11.a)Write a Java program to implement dynamic dispatching for calculating volume of different objects

```
// Define an interface for objects that have a volume.
interface VolumeCalculable {
    double calculateVolume();
}

// Create classes for different objects that implement the VolumeCalculable
interface.
class Sphere implements VolumeCalculable {
    private double radius;

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

    @Override
    public double calculateVolume() {
        return (4.0 / 3.0) * Math.PI * Math.pow(radius, 3);
    }
}

class Cube implements VolumeCalculable {
    private double sideLength;

    public Cube(double sideLength) {
        this.sideLength = sideLength;
    }

    @Override
    public double calculateVolume() {
        return Math.pow(sideLength, 3);
    }
}

class Cylinder implements VolumeCalculable {
    private double radius;
    private double height;

    public Cylinder(double radius, double height) {
        this.radius = radius;
        this.height = height;
    }

    @Override
    public double calculateVolume() {
        return Math.PI * Math.pow(radius, 2) * height;
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        // Create objects of different shapes
        VolumeCalculable sphere = new Sphere(5.0);
        VolumeCalculable cube = new Cube(3.0);
        VolumeCalculable cylinder = new Cylinder(4.0, 6.0);

        // Calculate and display the volumes using dynamic dispatch
        System.out.println("Volume of the Sphere: " + sphere.calculateVolume());
        System.out.println("Volume of the Cube: " + cube.calculateVolume());
        System.out.println("Volume of the Cylinder: " + cylinder.calculateVolume());
    }
}
```

ALGORITHM:

The program I provided demonstrates dynamic dispatching in Java to calculate the volume of different objects, such as spheres, cubes, and cylinders. Here's a step-by-step explanation of the algorithm and how dynamic dispatching works in this program:

1. **Interface Definition (`VolumeCalculable`)**:
   - We define an interface called `VolumeCalculable` that declares a single method `calculateVolume()`. This interface serves as a contract that any class implementing it must provide an implementation for the `calculateVolume` method.

2. **Object Classes (`Sphere`, `Cube`, and `Cylinder`)**:
   - Three classes, `Sphere`, `Cube`, and `Cylinder`, are created. Each of these classes implements the `VolumeCalculable` interface, which means they must provide an implementation for the `calculateVolume` method.
   - Each class stores the necessary data to calculate the volume of its corresponding shape. For example, the `Sphere` class stores the radius, the `Cube` class stores the side length, and the `Cylinder` class stores the radius and height.

3. **`calculateVolume` Method Implementation**:
   - Each of the object classes provides its own implementation of the `calculateVolume` method. These implementations use the specific formulas for calculating the volume of spheres, cubes, and cylinders based on their respective attributes (radius, side length, and height).

4. **Main Class (`Main`)**:
   - In the `Main` class, we create objects of the different shapes, namely a sphere, a cube, and a cylinder. For example:
     - `Sphere sphere = new Sphere(5.0);` creates a sphere object with a radius of 5.0.
     - `Cube cube = new Cube(3.0);` creates a cube object with a side length of 3.0.
     - `Cylinder cylinder = new Cylinder(4.0, 6.0);` creates a cylinder object with a radius of 4.0 and a height of 6.0.

5. **Dynamic Dispatching**:
   - When we call the `calculateVolume` method on these objects, Java performs dynamic dispatching. This means that at runtime, Java determines which version of the `calculateVolume` method to call based on the actual type of the object.
   - For example, when we call `sphere.calculateVolume()`, Java knows to invoke the `calculateVolume` method defined in the `Sphere` class to calculate the volume of a sphere.

6. **Output**:
   - The program prints the calculated volumes for each object type.

In summary, the program uses an interface to define a common method signature that multiple classes must implement. These classes provide their own implementations for calculating the volume of specific shapes. Dynamic dispatching allows Java to select the appropriate method at runtime based on the actual object type, which enables the program to calculate volumes for different objects using a unified interface.

OUTPUT:

```
Volume of the Sphere: 523.5987755982989
Volume of the Cube: 27.0
Volume of the Cylinder: 301.59289474462014
```

```
====================================================================================
=================================================================================
```

11.b)Write a Java program to print the Prime numbers between 0 and 1000

```java
public class PrimeNumbers {
    public static void main(String[] args) {
        // Loop through numbers from 0 to 1000
        for (int i = 0; i <= 1000; i++) {
            // Check if the current number is prime
            if (isPrime(i)) {
                System.out.print(i + " ");
            }
        }
    }

    // Function to check if a number is prime
    private static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        // Check for factors up to the square root of the number
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false; // If a factor is found, the number is not prime
            }
        }
        return true; // If no factors are found, the number is prime
    }
}
```

ALGORITHM:

Certainly! The algorithm for the provided Java program to print prime numbers between 0 and 1000 is as follows:

1. **Main Program (`PrimeNumbers`):**
   - Iterate through numbers from 0 to 1000 using a `for` loop.
   - For each number, check whether it is prime using the `isPrime` function.
   - If the number is prime, print it.

2. **isPrime Function:**
   - The `isPrime` function takes an integer `num` as input and returns a boolean value indicating whether the number is prime or not.
   - If `num` is less than or equal to 1, return `false` (1 is not a prime number, and 0 and 1 are not considered prime).
   - Iterate from 2 to the square root of `num` (inclusive) in a `for` loop.
     - If `num` is divisible evenly by any number in this range, return `false` (indicating that `num` has factors other than 1 and itself).
   - If no factors are found, return `true` (indicating that `num` is prime).

This algorithm follows the basic primality test by checking for factors up to the square root of the number. It prints the prime numbers between 0 and 1000 by leveraging the `isPrime` function to determine whether each number in the range is prime or not.

OUTPUT:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107
109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227
229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467
479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613
617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751
757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887
907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

```
=====================================================================================
=====================================================================================
==
```

12.a)Write a Java program to create own package with Matrix class to perform addition, subtraction and transpose and import the class.

```
// Matrix.java (inside the matrixoperations package)
package matrixoperations;

public class Matrix {
    private int rows;
    private int columns;
    private int[][] data;

    public Matrix(int rows, int columns, int[][] data) {
        this.rows = rows;
        this.columns = columns;
        this.data = data;
    }

    public Matrix add(Matrix other) {
        if (this.rows != other.rows || this.columns != other.columns) {
            throw new IllegalArgumentException("Matrices must have the same
dimensions for addition.");
        }

        int[][] resultData = new int[rows][columns];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                resultData[i][j] = this.data[i][j] + other.data[i][j];
            }
        }

        return new Matrix(rows, columns, resultData);
    }

    public Matrix subtract(Matrix other) {
        if (this.rows != other.rows || this.columns != other.columns) {
            throw new IllegalArgumentException("Matrices must have the same
dimensions for subtraction.");
        }

        int[][] resultData = new int[rows][columns];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                resultData[i][j] = this.data[i][j] - other.data[i][j];
            }
        }

        return new Matrix(rows, columns, resultData);
    }

    public Matrix transpose() {
        int[][] resultData = new int[columns][rows];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                resultData[j][i] = this.data[i][j];
            }
        }

        return new Matrix(columns, rows, resultData);
    }

    public void print() {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.print(data[i][j] + " ");
            }
            System.out.println();
        }
    }
}


// MainProgram.java
import matrixoperations.Matrix;

public class MainProgram {
    public static void main(String[] args) {
        int[][] data1 = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
        int[][] data2 = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

        Matrix matrix1 = new Matrix(3, 3, data1);
        Matrix matrix2 = new Matrix(3, 3, data2);

        System.out.println("Matrix 1:");
        matrix1.print();

        System.out.println("\nMatrix 2:");
        matrix2.print();

        Matrix sumMatrix = matrix1.add(matrix2);
        System.out.println("\nSum of matrices:");
        sumMatrix.print();

        Matrix diffMatrix = matrix1.subtract(matrix2);
        System.out.println("\nDifference of matrices:");
        diffMatrix.print();

        Matrix transposeMatrix1 = matrix1.transpose();
        System.out.println("\nTranspose of matrix 1:");
        transposeMatrix1.print();
    }
}
```

Alogorithm:

Matrix.java:
Matrix Class Definition:

Define a class named Matrix inside the package matrixoperations.

The class has private instance variables: rows, columns, and a 2D array data to store the matrix elements.
Constructor:

Define a constructor that initializes the matrix with the specified number of rows, columns, and data.
Matrix Operations:

add(Matrix other): Adds two matrices and returns a new matrix.
subtract(Matrix other): Subtracts one matrix from another and returns a new matrix.
transpose(): Transposes the matrix and returns a new matrix.
Printing Method:

print(): Prints the matrix elements.
MainProgram.java:
Import Matrix Class:

Import the Matrix class from the matrixoperations package.
Matrix Initialization:

Create two matrices (matrix1 and matrix2) with predefined data.
Matrix Printing:

Print the original matrices (matrix1 and matrix2).
Matrix Operations:

Perform matrix addition, subtraction, and transpose using the methods from the Matrix class.
Print the results of these operations.

STRUCTURE:

```
project/
|-- matrixoperations/
|   |-- Matrix.java
|
|-- MainProgram.java
```

Compilation:

```
javac matrixoperations/Matrix.java MainProgram.java
```

Run:
```
java MainProgram
```

OUTPUT:

```
Matrix 1:
1 2 3
```

```
4 5 6
7 8 9

Matrix 2:
9 8 7
6 5 4
3 2 1

Sum of matrices:
10 10 10
10 10 10
10 10 10

Difference of matrices:
-8 -6 -4
-2 0 2
4 6 8

Transpose of matrix 1:
1 4 7
2 5 8
3 6 9
```

==
==

13)Write a Java program to create own package with Array class to find maximum, minimum, sum and average and import the class

```
package custompackage;

public class Array {
    public static int findMaximum(int[] array) {
        int max = array[0];
        for (int value : array) {
            if (value > max) {
                max = value;
            }
        }
        return max;
    }

    public static int findMinimum(int[] array) {
        int min = array[0];
        for (int value : array) {
            if (value < min) {
                min = value;
            }
        }
        return min;
    }

    public static int findSum(int[] array) {
        int sum = 0;
        for (int value : array) {
            sum += value;
        }
        return sum;
    }

    public static double findAverage(int[] array) {
        int sum = findSum(array);
        return (double) sum / array.length;
    }
}


import custompackage.Array;

public class MainProgram {
    public static void main(String[] args) {
        int[] numbers = {5, 10, 3, 8, 15, 7};

        // Use the Array class to find maximum, minimum, sum, and average
        int max = Array.findMaximum(numbers);
        int min = Array.findMinimum(numbers);
        int sum = Array.findSum(numbers);
        double average = Array.findAverage(numbers);

        // Print the results
        System.out.println("Array: [5, 10, 3, 8, 15, 7]");
        System.out.println("Maximum: " + max);
        System.out.println("Minimum: " + min);
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);
    }
}
```

Algorithm:

Array.java:
Class Definition:

Define a class named Array in the package custompackage.
Maximum Value Method (findMaximum):

Define a static method findMaximum that takes an array of integers as input.
Initialize a variable max with the first element of the array.
Iterate through the array, updating max if a larger element is found.

Return the maximum value.
Minimum Value Method (findMinimum):

Define a static method findMinimum that takes an array of integers as input.
Initialize a variable min with the first element of the array.
Iterate through the array, updating min if a smaller element is found.
Return the minimum value.
Sum Method (findSum):

Define a static method findSum that takes an array of integers as input.
Initialize a variable sum to zero.
Iterate through the array, adding each element to sum.
Return the sum.
Average Method (findAverage):

Define a static method findAverage that takes an array of integers as input.
Use the findSum method to calculate the sum of the array.
Return the average by dividing the sum by the length of the array.
MainProgram.java:
Import Class:

Import the Array class from the custompackage package.
Main Method (main):

Define the main method.
Create an array of integers.
Maximum, Minimum, Sum, and Average:

Use the Array class methods to find the maximum, minimum, sum, and average of the array.
Print Results:

Print the original array.
Print the calculated maximum, minimum, sum, and average.

Output:

```
Array: [5, 10, 3, 8, 15, 7]
Maximum: 15
Minimum: 3
Sum: 48
Average: 8.0
```

==

14.(a) Write a Java program to store information into file and read information from file using ByteStream.

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

```
public class CopyFile {

public static void main(String[] args) throws IOException {

FileInputStream in = null;

FileOutputStream out = null;

try {
 int c;

in = new FileInputStream("input.txt");

out = new FileOutputStream("output.txt");

while ((c = in.read()) != -1) {
out.write(c);
System.out.println((char)c);

}
System.out.println("success");

}
catch(Exception e){
System.out.println(e);
}
finally {

if (in != null) { in.close(); }

if (out!= null) { out.close();}

}
}
}
```

Algorithm:
Certainly! Here's the algorithm for the provided Java program (`CopyFile.java`), which reads from an input file (`input.txt`) and copies its content to an output file (`output.txt`):

1. **Import Required Libraries:**
   - Import the necessary classes for handling input/output operations (`java.io.*`).

2. **Main Method (`main`):**
   - The program's entry point is the `main` method.

3. **File Input and Output Streams:**

   - Declare `FileInputStream` and `FileOutputStream` objects (`in` and `out`).
   - Initialize them to `null`.

4. **Try Block:**
   - Use a try block to handle potential exceptions.
   - Create a `FileInputStream` object (`in`) to read from the input file ("input.txt").
   - Create a `FileOutputStream` object (`out`) to write to the output file ("output.txt").

5. **Read and Write Content:**
   - Use a `while` loop to read each byte from the input file.
   - Write the byte to the output file using `out.write(c)`.
   - Print each character to the console using `System.out.println((char)c)`.

6. **Print Success Message:**
   - Print "success" after successfully copying the file.

7. **Catch Block:**
   - Catch any exceptions that might occur during file operations and print the exception message.

8. **Finally Block:**
   - Use a `finally` block to ensure that the input and output streams are closed, regardless of whether an exception occurred.
   - Close the `FileInputStream` (`in`) and `FileOutputStream` (`out`) objects.

```
OUTPUT
input.txt
i like java program

output.txt
i like java program

CopyFile
i like java program
success
====================================================================================
=======================================================================
```
14.b)Write a Java program to implement simple Applet to draw human face

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

public class FaceApplet extends Applet {

    public void paint(Graphics g) {
        // Draw the face outline
        g.drawOval(100, 100, 200, 200);
```

```java
        // Draw the eyes
        g.fillOval(150, 150, 20, 20); // Left eye
        g.fillOval(230, 150, 20, 20); // Right eye

        // Draw the mouth (smile)
        g.drawArc(150, 200, 100, 50, 180, 180);
    }
}
```

```html
<html>
<body>
    <applet code="FaceApplet.class" width="400" height="400"></applet>
</body>
</html>
```

Algorithm:

import java.applet.Applet;: Import the Applet class from the java.applet package.

import java.awt.Graphics;: Import the Graphics class from the java.awt package for drawing.

public class FaceApplet extends Applet {: Define a class named FaceApplet that extends the Applet class.

public void paint(Graphics g) {: Override the paint method, which is called when the applet is drawn.

g.drawOval(100, 100, 200, 200);: Draw the face outline using drawOval with parameters (x, y, width, height).

g.fillOval(150, 150, 20, 20);: Draw the left eye using fillOval with parameters (x, y, width, height).

g.fillOval(230, 150, 20, 20);: Draw the right eye.

g.drawArc(150, 200, 100, 50, 180, 180);: Draw a smile using drawArc with parameters (x, y, width, height, startAngle, arcAngle).

Compile:

```
javac FaceApplet.java
javac FaceApplet.html
appletviewer FaceApplet.html
```

Output:

smile emojii

==
==

15.Write a Java program to demonstrate different methods in String and StringBuffer classes.

```
public class StringDemo {

    public static void main(String[] args) {
        // String methods
        stringMethodsDemo();

        // StringBuffer methods
        stringBufferMethodsDemo();
    }

    // Demonstrate String methods
    private static void stringMethodsDemo() {
        // Creating strings
        String str1 = "Hello";
        String str2 = "World";

        // Concatenation
        String concatenated = str1.concat(" " + str2);
        System.out.println("Concatenated String: " + concatenated);

        // Length
        int length = concatenated.length();
        System.out.println("Length of String: " + length);

        // Substring
        String substring = concatenated.substring(6, 11);
        System.out.println("Substring: " + substring);

        // Uppercase and lowercase
        String uppercase = concatenated.toUpperCase();
        String lowercase = concatenated.toLowerCase();
        System.out.println("Uppercase: " + uppercase);
        System.out.println("Lowercase: " + lowercase);

        // Equality
        boolean isEqual = str1.equals(str2);
        System.out.println("Equality: " + isEqual);

        // Index of
        int index = concatenated.indexOf("World");
        System.out.println("Index of 'World': " + index);

        // Replace
        String replaced = concatenated.replace("Hello", "Hola");
        System.out.println("Replaced String: " + replaced);
    }
    // Demonstrate StringBuffer methods
    private static void stringBufferMethodsDemo() {
        // Creating StringBuffer
        StringBuffer stringBuffer = new StringBuffer("Java");

        // Append
        stringBuffer.append(" is").append(" awesome");
        System.out.println("Appended StringBuffer: " + stringBuffer);

        // Insert
        stringBuffer.insert(5, " programming");
        System.out.println("Inserted StringBuffer: " + stringBuffer);

        // Delete
        stringBuffer.delete(5, 16);
        System.out.println("Deleted StringBuffer: " + stringBuffer);

        // Reverse
        stringBuffer.reverse();
        System.out.println("Reversed StringBuffer: " + stringBuffer);
    }
}
```

Algorithm:

Certainly! Below is the algorithm for the Java program that demonstrates different methods in the `String` and `StringBuffer` classes:

### `StringDemo` Class:

#### Main Method:

1. **Start:**
   - Define the `StringDemo` class.

2. **Main Method (`main`):**
   - Start the `main` method.
   - Call `stringMethodsDemo` method to demonstrate `String` class methods.
   - Call `stringBufferMethodsDemo` method to demonstrate `StringBuffer` class methods.
   - End the `main` method.

#### `stringMethodsDemo` Method:

3. **String Methods Demo (`stringMethodsDemo`):**
   - Start the `stringMethodsDemo` method.
   - Create two strings (`str1` and `str2`) with values "Hello" and "World" respectively.
   - Concatenate the strings and print the result.

   - Get the length of the concatenated string and print it.
   - Extract a substring and print it.
   - Convert the string to uppercase and lowercase and print the results.
   - Check for equality between `str1` and `str2` and print the result.
   - Find the index of the substring "World" and print it.
   - Replace a substring and print the result.
   - End the `stringMethodsDemo` method.

#### `stringBufferMethodsDemo` Method:

4. **StringBuffer Methods Demo (`stringBufferMethodsDemo`):**
   - Start the `stringBufferMethodsDemo` method.
   - Create a `StringBuffer` with the initial value "Java".
   - Append and print additional content.
   - Insert and print additional content.
   - Delete a portion and print the result.
   - Reverse the `StringBuffer` and print it.
   - End the `stringBufferMethodsDemo` method.

5. **End:**
   - End the `StringDemo` class.

### Note:

- The `stringMethodsDemo` and `stringBufferMethodsDemo` methods demonstrate various methods of the `String` and `StringBuffer` classes respectively.
- Compile and run the program to see the output of each method.

This algorithm outlines the steps taken by the program to demonstrate the different methods of the `String` and `StringBuffer` classes.

OUTPUT:
```
Concatenated String: Hello World
Length of String: 11
Substring: World
Uppercase: HELLO WORLD
Lowercase: hello world
Equality: false
Index of 'World': 6
Replaced String: Hola World
Appended StringBuffer: Java is awesome
Inserted StringBuffer: Java  programmingis awesome
Deleted StringBuffer: Java gis awesome
Reversed StringBuffer: emosewa sig avaJ
```

==
===
StringBuilder

```
public class StringDemo {
```

```
public static void main(String[] args) {
    // String methods
    stringMethodsDemo();

    // StringBuilder methods
    stringBuilderMethodsDemo();
}

// Demonstrate String methods
private static void stringMethodsDemo() {
    // Creating strings
    String str1 = "Hello";
    String str2 = "World";

    // Concatenation
    String concatenated = str1.concat(" " + str2);
    System.out.println("Concatenated String: " + concatenated);

    // Length
    int length = concatenated.length();
    System.out.println("Length of String: " + length);

    // Substring
    String substring = concatenated.substring(6, 11);
    System.out.println("Substring: " + substring);

    // Uppercase and lowercase
    String uppercase = concatenated.toUpperCase();
    String lowercase = concatenated.toLowerCase();
    System.out.println("Uppercase: " + uppercase);
    System.out.println("Lowercase: " + lowercase);

    // Equality
    boolean isEqual = str1.equals(str2);
    System.out.println("Equality: " + isEqual);

    // Index of
    int index = concatenated.indexOf("World");
    System.out.println("Index of 'World': " + index);

    // Replace
    String replaced = concatenated.replace("Hello", "Hola");
    System.out.println("Replaced String: " + replaced);
}

// Demonstrate StringBuilder methods
private static void stringBuilderMethodsDemo() {
    // Creating StringBuilder
    StringBuilder stringBuilder = new StringBuilder("Java");
```

```
        // Append
        stringBuilder.append(" is").append(" awesome");
        System.out.println("Appended StringBuilder: " + stringBuilder);

        // Insert
        stringBuilder.insert(5, " programming");
        System.out.println("Inserted StringBuilder: " + stringBuilder);

        // Delete
        stringBuilder.delete(5, 16);
        System.out.println("Deleted StringBuilder: " + stringBuilder);

        // Reverse
        stringBuilder.reverse();
        System.out.println("Reversed StringBuilder: " + stringBuilder);
    }
}
```

algorithm:

Certainly! Here's the algorithm for the Java program that demonstrates different methods in the `String` and `StringBuilder` classes:

### `StringDemo` Class:

#### Main Method:

1. **Start:**
   - Define the `StringDemo` class.

2. **Main Method (`main`):**
   - Start the `main` method.
   - Call `stringMethodsDemo` method to demonstrate `String` class methods.
   - Call `stringBuilderMethodsDemo` method to demonstrate `StringBuilder` class methods.
   - End the `main` method.

#### `stringMethodsDemo` Method:

3. **String Methods Demo (`stringMethodsDemo`):**
   - Start the `stringMethodsDemo` method.
   - Create two strings (`str1` and `str2`) with values "Hello" and "World" respectively.
   - Concatenate the strings and print the result.
   - Get the length of the concatenated string and print it.
   - Extract a substring and print it.
   - Convert the string to uppercase and lowercase and print the results.
   - Check for equality between `str1` and `str2` and print the result.
   - Find the index of the substring "World" and print it.

   - Replace a substring and print the result.
   - End the `stringMethodsDemo` method.

#### `stringBuilderMethodsDemo` Method:

4. **StringBuilder Methods Demo (`stringBuilderMethodsDemo`):**
   - Start the `stringBuilderMethodsDemo` method.
   - Create a `StringBuilder` with the initial value "Java".
   - Append and print additional content.
   - Insert and print additional content.
   - Delete a portion and print the result.
   - Reverse the `StringBuilder` and print it.
   - End the `stringBuilderMethodsDemo` method.

5. **End:**
   - End the `StringDemo` class.

### Note:

- The `stringMethodsDemo` and `stringBuilderMethodsDemo` methods demonstrate various methods of the `String` and `StringBuilder` classes respectively.
- Compile and run the program to see the output of each method.

This algorithm outlines the steps taken by the program to demonstrate the different methods of the `String` and `StringBuilder` classes.

OUTPUT:

```
Concatenated String: Hello World
Length of String: 11
Substring: World
Uppercase: HELLO WORLD
Lowercase: hello world
Equality: false
Index of 'World': 6
Replaced String: Hola World
Appended StringBuilder: Java is awesome
Inserted StringBuilder: Java  programmingis awesome
Deleted StringBuilder: Java gis awesome
Reversed StringBuilder: emosewa sig avaJ
```

==
===