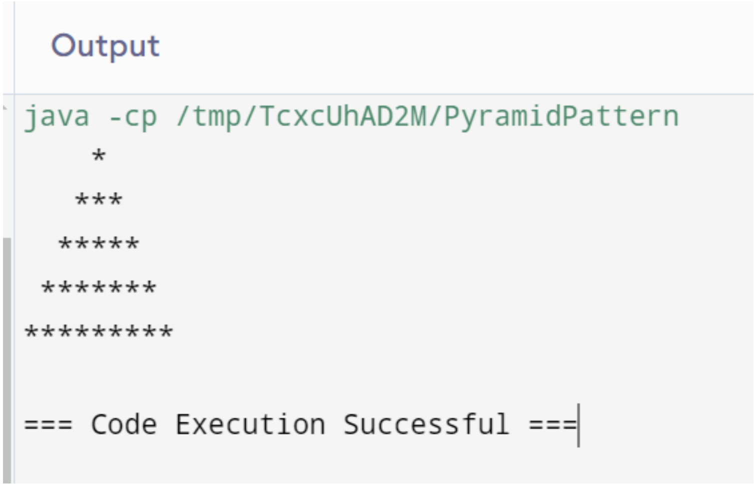
Flowchart:



Code:

```
public class PyramidPattern {

    public static void main(String[] args) {
        int rows = 5;  // Number of rows for the pyramid

        // Outer loop to handle the number of rows
        for (int i = 1; i <= rows; i++) {
            // Print leading spaces
            for (int j = rows; j > i; j--) {
                System.out.print(" ");
            }

            // Print stars for each row
            for (int k = 1; k <= (2 * i - 1); k++) {
                System.out.print("*");
            }

            // Move to the next line after each row
            System.out.println();
        }
    }
}
```

Output:



```
java -cp /tmp/TcxcUhAD2M/PyramidPattern
    *
   ***
  *****
 *******
*********

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

Problem Statement: 10. Write a JAVA program to find the sum of all elements of 2D array.

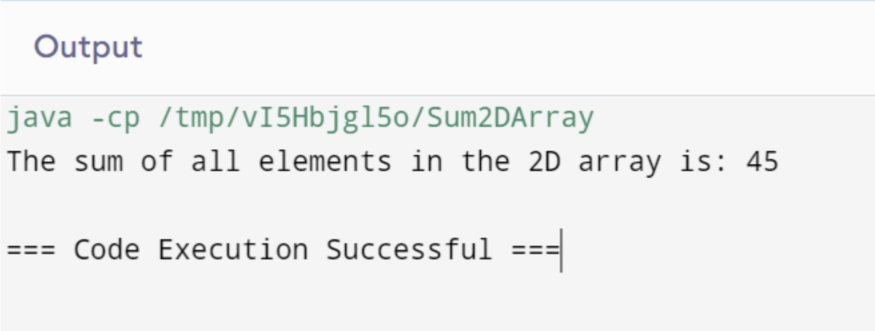
Aim: To write a Java program that calculates the sum of all elements in a 2D array.

Objective: To understand how to work with 2D arrays in Java.To learn how to traverse through rows and columns of a 2D array. To calculate the sum of all elements within the 2D array.

Code:

```
public class Sum2DArray {

    public static void main(String[] args) {
        // Define a 2D array with some elements
        int[][] array = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        int sum = 0;  // Variable to store the sum of all elements

        // Traverse through each element of the 2D array
        for (int i = 0; i < array.length; i++) {
            for (int j = 0; j < array[i].length; j++) {
                sum += array[i][j];  // Add each element to sum
            }
        }

        // Output the sum of all elements
        System.out.println("The sum of all elements in the 2D array is: " + sum);
    }
}
```

Output:



```
java -cp /tmp/vI5Hbjgl5o/Sum2DArray
The sum of all elements in the 2D array is: 45


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

Problem Statement: 11. Write a JAVA program to find the minimum and maximum from 1D array.

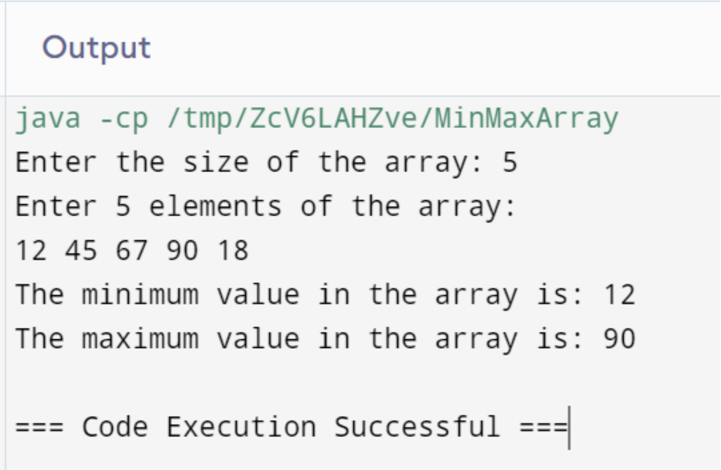
Aim: To write a Java program to find the minimum and maximum values from a 1D array.

Objective:
To understand how to work with 1D arrays in Java.
To traverse an array and compare its elements to find the minimum and maximum values.
To learn how to use basic conditional statements to track the smallest and largest elements in an array.

Code:

```
import java.util.Scanner;

public class MinMaxArray {

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

        // Get the size of the array
        System.out.print("Enter the size of the array: ");
        int size = scanner.nextInt();

        // Declare the array
        int[] array = new int[size];

        // Input elements of the array
        System.out.println("Enter " + size + " elements of the array:");
        for (int i = 0; i < size; i++) {
            array[i] = scanner.nextInt();
        }

        // Initialize min and max with the first element of the array
        int min = array[0];
        int max = array[0];

        // Traverse the array to find min and max

  for (int i = 1; i < size; i++) {
            if (array[i] < min) {
                min = array[i];  // Update min
            }
            if (array[i] > max) {
                max = array[i];  // Update max
            }
        }
        // Output the min and max
        System.out.println("The minimum value in the array is: " + min);
        System.out.println("The maximum value in the array is: " + max);
    }
}
```

Output:



```
java -cp /tmp/ZcV6LAHZve/MinMaxArray
Enter the size of the array: 5
Enter 5 elements of the array:
12 45 67 90 18
The minimum value in the array is: 12
The maximum value in the array is: 90

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

Problem Statement: 12. Write a JAVA program to perform the linear search on 1D array.

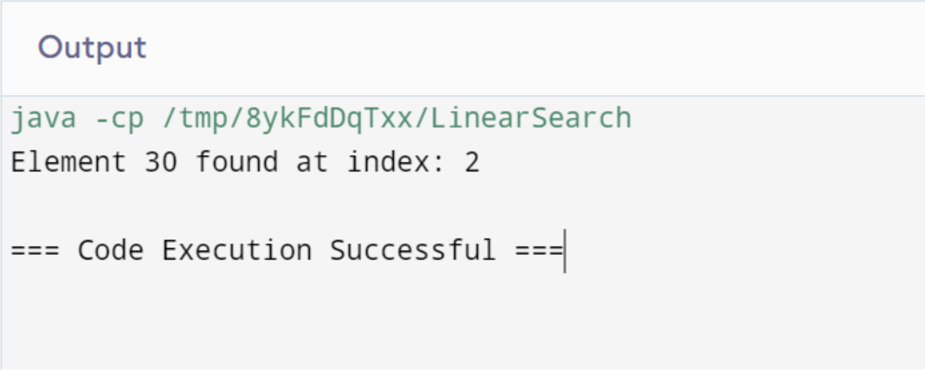
Aim: To demonstrate the implementation of a linear search algorithm in Java, which searches for a specified element within a one-dimensional array by checking each element sequentially.

Objective: Conceptual Understanding: To grasp the concept of linear search, which involves traversing through an array to find a specific value. Implementation: To write a method that performs the linear search by iterating over each element in the array and comparing it with the target value.Usage of Java Constructs: To utilize loops (for loop) and conditional statements (if statement) in Java to achieve the search functionality.

Code:

```
public class LinearSearch {

    public static int linearSearch(int[] array, int target) {
        // Iterate over each element in the array
        for (int i = 0; i < array.length; i++) {
            // If the current element matches the target, return the index
            if (array[i] == target) {
                return i;
            }
        }
        // Return -1 if the target is not found in the array
        return -1;
    }

    public static void main(String[] args) {
        // Example array to perform the search on
        int[] numbers = {10, 20, 30, 40, 50};

        // Target value we want to find in the array
        int target = 30;

        // Perform the linear search and capture the result
        int result = linearSearch(numbers, target);

        // Print the result of the search
        if (result == -1) {

  System.out.println("Element " + target + " not found in the array.");
        } else {
            System.out.println("Element " + target + " found at index: " + result);
        }
    }
}
```

Output:



```
java -cp /tmp/8ykFdDqTxx/LinearSearch
Element 30 found at index: 2

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

Problem Statement: 13. Write a JAVA program to Concatenate two strings.

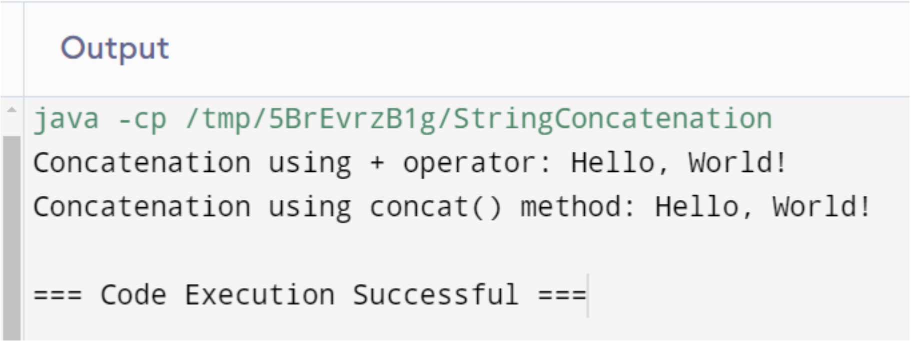
Aim: The aim of this program is to demonstrate how to concatenate two strings in Java.

Objectives: 1. To understand and apply the concept of string concatenation. To use the `+` operator and the `concat()` method for string concatenation.To observe the differences in using different methods for concatenation and understand their behavior.

Code:

```
public class StringConcatenation {

    public static void main(String[] args) {
        // Initialize two strings to concatenate
        String string1 = "Hello, ";
        String string2 = "World!";

        // Concatenate using the + operator
        String concatenatedString1 = string1 + string2;

        // Concatenate using the concat() method
        String concatenatedString2 = string1.concat(string2);

        // Print the results
        System.out.println("Concatenation using + operator: " + concatenatedString1);
        System.out.println("Concatenation using concat() method: " + concatenatedString2);
    }
}
```

Output:



```
java -cp /tmp/5BrEvrzB1g/StringConcatenation
Concatenation using + operator: Hello, World!
Concatenation using concat() method: Hello, World!

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

Problem Statement: 14. Write a JAVA program to Convert a string to uppercase or lowercase:

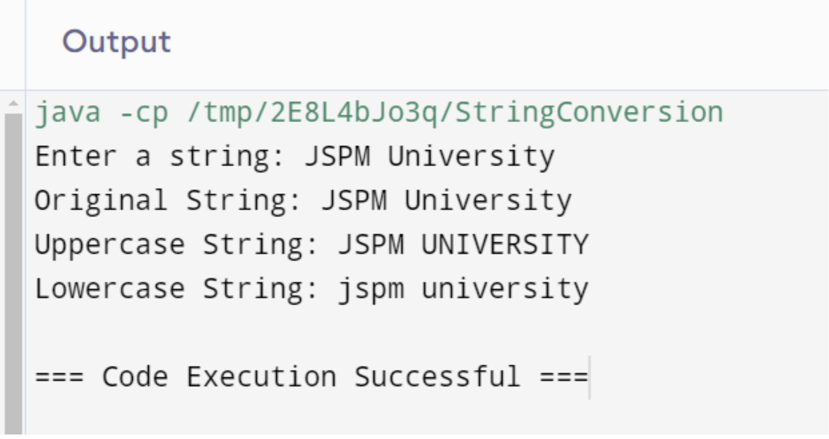
Aim: The aim of this program is to demonstrate how to convert a string to uppercase and lowercase in Java.

Objectives: To understand and apply the `toUpperCase()` and `toLowerCase()` methods of the `String` class.
To get user input for the string and display the converted results.
To observe the behavior of string conversion methods.

Code:

```
public class StringConversion {

    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a string
        System.out.print("Enter a string: ");
        String inputString = scanner.nextLine();

        // Convert the string to uppercase
        String uppercaseString = inputString.toUpperCase();

        // Convert the string to lowercase
        String lowercaseString = inputString.toLowerCase();

        // Display the results
        System.out.println("Original String: " + inputString);
        System.out.println("Uppercase String: " + uppercaseString);
        System.out.println("Lowercase String: " + lowercaseString);

        // Close the scanner
        scanner.close();
    }
}
```

Output:



```
java -cp /tmp/2E8L4bJo3q/StringConversion
Enter a string: JSPM University
Original String: JSPM University
Uppercase String: JSPM UNIVERSITY
Lowercase String: jspm university

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

Problem Statement: 15. Write a Java program to perform a binary search on a sorted array.

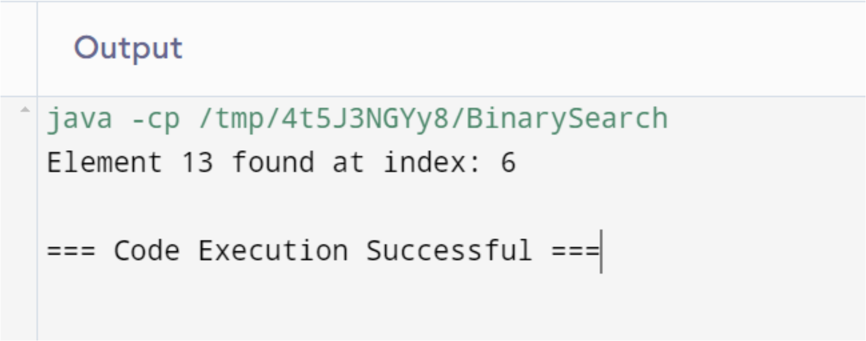
Aim: The aim of this program is to demonstrate how to perform a binary search on a sorted array in Java.

Objectives: To understand the binary search algorithm, which works on the principle of divide and conquer.
To implement a method that searches for a target value within a sorted array and returns its index.
To observe the efficiency of binary search compared to linear search.

Code:

```
public class BinarySearch {


    public static int binarySearch(int[] array, int target) {
        int left = 0;
        int right = array.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            // Check if target is present at mid
            if (array[mid] == target) {
                return mid;
            }

            // If target is greater, ignore the left half
            if (array[mid] < target) {
                left = mid + 1;
            }


// If target is smaller, ignore the right half
            else {
                right = mid - 1;
            }
        }

        // Target was not found
        return -1;
    }

    public static void main(String[] args) {
        // Example sorted array
        int[] sortedArray = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};
        // Target value to search for
        int target = 13;

        // Perform binary search
        int result = binarySearch(sortedArray, target);

        // Print the result
        if (result == -1) {
            System.out.println("Element " + target + " not found in the array.");
        } else {
            System.out.println("Element " + target + " found at index: " + result);
        }
    }
}
```

Output:



```
java -cp /tmp/4t5J3NGYy8/BinarySearch
Element 13 found at index: 6

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

Problem Sataement 16: Create a class named Person with attributes name, age, and address. Create objects of this class and assign values to their attributes. Print the details of these objects.

Aim: The aim of the task is to understand the concepts of object-oriented programming (OOP) in Python, specifically focusing on:

- **Class creation:** Creating a class (Person) with attributes like name, age, and address.
- **Object instantiation:** Creating instances (objects) of the Person class.
- **Attribute assignment:** Assigning values to the attributes of each object.
- **Object details display:** Printing out the details of the created objects.

Objective: This exercise helps you understand how to define a class, create objects from it, assign values to object attributes, and access these attributes.

Source Code:

```
// Define the Person class
class Person {
    // Attributes of the class
    String name;
    int age;
    String address;

    // Method to print the details of the person
    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
    }

    public static void main(String[] args) {
        // Create objects of the Person class
        Person person1 = new Person();
        Person person2 = new Person();

        // Assign values to the attributes for person1
        person1.name = "Alice";
        person1.age = 25;
        person1.address = "123 Main St";

        // Assign values to the attributes for person2
        person2.name = "Bob";
        person2.age = 30;
        person2.address = "456 Maple Ave";

        // Print the details of each person
        person1.printDetails();
        System.out.println();  // Print a blank line between persons
        person2.printDetails();
    }
}
```

Output:

```
Name: Alice
Age: 25
Address: 123 Main St

Name: Bob
Age: 30
Address: 456 Maple Ave
```

Problem Sataement 17: Define a class Animal with attributes name and sound. Create objects of different animal classes (e.g., Dog, Cat, Bird) that inherit from Animal, and demonstrate polymorphism by calling their specific makeSound() methods.

Aim: of this task is to demonstrate **inheritance** and **polymorphism** in object-oriented programming (OOP). Specifically, it involves creating a base class Animal with common attributes, and then extending this class to create subclasses for specific animals (like Dog, Cat, Bird). The focus is on how these subclasses can have their own unique implementation of a method (makeSound()), illustrating polymorphism.

Objectives:

1. **Class Definition and Inheritance**:
   - Define a base class Animal with common attributes (e.g., name and sound).
   - Create subclasses (e.g., Dog, Cat, Bird) that inherit from Animal.
2. **Method Overriding**:
   - Override the makeSound() method in each subclass to give a specific behavior (e.g., a dog barks, a cat meows, etc.).
3. **Polymorphism**:
   - Demonstrate polymorphism by calling the makeSound() method on objects of different subclasses through the Animal reference, showing how the same method can behave differently based on the object.
4. **Object Creation**:
   - Instantiate objects of the subclasses and call their respective methods to show their unique sounds.
5. **Code Reusability**:
   - Reuse common attributes and methods through inheritance, reducing redundancy.

Source Code:

```
// Base class Animal
class Animal {
    // Attributes
    String name;

    // Constructor for Animal class
    public Animal(String name) {
        this.name = name;
    }

    // Method to be overridden by subclasses
    public void makeSound() {
        System.out.println("This animal makes a sound.");
    }
}

// Subclass Dog that inherits from Animal
class Dog extends Animal {

    // Constructor for Dog class
    public Dog(String name) {
        super(name);  // Call the constructor of the Animal class
    }

    // Overriding the makeSound method
    @Override
    public void makeSound() {
        System.out.println(name + " says: Woof Woof!");
    }
}

// Subclass Cat that inherits from Animal
class Cat extends Animal {

    // Constructor for Cat class
    public Cat(String name) {
        super(name);  // Call the constructor of the Animal class
    }

    // Overriding the makeSound method
    @Override
    public void makeSound() {
        System.out.println(name + " says: Meow!");
    }
}

// Subclass Bird that inherits from Animal
class Bird extends Animal {

    // Constructor for Bird class
    public Bird(String name) {
        super(name);  // Call the constructor of the Animal class
    }

    // Overriding the makeSound method
    @Override
    public void makeSound() {
        System.out.println(name + " says: Tweet Tweet!");
    }
}

// Main class to demonstrate polymorphism
public class Main {
    public static void main(String[] args) {
        // Creating objects of different animal types
        Animal myDog = new Dog("Buddy");
        Animal myCat = new Cat("Whiskers");
        Animal myBird = new Bird("Tweety");

        // Demonstrating polymorphism by calling makeSound on different animals
        myDog.makeSound();  // Output: Buddy says: Woof Woof!
        myCat.makeSound();  // Output: Whiskers says: Meow!
        myBird.makeSound(); // Output: Tweety says: Tweet Tweet!
    }
}
```

Output:

```
Buddy says: Woof Woof!
Whiskers says: Meow!
Tweety says: Tweet Tweet!
```

Problem Sataement 18: Create two classes: Shape (superclass) with a method draw(), and Circle (subclass) that overrides the draw() method to display "Drawing Circle". Create objects of both classes and call the draw() method.

Aim:

The aim of this task is to demonstrate **inheritance** and **method overriding** in object-oriented programming (OOP). Specifically, it involves creating a superclass Shape with a general method draw(), and then creating a subclass Circle that overrides this method to provide a specific implementation. The task also involves showing how the draw() method behaves differently when called on an object of the superclass and the subclass, illustrating polymorphism.

Objectives:

1. **Class Definition and Inheritance**:
    - Define a base class Shape that contains a method draw() meant to be overridden by subclasses.
    - Create a subclass Circle that inherits from Shape and provides its own implementation of the draw() method.
2. **Method Overriding**:
    - Demonstrate **method overriding** by implementing the draw() method in the Circle class to display "Drawing Circle".
3. **Polymorphism**:
    - Show polymorphism by creating objects of both the Shape and Circle classes and calling the draw() method on each. This demonstrates how the method behaves differently for the parent class (Shape) and the child class (Circle).
4. **Object Creation**:
    - Instantiate objects of both the Shape and Circle classes and call their respective draw() methods to highlight the differences in behavior.
5. **Code Reusability**:
    - Reuse the draw() method signature through inheritance and override it in the subclass to provide specific functionality for Circle.

Source Code:

```
// Base class Shape
class Shape {
   // Method to be overridden by subclasses
   public void draw() {
      System.out.println("Drawing Shape");
   }
}
```

```
// Subclass Circle that inherits from Shape
class Circle extends Shape {
    // Overriding the draw method
    @Override
    public void draw() {
        System.out.println("Drawing Circle");
    }
}

// Main class to demonstrate method overriding and polymorphism
public class Main {
    public static void main(String[] args) {
        // Create an object of the Shape class
        Shape shape = new Shape();
        // Create an object of the Circle class
        Circle circle = new Circle();

        // Call the draw method on the Shape object
        shape.draw();  // Output: Drawing Shape

        // Call the draw method on the Circle object
        circle.draw();  // Output: Drawing Circle
    }
}
```

Output:

```
Drawing Shape
Drawing Circle
```

Problem Sataement 19: Create an interface named Animal with the following methods:

- void makeSound()
- void eat()

Create two classes, Dog and Cat, that implement the Animal interface. Implement the makeSound() and eat() methods for each class to represent the specific behaviors of dogs and cats

Aim:

The aim of this task is to demonstrate the principles of **interface implementation** in object-oriented programming (OOP). It showcases how multiple classes can implement a common interface and provide their own specific behavior for the methods defined in the interface.

Objective:

1. **Define an Interface**: Create an interface named Animal that declares two methods: makeSound() and eat(). The interface serves as a blueprint for animal behavior.
2. **Implement the Interface**: Create two classes, Dog and Cat, that implement the Animal interface. Each class will provide concrete implementations of the makeSound() and eat() methods to reflect the distinct behaviors of dogs and cats.
3. **Demonstrate Polymorphism**: By using the interface, it is possible to treat objects of different types (e.g., Dog and Cat) uniformly through polymorphism. The program should allow calling these methods regardless of the underlying class type.

Source Code:

```
// Define the Animal interface
interface Animal {
    void makeSound();
    void eat();
}

// Implement the Animal interface in Dog class
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }

    @Override
    public void eat() {
        System.out.println("Dog eats bone");
    }
}

// Implement the Animal interface in Cat class
class Cat implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat meows");
    }

    @Override
    public void eat() {
        System.out.println("Cat eats fish");
    }
}

// Main class to test the implementation
public class Main {
    public static void main(String[] args) {
        // Create objects of Dog and Cat classes
        Animal dog = new Dog();
        Animal cat = new Cat();

        // Call methods on Dog object
        dog.makeSound();
        dog.eat();

        // Call methods on Cat object
        cat.makeSound();
        cat.eat();
    }
}
```

Output:

```
Dog barks
Dog eats bone
Cat meows
Cat eats fish
```

Problem Statement 20: Create a package named shapes and define two classes within it:

- Circle: This class should have an attribute radius and methods like calculateArea(), calculatePerimeter(), and displayShape().
- Rectangle: This class should have attributes length and breadth and methods like calculateArea(), calculatePerimeter(), and displayShape().

Create a separate class named ShapeTester in the default package (or another package if desired) to demonstrate the usage of the shapes package. Create instances of Circle and Rectangle objects and call their methods.

Aim: Aim is to test your understanding of object-oriented programming concepts, specifically:

- **Package creation:** Creating a package to organize related classes.
- **Class definition:** Defining classes with attributes and methods.
- **Object instantiation:** Creating objects from classes.
- **Method invocation:** Calling methods on objects to perform actions.

Objective:

- Design and implement a modular code structure using packages.
- Create classes that represent real-world entities.
- Encapsulate data and behavior within classes.
- Use objects to interact with and manipulate data.

Source Code:

shapes/Circle.java

```
package shapes;

public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

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

    public double calculatePerimeter() {
        return 2 * Math.PI * radius;
    }

    public void displayShape() {
        System.out.println("Circle with radius " + radius);
    }
}
```

shapes/Rectangle.java

```
package shapes;

public class Rectangle {
    private double length;
    private double breadth;

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

    public double calculateArea() {
        return length * breadth;
    }

    public double calculatePerimeter() {
        return 2 * (length + breadth);
    }

    public void displayShape() {
        System.out.println("Rectangle with length " + length + " and breadth " + breadth);
    }
}
```

ShapeTester.java

```
import shapes.Circle;
import shapes.Rectangle;

public class ShapeTester {
    public static void main(String[] args) {
        Circle circle = new Circle(5.0);
        Rectangle rectangle = new Rectangle(4.0, 3.0);

        System.out.println("Circle:");
        circle.displayShape();
        System.out.println("Area: " + circle.calculateArea());
        System.out.println("Perimeter: " + circle.calculatePerimeter());

        System.out.println("\nRectangle:");
        rectangle.displayShape();
        System.out.println("Area: " + rectangle.calculateArea());
        System.out.println("Perimeter: " + rectangle.calculatePerimeter());
    }
}
```

Output:

```
Circle:
Circle with radius 5.0
Area: 78.53981633974483
Perimeter: 31.41592653589793

Rectangle:
Rectangle with length 4.0 and breadth 3.0
 Area: 12.0
Perimeter: 14.0
```

Problem Statement 21: Create a package named mypackage. Inside this package, define a class MyClass with a method displayMessage () that prints "Hello from MyClass!". Write another class outside the package to import and use this class..

A**im** of this exercise is to demonstrate the creation, organization, and utilization of Python packages and modules, which is a fundamental aspect of structuring Python projects for better code reusability and maintainability.

O**bjectives** are as follows:

1. **Package Creation**: Learn how to create a Python package by defining a directory with an __init__.py file.
2. **Class Definition**: Define a class (MyClass) inside the package with a specific method (displayMessage()), which will serve as a reusable component.
3. **Method Implementation**: Implement a simple method within the class that outputs a message.
4. **Importing from a Package**: Write another class or script outside the package that demonstrates how to import and use the class (MyClass) from the package.
5. **Practical Understanding of Python Modules**: Understand how Python packages work in practice, enabling the creation of modular and organized codebases.

Source Code:

Here’s how you can implement the same functionality in Java:

Step 1: Create the package and class

1. Create a folder structure for the package.

The package mypackage will contain the class MyClass.

```
mypackage/
   MyClass.java
```

Step 2: Define the MyClass inside the package

```
// mypackage/MyClass.java
package mypackage;

public class MyClass {
   public void displayMessage() {
      System.out.println("Hello from MyClass!");
   }
}
```

Step 3: Create another class outside the package to use MyClass

Create a Main.java file outside the mypackage directory:

```
// Main.java
import mypackage.MyClass;

public class Main {
   public static void main(String[] args) {
        // Create an instance of MyClass
        MyClass myObject = new MyClass();

        // Call the displayMessage method
        myObject.displayMessage();
    }
}
```

Step 4: Compile and run the code

1. **Compile the package and the Main class**:

Open a terminal, navigate to the folder where Main.java and mypackage are located, and run:

```
javac mypackage/MyClass.java Main.java
```

2. **Run the program**:

```
java Main
```

Output:

```
Hello from MyClass!
```

Problem statement 22: Write a JAVA program to display the use of multithreading.

Aim of this exercise is to demonstrate the concept of **multithreading** in Java, which allows multiple threads to run concurrently, improving the performance and responsiveness of programs, especially for tasks that can be executed independently.

O**bjectives** of this exercise are:

1. **Understand Multithreading**: Learn how Java supports multithreading by using the Thread class or implementing the Runnable interface.
2. **Thread Creation**: Practice creating multiple threads and running them concurrently.
3. **Thread Execution**: Demonstrate how multiple threads can execute independently, showcasing the parallel execution of tasks.
4. **Synchronization (optional)**: Optionally, touch on thread synchronization to handle situations where multiple threads access shared resources.
5. **Efficiency**: Understand how multithreading can improve the efficiency of a program by running tasks in parallel rather than sequentially.

Source Code:

```
// ThreadExample.java
class MyThread extends Thread {
    private String threadName;

    MyThread(String name) {
        threadName = name;
    }

    // The run method contains the code that will be executed by the thread
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(threadName + " is running: " + i);
            try {
                Thread.sleep(500); // Pause for 500 milliseconds
            } catch (InterruptedException e) {
                System.out.println(threadName + " interrupted.");
            }
        }
        System.out.println(threadName + " finished.");
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        // Create two threads
        MyThread thread1 = new MyThread("Thread 1");
        MyThread thread2 = new MyThread("Thread 2");

        // Start the threads
        thread1.start();
        thread2.start();
    }
}
```

Output:

```
Thread 1 is running: 1
Thread 2 is running: 1
Thread 1 is running: 2
Thread 2 is running: 2
Thread 1 is running: 3
Thread 2 is running: 3
Thread 1 is running: 4
Thread 2 is running: 4
Thread 1 is running: 5
Thread 2 is running: 5
Thread 1 finished.
Thread 2 finished.
```

Problem Statement 23: Write a JAVA program to demonstrate the use for exception handling.

Aim:

The aim of this program is to **demonstrate the concept of exception handling in Java**. Exception handling is a mechanism to handle runtime errors, ensuring the normal flow of the application is maintained.

Objectives:

1. **Understand Java Exception Hierarchy**: Learn about checked and unchecked exceptions, and how they are organized within Java.
2. **Implement try, catch, finally blocks**: Practice writing blocks of code that handle potential exceptions.
3. **Use throw and throws keywords**: Learn how to manually throw exceptions and declare exceptions that might occur.
4. **Handle multiple exceptions**: Demonstrate handling of different types of exceptions in a single program.
5. **Maintain program flow**: Understand how exception handling allows a program to continue running smoothly even when errors occur.
6. **Write clean and maintainable code**: Ensure that exception handling is applied where necessary, preventing crashes and improper terminations.

Source Code:

```
import java.util.Scanner;

public class ExceptionHandlingDemo {

    // Method to demonstrate exception handling
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            // Input from user
            System.out.print("Enter a number: ");
            int number = scanner.nextInt();

            // Arithmetic Exception (e.g., division by zero)
            int result = divideByNumber(number);
            System.out.println("Result of division by " + number + " is: " + result);

        } catch (ArithmeticException e) {
            // Catch block for handling division by zero
            System.out.println("Error: Division by zero is not allowed.");
        } catch (Exception e) {
            // Catch block for handling any other exception
            System.out.println("An error occurred: " + e.getMessage());

        } finally {
            // This block will always execute
            System.out.println("Program execution complete.");
            scanner.close(); // Closing scanner to avoid resource leaks
        }
    }

    // Method that throws an ArithmeticException
    public static int divideByNumber(int number) throws ArithmeticException {
        return 100 / number;
    }
}
```

Output:

```
Enter a number: 0
Error: Division by zero is not allowed.
Program execution complete.
```

Problem Statement 24: Write a JAVA program to read the contents of a text file and print them to the console.

Aim: The aim of this program is to **demonstrate how to read the contents of a text file in Java** and print them to the console. This helps in understanding file handling in Java, which is essential for reading from and writing to external files.

Objectives:

1. **Understand Java File I/O**: Learn how to use Java classes such as FileReader, BufferedReader, and File to handle file input operations.
2. **Practice Exception Handling in File I/O**: Handle potential exceptions like FileNotFoundException and IOException to ensure the program can deal with unexpected file issues gracefully.
3. **Read data from a text file**: Learn how to read data line by line or character by character from a text file.
4. **Print file contents to the console**: Display the contents of the file to the console, ensuring that the file was read successfully.
5. **Handle resource management**: Learn how to manage file resources effectively using try-with-resources or finally block to ensure file streams are properly closed after use.
6. **Write efficient and readable code**: Practice writing clean code for reading and processing files in Java.

Source code:

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

public class FileReadDemo {

    public static void main(String[] args) {
        String filePath = "example.txt"; // Replace with the path of your text file

        // Using try-with-resources to ensure the file is closed after reading
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;

            // Reading the file line by line
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            // Handling exceptions related to file reading
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }
}
```

Steps to run:

1. Create a text file named example.txt (or provide a different file path).
2. Make sure the file is accessible from the location specified in filePath.
3. Run the program, and it will print the contents of the file to the console.

Sample Output:

If example.txt contains:

```
Hello, world!
This is a test file.
```

The output will be:

```
Hello, world!
This is a test file.
```

Problem statement 25: Write a Java program to read from a file, write to a file, and copy content from one file to another.

Aim:

The aim of this program is to **demonstrate how to read from a file, write to a file, and copy content from one file to another** in Java. This covers key aspects of file handling, including both input and output operations, and shows how data can be transferred between files.

Objectives:

1. **Understand file input and output operations in Java**: Learn how to use classes such as FileReader, BufferedReader, FileWriter, and BufferedWriter to handle file reading and writing.
2. **Read data from a file**: Implement file reading functionality to access the contents of a file, either line by line or as a stream of data.
3. **Write data to a file**: Implement file writing functionality to write data to a text file.
4. **Copy content between files**: Demonstrate how to transfer data from one file to another, ensuring that the contents are read from a source file and written to a destination file.
5. **Implement exception handling for file operations**: Use exception handling (IOException, FileNotFoundException) to manage potential errors that might arise while performing file I/O operations.
6. **Practice efficient resource management**: Use techniques such as try-with-resources or explicit stream closing to handle file resources properly and avoid resource leaks.
7. **Write reusable and maintainable code**: Focus on structuring code that can easily handle various file operations, making it adaptable for future use in different file-related tasks.

Source Code:

```
import java.io.*;

public class FileOperationsDemo {

    public static void main(String[] args) {
        String sourceFilePath = "source.txt";  // Path to the source file
        String destinationFilePath = "destination.txt";  // Path to the destination file

        // Step 1: Read content from the source file
        System.out.println("Reading content from the source file:");
        readFile(sourceFilePath);

        // Step 2: Write content to a destination file
        String contentToWrite = "This is a sample content written to the destination file.";
        writeFile(destinationFilePath, contentToWrite);
        System.out.println("\nContent written to the destination file successfully.");

        // Step 3: Copy content from the source file to the destination file
        copyFileContent(sourceFilePath, destinationFilePath);
        System.out.println("\nContent copied from the source file to the destination file
successfully.");

        // Step 4: Display the content of the destination file after copying
        System.out.println("\nReading content from the destination file:");
        readFile(destinationFilePath);
    }

    // Method to read content from a file
    public static void readFile(String filePath) {
        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }

    // Method to write content to a file
    public static void writeFile(String filePath, String content) {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(filePath))) {
            bw.write(content);
        } catch (IOException e) {
            System.out.println("An error occurred while writing to the file: " +
e.getMessage());
        }
    }

    // Method to copy content from one file to another
    public static void copyFileContent(String sourceFilePath, String destinationFilePath) {
        try (BufferedReader br = new BufferedReader(new FileReader(sourceFilePath));
             BufferedWriter bw = new BufferedWriter(new FileWriter(destinationFilePath,
true))) { // Append mode

            String line;
            while ((line = br.readLine()) != null) {
                bw.write(line);
                bw.newLine(); // To ensure lines are properly formatted in the destination file
            }
        } catch (IOException e) {
            System.out.println("An error occurred while copying the file content: " +
e.getMessage());
        }
    }
}
```

Steps to Run:

1. Create two text files, source.txt (with some content) and destination.txt (it can be empty initially).
2. Run the program, and it will:
   - Read from the source.txt.
   - Write new content to destination.txt.
   - Copy the content of source.txt to destination.txt.
3. The final content of destination.txt will include both the written content and the copied content.

Sample Output:

```
Reading content from the source file:
Hello, this is the content of the source file.

Content written to the destination file successfully.

Content copied from the source file to the destination file successfully.

Reading content from the destination file:
This is a sample content written to the destination file.
Hello, this is the content of the source file.
```

Problem statement 26: Write a JAVA program to Use a FileReader to read characters from a file one at a time.

Aim:

The aim of this program is to **demonstrate how to use FileReader to read characters from a file one at a time** in Java. This helps in understanding how to handle character-based file input in a low-level manner.

Objectives:

1. **Understand FileReader Class**: Learn about the FileReader class in Java, which is used to read data from files as a stream of characters.
2. **Read Characters Individually**: Practice reading characters from a file one by one, which helps in understanding how character streams work.
3. **Handle File Not Found**: Implement error handling to manage scenarios where the file might not be found or accessible.
4. **Efficient Resource Management**: Use techniques to ensure that the file resources are properly closed after reading to prevent resource leaks.
5. **Basic File I/O Operations**: Gain a foundational understanding of file input operations and how to work with files at a character level in Java.
6. **Write Readable and Maintainable Code**: Structure the code in a way that demonstrates best practices for reading files character by character.

Source Code:

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

public class FileReaderDemo {

    public static void main(String[] args) {
        String filePath = "example.txt"; // Replace with the path to your text file

        try (FileReader fr = new FileReader(filePath)) {
            int character;
            // Read characters one at a time
            while ((character = fr.read()) != -1) {
                // Convert the integer value to a character and print it
                System.out.print((char) character);
            }
        } catch (IOException e) {
            // Handle file-related exceptions
            System.out.println("An error occurred while reading the file: " + e.getMessage());
        }
    }
}
```

Steps to Run:

1. Create a text file named example.txt (or use a different path).
2. Populate it with some content.
3. Run the program, and it will read and print the content of the file one character at a time.

Sample Output:

If example.txt contains:

```
Hello, world!
```

The output will be:

```
Hello, world!
```

Problem Statement 27: Write a JAVA program to Create a simple applet that displays a message and an image.

Aim:

The aim of this program is to **demonstrate how to create a simple Java applet** that displays a message and an image. This helps in understanding the basics of applet programming and how to render both text and images within an applet.

Objectives:

1. **Understand Java Applets**: Learn the basics of Java applets, including their lifecycle methods (init(), start(), paint(), etc.) and how they are integrated into a Java application.

2. **Create a Simple Applet**: Implement a basic applet that displays a static message on the screen.

3. **Load and Display an Image**: Use the Image class to load an image file and display it within the applet.

4. **Utilize Graphics Methods**: Utilize the Graphics class methods to draw text and images on the applet's display area.

5. **Handle Applet Lifecycle**: Understand how applet methods are called and how to properly initialize and manage applet components.

6. **Work with Applet HTML**: Learn how to embed the applet into an HTML file to view it in a web browser or applet viewer.

Source Code:

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.MediaTracker;

/*
<applet code="SimpleApplet" width=300 height=300>
</applet>
*/

public class SimpleApplet extends Applet {
    private Image image;

    @Override
    public void init() {
        // Load the image
        image = getImage(getCodeBase(), "example.jpg"); // Ensure "example.jpg" is in the
same directory
    }
    @Override
    public void paint(Graphics g) {
        // Display the message
        g.drawString("Hello, this is a simple applet!", 20, 20);

        // Display the image
        if (image != null) {
            // Wait until the image is fully loaded
            MediaTracker tracker = new MediaTracker(this);
            tracker.addImage(image, 0);
            try {
                tracker.waitForAll();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // Draw the image at coordinates (20, 40)
            g.drawImage(image, 20, 40, this);
        }
    }
}
```

□ **Prepare the Image:**

- Place an image named example.jpg in the same directory as your SimpleApplet.java file.

□ **Compile the Applet Code:**

- Open a terminal or command prompt.
- Navigate to the directory containing your SimpleApplet.java file.
- Compile the applet using the javac command:

```
javac SimpleApplet.java
```

Run the Applet Using appletviewer:

- In the terminal or command prompt, navigate to the directory containing your applet.html file.
- Run the applet using the appletviewer command:

```
appletviewer applet.html
```

Problem statement 28: Write a JAVA program to Use the drawLine() method of the Graphics object to draw lines.

Aim:

The aim of this program is to demonstrate how to use the drawLine() method of the Graphics class in Java to draw lines on a graphical interface. This helps in understanding basic drawing operations within Java's graphics framework.

Objectives:

1. **Understand the Graphics Class:** Learn how the Graphics class in Java provides methods for drawing on components and canvases.
2. **Use the drawLine() Method:** Implement the drawLine() method to draw straight lines between specified points.
3. **Handle Basic Drawing Operations:** Practice basic graphical operations such as setting colors and drawing multiple lines.
4. Understand Coordinate Systems: Learn how to work with the coordinate system used in Java's graphics environment.
5. **Implement a GUI Component:** Integrate the drawing functionality into a simple Java GUI component, such as a JPanel or Applet.
6. **Develop Simple Graphics Applications:** Use this method to create simple graphics applications and understand how to handle graphical rendering in Java.

Source Code:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;

public class LineDrawingDemo extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Draw a line from (10, 10) to (100, 100)
        g.drawLine(10, 10, 100, 100);

        // Draw a line from (100, 100) to (200, 50)
        g.drawLine(100, 100, 200, 50);

        // Draw a line from (200, 50) to (10, 10)
        g.drawLine(200, 50, 10, 10);
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Line Drawing Demo");
        LineDrawingDemo panel = new LineDrawingDemo();

        frame.add(panel);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Steps to Run:

1. **Create the Java File**: Save the code in a file named LineDrawingDemo.java.
2. **Compile the Code**: Open a terminal or command prompt, navigate to the directory containing the file, and compile it using:

```
javac LineDrawingDemo.java
```

Run the Program: Run the compiled class file using:

```
java LineDrawingDemo
```

Problem statement 29: Write a JAVA program to Use the drawOval() method of the Graphics object to draw lines.

Aim:
The aim of this program is to demonstrate how to use the drawOval() method of the Graphics class in Java to draw ellipses and circles. This helps in understanding how to create circular and elliptical shapes within a graphical interface.

Objectives:

1. Understand the Graphics Class: Learn about the Graphics class and how it provides methods for drawing shapes on components and canvases.
2. Use the drawOval() Method: Implement the drawOval() method to draw ellipses and circles by specifying their bounding rectangle.
3. Explore Basic Drawing Operations: Practice drawing shapes and understand how to set up the dimensions and positioning for accurate rendering.
4. Understand Coordinate Systems: Learn about how coordinates work in Java's graphical environment, particularly for drawing shapes.
5. Integrate Drawing into a GUI Component: Embed the drawing functionality into a Java GUI component, such as a JPanel or Applet.
6. Create Simple Graphics Applications: Develop a basic graphics application to illustrate how to use drawing methods to create visual elements.

Source Code:

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Graphics;

public class OvalDrawingDemo extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Draw a circle with a radius of 50 pixels
        g.drawOval(50, 50, 100, 100); // (x, y, width, height)

        // Draw an ellipse with specified width and height
        g.drawOval(200, 50, 150, 75); // (x, y, width, height)

        // Draw a larger ellipse
        g.drawOval(100, 150, 200, 100); // (x, y, width, height)
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Oval Drawing Demo");
        OvalDrawingDemo panel = new OvalDrawingDemo();

        frame.add(panel);
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Steps to Run:

1. **Create the Java File**: Save the code in a file named OvalDrawingDemo.java.

2. **Compile the Code**: Open a terminal or command prompt, navigate to the directory containing the file, and compile it using:

javac OvalDrawingDemo.java

Run the Program: Run the compiled class file using:

java OvalDrawingDemo

Problem Statement 30: Write a Demo program in JAVA on Event handling.

Aim:

The aim of this program is to demonstrate basic event handling in Java by creating a simple application that responds to user actions such as button clicks, mouse movements, or keyboard inputs. This helps in understanding how to manage and respond to user-generated events in a Java application.

Objectives:

1. Understand Event Handling: Learn the concept of event handling in Java, including how to use event listeners to respond to user actions.
2. Implement Event Listeners: Create event listeners for various types of events, such as button clicks, mouse movements, or key presses.
3. Use Swing Components: Utilize Swing components (e.g., JButton, JFrame) to create a graphical user interface for the demo application.
4. Manage Event Sources and Listeners: Understand how to register event listeners with event sources and handle different types of events.
5. Create a Responsive GUI: Develop a simple GUI application that responds to user interactions, demonstrating practical event handling techniques.

Source Code:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EventHandlingDemo {

    public static void main(String[] args) {
        // Create a JFrame instance
        JFrame frame = new JFrame("Event Handling Demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 200);

        // Create a JPanel to hold components
        JPanel panel = new JPanel();

        // Create a JButton with a label
        JButton button = new JButton("Click Me");

        // Add ActionListener to the button
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Action to be performed when the button is clicked
                System.out.println("Button was clicked!");
            }
        });

        // Add the button to the panel
        panel.add(button);

        // Add the panel to the frame
        frame.add(panel);

        // Make the frame visible
        frame.setVisible(true);
    }
}
```

Steps to Run:

1. **Create the Java File**: Save the code in a file named EventHandlingDemo.java.
2. **Compile the Code**: Open a terminal or command prompt, navigate to the directory containing the file, and compile it using:

```
javac EventHandlingDemo.java
```

Run the Program: Run the compiled class file using:

```
java EventHandlingDemo
```