KGiSL MicroCollege
Genius as a Service (GaaS)
TECHNICAL TEAM

KGCAS
KG COLLEGE OF ARTS AND SCIENCE
co-Kreate your Genius
Autonomous Institution - Affiliated to Bharathiar University

LABORATORY MANUAL

JAVA PROGRAMMING

B.Sc. CS / BCA / B.Sc. CT / B.Sc. IT / B.Sc. AI & DS

2024 BATCH
(Autonomous)

KGCAS
KG COLLEGE OF ARTS AND SCIENCE
co-Kreate your Genius
Autonomous Institution - Affiliated to Bharathiar University

COURSE OBJECTIVES

1. Develop a deep understanding of Java programming basics, including syntax, control structures, and data types.
2. Learn the use of arrays, strings, and file handling for problem-solving.
3. Foster algorithmic thinking to solve mathematical, logical, and real-world problems like prime numbers, matrix multiplication, and string operations.
4. Understand and implement Java's multithreading features for concurrent programming.
5. Perform operations using different string-handling classes, such as String, StringBuffer, and character arrays.
6. Work with file operations to perform data analysis, retrieve file properties, and implement user input/output.
7. Design graphical user interfaces (GUIs) using Java Swing for applications like calculators, traffic light simulations, and mouse event handlers.
8. Learn Java's event-handling model to manage user interactions like mouse clicks, key presses, and GUI controls.

COURSE OUTCOMES

1. Develop and test Java programs for basic operations, control structures, and computational problems.
2. Solve mathematical problems like prime number detection and matrix multiplication using efficient algorithms.
3. Use String, StringBuffer, and character arrays to perform operations like concatenation, substring extraction, reversal, and searching.
4. Design and implement multithreaded programs to solve concurrent tasks, including random number processing and asynchronous method execution.
5. Write robust Java programs that use exception handling for common runtime errors like ArithmeticException, NumberFormatException, and ArrayIndexOutOfBoundsException.
6. Perform file operations to read, write, and retrieve file properties such as size, readability, and writability.

7. Create interactive applications using Java Swing, such as calculators and traffic light simulations, employing effective layouts and event handling.
8. Use listener interfaces and adapter classes to handle events like mouse clicks and user actions effectively.

Mapping Objectives to Outcomes

Objective	Outcome
Strengthen Core Java Fundamentals	**CO1**
Enhance Problem-Solving Skills	**CO2**
Develop Multithreading and Exception Handling Skills	**CO4, CO5**
Learn String Manipulation Techniques	**CO3**
Master File Handling and I/O Operations	**CO6**
Introduce GUI Programming	**CO7**
Understand Event Handling	**CO8**

SYLLABUS

S. NO.	LIST OF PROGRAMS
1	Basic Java programs.
2	Java program that prompts the user for an integer and then prints out all the prime numbers up to that Integer.
3	Java program to multiply two given matrices.
4	Java program that displays the number of characters, lines, and words in a text.
5	Generate random numbers between two given limits using Random class and print messages according to the value range generated.
6	Java program to do String Manipulation using Character Array and perform the following string operations: a) String length b) Finding a character at a particular position c) Concatenating two strings.
7	Java program to perform the following string operations using the String class: a) String Concatenation b) Search a substring c) To extract a substring from the given.
8	Java program to perform string operations using the String Buffer class: a) Length of a string b) Reverse a string c) Delete a substring from the given string.
9	Java program that implements a multi-thread application that has three threads. The first thread generates a random integer every 1 second and if the value is even, the second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of the cube of the number.
10	Threading program that uses the same method asynchronously to print the numbers 1 to 10 using Thread1 and to print 90 to 100 using Thread2.
11	Java program will demonstrate the use of the following exceptions: a) ArithmeticException b) NumberFormatException c) ArrayIndexOutofBoundException d) NegativeArraySizeException.

12	Java program reads the file name from the user, then displays information about whether the file exists, whether the file is readable, whether the file is writable, the type of file, and the length of the file in bytes.
13	Java program to accept a text and change its size and font. Include bold italic options. Use frames and controls.
14	Java program that handles all mouse events and shows the event name at the center of the window when a mouse event is fired. (Use adapter classes).
15	Java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the +, -, *, and % operations. Add a text field to display the result. Handle any possible exceptions like divide by zero.
16	Java program that simulates a traffic light. The program lets the user select one of three lights: red, yellow, or green with radio buttons. On selecting a button, an appropriate message with “stop” “ready” or “go” should appear above the buttons in a selected color. Initially, there is no message shown.

LIST OF EXPERIMENTS

PREREQUISITES

General Knowledge Prerequisites

a. Core Java Concepts

- **Basic Syntax**: Familiarity with main() method, data types, variables, operators, and control structures (if, for, while).
- **Object-Oriented Programming (OOP)**: Understanding classes, objects, constructors, and basic inheritance.
- **Exception Handling**: Knowledge of try-catch-finally blocks and built-in Java exceptions.
- **Multithreading**: Basics of threads, Thread class, and Runnable interface.
- **String Handling**: Familiarity with String, StringBuffer, and StringBuilder classes.

b. File I/O in Java

- Understanding file handling using File, FileReader, and FileWriter.
- Basic file operations like checking file properties and reading/writing files.

c. Event Handling

- Knowledge of event-driven programming in Java.
- Familiarity with listener interfaces like MouseListener and adapter classes.

d. GUI Development

- Basics of **Swing** framework for creating graphical user interfaces.
- Understanding of Java layouts (FlowLayout, GridLayout) and components (JButton, JTextField, JLabel, etc.).

Tools and Software Setup

a. Java Development Environment

- **JDK**:
  - Install the latest JDK (Java SE 17 or later).
  - Set up environment variables (JAVA_HOME and PATH).
  - Verify installation:

```
java -version
javac -version
```

b. Integrated Development Environment (IDE)

- Recommended: **IntelliJ IDEA**, **Eclipse IDE**, or **NetBeans**.
- Lightweight alternative: **VS Code** with Java extensions.

c. Text Editor

- For quick programs, use simple editors like **Notepad++** or **Visual Studio Code**.

Program-Specific Prerequisites

1. Basic Java Programs

- **Required Knowledge**: Understanding basic syntax, System.out.println(), variables, and data types.
- **Setup**: A simple text editor or IDE is sufficient.

2. Prime Numbers

- **Required Knowledge**:
  - Loops and conditional statements.
  - Mathematical logic for prime number detection.

3. Matrix Multiplication

- **Required Knowledge**:
  - Arrays in Java (1D and 2D).
  - Nested loops for matrix operations.

4. File Analysis (Characters, Lines, Words)

- **Required Knowledge**:
  - File I/O using FileReader or Scanner.
  - String manipulation for counting words and lines.
- **Tools**: Basic text file for input.

5. Random Numbers

- **Required Knowledge**:
  - Random class for generating random numbers.
  - Conditional statements to handle ranges.

6-8. String Manipulations

- **Required Knowledge**:
  - Familiarity with String, StringBuffer, and character arrays.
  - Operations like concatenation, reversing, and substring handling.

9-10. Multithreading

- **Required Knowledge**:
  - Basics of the Thread class and Runnable interface.
  - Using Thread.sleep() for delays.
  - Synchronization to manage shared resources (if required).

11. Exception Handling

- **Required Knowledge**:
  - Understanding built-in exceptions like ArithmeticException, NumberFormatException, etc.
  - Using try-catch blocks for handling errors.

12. File Information

- **Required Knowledge**:
  - File class for checking file properties (readable, writable, size, etc.).
- **Tools**: Sample file for testing.

13. Text Styling (Swing GUI)

- **Required Knowledge**:
  - Java Swing components: JFrame, JTextField, and JButton.
  - Font handling using Font class.

14. Mouse Events

- **Required Knowledge**:
  - Event handling using MouseListener and adapter classes.
  - Displaying events using JLabel.

15. Calculator Application

- **Required Knowledge**:
  - Swing layout managers (e.g., GridLayout).
  - Event handling for button clicks.

16. Traffic Light Simulation

- **Required Knowledge**:
  - Swing components like JRadioButton and JPanel.
  - Conditional logic for handling radio button selections.
  - Color manipulation using Color class.

EXPERIMENT - 1

AIM:

To write a Java program that swaps the values of two numbers without using a third variable, using the Scanner class for input.

ALGORITHM:

Step 1: Start the process.

Step 2: Open the Eclipse IDE.

Step 3: Declare two integer variables a and b and use the Scanner class to assign values to a and b from user input.

Step 4: Print the original values of a and b.

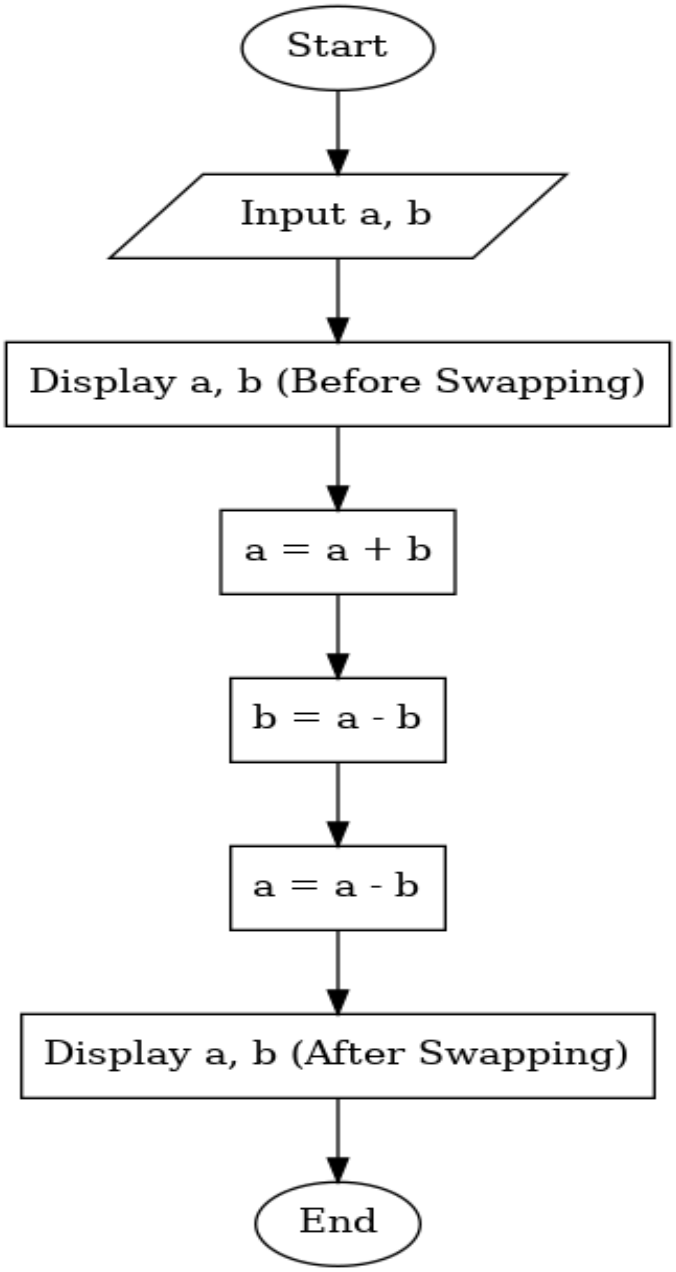
Step 5: Swap the values of a and b without using a third variable:

- Set a = a + b, then set b = a - b, and finally set a = a - b.

Step 6: Print the swapped values of a and b.

Step 7: End the process.

FLOW CHART:



SOURCE CODE:

```
import java.util.Scanner;
public class SwapNumbers {
        public static void main(String[] args) {
        int a, b;
```

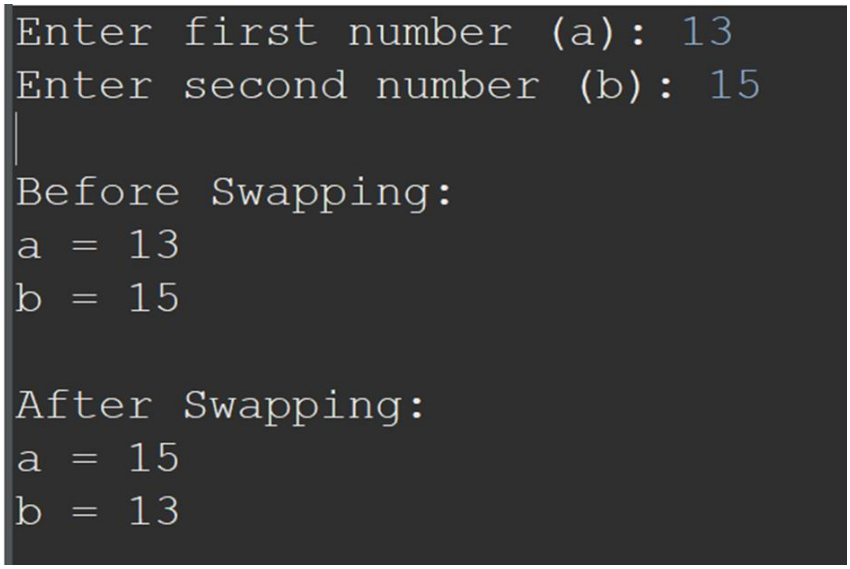
```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter first number (a): ");
        a = scanner.nextInt();
        System.out.print("Enter second number (b): ");
        b = scanner.nextInt();
        System.out.println("\nBefore Swapping:");
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        a = a + b;
        b = a - b;
        a = a - b;
        System.out.println("\nAfter Swapping:");
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        scanner.close();
        }
}
```

CODE EXPLANATION:

1. **import java.util.Scanner;**
   - Imports the Scanner class, which is used to read user input from the console.
2. **public class SwapNumbers {**
   - Defines the class SwapNumbers, which contains the program logic.
3. **public static void main(String[] args) {**
   - The entry point of the program. This method is executed when the program runs.
4. **int a, b;**
   - Declares two integer variables a and b to store the numbers.
5. **Scanner scanner = new Scanner(System.in);**
   - Creates a Scanner object to read input from the console.
6. **System.out.print("Enter first number (a): ");**
   - Prompts the user to enter the first number.

7. **a = scanner.nextInt();**
    - Reads an integer from the user and assigns it to the variable a.
8. **System.out.print("Enter second number (b): ");**
    - Prompts the user to enter the second number.
9. **b = scanner.nextInt();**
    - Reads another integer from the user and assigns it to the variable b.
10. **System.out.println("\nBefore Swapping:");**
    - Prints a message indicating the start of the "before swapping" section.
11. **System.out.println("a = " + a);**
    - Displays the value of a before swapping.
12. **System.out.println("b = " + b);**
    - Displays the value of b before swapping.
13. **a = a + b;**
    - Adds the values of a and b and stores the result in a. At this point, a holds the sum of the two numbers.
14. **b = a - b;**
    - Subtracts b (the original value) from a (the sum of a and b). This operation effectively assigns the original value of a to b.
15. **a = a - b;**
    - Subtracts the new b (which now holds the original value of a) from a (the sum of the original a and b). This operation assigns the original value of b to a.
16. **System.out.println("\nAfter Swapping:");**
    - Prints a message indicating the start of the "after swapping" section.
17. **System.out.println("a = " + a);**
    - Displays the value of a after swapping.
18. **System.out.println("b = " + b);**
    - Displays the value of b after swapping.
19. **scanner.close();**
    - Closes the Scanner object to release system resources. It is good practice to close the scanner after use.

OUTPUT:



RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 2

AIM:

Write a Java program that prompts the user for an integer and displays all prime numbers up to that integer.

ALGORITHM:

Step 1: Start the process.

Step 2: Open the Eclipse IDE.

Step 3: Create a Scanner object to read input from the user.

Step 4: Prompt the user to enter an integer.

Step 5: Store the input integer in a variable n.

Step 6: Print a message indicating the prime numbers up to n.

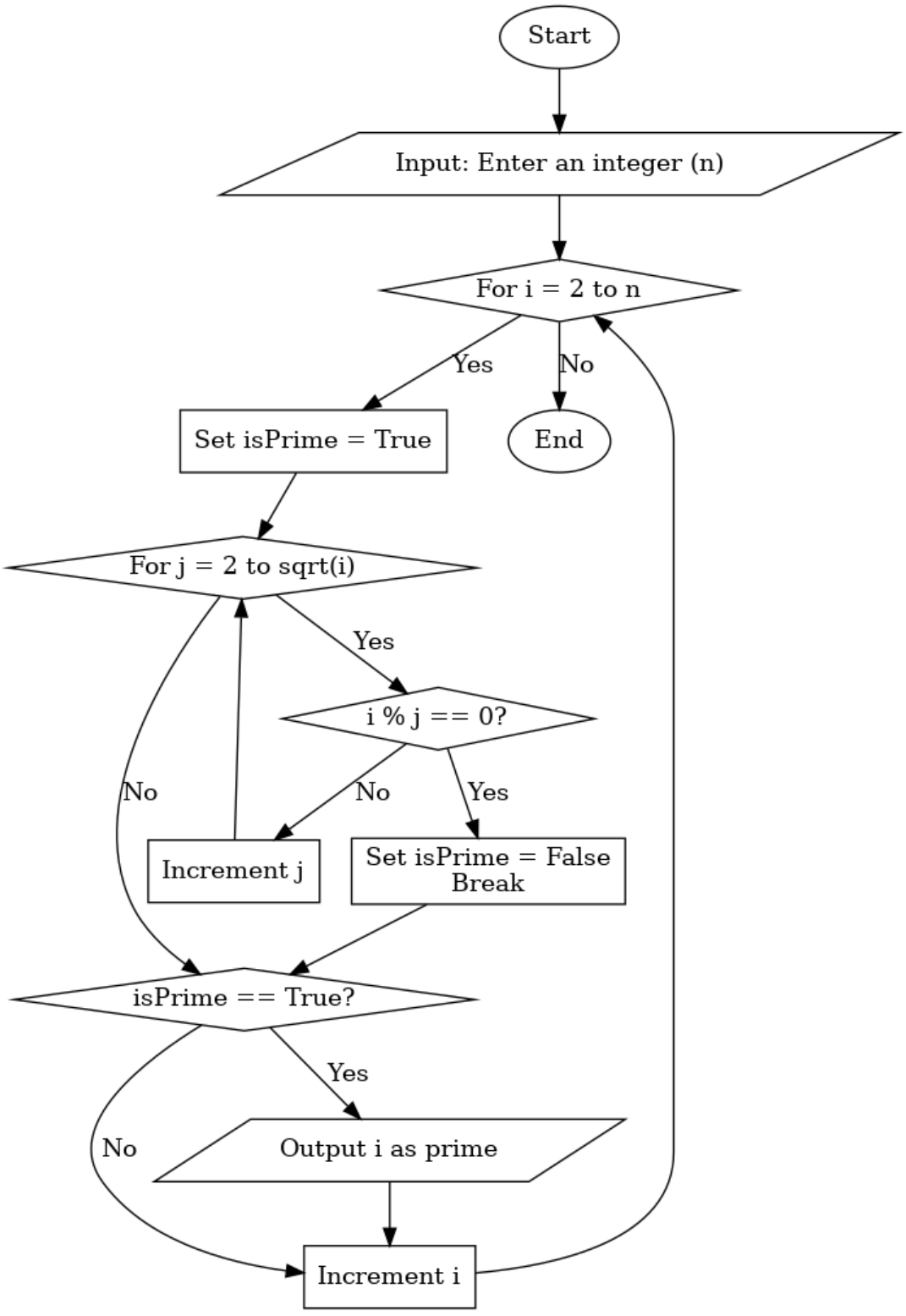
Step 7: For each number i from 2 to n:

- Now Initialize a flag variable isPrime as true. For each number j from 2 to the square root of I, If i is divisible by j, set isPrime to false and break the loop.

Step 8: If isPrime is still true, print i as a prime number.

Step 9: End the process.

FLOW CHART:


Start
Input: Enter an integer (n)
For i = 2 to n
Yes
No
Set isPrime = True
End
For j = 2 to sqrt(i)
Yes
i % j == 0?
No
No
Yes
Increment j
Set isPrime = False
Break
isPrime == True?
Yes
No
Output i as prime
Increment i

SOURCE CODE:

```
public class PrimeNumbers {
        public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int n = scanner.nextInt();
        System.out.println("Prime numbers up to " + n + " are:");
        for (int i = 2; i <= n; i++) {
        boolean isPrime = true;
        for (int j = 2; j * j <= i; j++) {
                if (i % j == 0) {
                isPrime = false; // i is not prime
                break; // exit the inner loop
                }
        }
        if (isPrime) {
                System.out.print(i + " "); number
        }
        }
        scanner.close();
        }
}
```

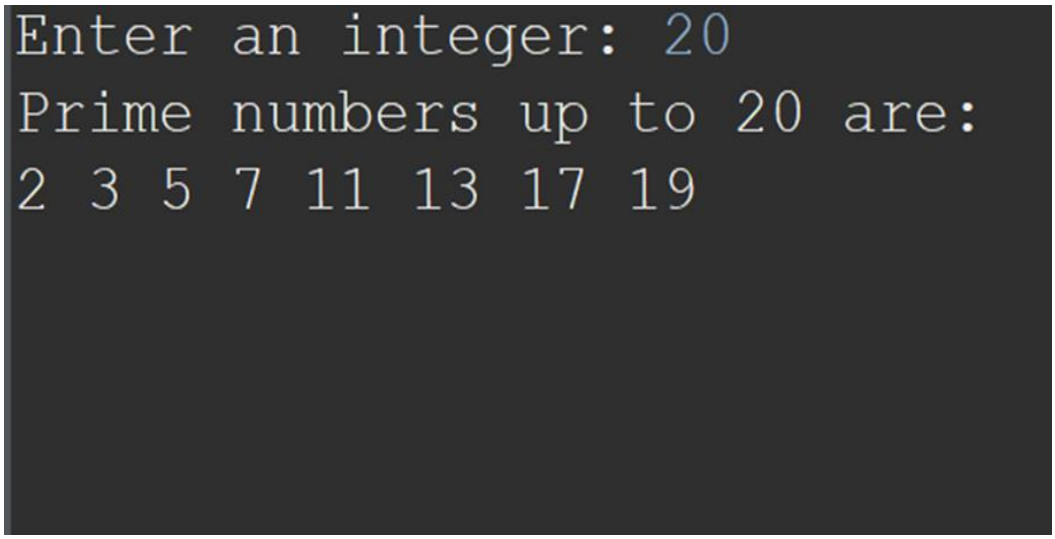
CODE EXPLANATION:

1. **public class PrimeNumbers {**
   - Declares a class named PrimeNumbers, which contains the logic to find prime numbers.
2. **public static void main(String[] args) {**
   - The entry point of the program. This method is executed when the program runs.
3. **Scanner scanner = new Scanner(System.in);**
   - Creates a Scanner object to read input from the user.

4. **System.out.print("Enter an integer: ");**
   - Prompts the user to enter an integer value.
5. **int n = scanner.nextInt();**
   - Reads the integer entered by the user and stores it in the variable n.
6. **System.out.println("Prime numbers up to " + n + " are:");**
   - Prints a message indicating the program will display prime numbers up to n.
7. **for (int i = 2; i <= n; i++) {**
   - Starts a loop to check all numbers from 2 to n. The variable i represents the current number being checked.
8. **boolean isPrime = true;**
   - Assumes initially that the current number i is prime. This variable will be updated if the number is found to be non-prime.
9. **for (int j = 2; j * j <= i; j++) {**
   - Starts a loop to check divisors for the current number i.
   - j starts at 2 and runs up to the square root of i (j * j <= i) to optimize the check.
10. **if (i % j == 0) {**
    - Checks if i is divisible by j. If true, i is not a prime number.
11. **isPrime = false;**
    - Sets isPrime to false, indicating that i is not a prime number.
12. **break;**
    - Exits the inner loop early since i is confirmed not to be prime.
13. **}**
    - Ends the inner for loop.
14. **if (isPrime) {**
    - Checks if isPrime is still true after the inner loop. If so, i is a prime number.
15. **System.out.print(i + " ");**
    - Prints the current prime number i, followed by a space.
16. **}**
    - Ends the outer for loop.
17. **scanner.close();**
    - Closes the Scanner object to release system resources.

18. }
    - Ends the main method.
19. }
    - Ends the PrimeNumbers class.

OUTPUT:



RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 3

AIM:

Write a Java program that multiplies two matrices, taking input from the user, and then calculates and prints the product of the matrices.

ALGORITHM:

Step 1: Start the process.

Step 2: Open the Eclipse IDE.

Step 3: Input the number of rows and columns for both matrices (matrix A and matrix B) from the user.

Step 4: Check if matrix multiplication is possible:

- Ensure that the number of columns of matrix A is equal to the number of rows of matrix B.
- If the condition is not satisfied, print an error message and terminate the process.

Step 5: Input the elements of matrix A from the user.

Step 6: Input the elements of matrix B from the user.

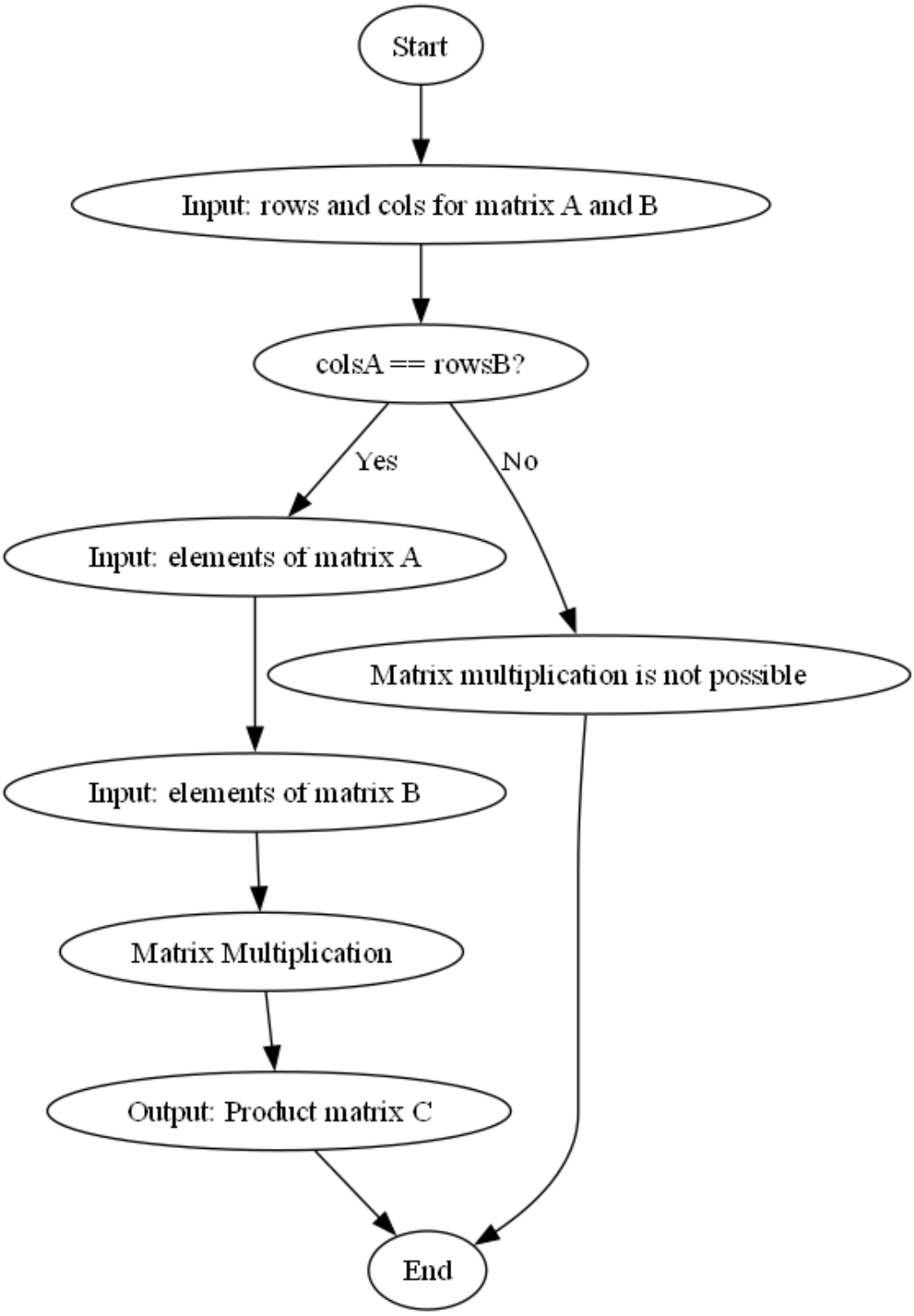
Step 7: Perform matrix multiplication:

- Multiply each row element of matrix A by each column element of matrix B.
- For each element of the result matrix, calculate the sum of the products.

Step 8: Store the result of the matrix multiplication in a new matrix, and print the resultant matrix.

Step 9: End the process.

FLOW CHART:


Start
Input: rows and cols for matrix A and B
colsA == rowsB?
Yes
No
Input: elements of matrix A
Matrix multiplication is not possible
Input: elements of matrix B
Matrix Multiplication
Output: Product matrix C
End

SOURCE CODE:

```
import java.util.Scanner;
public class MatrixMultiplication {
        public static void main(String[] args) {
        Scanner myobj = new Scanner(System.in);
        System.out.print("Enter the number of rows for matrix A: ");
        int rowsA = myobj.nextInt();
        System.out.print("Enter the number of columns for matrix A: ");
        int colsA = myobj.nextInt();
        System.out.print("Enter the number of rows for matrix B: ");
        int rowsB = myobj.nextInt();
        System.out.print("Enter the number of columns for matrix B: ");
        int colsB = myobj.nextInt();
        if (colsA != rowsB) {
        System.out.println("Matrix multiplication is not possible.");
        return;
        }
        int[][] a = new int[rowsA][colsA];
        int[][] b = new int[rowsB][colsB];
        int[][] c = new int[rowsA][colsB];
        System.out.println("Enter elements of matrix A:");
        for (int i = 0; i < rowsA; i++)
        {
        for (int j = 0; j < colsA; j++)
        {
                a[i][j] = myobj.nextInt();
        }
        }
        System.out.println("Enter elements of matrix B:");
        for (int i = 0; i < rowsB; i++)
        {
```

```
        for (int j = 0; j < colsB; j++)
        {
                b[i][j] = myobj.nextInt();
        }
        }
        for (int i = 0; i < rowsA; i++)
        {
        for (int j = 0; j < colsB; j++)
        {
                for (int k = 0; k < colsA; k++)
                {
                c[i][j] += a[i][k] * b[k][j];
                }
        }
        }
        System.out.println("The product of the matrices is:");
        for (int i = 0; i < rowsA; i++)
        {
        for (int j = 0; j < colsB; j++)
        {
                System.out.print(c[i][j] + " ");
        }
        System.out.println();
        }
        myobj.close();
        }
}
```

CODE EXPLANATION:

1. **import java.util.Scanner;**
   - Imports the Scanner class for user input.
2. **public class MatrixMultiplication {**
   - Declares a class named MatrixMultiplication.
3. **public static void main(String[] args) {**
   - The main method where the program execution begins.
4. **Scanner myobj = new Scanner(System.in);**
   - Creates a Scanner object to read input from the user.
5. **System.out.print("Enter the number of rows for matrix A: ");**
   - Prompts the user to enter the number of rows for matrix A.
6. **int rowsA = myobj.nextInt();**
   - Reads the number of rows for matrix A and stores it in rowsA.
7. **System.out.print("Enter the number of columns for matrix A: ");**
   - Prompts the user to enter the number of columns for matrix A.
8. **int colsA = myobj.nextInt();**
   - Reads the number of columns for matrix A and stores it in colsA.
9. **System.out.print("Enter the number of rows for matrix B: ");**
   - Prompts the user to enter the number of rows for matrix B.
10. **int rowsB = myobj.nextInt();**
    - Reads the number of rows for matrix B and stores it in rowsB.
11. **System.out.print("Enter the number of columns for matrix B: ");**
    - Prompts the user to enter the number of columns for matrix B.
12. **int colsB = myobj.nextInt();**
    - Reads the number of columns for matrix B and stores it in colsB.
13. **if (colsA != rowsB) {**
    - Checks if matrix multiplication is possible. The number of columns in matrix A must equal the number of rows in matrix B.
14. **System.out.println("Matrix multiplication is not possible.");**
    - Prints a message if matrix multiplication is not possible.
15. **return;**

- Exits the program if multiplication cannot be performed.

16. **int[][] a = new int[rowsA][colsA];**
    - Declares a 2D array to store matrix A.
17. **int[][] b = new int[rowsB][colsB];**
    - Declares a 2D array to store matrix B.
18. **int[][] c = new int[rowsA][colsB];**
    - Declares a 2D array to store the result of matrix multiplication.
19. **System.out.println("Enter elements of matrix A:");**
    - Prompts the user to enter the elements of matrix A.
20. **for (int i = 0; i < rowsA; i++) {**
    - Loops through each row of matrix A.
21. **for (int j = 0; j < colsA; j++) {**
    - Loops through each column of matrix A.
22. **a[i][j] = myobj.nextInt();**
    - Reads the value for matrix A at position [i][j].
23. **System.out.println("Enter elements of matrix B:");**
    - Prompts the user to enter the elements of matrix B.
24. **for (int i = 0; i < rowsB; i++) {**
    - Loops through each row of matrix B.
25. **for (int j = 0; j < colsB; j++) {**
    - Loops through each column of matrix B.
26. **b[i][j] = myobj.nextInt();**
    - Reads the value for matrix B at position [i][j].
27. **for (int i = 0; i < rowsA; i++) {**
    - Loops through each row of the result matrix.
28. **for (int j = 0; j < colsB; j++) {**
    - Loops through each column of the result matrix.
29. **for (int k = 0; k < colsA; k++) {**
    - Loops through the elements of the row of matrix A and column of matrix B to calculate the product.
30. **c[i][j] += a[i][k] * b[k][j];**

- Multiplies the corresponding elements and adds to the result matrix at [i][j].

31. **System.out.println("The product of the matrices is:");**
    - Prints the result matrix.
32. **for (int i = 0; i < rowsA; i++) {**
    - Loops through each row of the result matrix.
33. **for (int j = 0; j < colsB; j++) {**
    - Loops through each column of the result matrix.
34. **System.out.print(c[i][j] + " ");**
    - Prints the value of the result matrix at position [i][j].
35. **System.out.println();**
    - Moves to the next line after printing one row of the result matrix.
36. **myobj.close();**
    - Closes the Scanner to release resources.
37. **}**
    - Closes the main method.
38. **}**
    - Closes the MatrixMultiplication class.

OUTPUT:

```
Enter the number of rows for matrix A: 3
Enter the number of columns for matrix A: 3
Enter the number of rows for matrix B: 3
Enter the number of columns for matrix B: 3
Enter elements of matrix A:
9 8 7
6 5 4
3 2 1
Enter elements of matrix B:
1 2 3
4 5 6
7 8 9
The product of the matrices is:
90 114 138
54 69 84
18 24 30
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 4

AIM:

Write a Java program that reads a text file and displays the number of characters, lines, and words in the file.

ALGORITHM:

Step 1: Start the process.

Step 2: Create a Java class and import necessary classes (BufferedReader, FileReader, and IOException).

Step 3: Define the file path of the text file to be read.

Step 4: Use a BufferedReader to read the file line by line.

Step 5: Initialize counters for characters, lines, and words.

Step 6: Read each line from the file.

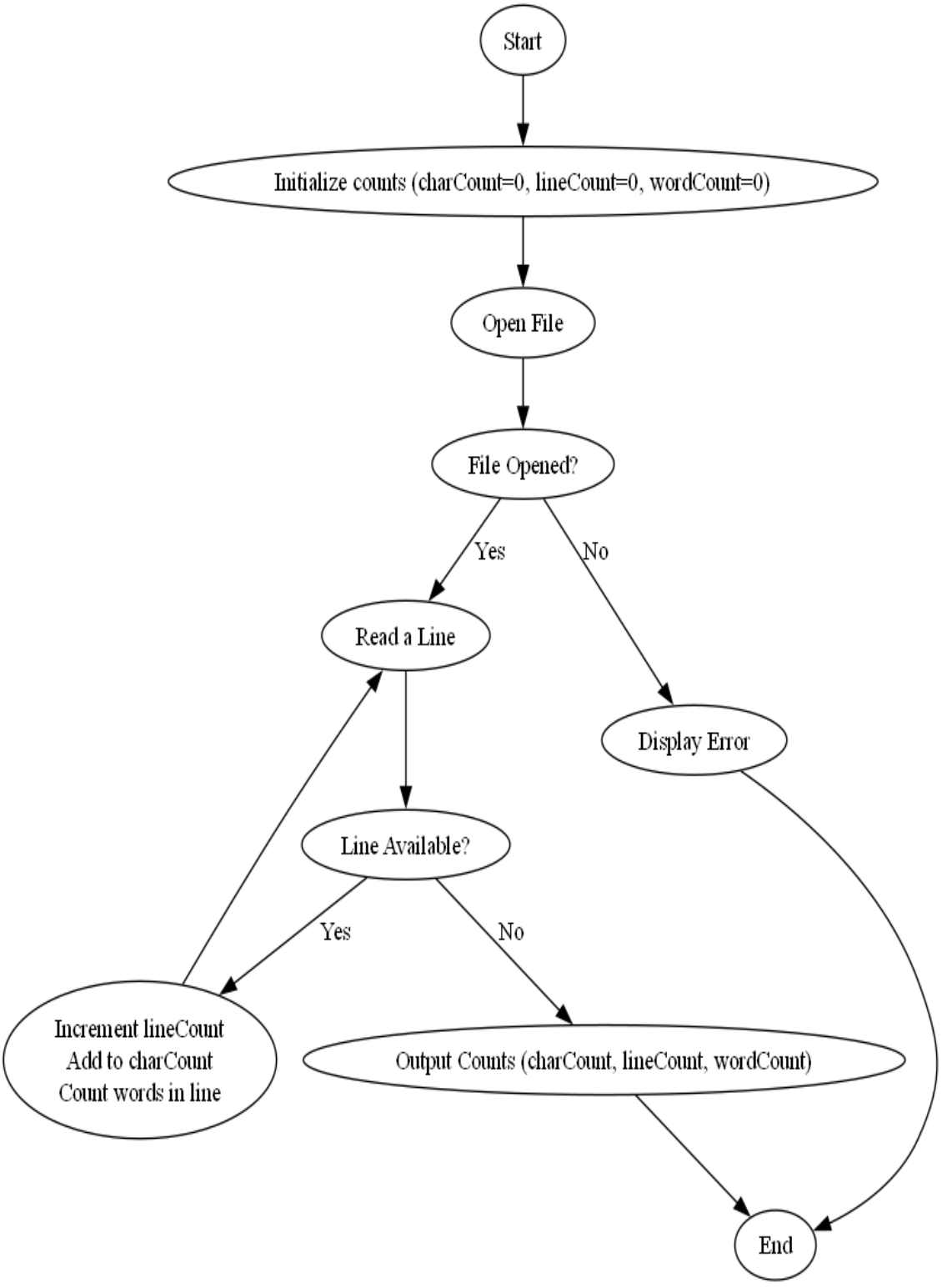
- Increment the line count for each line read.
- Increment the character count by the length of the line.
- Split the line into words and increment the word count based on the number of words.

Step 7: After reading the entire file, print the total number of characters, lines, and words.

Step 8: Handle any exceptions using a try-catch block for IOException.

Step 9: End the process.

FLOW CHART:


Start
Initialize counts (charCount=0, lineCount=0, wordCount=0)
Open File
File Opened?
Yes
No
Read a Line
Display Error
Line Available?
Yes
No
Increment lineCount
Add to charCount
Count words in line
Output Counts (charCount, lineCount, wordCount)
End

SOURCE CODE:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class TextStatistics {
    public static void main(String[] args) {
        String filePath = "path/to/your/textfile.txt"; // Change to your file path
         try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
            String line;
            int characterCount = 0;
            int lineCount = 0;
            int wordCount = 0;
            while ((line = reader.readLine()) != null) {
                lineCount++;
                characterCount += line.length();
                wordCount += line.split("\\s+").length; // Split by whitespace
            }
            System.out.println("Characters: " + characterCount);
            System.out.println("Lines: " + lineCount);
            System.out.println("Words: " + wordCount);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CODE EXPLANATION:

import java.io.BufferedReader;

- **Purpose**: Imports the BufferedReader class to read text from a file efficiently, line by line.

import java.io.FileReader;

- **Purpose**: Imports the FileReader class, which is used to read the contents of a file.

import java.io.IOException;

- **Purpose**: Imports the IOException class, which is needed to handle input-output-related exceptions (e.g., file not found or read errors).

public class TextStatistics {

- **Purpose**: Declares a class named TextStatistics. This is the program's main class.

public static void main(String[] args) {

- **Purpose**: The main method is the entry point of the program where execution begins.

String filePath = "path/to/your/textfile.txt";

- **Purpose**: Defines a string variable filePath that stores the path to the text file to be analyzed. You need to replace "path/to/your/textfile.txt" with the actual file path.

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

- **Purpose**:
    1. Creates a BufferedReader object reader to read the file.
    2. The new FileReader(filePath) reads the file at the specified path.
    3. The try-with-resources ensures the BufferedReader is automatically closed after the operations complete, preventing resource leaks.

String line;

- **Purpose**: Declares a variable line to store each line of text read from the file.

int characterCount = 0;

- **Purpose**: Initializes a variable characterCount to 0. It will store the total number of characters in the file.

int lineCount = 0;

- **Purpose**: Initializes a variable lineCount to 0. It will store the total number of lines in the file.

int wordCount = 0;

- **Purpose**: Initializes a variable wordCount to 0. It will store the total number of words in the file.

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

- **Purpose**: Reads each line from the file until there are no more lines (null indicates the end of the file).
- **Explanation**:
  - reader.readLine() reads a single line of text from the file.
  - The line variable stores the current line.

lineCount++;

- **Purpose**: Increments the lineCount by 1 for each line read, keeping track of the total number of lines.

characterCount += line.length();

- **Purpose**: Adds the length of the current line to characterCount, counting all the characters (including spaces and punctuation) in the file.

wordCount += line.split("\\s+").length;

- **Purpose**: Counts the words in the current line and adds them to wordCount.
- **Explanation**:
  - line.split("\\s+") splits the line into an array of words, using one or more whitespace characters (\\s+) as the delimiter.
  - .length gets the number of elements in the array, which represents the number of words.

}

- **Purpose**: Ends the while loop after processing all lines in the file.

System.out.println("Characters: " + characterCount);

- **Purpose**: Prints the total number of characters counted in the file.

System.out.println("Lines: " + lineCount);

- **Purpose**: Prints the total number of lines counted in the file.

System.out.println("Words: " + wordCount);

- **Purpose**: Prints the total number of words counted in the file.

} catch (IOException e) {

- **Purpose**: Catches any IOException that might occur during file reading (e.g., file not found or read errors).

e.printStackTrace();

- **Purpose**: Prints the stack trace of the exception to the console for debugging purposes.

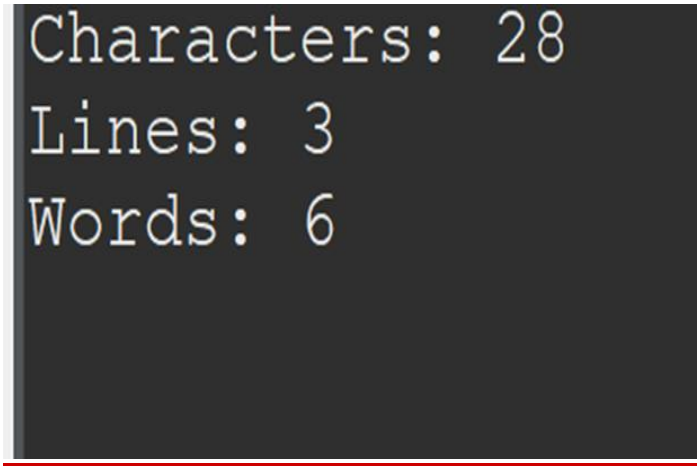
}

- **Purpose**: Ends the try-catch block

}

- **Purpose**: Ends the main method and the program.

OUTPUT:



RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 5

AIM:

Generate random numbers between two given limits using Random class and print messages according to the value range generated.

ALGORITHM:

Step 1: Start the Process.

Step 2: Import **Random** and **Scanner.**

Step 3: Create **a Scanner** and **Random** objects.

Step 4: Prompt for and read **lowerLimit** and **upperLimit.**

Step 5: Calculate **randomNumber** using

random.nextInt(upperLimit - lowerLimit + 1) + lowerLimit.

Step 6: Print the generated **randomNumber.**

Step 7: If **randomNumber < 0,** print "The number is negative.";

else if **0 ≤ randomNumber ≤ 10,** print "The number is between 0 and 10."; else if **11 ≤ randomNumber ≤ 50,** print "The number is between 11 and 50.";

else print "The number is greater than 50."

Step 8: End the Process.

FLOW CHART:

SOURCE CODE:

```
import java.util.Random;
import java.util.Scanner;
public class RandomNumberGenerator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();
        // Input for range limits
        System.out.print("Enter the lower limit: ");
```

```
        int lowerLimit = scanner.nextInt();
        System.out.print("Enter the upper limit: ");
        int upperLimit = scanner.nextInt();
        // Generate a random number within the given limits
        int randomNumber = random.nextInt(upperLimit - lowerLimit + 1) + lowerLimit;
        // Print messages based on the random number generated
        System.out.println("Generated Random Number: " + randomNumber);
        if (randomNumber < 0) {
            System.out.println("The number is negative.");
        } else if (randomNumber >= 0 && randomNumber <= 10) {
            System.out.println("The number is between 0 and 10.");
        } else if (randomNumber > 10 && randomNumber <= 50) {
            System.out.println("The number is between 11 and 50.");
        } else {
            System.out.println("The number is greater than 50.");
        }
    }
}
```

CODE EXPLANATION:

1. **import java.util.Random;**

   This imports the Random class, which is used for generating random numbers.

2. **import java.util.Scanner;**

   This imports the Scanner class, which is used for reading user input from the console.

3. **public class RandomNumberGenerator {**

   Declares the class named RandomNumberGenerator.

4. **public static void main(String[] args) {**

   Defines the main method, which is the starting point of the program.

5. **Scanner scanner = new Scanner(System.in);**

   Creates a Scanner object to take user input from the console.

6. **Random random = new Random();**

   Creates a Random object to generate random numbers.

7. **System.out.print("Enter the lower limit: ");**

   Prompts the user to enter the lower limit for the random number range.

8. **int lowerLimit = scanner.nextInt();**

   Reads an integer value from the user as the lower limit.

9. **System.out.print("Enter the upper limit: ");**

   Prompts the user to enter the upper limit for the random number range.

10. **int upperLimit = scanner.nextInt();**

    Reads an integer value from the user as the upper limit.

11. **int randomNumber = random.nextInt(upperLimit - lowerLimit + 1) + lowerLimit;**

    Generates a random number within the range [lowerLimit, upperLimit]. The formula ensures the number is inclusive of both limits.

12. **System.out.println("Generated Random Number: " + randomNumber);**

    Displays the generated random number to the user.

13. **if (randomNumber < 0) {**

    Checks if the generated random number is negative.

14. **System.out.println("The number is negative.");**

    Prints a message indicating the number is negative.

15. **} else if (randomNumber >= 0 && randomNumber <= 10) {**

    Checks if the generated number is between 0 and 10, inclusive.

16. **System.out.println("The number is between 0 and 10.");**

    Prints a message indicating the number falls in the range [0, 10].

17. **} else if (randomNumber > 10 && randomNumber <= 50) {**

    Checks if the generated number is between 11 and 50, inclusive.

18. **System.out.println("The number is between 11 and 50.");**

    Prints a message indicating the number falls in the range [11, 50].

19. **} else {**

    If none of the above conditions are met, this block executes.

20. **System.out.println("The number is greater than 50.");**

    Prints a message indicating the number is greater than 50.

21. }

Closes the if-else block.

22. }

Closes the main method.

23. }

Closes the class definition.

OUTPUT:

```
Enter the lower limit: 2
Enter the upper limit: 40
Generated Random Number: 7
The number is between 0 and 10.
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 6

AIM:

Java program to do String Manipulation using CharacterArray and perform the following string operations.

ALGORITHM:

Step 1: Start the Process.

Step 2: Import **Scanner** for reading user input.

Step 3: Create a **Scanner** object for user input.

Step 4: Prompt the user to enter the first string, read it into **firstString,** and convert it to a character array **firstCharArray.**

Step 5: Get the length of the **firstCharArray** and print the length of the first string.

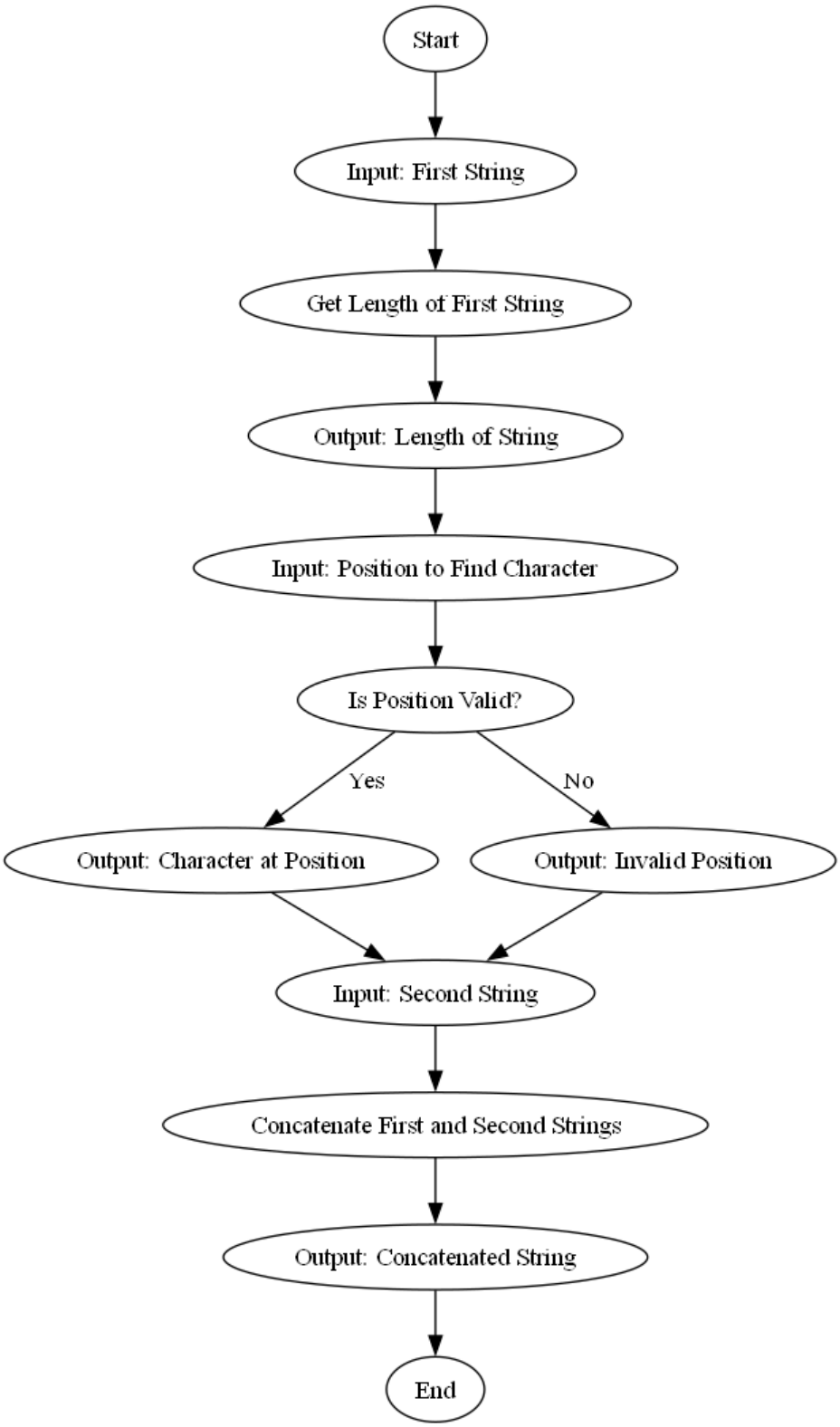
Step 6: Prompt the user to enter a position (index) to find the character, and read it into **position.** If valid **(0 to length - 1),** print **firstCharArray[position];** else print "Invalid position!"

Step 7: Clear the input buffer (optional), prompt the user to enter the second string, and read it into **secondString.**

Step 8: Concatenate **firstString** and **secondString** into a **concatenated string** and print it.

Step 9: End the Process.

FLOW CHART:


Start
Input: First String
Get Length of First String
Output: Length of String
Input: Position to Find Character
Is Position Valid?
Yes
No
Output: Character at Position
Output: Invalid Position
Input: Second String
Concatenate First and Second Strings
Output: Concatenated String
End

SOURCE CODE:

```
import java.util.Scanner;
public class StringManipulation {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input for the first string
        System.out.print("Enter the first string: ");
        String firstString = scanner.nextLine();
        char[] firstCharArray = firstString.toCharArray(); // Convert to character array
        // a) Get the length of the string
        int length = firstCharArray.length;
        System.out.println("Length of the first string: " + length);
        // b) Finding a character at a particular position
        System.out.print("Enter a position to find the character (0 to " + (length - 1) + "): ");
        int position = scanner.nextInt();
        if (position >= 0 && position < length) {
            char characterAtPosition = firstCharArray[position];
            System.out.println("Character at position " + position + ": " + characterAtPosition);
        } else {
            System.out.println("Invalid position!");
        }
        // Input for the second string
        scanner.nextLine(); // Clear the buffer
        System.out.print("Enter the second string: ");
        String secondString = scanner.nextLine();
        // c) Concatenating two strings
        String concatenatedString = firstString + secondString;
        System.out.println("Concatenated string: " + concatenatedString);
    }
}
```

CODE EXPLANATION:

1. **import java.util.Scanner;**
   - This imports the Scanner class, which is used to read input from the user.
2. **public class StringManipulation {**
   - Declares a class named StringManipulation, which contains the program logic.
3. **public static void main(String[] args) {**
   - Defines the main method, which is the entry point of the program.
4. **Scanner scanner = new Scanner(System.in);**
   - Creates a Scanner object to read input from the console.
5. **System.out.print("Enter the first string: ");**
   - Prompts the user to enter the first string.
6. **String firstString = scanner.nextLine();**
   - Reads the first string input from the user, including spaces.
7. **char[] firstCharArray = firstString.toCharArray();**
   - Converts the first string into a character array for easier manipulation.
8. **int length = firstCharArray.length;**
   - Calculates the length of the first string using the length property of the character array.
9. **System.out.println("Length of the first string: " + length);**
   - Prints the length of the first string.
10. **System.out.print("Enter a position to find the character (0 to " + (length - 1) + "): ");**
    - Prompts the user to enter a position to retrieve a character from the first string.
11. **int position = scanner.nextInt();**
    - Reads the position input as an integer.
12. **if (position >= 0 && position < length) {**
    - Checks if the entered position is within the valid range (0 to length-1).
13. **char characterAtPosition = firstCharArray[position];**
    - Retrieves the character at the specified position from the character array.
14. **System.out.println("Character at position " + position + ": " + characterAtPosition);**
    - Prints the character found at the specified position.
15. **} else {**

- Executes if the position entered is invalid.

16. **System.out.println("Invalid position!");**
    - Prints a message indicating the entered position is not valid.
17. **scanner.nextLine();**
    - Clears the scanner buffer to avoid issues when switching between nextInt and nextLine.
18. **System.out.print("Enter the second string: ");**
    - Prompts the user to enter the second string.
19. **String secondString = scanner.nextLine();**
    - Reads the second string input from the user, including spaces.
20. **String concatenatedString = firstString + secondString;**
    - Concatenates the first and second strings using the + operator.
21. **System.out.println("Concatenated string: " + concatenatedString);**
    - Prints the concatenated result of the two strings.
22. **}**
    - Closes the main method.
23. **}**
    - Closes the class definition.

OUTPUT:

```
Enter the first string: Java
Length of the first string: 4
Enter a position to find the character (0 to 3): 2
Character at position 2: v
Enter the second string:  is a high level language
Concatenated string: Java is a high level language
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 7

AIM:

To develop a Java program that demonstrates fundamental string operations, including String Concatenation, Substring Search, and Substring Extraction.

ALGORITHM:

Step 1: Start the Process.

Step 2: Initialize Scanner for user input.

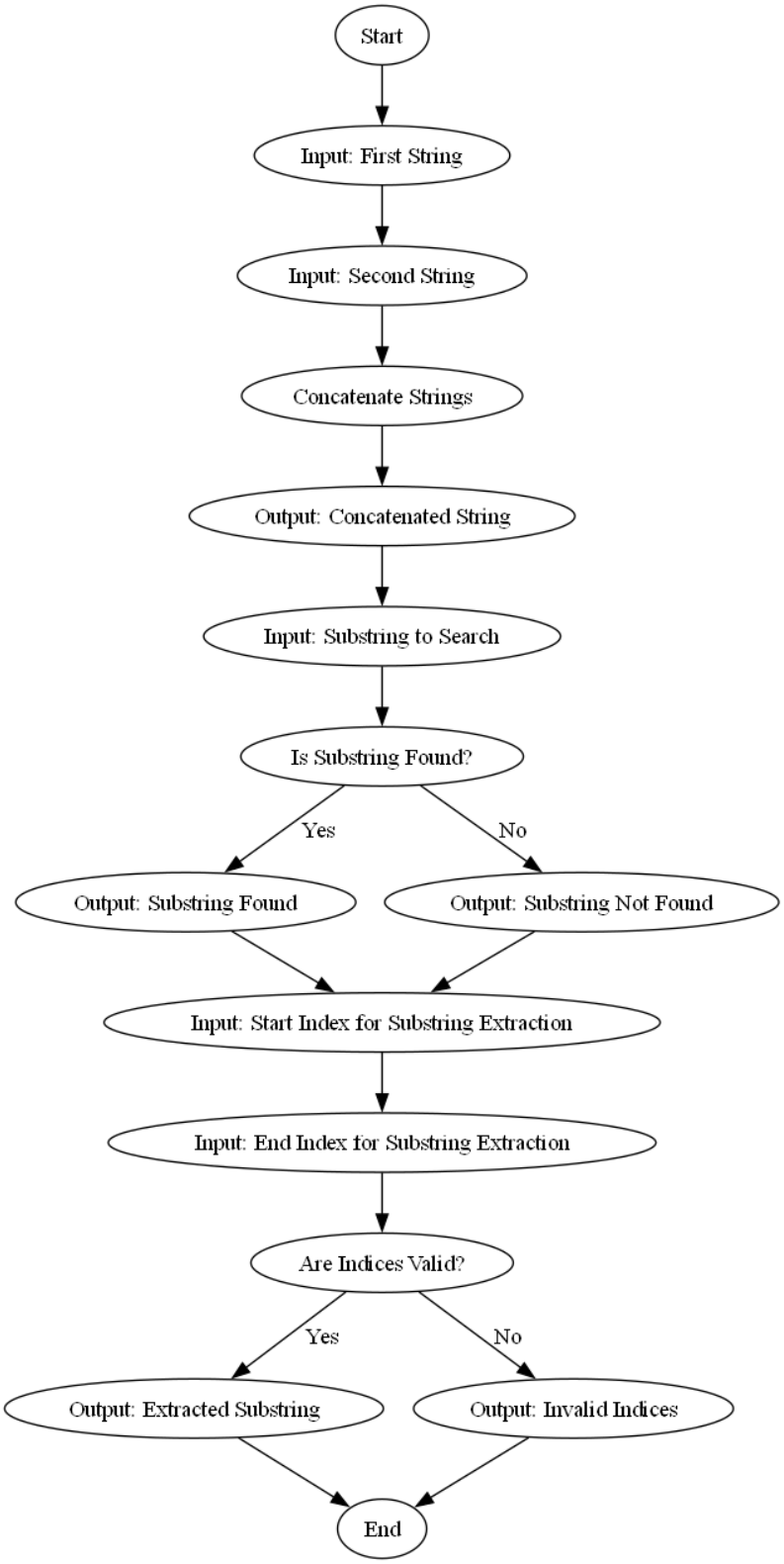
Step 3: Read the first string (str1) and second string (str2). Concatenate them into **concatenated** and display.

Step 4: Read the substring to search. Check if **concatenated** contains it; display "Substring found!" or "Substring not found.".

Step 5: Read start and end indices. If valid, extract substring from **concatenated** and display; else, display "Invalid indices.".

Step 6: End the Process.

FLOW CHART:


Start
Input: First String
Input: Second String
Concatenate Strings
Output: Concatenated String
Input: Substring to Search
Is Substring Found?
Yes
No
Output: Substring Found
Output: Substring Not Found
Input: Start Index for Substring Extraction
Input: End Index for Substring Extraction
Are Indices Valid?
Yes
No
Output: Extracted Substring
Output: Invalid Indices
End

SOURCE CODE:

```
import java.util.Scanner;
public class SimpleStringOperations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Concatenation
        System.out.print("Enter the first string: ");
        String str1 = scanner.nextLine();
        System.out.print("Enter the second string: ");
        String str2 = scanner.nextLine();
        String concatenated = str1 + str2;
        System.out.println("Concatenated String: " + concatenated);
        // Search for a substring
        System.out.print("Enter a substring to search: ");
        String substring = scanner.nextLine();
        if (concatenated.contains(substring)) {
            System.out.println("Substring found!");
        } else {
            System.out.println("Substring not found.");
        }
            // Extract a substring
        System.out.print("Enter start index: ");
        int start = scanner.nextInt();
        System.out.print("Enter end index: ");
        int end = scanner.nextInt();
        if (start >= 0 && end <= concatenated.length()) {
            String extracted = concatenated.substring(start, end);
            System.out.println("Extracted Substring: " + extracted);
        } else {
            System.out.println("Invalid indices.");
        }
```

```
    }
}
```

CODE EXPLANATION:

1. **import java.util.Scanner;**
   - Imports the Scanner class, which allows reading user input from the console.
2. **public class SimpleStringOperations {**
   - Declares the class SimpleStringOperations, where the program logic will be implemented.
3. **public static void main(String[] args) {**
   - Defines the main method, which is the entry point of the program.
4. **Scanner scanner = new Scanner(System.in);**
   - Creates a Scanner object named scanner to read input from the user.
5. **System.out.print("Enter the first string: ");**
   - Prompts the user to enter the first string.
6. **String str1 = scanner.nextLine();**
   - Reads the first string entered by the user and stores it in the variable str1.
7. **System.out.print("Enter the second string: ");**
   - Prompts the user to enter the second string.
8. **String str2 = scanner.nextLine();**
   - Reads the second string entered by the user and stores it in the variable str2.
9. **String concatenated = str1 + str2;**
   - Concatenates str1 and str2 using the + operator and stores the result in the variable concatenated.
10. **System.out.println("Concatenated String: " + concatenated);**
    - Prints the concatenated string to the console.
11. **System.out.print("Enter a substring to search: ");**
    - Prompts the user to enter a substring that they want to search for in the concatenated string.
12. **String substring = scanner.nextLine();**
    - Reads the substring entered by the user and stores it in the variable substring.

13. **if (concatenated.contains(substring)) {**
    - Checks if the concatenated string contains the entered substring using the contains method.
14. **System.out.println("Substring found!");**
    - If the substring is found, it prints "Substring found!".
15. **} else {**
    - If the substring is not found, it executes the code in this block.
16. **System.out.println("Substring not found.");**
    - Prints "Substring not found." if the substring is not found in the concatenated string.
17. **System.out.print("Enter start index: ");**
    - Prompts the user to enter the start index for the substring extraction.
18. **int start = scanner.nextInt();**
    - Reads the start index entered by the user and stores it in the variable start.
19. **System.out.print("Enter end index: ");**
    - Prompts the user to enter the end index for the substring extraction.
20. **int end = scanner.nextInt();**
    - Reads the end index entered by the user and stores it in the variable end.
21. **if (start >= 0 && end <= concatenated.length()) {**
    - Checks if the entered start and end indices are valid. Specifically, it ensures that the start index is greater than or equal to 0 and the end index is less than or equal to the length of the concatenated string.
22. **String extracted = concatenated.substring(start, end);**
    - If the indices are valid, it extracts the substring from the concatenated string using the substring method, from start to end index, and stores it in the extracted variable.
23. **System.out.println("Extracted Substring: " + extracted);**
    - Prints the extracted substring to the console.
24. **} else {**
    - If the indices are invalid, this block is executed.
25. **System.out.println("Invalid indices.");**
    - Prints an error message indicating the indices provided are invalid.
26. **}**

- Closes the if-else block for extracting the substring.

27. }

- Closes the main method.

28. }

- Closes the class definition.

OUTPUT:

```
Enter the first string: Java Programming
Enter the second string:  Language
Concatenated String: Java Programming Language
Enter a substring to search: Program
Substring found!
Enter start index: 0
Enter end index: 15
Extracted Substring: Java Programmin
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 8

AIM:

To create a Java program that utilizes the StringBuffer class to perform string operations: determine string length, reverse the string, and delete a specified substring.

ALGORITHM:

Step 1: Start the Process.

Step 2: Initialize Scanner for user input.

Step 3: Read a string to create a StringBuffer object (sb).

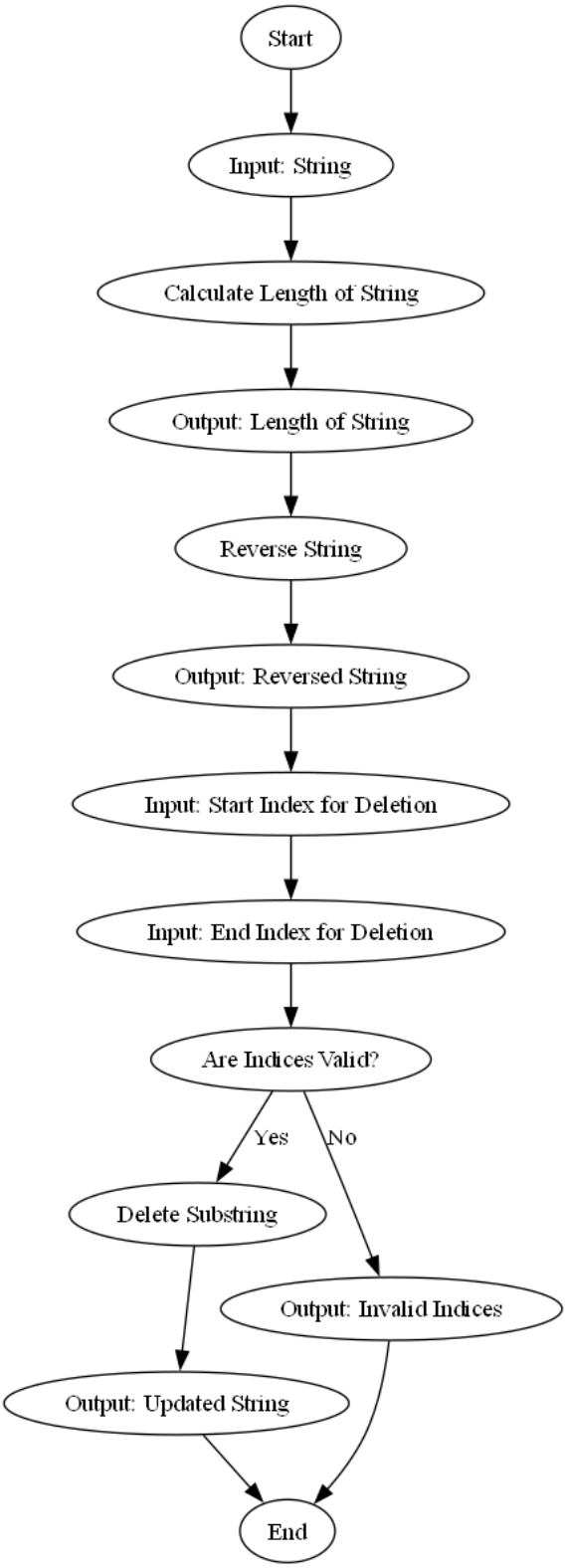
Step 4: Find and display the length of sb using the length() method.

Step 5: Reverse sb using the reverse() method and display it.

Step 6: Read start and end indices for deletion. If valid (start >= 0, end <= length of sb, start < end), delete the substring using delete(start, end) and display the modified string; else, display "Invalid indices.".

Step 7: End the Process.

FLOW CHART:


Start
Input: String
Calculate Length of String
Output: Length of String
Reverse String
Output: Reversed String
Input: Start Index for Deletion
Input: End Index for Deletion
Are Indices Valid?
Yes
No
Delete Substring
Output: Invalid Indices
Output: Updated String
End

SOURCE CODE:

```
import java.util.Scanner;
public class SimpleStringBuffer {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Input string
        System.out.print("Enter a string: ");
        StringBuffer sb = new StringBuffer(scanner.nextLine());
        // Length of the string
        System.out.println("Length: " + sb.length());
        // Reverse the string
        System.out.println("Reversed: " + sb.reverse());
        // Delete a substring
        System.out.print("Start index to delete: ");
        int start = scanner.nextInt();
        System.out.print("End index to delete: ");
        int end = scanner.nextInt();
        if (start >= 0 && end <= sb.length() && start < end) {
            sb.delete(start, end);
            System.out.println("After deletion: " + sb);
        } else {
            System.out.println("Invalid indices.");
        }
    }
}
```

CODE EXPLANATION:

1. **import java.util.Scanner;**
    - Imports the Scanner class to read user input from the console.
2. **public class SimpleStringBuffer {**

- Defines the class SimpleStringBuffer, where the program logic will be implemented.

3. **public static void main(String[] args) {**
   - Declares the main method, which serves as the entry point of the program.
4. **Scanner scanner = new Scanner(System.in);**
   - Creates a Scanner object called scanner to read input from the user.
5. **System.out.print("Enter a string: ");**
   - Prompts the user to enter a string.
6. **StringBuffer sb = new StringBuffer(scanner.nextLine());**
   - Reads the string entered by the user and stores it in a StringBuffer object named sb. StringBuffer allows us to modify the string.
7. **System.out.println("Length: " + sb.length());**
   - Prints the length of the string using the length() method of StringBuffer.
8. **System.out.println("Reversed: " + sb.reverse());**
   - Reverses the string using the reverse() method of StringBuffer and prints the reversed string.
9. **System.out.print("Start index to delete: ");**
   - Prompts the user to enter the start index for deleting a substring.
10. **int start = scanner.nextInt();**
    - Reads the start index entered by the user and stores it in the variable start.
    -
11. **System.out.print("End index to delete: ");**
    - Prompts the user to enter the end index for deleting a substring.
12. **int end = scanner.nextInt();**
    - Reads the end index entered by the user and stores it in the variable end.
13. **if (start >= 0 && end <= sb.length() && start < end) {**
    - Checks if the entered indices are valid. The start index must be greater than or equal to 0, the end index must be less than or equal to the length of the string, and the start index must be less than the end index.
14. **sb.delete(start, end);**

- If the indices are valid, it deletes the substring from the start index to the end index using the delete() method of StringBuffer.

15. **System.out.println("After deletion: " + sb);**
    - Prints the modified string after the deletion.
16. **} else {**
    - If the indices are invalid, this block is executed.
17. **System.out.println("Invalid indices.");**
    - Prints an error message indicating that the indices provided are invalid.
18. **}**
    - Closes the if-else block.
19. **}**
    - Closes the main method.
20. **}**
    - Closes the class definition.

OUTPUT:

```
Enter a string: Program Language
Length: 16
Reversed: egaugnaL margorP
Start index to delete: 0
End index to delete: 5
After deletion: naL margorP
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 9

AIM:

To implement a multi-threaded Java application that generates random integers and computes their squares if even or cubes if odd, using three separate threads.

ALGORITHM:

Step 1: Start the Process.

Step 2: Create the **RandomNumberGenerator** Class. Create a **NumberProcessor** object. In the **run()** method, generate a random integer between 0 and 99, print it, call **processNumber(number),** and sleep for 1 second.

Step 3: Create the **NumberProcessor** Class. Declare an integer called **number.** In the **processNumber(int number)** method, set **number** and notify waiting threads. In the **square()** method, wait for a number; if it's even, print its square. In the **cube()** method, wait for a number; if it's odd, print its cube.

Step 4: Create the **MultiThreadedRandomNumber** Class. In the **main()** method**,** create a **NumberProcessor** instance. Start the **RandomNumberGenerator** thread, the **square()** thread, and the **cube()** thread.

Step 5: End the Process.

FLOW CHART:

SOURCE CODE:

```
import java.util.Random;
class RandomNumberGeneratorEx extends Thread {
    private final NumberProcessor processor;
    public RandomNumberGeneratorEx(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.processNumber(number);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                break;
            }
        }
    }
}
class NumberProcessor {
    private int number;
    public synchronized void processNumber(int number) {
        this.number = number;
        notifyAll();
    }
    public void square() {
        while (true) {
            synchronized (this) {
                try {
                    wait();
                    if (number % 2 == 0) {
                        System.out.println("Square: " + (number * number));
                    }
                } catch (InterruptedException e) {
                    break;
```

```
            }
        }
    }
  }
  public void cube() {
    while (true) {
      synchronized (this) {
        try {
          wait();
          if (number % 2 != 0) {
            System.out.println("Cube: " + (number * number * number));
          }
        } catch (InterruptedException e) {
          break;
        }
      }
    }
  }
}

public class MultiThreadedRandomNumber {
  public static void main(String[] args) {
    NumberProcessor processor = new NumberProcessor();
    RandomNumberGeneratorEx generator = new RandomNumberGeneratorEx(processor);
    Thread squareThread = new Thread(processor::square);
    Thread cubeThread = new Thread(processor::cube);
    generator.start();
    squareThread.start();
    cubeThread.start();
  }
}
```

CODE EXPLANATION:

1. **RandomNumberGeneratorEx extends Thread {**
   - The RandomNumberGeneratorEx class is a subclass of the Thread class, meaning it will execute in its own thread of execution.
2. **private final NumberProcessor processor;**
   - A private member processor of type NumberProcessor is declared. This will be used to process numbers.
3. **public RandomNumberGeneratorEx(NumberProcessor processor) {**
   - Constructor to initialize the processor object. This is called when a RandomNumberGeneratorEx object is created.
4. **this.processor = processor;**
   - Inside the constructor, the processor parameter is assigned to the class's processor variable.
5. **@Override public void run() {**
   - The run() method is overridden. This is the entry point for the thread execution.
6. **Random random = new Random();**
   - A Random object is created to generate random numbers.
7. **while (true) {**
   - Starts an infinite loop where random numbers will continuously be generated.
8. **int number = random.nextInt(100);**
   - Generates a random integer between 0 and 99.
9. **System.out.println("Generated: " + number);**
   - Prints the generated random number to the console.
10. **processor.processNumber(number);**
    - Passes the generated number to the processNumber() method of NumberProcessor.
11. **try { Thread.sleep(1000); }**
    - The thread sleeps for 1000 milliseconds (1 second), creating a pause before generating the next random number.
    -
12. **catch (InterruptedException e) { break; }**

- If the thread is interrupted, it will break out of the loop and stop execution.

13. **}**
    - Ends the run() method.

NumberProcessor Class:

14. **class NumberProcessor {**
    - The NumberProcessor class is responsible for processing the generated numbers.
15. **private int number;**
    - A private number variable that stores the current number being processed.
16. **public synchronized void processNumber(int number) {**
    - This method is synchronized to ensure that only one thread at a time can execute it.
    - It takes the number as input and processes it.
17. **this.number = number;**
    - The input number is assigned to the class's number variable.
18. **notifyAll();**
    - This call notifies all waiting threads that they can proceed (i.e., they are allowed to process the number).
19. **}**
    - Ends the processNumber() method.

Square Method:

20. **public void square() {**
    - Defines the square() method to process the number and compute its square.
21. **while (true) {**
    - Starts an infinite loop that continues to process numbers.
22. **synchronized (this) {**
    - This block is synchronized to ensure mutual exclusion while accessing the shared number variable.
23. **try { wait(); }**
    - The wait() method causes the thread to wait until notified by another thread.

24. **if (number % 2 == 0) {**
    - Check if the current number is even.
25. **System.out.println("Square: " + (number * number));**
    - If the number is even, it calculates the square and prints it.
26. **} catch (InterruptedException e) { break; }**
    - If the thread is interrupted, it breaks out of the loop and stops execution.
27. **}**
    - Ends the synchronized block.
28. **}**
    - Ends the square() method.

Cube Method:

29. **public void cube() {**
    - Defines the cube() method to process the number and compute its cube.
30. **while (true) {**
    - Starts an infinite loop to process numbers.
31. **synchronized (this) {**
    - The synchronized block ensures that only one thread can process the number at a time.
32. **try { wait(); }**
    - Causes the thread to wait until it is notified by another thread.
33. **if (number % 2 != 0) {**
    - Check if the current number is odd.
34. **System.out.println("Cube: " + (number * number * number));**
    - If the number is odd, it calculates and prints the cube.
35. **} catch (InterruptedException e) { break; }**
    - If the thread is interrupted, it exits the loop and stops execution.
36. **}**
    - Ends the synchronized block.
37. **}**
    - Ends the cube() method.

MultiThreadedRandomNumber (Main Class):

38. **public class MultiThreadedRandomNumber {**
    - The main class where the program execution begins.
39. **public static void main(String[] args) {**
    - The main() method is the entry point of the program.
40. **NumberProcessor processor = new NumberProcessor();**
    - Creates a new instance of NumberProcessor, which will process the random numbers.
41. **RandomNumberGeneratorEx generator = new RandomNumberGeneratorEx(processor);**
    - Creates a new RandomNumberGeneratorEx object, passing the processor to it.
42. **Thread squareThread = new Thread(processor::square);**
    - Creates a new thread that will run the square() method from NumberProcessor.
43. **Thread cubeThread = new Thread(processor::cube);**
    - Creates a new thread that will run the cube() method from NumberProcessor.
44. **generator.start();**
    - Starts the RandomNumberGeneratorEx thread, which begins generating random numbers.
45. **squareThread.start();**
    - Starts the thread that calculates the square of even numbers.
46. **cubeThread.start();**
    - Starts the thread that calculates the cube of odd numbers.
47. **}**
    - Ends the main() method.
48. **}**
    - Ends the MultiThreadedRandomNumber class.

OUTPUT:

```
Generated: 85
Cube: 614125
Generated: 8
Square: 64
Generated: 11
Cube: 1331
Generated: 35
Cube: 42875
Generated: 40
Square: 1600
Generated: 76
Square: 5776
Generated: 24
Square: 576
Generated: 61
Cube: 226981
Generated: 34
Square: 1156
Generated: 33
Cube: 35937
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 10

AIM:

To implement a Java program that uses multi-threading to print numbers from 1 to 10 and from 90 to 100 asynchronously using the same method.

ALGORITHM:

Step 1: Start.

Step 2: Define the NumberPrinter Class.

Step 3: Declare attributes int start for the starting number and int end for the ending number.

Step 4: Create a constructor to initialize start and end with given values.

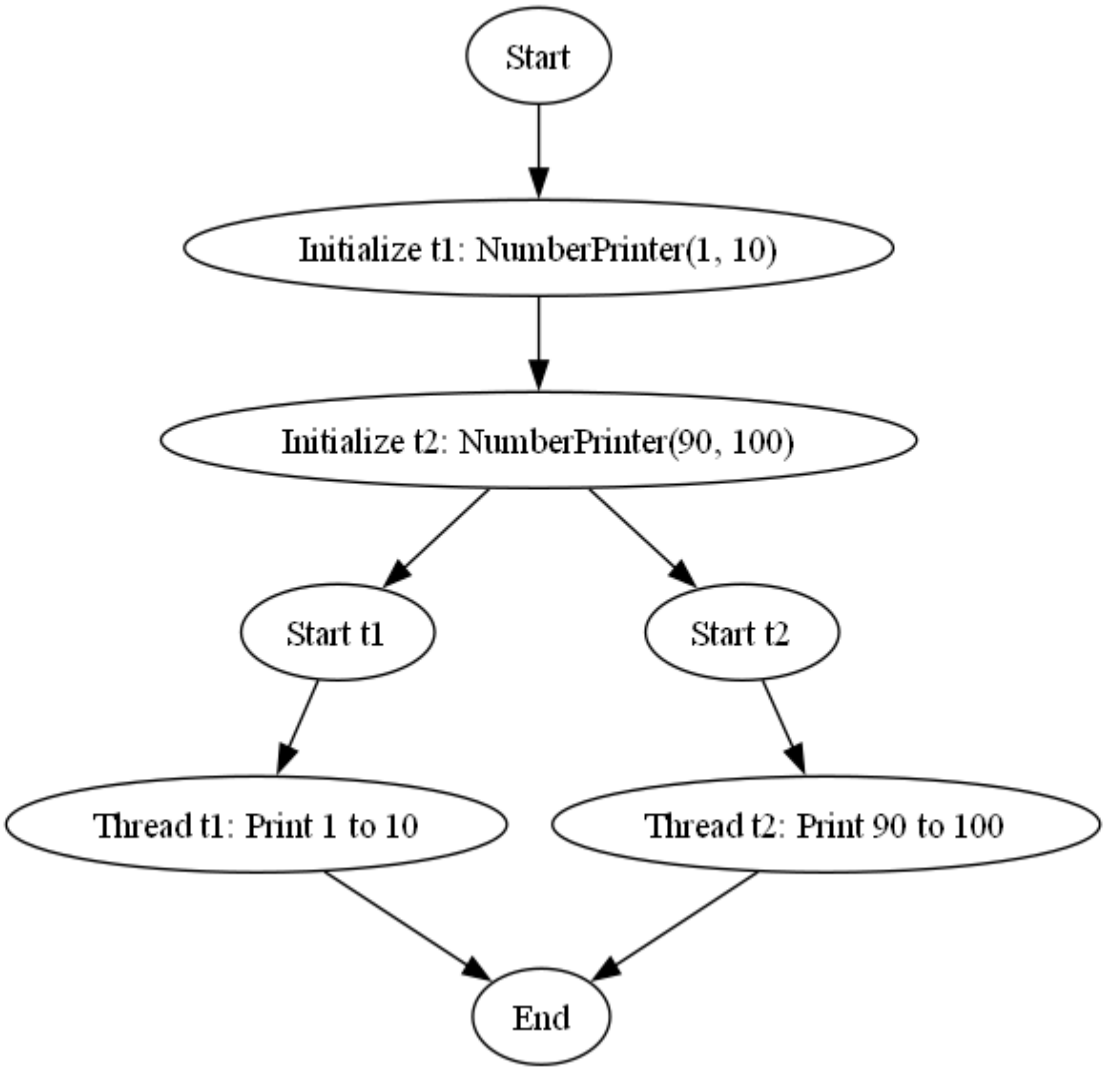
Step 5: In the run() method, loop from start to end (inclusive) and print the current number. Sleep for 500 milliseconds and handle any InterruptedException by restoring the thread's interrupted status.

Step 6: Define the AsyncNumberPrinting Class. In the main() method, create a Thread object t1 for printing numbers from 1 to 10 using NumberPrinter.

Step 7: Create another Thread object t2 for printing numbers from 90 to 100 using NumberPrinter. Start both threads using t1.start() and t2.start().

Step 8: End the Process.

FLOW CHART:



SOURCE CODE:

```
class NumberPrinter extends Thread {
    private final int start;
    private final int end;
    public NumberPrinter(int start, int end) {
        this.start = start;
        this.end = end;
    }
    @Override
```

```
    public void run() {
        for (int i = start; i <= end; i++) {
            System.out.println(i);
            try {
                Thread.sleep(500); // Pause for visibility
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
public class AsyncNumberPrinting {
    public static void main(String[] args) {
        Thread t1 = new NumberPrinter(1, 10);
        Thread t2 = new NumberPrinter(90, 100);
        t1.start();
        t2.start();
    }
}
```

CODE EXPLANATION:

NumberPrinter Class:

1. **class NumberPrinter extends Thread {**
    - Defines the NumberPrinter class, which extends the Thread class. This means the class will run in its own separate thread of execution.
2. **private final int start;**
    - Declares a private member variable start of type int. This will store the starting number from which the printing begins.

3. **private final int end;**
   - Declares a private member variable end of type int. This will store the ending number up to which the numbers will be printed.
4. **public NumberPrinter(int start, int end) {**
   - The constructor for the NumberPrinter class, which takes two parameters start and end. These parameters define the range of numbers to print.
5. **this.start = start;**
   - Assigns the start parameter value to the start member variable of the NumberPrinter class.
6. **this.end = end;**
   - Assigns the end parameter value to the end member variable of the NumberPrinter class.
7. **@Override public void run() {**
   - Overrides the run() method from the Thread class. This is the method that gets executed when the thread starts.
8. **for (int i = start; i <= end; i++) {**
   - Starts a for loop that begins at the start value and runs until i equals end. Each iteration prints a number.
9. **System.out.println(i);**
   - Prints the current value of i to the console, which is the current number in the loop.
10. **try { Thread.sleep(500); }**
    - Pauses the execution of the current thread for 500 milliseconds (half a second) to allow for better visibility of the printed numbers.
11. **catch (InterruptedException e) {**
    - Catches any InterruptedException that may occur during the Thread.sleep() operation. This exception occurs if the thread is interrupted while sleeping.
12. **Thread.currentThread().interrupt();**
    - Restores the interrupted status of the current thread after the exception is caught. This ensures that the interruption is not lost, and can be handled appropriately if needed.
13. **}**

- Ends the try-catch block.

14. **}**
    - Ends the for loop.
15. **}**
    - Ends the run() method.
16. **}**
    - Ends the NumberPrinter class.

AsyncNumberPrinting (Main Class):

17. **public class AsyncNumberPrinting {**
    - Defines the AsyncNumberPrinting class, which contains the main() method where the execution begins.
18. **public static void main(String[] args) {**
    - The main() method is the entry point of the program. It's where the execution starts when you run the program.
19. **Thread t1 = new NumberPrinter(1, 10);**
    - Creates a new instance of the NumberPrinter class called t1, and initializes it with a range from 1 to 10. This thread will print the numbers from 1 to 10.
20. **Thread t2 = new NumberPrinter(90, 100);**
    - Creates a new instance of the NumberPrinter class called t2, and initializes it with a range from 90 to 100. This thread will print the numbers from 90 to 100.
21. **t1.start();**
    - Starts the thread t1, which invokes the run() method of NumberPrinter. This will begin printing the numbers from 1 to 10.
22. **t2.start();**
    - Starts the thread t2, which also invokes the run() method of NumberPrinter. This will begin printing the numbers from 90 to 100.
23. **}**
    - Ends the main() method.
24. **}**

- Ends the AsyncNumberPrinting class.

OUTPUT:

```
90
1
91
2
3
92
4
93
94
5
95
6
96
7
97
8
98
9
10
99
100
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT – 11

AIM:

To illustrate the handling of various common exceptions in Java, including ArithmeticException, NumberFormatException, ArrayIndexOutOfBoundsException, and NegativeArraySizeException.

ALGORITHM:

Step 1: Start.

Step 2: Initialize Scanner for user input.

Step 3: Arithmetic Exception: Prompt for a number to divide 10 by. Try to calculate and print 10 divided by the number. If ArithmeticException occurs, print "Error: Cannot divide by zero!"

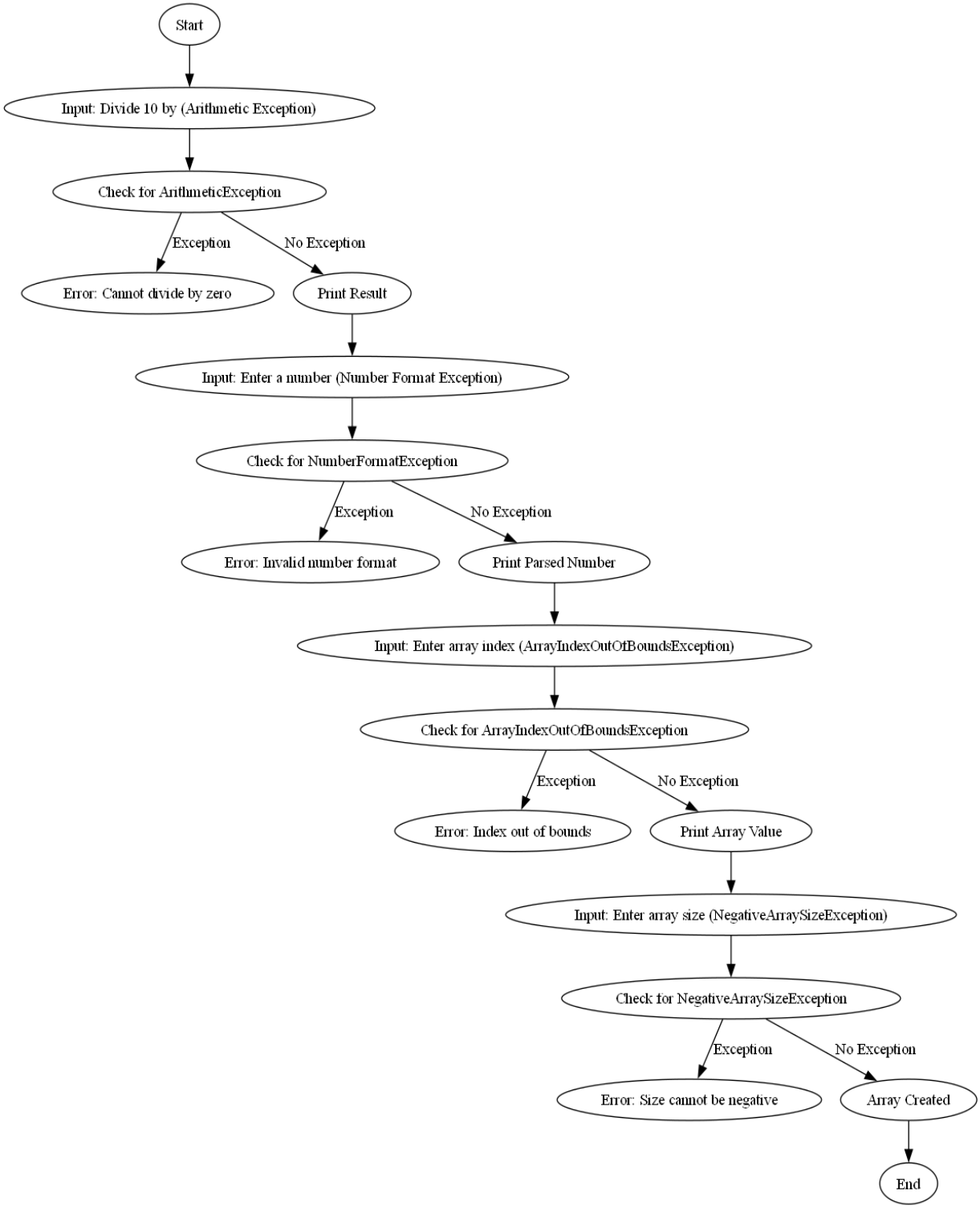
Step 4: Number Format Exception: Prompt for a number. Try to read it as a string and convert it to an integer. If NumberFormatException occurs, print "Error: Invalid number format!"

Step 5: ArrayIndexOutOfBoundsException: Initialize an array {1, 2, 3}. Prompt for an index. Try to print the array value at that index. If **ArrayIndexOutOfBoundsException** occurs, print "Error: Index out of bounds!"

Step 6: NegativeArraySizeException: Prompt for an array size. Try to create an array of that size and print a success message. If **NegativeArraySizeException** occurs, print "Error: Size cannot be negative!"

Step 7: End the Process.

FLOW CHART:


Start
Input: Divide 10 by (Arithmetic Exception)
Check for ArithmeticException
Exception
No Exception
Error: Cannot divide by zero
Print Result
Input: Enter a number (Number Format Exception)
Check for NumberFormatException
Exception
No Exception
Error: Invalid number format
Print Parsed Number
Input: Enter array index (ArrayIndexOutOfBoundsException)
Check for ArrayIndexOutOfBoundsException
Exception
No Exception
Error: Index out of bounds
Print Array Value
Input: Enter array size (NegativeArraySizeException)
Check for NegativeArraySizeException
Exception
No Exception
Error: Size cannot be negative
Array Created
End

SOURCE CODE:

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

        // a) Arithmetic Exception
        try {
            System.out.print("Divide 10 by: ");
            int num = scanner.nextInt();
            System.out.println("Result: " + (10 / num));
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }

        // b) Number Format Exception
        try {
            System.out.print("Enter a number: ");
            int parsedNumber = Integer.parseInt(scanner.next());
            System.out.println("Parsed Number: " + parsedNumber);
        } catch (NumberFormatException e) {
            System.out.println("Error: Invalid number format!");
        }

        // c) ArrayIndexOutOfBoundsException
        try {
            int[] array = {1, 2, 3};
            System.out.print("Access index (0-2): ");
            int index = scanner.nextInt();
            System.out.println("Array value: " + array[index]);
        } catch (ArrayIndexOutOfBoundsException e) {
```

```
            System.out.println("Error: Index out of bounds!");
        }

        // d) NegativeArraySizeException
        try {
            System.out.print("Enter array size: ");
            int size = scanner.nextInt();
            int[] array = new int[size]; // May throw NegativeArraySizeException
            System.out.println("Array created with size: " + size);
        } catch (NegativeArraySizeException e) {
            System.out.println("Error: Size cannot be negative!");
        }
    }
}
```

CODE EXPLANATION:

import java.util.Scanner;

- Imports the Scanner class from the java.util package, allowing user input via the console.

public class SimpleExceptionDemo {

- Defines the SimpleExceptionDemo class. This is the main class that contains the program's logic.

public static void main(String[] args) {

- Defines the main method. The entry point of the program where the execution begins.

Scanner scanner = new Scanner(System.in);

- Creates a Scanner object named scanner to read input from the console (standard input stream, System.in).

// a) Arithmetic Exception

- This is a comment that marks the start of the block of code handling the ArithmeticException.

try {

- Begins the try block. Any code that may throw an exception is placed inside this block.

System.out.print("Divide 10 by: ");

- Prompts the user to input a number to divide 10 by.

int num = scanner.nextInt();

- Reads an integer input from the user and stores it in the variable num.

System.out.println("Result: " + (10 / num));

- Attempts to divide 10 by num and print the result. If num is 0, an ArithmeticException will occur.

} catch (ArithmeticException e) {

- Catches the ArithmeticException if it occurs (e.g., division by zero). This is part of the exception handling mechanism.

System.out.println("Error: Cannot divide by zero!");

- Prints a custom error message if a division by zero occurs.

// b) Number Format Exception

- This is a comment marking the start of the block of code handling the NumberFormatException.

try {

- Begins another try block. Any code inside this block may potentially throw an exception, in this case, related to number parsing.

System.out.print("Enter a number: ");

- Prompts the user to enter a number.

int parsedNumber = Integer.parseInt(scanner.next());

- Reads the user input as a string and attempts to parse it into an integer using Integer.parseInt(). If the user enters a non-integer value, a NumberFormatException will be thrown.

System.out.println("Parsed Number: " + parsedNumber);

- If no exception occurs, it prints the parsed number.

} catch (NumberFormatException e) {

- Catches the NumberFormatException if it occurs (e.g., the user enters a non-integer string). This ensures that the program does not crash and handles the error gracefully.

System.out.println("Error: Invalid number format!");

- Prints a custom error message if the input is not a valid integer.

// c) ArrayIndexOutOfBoundsException

- This is a comment marking the start of the block of code handling the ArrayIndexOutOfBoundsException.

try {

- Begins another try block. This block contains code that may throw an ArrayIndexOutOfBoundsException.

int[] array = {1, 2, 3};

- Declares and initializes an array with three elements: 1, 2, and 3.

System.out.print("Access index (0-2): ");

- Prompts the user to input an index to access an element in the array.

int index = scanner.nextInt();

- Reads the user's input as an integer, which represents the index to access in the array.

System.out.println("Array value: " + array[index]);

- Tries to access the value at the specified index in the array and print it. If the index is out of bounds (less than 0 or greater than 2), it will throw an ArrayIndexOutOfBoundsException.

} catch (ArrayIndexOutOfBoundsException e) {

- Catches the ArrayIndexOutOfBoundsException if the user enters an invalid index (outside the range of the array's valid indices).

System.out.println("Error: Index out of bounds!");

- Prints a custom error message if the index is out of bounds.

// d) NegativeArraySizeException

- This is a comment marking the start of the block of code handling the NegativeArraySizeException.

try {

- Begins another try block. This block contains code that may throw a NegativeArraySizeException.

System.out.print("Enter array size: ");

- Prompts the user to input a size for an array.

int size = scanner.nextInt();

- Reads the user's input as an integer, which represents the size of the array to be created.

int[] array = new int[size];

- Attempts to create an array of integers with the size specified by the user. If the user enters a negative number, a NegativeArraySizeException will be thrown.

System.out.println("Array created with size: " + size);

- If the array is successfully created, this line prints the size of the array.

} catch (NegativeArraySizeException e) {

- Catches the NegativeArraySizeException if the user enters a negative size for the array.

System.out.println("Error: Size cannot be negative!");

- Prints a custom error message if the array size is negative.

}

- Closes the main method.

}

- Closes the SimpleExceptionDemo class.

OUTPUT:

```
Divide 10 by: 5
Result: 2
Enter a number: 25
Parsed Number: 25
Access index (0-2): 1
Array value: 2
Enter array size: 10
Array created with size: 10
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT – 12

AIM:

To create a Java program that retrieves and displays file information, including existence, readability, writability, type, and size, based on user-provided filename.

ALGORITHM:

Step 1: Start the Process.

Step 2: Initialize the Scanner to read user input.

Step 3: Prompt for filename and read input.

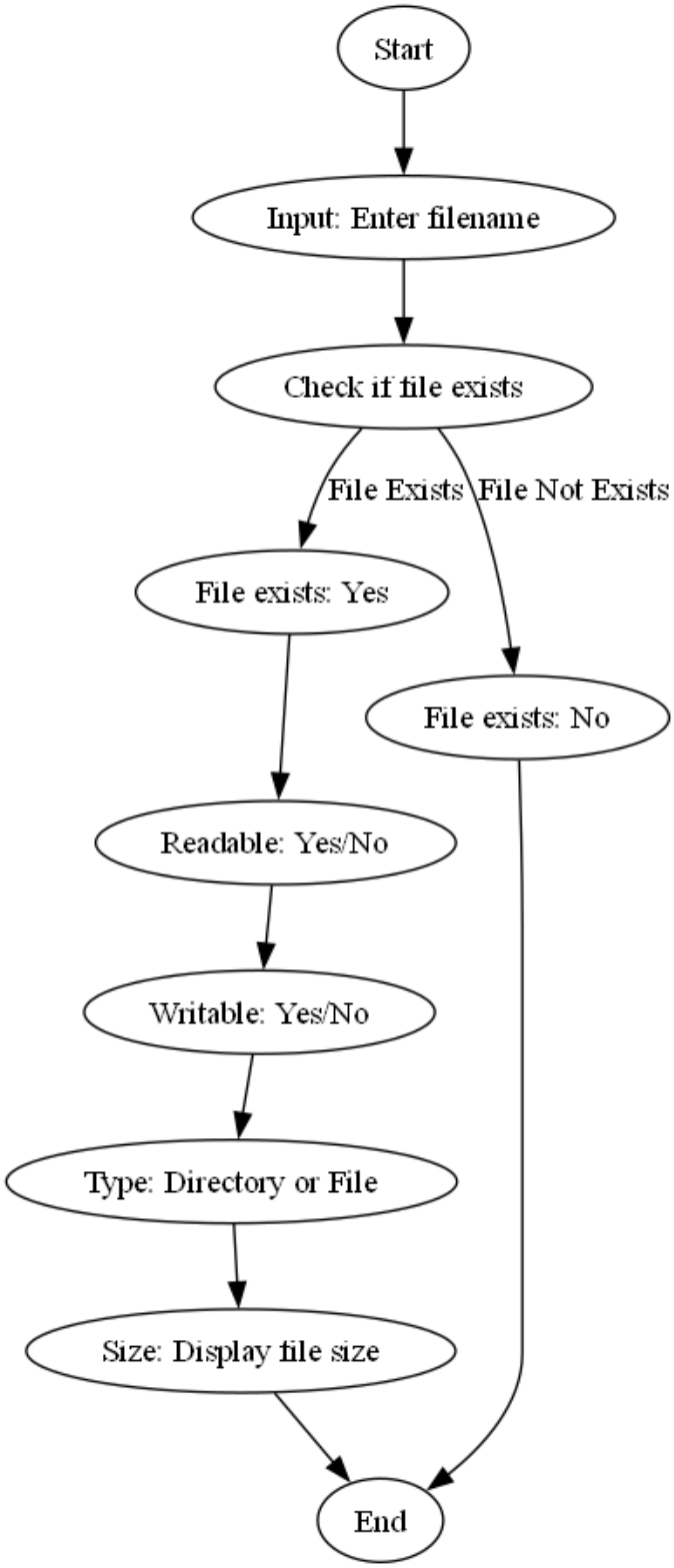
Step 4: Create a File object using the filename.

Step 5: Check if the file exists. If it exists, print "File exists: Yes." Check if it is readable and writable, then print the results. Determine if it's a directory or a file, and print the type. Finally, print the size in bytes.

Step 6: If the file does not exist, print "File exists: No."

Step 7: End the Process.

FLOW CHART:


Start
Input: Enter filename
Check if file exists
File Exists
File Not Exists
File exists: Yes
File exists: No
Readable: Yes/No
Writable: Yes/No
Type: Directory or File
Size: Display file size
End

SOURCE CODE:

```
import java.io.File;
import java.util.Scanner;
public class SimpleFileInfo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the filename: ");
        String filename = scanner.nextLine();
        File file = new File(filename);
       if (file.exists()) {
            System.out.println("File exists: Yes");
            System.out.println("Readable: " + file.canRead());
            System.out.println("Writable: " + file.canWrite());
            System.out.println("Type: " + (file.isDirectory() ? "Directory" : "File"));
            System.out.println("Size: " + file.length() + " bytes");
        } else {
            System.out.println("File exists: No");
        }
    }
}
```

CODE EXPLANATION:

import java.io.File;

- This line imports the Fileclass from the java.iopackage,which is used to create and manage file and directory pathnames in Java.

import java.util.Scanner;

- This line imports the Scannerclass from the java.util package, which is used to read input from the user.

public class SimpleFileInfo {

- This line declares the SimpleFileInfo class, which is themain class of the program.

public static void main(String[] args) {

- This line declares the mainmethod, the entry point of theprogram, where the execution begins.

Scanner scanner = new Scanner(System.in);

- Creates a Scannerobject named scannerto read input fromthe console (standard input stream System.in).

System.out.print("Enter the filename: ");

- Prints the message "Enter the filename: " to the console,prompting the user to input the name of a file.

String filename = scanner.nextLine();

- Reads the user input as a string (the filename) and stores it inthe variable filename.

File file = new File(filename);

- Creates a Fileobject named fileusing the filename providedby the user. This object will represent the file or directory located at the path specified by filename.

if (file.exists()) {

- Checks if the file (or directory) exists at the specified path using the exists() method. If the file exists, the programenters the ifblock.

System.out.println("File exists: Yes");

- If the file exists, this line prints "File exists: Yes" to theconsole.

System.out.println("Readable: " + file.canRead());

- Prints whether the file is readable by calling the canRead()method on the file object. If the file is readable, it will printtrue; otherwise, it will print false.

System.out.println("Writable: " + file.canWrite());

- Prints whether the file is writable by calling the canWrite()method on the file object. If the file is writable, it will printtrue; otherwise, it will print false.

System.out.println("Type: " + (file.isDirectory() ?"Directory" : "File"));

- Checks if the fileis a directory using the isDirectory()method. If it is a directory, the program prints "Directory"; otherwise, it prints "File".

System.out.println("Size: " + file.length() + " bytes");

- Prints the size of the file in bytes using the length() method.If the file is a directory, this will print the size of the directory (which is typically 0 bytes).

}

else {

- This line marks the beginning of the elseblock, which isexecuted if the file does not exist.

System.out.println("File exists: No");

- If the file does not exist, this line prints "File exists: No" to theconsole.

}

- Closes the if-elseblock.

}

- Closes the mainmethod.

}

- Closes the SimpleFileInfoclass.

OUTPUT:

```
Enter the filename: E:\JAVA\Day-wise-Notes\Demo.txt
File exists: Yes
Readable: true
Writable: true
Type: File
Size: 270 bytes
```

RESULT:

Thus, the program had been successfully executed.

EXPERIMENT - 13

AIM:

To create a simple Java GUI program that allows users to input text and display it with basic formatting options.

ALGORITHM:

Step 1: Start the Process.

Step 2: Initialize Frame: Create a JFrame titled "Text Formatter", set size to 300x250 pixels, set close operation to exit, and set layout to FlowLayout.

Step 3: Create Components: Create a JTextField for input (15 columns), a JButton labeled "Format Text", and a non-editable JTextArea for displaying formatted text (5 rows, 20 columns).

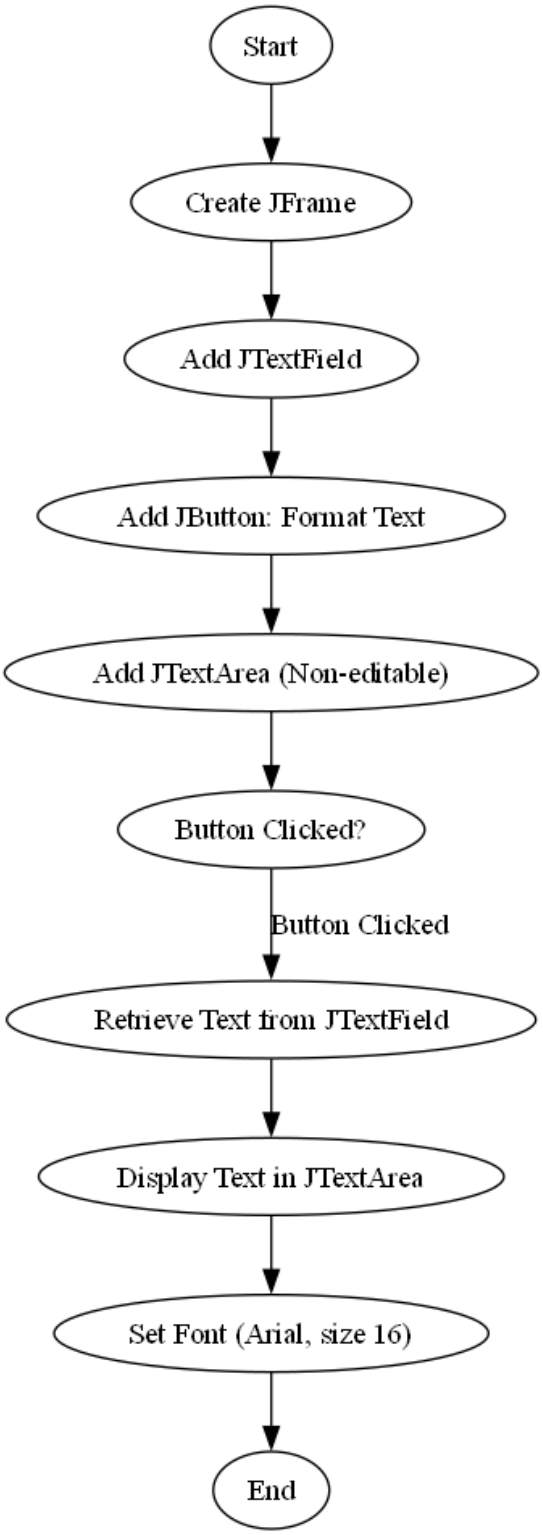
Step 4: Add Components to Frame: Add the JTextField, JButton, and JTextArea (inside a JScrollPane) to the frame.

Step 5: Add Action Listener to Button: On button click, retrieve text from JTextField, display it in JTextArea, and set font to Arial, plain style, size 16.

Step 6: Make Frame Visible: Set the frame to visible.

Step 7: End the Process.

FLOW CHART:


Start
Create JFrame
Add JTextField
Add JButton: Format Text
Add JTextArea (Non-editable)
Button Clicked?
Button Clicked
Retrieve Text from JTextField
Display Text in JTextArea
Set Font (Arial, size 16)
End

SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleTextFormatter extends JFrame {
    private JTextField textField;
    private JTextArea textArea;
    public SimpleTextFormatter() {
        // Frame setup
        setTitle("Text Formatter");
        setSize(300, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        // Text input
        textField = new JTextField(15);
        add(textField);
        // Button to apply formatting
        JButton formatButton = new JButton("Format Text");
        add(formatButton);
        // Display area for formatted text
        textArea = new JTextArea(5, 20);
        textArea.setEditable(false);
        add(new JScrollPane(textArea));
        // Button action
        formatButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                formatText();
            }
        });
```

```
        setVisible(true);
    }
    private void formatText() {
        String text = textField.getText();
        textArea.setText(text);  // Display the text
        textArea.setFont(new Font("Arial", Font.PLAIN, 16)); // Set a default font
    }
    public static void main(String[] args) {
        new SimpleTextFormatter();
    }
}
```

CODE EXPLANATION:

import javax.swing.*;

- Imports all classes from the javax.swing package. This package is used to create graphical user interfaces (GUIs) inJava, including components like JFrame, JTextField, JTextArea, and JButton.

import java.awt.*;

- Imports all classes from the java.awt package. This packageprovides graphical components and utilities for handling graphical user interfaces, including FlowLayoutand Font.

import java.awt.event.ActionEvent;

- Imports the ActionEvent class from the java.awt.event package, which is used to handle action events like buttonclicks.

import java.awt.event.ActionListener;

- Imports the ActionListenerinterface from the java.awt.event package. This interface allows objects tolisten for and handle action events like button presses.

public class SimpleTextFormatter extends JFrame {

- Declares the SimpleTextFormatter class that extends JFrame. By extending JFrame, the class inherits all functionality of a **J**Frame window and allows the creation of a graphical window.

private JTextField textField;

- Declares a JTextFieldobject called textField. This will beused to allow the user to input text.

private JTextArea textArea;

- Declares a JTextAreaobject called textArea. This will displaythe formatted text after the user clicks the "Format Text" button.

public SimpleTextFormatter() {

- Defines the constructor for the SimpleTextFormatterclass.This constructor is called when an instance of the class is created and is used to initialize the GUI components.

setTitle("Text Formatter");

- Sets the title of the **J**Frame window to "Text Formatter".

setSize(300, 250);

- Sets the size of the **J**Frame window to be 300 pixels wide and 250 pixels tall.

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- Specifies that the application will close when the user clicksthe "X" button on the window.

setLayout(new FlowLayout());

- Sets the layout manager for the **J**Frame to FlowLayout. This layout manager arranges components in a left-to-right flow, one after another.

textField = new JTextField(15);

- Initializes the textFieldobject as a new JTextFieldwith a column width of 15 characters. This field will be used to inputtext.

add(textField);

- Adds the textFieldcomponent to the **J**Frame window so that the user can see and interact with it.

JButton formatButton = new JButton("Format Text");

- Creates a new JButton with the label "Format Text". This button will trigger the text formatting action when clicked.

add(formatButton);

- Adds the formatButtonto the **J**Frame window so the user can click it to format the text.

textArea = new JTextArea(5, 20);

- Initializes the textArea object as a new JTextArea with 5 rows and 20 columns. This area will be used to display theformatted text.

textArea.setEditable(false);

- Sets the textArea to be non-editable by the user. This ensures that the user can only see the formatted text, notmodify it directly.

add(new JScrollPane(textArea));

- Wraps the textAreain a JScrollPaneto allow scrolling whenthe content exceeds the visible area. The JScrollPane is thenadded to the **J**Frame.

formatButton.addActionListener(new ActionListener() {

- Adds an ActionListenerto the formatButton. This listenerresponds to the button's action (when it is clicked).

@Override

- The @Override annotation indicates that the following methodis overriding a method from the ActionListener interface (specifically, the actionPerformedmethod).

public void actionPerformed(ActionEvent e) {

- Defines the actionPerformedmethod, which is called whenthe user clicks the "Format Text" button.

formatText();

- Calls the formatText() method, which will format the textentered in the textFieldand display it in the textArea.

});

- Closes the addActionListenermethod.

setVisible(true);

- Makes the JFrame visible on the screen. Without this, the window would not be displayed.

private void formatText() {

- Defines the formatText()method, which formats anddisplays the text input by the user.

String text = textField.getText();

- Retrieves the text entered by the user in the textFieldandstores it in the text variable.

textArea.setText(text);

- Sets the text in the textAreato the value of textretrievedfrom the textField.

textArea.setFont(new Font("Arial", Font.PLAIN, 16));

- Sets the font of the text in the textAreato Arial with a plainstyle and a size of 16.

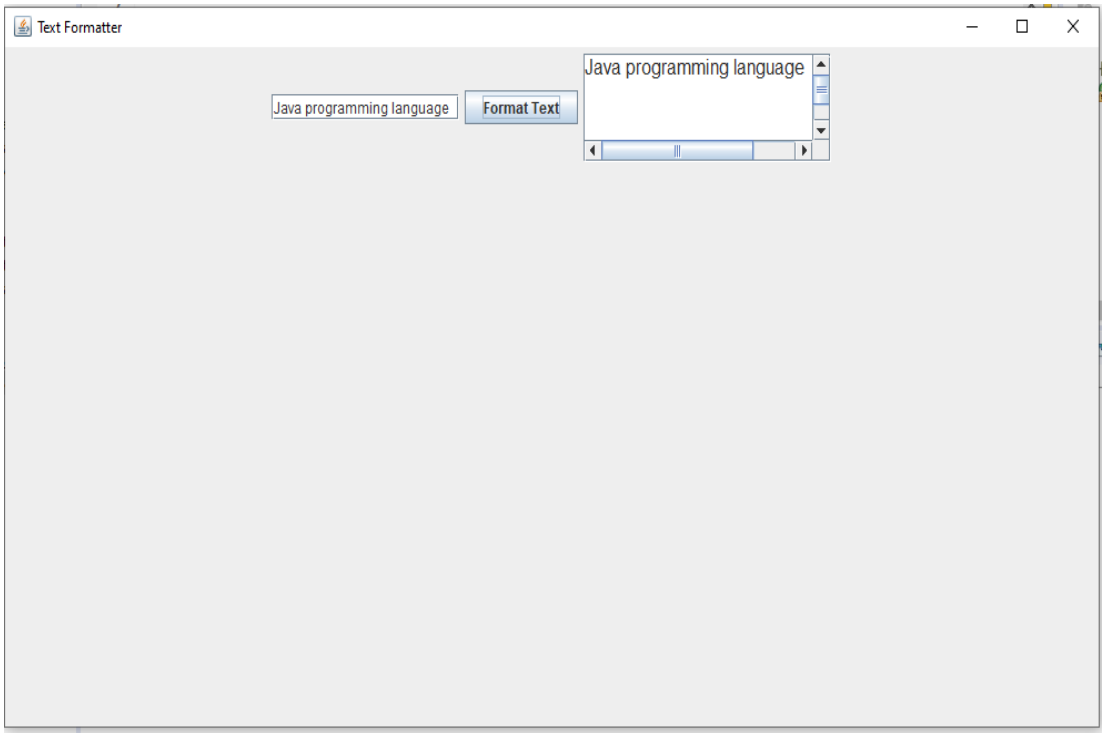
public static void main(String[] args) {

- Defines the mainmethod, which is the entry point for runningthe program.

new SimpleTextFormatter();

- Creates a new instance of the SimpleTextFormatterclass,which initializes and displays the GUI.

OUTPUT



RESULT:

Thus, the program had been successfully executed.

EXPERIMENT – 14

AIM:

To create a simple Java GUI program that handles mouse events and displays the corresponding event name in the center of the window.

ALGORITHM:

Step 1: Start the Process.

Step 2: Initialize Frame: Create a JFrame titled "Mouse Event Demo", set size to 400x300 pixels, set close operation to exit, and set layout to BorderLayout.

Step 3: Create Label: Create a JLabel for displaying mouse event names, centered in the window, and set font to Arial, size 24.

Step 4: Add Label to Frame: Add the JLabel to the center of the frame.

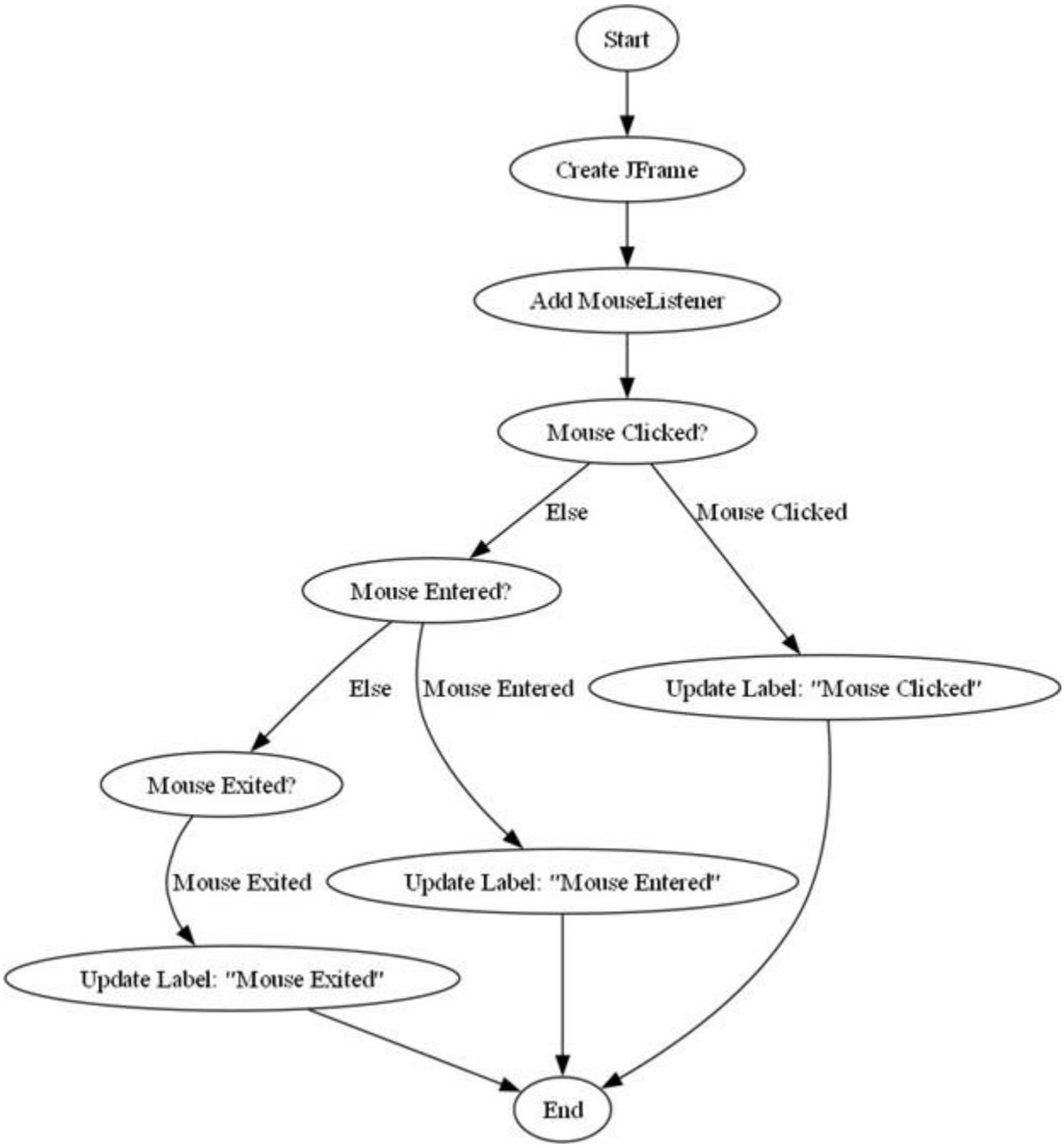
Step 5: Add Mouse Listener: Use MouseAdapter to handle events:

- On mouse click, set label text to "Mouse Clicked".
- On mouse enter set label text to "Mouse Entered".
- On mouse exit, set label text to "Mouse Exited".

Step 6: Make Frame Visible: Set the frame to visible.

Step 7: End the Process.

FLOW CHART:



SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
```

```
public class SimpleMouseEventDemo extends JFrame {
    private JLabel label;
    public SimpleMouseEventDemo() {
        setTitle("Mouse Event Demo");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        label = new JLabel("", SwingConstants.CENTER);
        label.setFont(new Font("Arial", Font.PLAIN, 24));
        add(label, BorderLayout.CENTER);
        addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                label.setText("Mouse Clicked");
            }
            public void mouseEntered(MouseEvent e) {
                label.setText("Mouse Entered");
            }
            public void mouseExited(MouseEvent e) {
                label.setText("Mouse Exited");
            }
        });

        setVisible(true);
    }
    public static void main(String[] args) {
        new SimpleMouseEventDemo();
    }
}
```

CODE EXPLANATION:

import javax.swing.*;

- Imports all classes from the javax.swing package, which provides components like JFrame, JLabel, and other GUI elements used in the program.

import java.awt.*;

- Imports all classes from the java.awt package, which includes components and utilities for handling graphical userinterfaces, such as BorderLayoutand Font.

import java.awt.event.MouseAdapter;

- Imports the MouseAdapterclass, which is a convenience classthat implements the MouseListener interface. You can override only the methods you need, such as mouseClicked, mouseEntered, and mouseExited.

import java.awt.event.MouseEvent;

- Imports the MouseEventclass, which encapsulates the detailsof a mouse event (like clicks, movements, and other mouse-related actions).

public class SimpleMouseEventDemo extends JFrame {

- Declares the SimpleMouseEventDemo class that extends JFrame. By extending JFrame, this class can create a windowwith GUI components like buttons, labels, and listeners for handling events.

private JLabel label;

- Declares a JLabel object called label, which will be used todisplay text that responds to mouse events.

public SimpleMouseEventDemo() {

- Defines the constructor for the SimpleMouseEventDemo class. This constructor is used to set up the JFrame window and add components (like the label and mouse listeners).

setTitle("Mouse Event Demo");

- Sets the title of the JFrame window to "Mouse Event Demo". This title will appear at the top of the window.

setSize(400, 300);

- Sets the size of the window to be 400 pixels wide and 300pixels tall.

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

- Specifies that the program should terminate when the userclicks the "close" button (the "X" in the window's corner).

setLayout(new BorderLayout());

- Sets the layout manager for the JFrame to BorderLayout. This layout manager divides the window into five regions (North, South, East, West, and Center). In this case, the labelwill be placed in the center.

label = new JLabel("", SwingConstants.CENTER);

- Initializes the labelas a new JLabel. The text of the label isinitially set to an empty string (""), and the SwingConstants.CENTER argument ensures that the label'stext is centered within the label.

label.setFont(new Font("Arial", Font.PLAIN, 24));

- Sets the font of the label's text to Arial, with a plain style and asize of 24 points.

add(label, BorderLayout.CENTER);

- Adds the labelto the center of the JFrame using the BorderLayout.CENTER position. This places the label in thecentral part of the window.

addMouseListener(new MouseAdapter() {

- Adds a MouseListenerto the JFrame. The MouseAdapteris a convenience class that allows you to override only the methodsyou are interested in, such as mouseClicked, mouseEntered, and mouseExited.

public void mouseClicked(MouseEvent e) {

- Defines the mouseClicked method, which is called when theuser clicks the mouse inside the JFrame.

label.setText("Mouse Clicked");

- Sets the text of the label to "Mouse Clicked" when the mouse isclicked.

public void mouseEntered(MouseEvent e) {

- Defines the mouseEntered method, which is called when themouse pointer enters the JFrame window.

label.setText("Mouse Entered");

- Sets the text of the label to "Mouse Entered" when the mousepointer enters

the window.

public void mouseExited(MouseEvent e) {

- Defines the mouseExited method, which is called when themouse pointer exits the JFrame window.

label.setText("Mouse Exited");

- Sets the text of the label to "Mouse Exited" when the mousepointer exits the window.

});

- Closes the MouseAdapter declaration and the addMouseListenermethod. The program is now listening formouse events.

setVisible(true);

- Makes the JFrame window visible on the screen. Without this, the window would remain hidden.

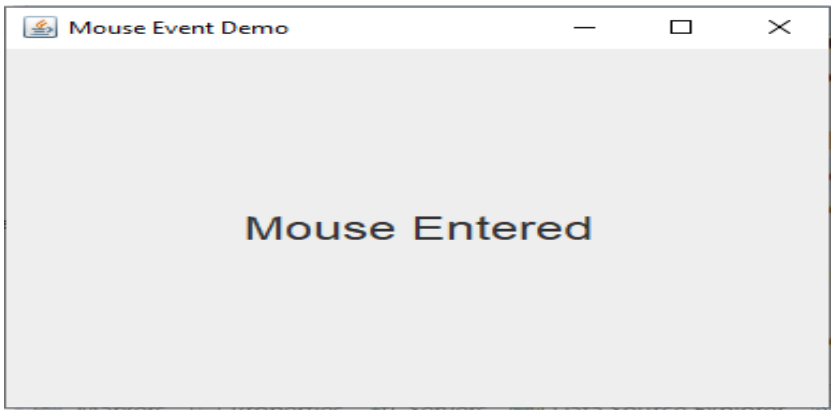
public static void main(String[] args) {

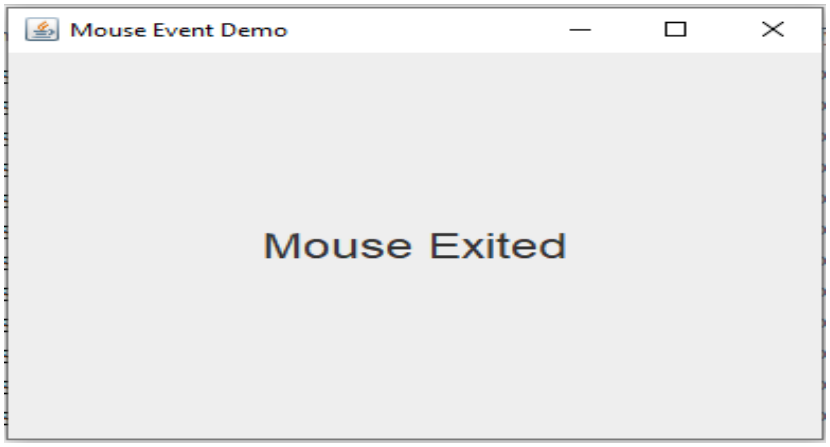
- Defines the mainmethod, which is the entry point for runningthe program.

new SimpleMouseEventDemo();

- Creates a new instance of the SimpleMouseEventDemo class,which initializes and displays the JFrame window and its components.

OUTPUT





RESULT:

Thus, the program had been successfully executed.

EXPERIMENT – 15

AIM:

To create a simple Java calculator using a GUI that performs basic arithmetic operations and handles exceptions like division by zero.

ALGORITHM:

Step 1: Start the Process.

Step 2: Create GUI: Set up JFrame, JTextField, and buttons.

Step 3: Set layout: Use BorderLayout for the frame and GridLayout for buttons.

Step 4: Create buttons: Define button labels for digits and operations in an array.

Step 5: Add buttons: Loop through the array to create and add buttons.

Step 6: Handle clicks: Attach action listeners to all buttons.

Step 7: Input number: Append digits to the display when pressed.

Step 8: Input operator: Store the first number and operator when an operator is pressed.

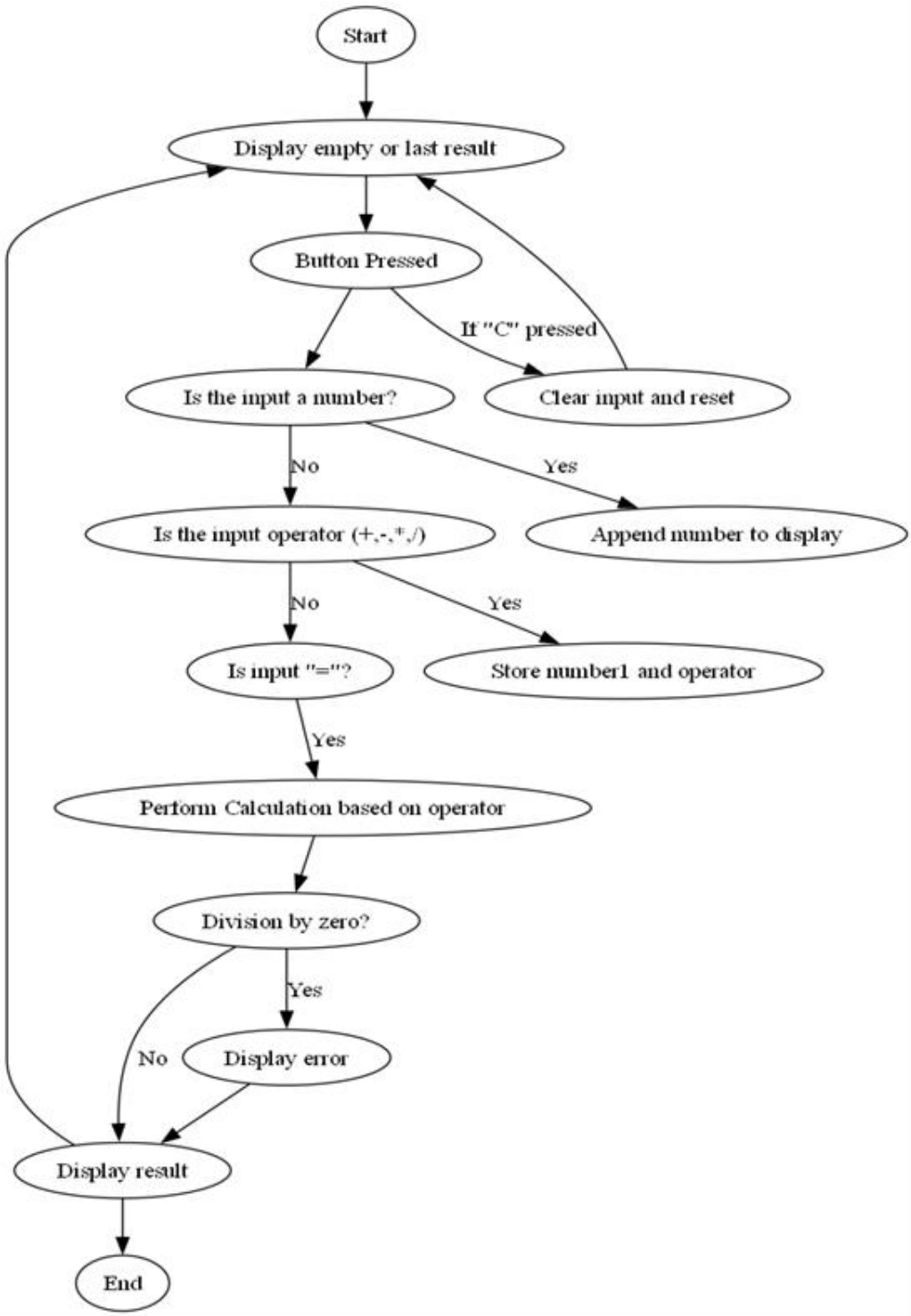
Step 9: Calculate: Operate on the second number when = is pressed.

Step 10: Show result: Display the result, and handle division by zero error.

Step 11: Clear: Reset everything when C is pressed.

Step 12: End the Process.

FLOW CHART:


Start
Display empty or last result
Button Pressed
If "C" pressed
Is the input a number?
Clear input and reset
No
Yes
Is the input operator (+,-,*,/)
Append number to display
No
Yes
Is input "="?
Store number1 and operator
Yes
Perform Calculation based on operator
Division by zero?
Yes
No
Display error
Display result
End

SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleCalculator extends JFrame implements ActionListener {
    // Components
    JTextField display;
    double num1 = 0, num2 = 0, result = 0;
    char operator;
    // Constructor
    public SimpleCalculator() {
        // Frame settings
        setTitle("Simple Calculator");
        setSize(300, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        // Create display field
        display = new JTextField();
        display.setEditable(false);
        add(display, BorderLayout.NORTH);
        // Create panel for buttons
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 4));
        // Button labels
        String[] buttons = {
            "7", "8", "9", "+",
            "4", "5", "6", "-",
            "1", "2", "3", "*",
            "C", "0", "=", "/"
        };
```

```
    // Add buttons to panel
    for (String text : buttons) {
        JButton button = new JButton(text);
        button.addActionListener(this);
        panel.add(button);
    }
    add(panel);
    // Set frame visibility
    setVisible(true);
}
// Action handling
@Override
public void actionPerformed(ActionEvent e) {
    String command = e.getActionCommand();
    if (command.charAt(0) >= '0' && command.charAt(0) <= '9') {
        // If a number is pressed, append to display
        display.setText(display.getText() + command);
    } else if (command.equals("C")) {
        // Clear the display
        display.setText("");
        num1 = num2 = result = 0;
    } else if (command.equals("=")) {
        // Perform calculation
        num2 = Double.parseDouble(display.getText());
        switch (operator) {
            case '+': result = num1 + num2; break;
            case '-': result = num1 - num2; break;
            case '*': result = num1 * num2; break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
```

```
                } else {
                    display.setText("Error");
                    return;
                }
                break;
            }
            display.setText(String.valueOf(result));
        } else {
            // Store the first number and the operator
            num1 = Double.parseDouble(display.getText());
            operator = command.charAt(0);
            display.setText("");
        }
    }
    // Main method to run the calculator
    public static void main(String[] args) {
        new SimpleCalculator();
    }
}
```

CODE EXPLANATION:

import javax.swing.*;

Imports the necessary classes from the javax.swing package,which is used for building graphical user interface (GUI) components such as JFrame, JButton, and JTextField.

import java.awt.*;

Imports the java.awt package, which provides classes for creating and managing user interfaces, layout managers like BorderLayoutand GridLayout, and other UI components likeFontand Color.

import java.awt.event.ActionEvent;

Imports the ActionEventclass, which represents an actionevent, such as when a button is clicked

import java.awt.event.ActionListener;

Imports the ActionListener interface, which is used to handle action events. The

program implements this interfaceto handle button clicks.

public class SimpleCalculator extends JFrame implementsActionListener {

Declares the SimpleCalculatorclass, which extends JFrame(for creating a window) and implements the ActionListenerinterface (for handling button clicks).

JTextField display;

Declares a JTextFieldnamed displayto show the userinput and the result of the calculations.

double num1 = 0, num2 = 0, result = 0;

Declares and initializes variables num1, num2, and result to store the two numbers and the result of the calculation. Theseare initialized to 0.

char operator;

Declares a charvariable operatorto store the operator (+, -, *, or /) for the calculation.

public SimpleCalculator() {

Defines the constructor for the SimpleCalculatorclass. Thisconstructor sets up the frame, display, and buttons for the calculator.

setTitle("Simple Calculator");

Sets the title of the JFrame window to "Simple Calculator" (this will appear at the top of the window).

setSize(300, 400);

Sets the size of the JFrame window to 300 pixels wide and 400 pixels tall.

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Ensures that when the user closes the JFrame window, the program terminates by calling System.exit(0).

setLayout(new BorderLayout());

Sets the layout manager of the JFrame to BorderLayout. This layout divides the window into five regions: North, South, East, West, and Center. The calculator display will be placed inthe North, and the buttons will be arranged in the Center.

display = new JTextField();

Initializes the displayas a new JTextField, where the usercan see the numbers and results. This field will be set to read-only.

display.setEditable(false);

Sets the display field to non-editable. This means the usercannot directly type in the field, only through the buttons.

add(display, BorderLayout.NORTH);

Adds the displayfield to the North region of the BorderLayout. This places the display at the top of theJFrame.

JPanel panel = new JPanel();

Creates a new JPanel object called panel, which will hold thecalculator buttons.
panel.setLayout(new GridLayout(4, 4));

Sets the layout manager of the panelto GridLayout(4, 4).This arranges the buttons in a 4x4 grid (4 rows and 4 columns).
String[] buttons = { ... };

Declares an array of strings containing the labels for the buttons. This array includes the digits 0-9, operators (+, -, *, /), the "C" (clear) button, and the "=" (equals) button.

for (String text : buttons) {

Loops through the buttonsarray. For each button label(stored in the variable text), a new JButtonis created.
JButton button = new JButton(text);

Creates a new JButtonwith the label text.
button.addActionListener(this);

Adds the current instance of SimpleCalculator (this) as anActionListener to the button. This means that the actionPerformed()method will be called when the button isclicked.
panel.add(button);

Adds the created button to the panel.
add(panel);

Adds the panel(which contains the buttons) to the **J**Frame. The buttons will be displayed in the center of the window.
setVisible(true);

Makes the **J**Frame window visible. Without this, the window would not appear on the screen.
@Override public void actionPerformed(ActionEvent e) {

Overrides the actionPerformedmethod from the ActionListenerinterface. This method is triggered when abutton is clicked.
String command = e.getActionCommand();

Retrieves the command (label) of the button that was clicked.For example, if the "7" button is clicked, the command will be"7".
if (command.charAt(0) >= '0' && command.charAt(0) <='9') {

Check if the clicked button is a digit (0-9) by comparing the first character (command.charAt(0)) to see if it falls within therange of '0' to '9'.
display.setText(display.getText() + command);

Appends the clicked digit to the current text in the displayfield. For example, if "7" is

clicked, the display will show "7",and if "1" is clicked next, the display will show "71".

else if (command.equals("C")) {

Check if the "C" button was clicked. If it is clicked, it clearsthe display and resets the numbers and result.

display.setText("");

Clears the display field.

num1 = num2 = result = 0;

Resets all the variables (num1, num2, and result) to 0.

else if (command.equals("=")) {

Checks if the "=" button was clicked, which triggers thecalculation.

num2 = Double.parseDouble(display.getText());

Converts the text in the display field to a double value andstores it in num2.

switch (operator) {

Uses a switchstatement to perform the calculation based onthe selected operator.

case '+': result = num1 + num2; break;

If the operator is '+', it adds num1and num2and stores theresult.

case '-': result = num1 - num2; break;

If the operator is '-', it subtracts num2from num1and storesthe result.

case '*': result = num1 * num2; break;

If the operator is '*', it multiplies num1and num2and stores theresult.

case '/':

If the operator is '/', it performs division. If num2is not 0, itdivides num1 by num2 and stores the result. If num2 is 0, itdisplays an "Error" message in the display field.

display.setText(String.valueOf(result));

Converts the resultto a string and displays it in the display field.

else {

If the clicked button is an operator (like +, -, *, or /), the current number in the display is stored as num1, and theoperator is saved.

num1 = Double.parseDouble(display.getText());

Converts the current value in the display to a double andstores it in num1.

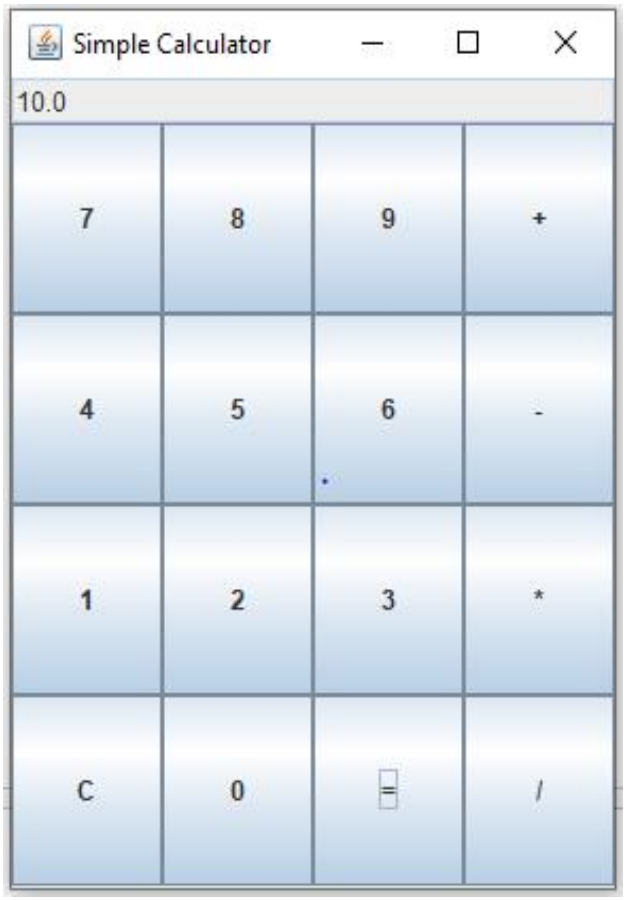
operator = command.charAt(0);

Sets the operatorvariable to the first character of thecommand (which is the operator symbol).

display.setText("");

Clears

OUTPUT:



RESULT:

Thus, the program had been successfully executed.

EXPERIMENT – 16

AIM:

To create a simple Java GUI application that simulates a traffic light using radio buttons to display corresponding messages and colors for "Stop," "Ready," and "Go."

ALGORITHM:

Step 1: Start.

Step 2: Initialize Frame: Create a JFrame titled "Traffic Light Simulator", set size to 300x200 pixels, set close operation to exit, and set layout to FlowLayout.

Step 3: Create Message Label: Create a JLabel for traffic light messages, set font to Arial, bold, size 24, and add it to the frame (initially empty).

Step 4: Create Radio Buttons: Create JRadioButton objects for "Red", "Yellow", and "Green".

Step 5: Group Radio Buttons: Use a ButtonGroup to allow only one radio button to be selected at a time.

Step 6: Add Buttons to Frame: Add the radio buttons to the frame.

Step 7: Add Action Listeners: For each radio button:

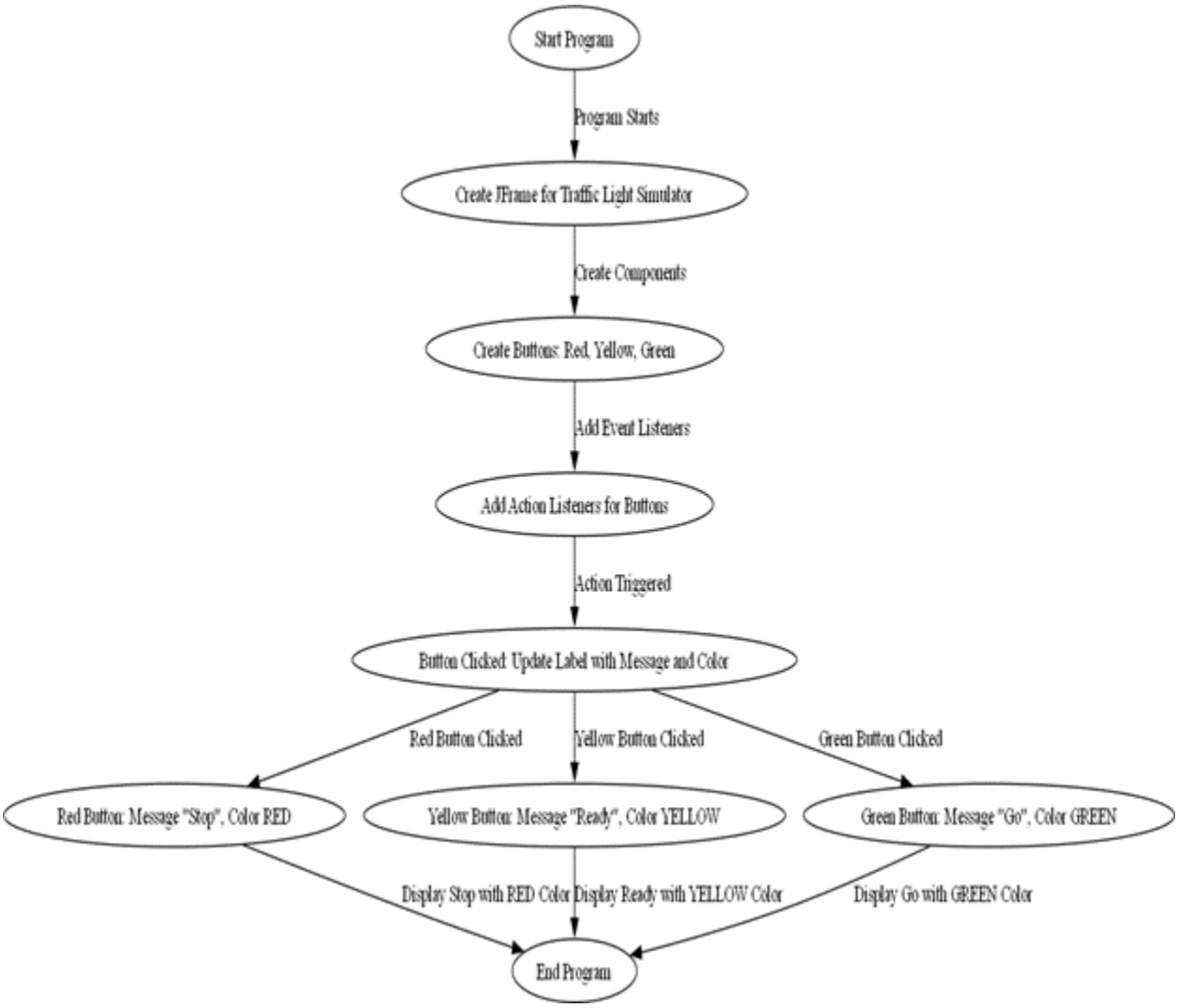
- Red: Call updateMessage("Stop", Color.RED).
- Yellow: Call updateMessage("Ready", Color.YELLOW).
- Green: Call updateMessage("Go", Color.GREEN).

Step 8: Define updateMessage Method: Set the label text and color based on input parameters.

Step 9: Make Frame Visible: Set the frame to visible.

Step 10: End the Process.

FLOW CHART:



SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleTrafficLightSimulator extends JFrame {
    private JLabel messageLabel;
    public SimpleTrafficLightSimulator() {
        setTitle("Traffic Light Simulator");
```

```
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        messageLabel = new JLabel(""); // Initial empty message
        messageLabel.setFont(new Font("Arial", Font.BOLD, 24));
        add(messageLabel);
        // Create radio buttons for traffic lights
        JRadioButton redButton = new JRadioButton("Red");
        JRadioButton yellowButton = new JRadioButton("Yellow");
        JRadioButton greenButton = new JRadioButton("Green");
        // Group buttons so only one can be selected
        ButtonGroup group = new ButtonGroup();
        group.add(redButton);
        group.add(yellowButton);
        group.add(greenButton);
        add(redButton);
        add(yellowButton);
        add(greenButton);
        // Action listener for buttons
        redButton.addActionListener(e -> updateMessage("Stop", Color.RED));
        yellowButton.addActionListener(e -> updateMessage("Ready", Color.YELLOW));
        greenButton.addActionListener(e -> updateMessage("Go", Color.GREEN));
        setVisible(true);
    }
    private void updateMessage(String message, Color color) {
        messageLabel.setText(message);
        messageLabel.setForeground(color);
    }
    public static void main(String[] args) {
        new SimpleTrafficLightSimulator();
    }
```

}

CODE EXPLANATION:

Importing Libraries:
The code begins by importing necessary Java libraries for creatinggraphical user interfaces (GUI) using Swing, and for handling events:

- javax.swing.*for GUI components like buttons and labels.
- java.awt.*for layouts and colors.
- java.awt.event.ActionEvent and java.awt.event.ActionListenerto handle user actions(like button clicks).

Class Definition:
The SimpleTrafficLightSimulator class extends JFrame, meaning it represents a window where the GUI will be displayed.

Constructor (Setting up the GUI):
The constructor SimpleTrafficLightSimulator()is where theGUI components and layout are defined:

- setTitle("Traffic Light Simulator"): This sets the titleof the window to "Traffic Light Simulator".
- setSize(300, 200): Specifies the window size (300 pixelswide and 200 pixels tall).
- setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE): Thisensures that the program will close when the window is closed.
- setLayout(new FlowLayout()): The layout manager is set toFlowLayout, meaning components will be added sequentially from left to right.

Message Label:

- messageLabel = new JLabel("");: Creates a label that willdisplay the traffic light message (initially empty).
- messageLabel.setFont(new Font("Arial", Font.BOLD,24)): Sets the font style and size for the label.
- add(messageLabel): Adds the label to the window.

Radio Buttons for Traffic Lights:

- Three JRadioButtoncomponents (redButton, yellowButton,greenButton) are created for the red, yellow, and green lights.

- ButtonGroup group = new ButtonGroup(): This groups theradio buttons, ensuring that only one button can be selected at a time.
- group.add(redButton); group.add(yellowButton); group.add(greenButton): The radio buttons are added to thegroup.

Adding Buttons to the Window:

- add(redButton); add(yellowButton); add(greenButton): These buttons are added to the window.

Action Listeners for Buttons:

- Each radio button has an action listener attached to it. The action listener will update the message displayed on the labelwhen a button is clicked:
- redButton.addActionListener(e -> updateMessage("Stop", Color.RED)): When the red buttonis clicked, it updates the message to "Stop" and changes the text color to red.
- yellowButton.addActionListener(e -> updateMessage("Ready", Color.YELLOW)): When the yellowbutton is clicked, it updates the message to "Ready" and changes the text color to yellow.
- greenButton.addActionListener(e -> updateMessage("Go", Color.GREEN)): When the green button is clicked, it updates the message to "Go" and changes the text color to green.

Method for Updating the Message:

- private void updateMessage(String message, Colorcolor): This method updates the message shown on the messageLabeland sets its color.
- messageLabel.setText(message): Changes the text on the label.
- messageLabel.setForeground(color): Changes the color ofthe text to match the traffic light color.

Main Method:

- public static void main(String[] args): This is theentry point for the program.
- new SimpleTrafficLightSimulator(): Creates an instance of the SimpleTrafficLightSimulatorclass, initializing the GUI.

OUTPUT:

RESULT:

Thus, the program had been successfully executed.