JAVA LAB INDEX PROGRAM

1. Write a Java Program to define a class, define instance methods for setting and Retrieving values of instance variables and instantiate its object.

```
// Define the class
class Student {
    // Instance variables
    private String name;
    private int age;
    private double grade;
    // Constructor
    public Student() {
        // Default constructor
        this.name = "";
        this.age = 0;
        this.grade = 0.0;
    }
    // Setter method for 'name'
    public void setName(String name) {
        this.name = name;
    }
    // Setter method for 'age'
    public void setAge(int age) {
        this.age = age;
    }
    // Setter method for 'grade'
    public void setGrade(double grade) {
        this.grade = grade;
    }
    // Getter method for 'name'
    public String getName() {
        return this.name;
```

```
    }
    // Getter method for 'age'
    public int getAge() {
        return this.age;
    }
    // Getter method for 'grade'
    public double getGrade() {
        return this.grade;
    }}
public class Main {
    public static void main(String[] args) {
        // Instantiate an object of the Student class
        Student student1 = new Student();
        // Setting values using setter methods
        student1.setName("John Doe");
        student1.setAge(20);
        student1.setGrade(88.5);
        // Retrieving values using getter methods
        System.out.println("Student Name: " + student1.getName());
        System.out.println("Student Age: " + student1.getAge());
        System.out.println("Student Grade: " + student1.getGrade());
    }}
```

O/P:

Student Name: John Doe
Student Age: 20
Student Grade: 88.5

2. **Write a program to find the volume of a box having its side w,h,d means width ,height and depth. Its volume is v=w*h*d and also find the surface area given by the formula s=2(wh+hd+dw). Use appropriate constructors for the above.**

```
// Define the class Box
class Box {
    // Instance variables for width, height, and depth
    private double width;
    private double height;
    private double depth;

    // Constructor to initialize the dimensions of the box
    public Box(double w, double h, double d) {
        this.width = w;
        this.height = h;
        this.depth = d;
    }
    // Method to calculate the volume of the box (v = w * h * d)
    public double calculateVolume() {
        return width * height * depth;
    }
    // Method to calculate the surface area of the box (s = 2(w*h + h*d + d*w))
    public double calculateSurfaceArea() {
        return 2 * (width * height + height * depth + depth * width);
    }
    // Getter methods (optional, if needed for accessing the dimensions)
    public double getWidth() {
        return width;
    }
```

```
    public double getHeight() {
        return height;
    }
    public double getDepth() {
        return depth;
    }}
public class Main {
    public static void main(String[] args) {
        // Create an object of the Box class with specific dimensions
        Box box1 = new Box(5.0, 3.0, 4.0);
        // Calculate the volume and surface area
        double volume = box1.calculateVolume();
        double surfaceArea = box1.calculateSurfaceArea();
        // Print the results
        System.out.println("Volume of the box: " + volume);
        System.out.println("Surface area of the box: " + surfaceArea);
    }
}
```

O/P:

Volume of the box: 60.0

Surface area of the box: 94.0

3. Create a base class called shape. Apart from Constructors, It contains two methods get xyvalue() and show xyvalue() for accepting co-ordinates and to display the same. Create the sub class Called Rectangle which contains a method to display the length and breadth of the rectangle called showxyvalue().Illustrate the concepts of Overriding and Constructor call sequence.

```
// Base class: Shape
class Shape {
    // Instance variables for the coordinates
    private int x;
    private int y;
    // Constructor of the base class
    public Shape() {
        System.out.println("Shape Constructor called");    }
    // Method to accept coordinates (x, y)
    public void getXYValue(int x, int y) {
        this.x = x;
        this.y = y;    }
    // Method to display coordinates (x, y)
    public void showXYValue() {
        System.out.println("Coordinates of Shape: (" + x + ", " + y + ")");    }}
// Subclass: Rectangle extends Shape
class Rectangle extends Shape {
    // Instance variables for length and breadth
    private int length;
    private int breadth;
    // Constructor of the subclass
    public Rectangle(int length, int breadth) {
        // Calling the constructor of the base class (Shape)
        super();
        System.out.println("Rectangle Constructor called");
        this.length = length;
```

```
        this.breadth = breadth;    }
    // Overriding the showXYValue method to also display length and breadth
    @Override
    public void showXYValue() {
// First calling the base class method to display coordinates
        super.showXYValue();
 // Then showing the length and breadth of the rectangle
        System.out.println("Length of Rectangle: " + length);
        System.out.println("Breadth of Rectangle: " + breadth);    }}
public class Main {
    public static void main(String[] args) {
        // Create an object of Rectangle class
        Rectangle rect = new Rectangle(10, 5);
        // Setting the coordinates for the rectangle
        rect.getXYValue(2, 3);
        // Displaying the coordinates and dimensions
        rect.showXYValue();  // This will call the overridden method    }}
```

O/P:

Shape Constructor called

Rectangle Constructor called

Coordinates of Shape: (2, 3)

Length of Rectangle: 10

Breadth of Rectangle: 5

4. Write a program in JAVA to create a class Bird also declares the different parameterized constructor to display the name of Birds.

```
// Bird class definition
public class Bird {

    // Instance variable to store the name of the bird
    private String name;

    // Default constructor
```

```
  public Bird() {
    this.name = "Unknown Bird";  // Default name if no argument is passed
  }

  // Parameterized constructor to set the bird's name
  public Bird(String name) {
    this.name = name;
  }

  // Another parameterized constructor (for demonstration) with a default species
  public Bird(String name, String species) {
    this.name = name + " the " + species;
  }

  // Method to display the bird's name
  public void display() {
    System.out.println("Bird's name: " + name);
  }

  // Main method to test the Bird class
  public static void main(String[] args) {
    // Creating objects using different constructors
    Bird bird1 = new Bird();  // Using default constructor
    bird1.display();

    Bird bird2 = new Bird("Eagle");  // Using constructor with bird name
    bird2.display();

    Bird bird3 = new Bird("Parrot", "Macaw");  // Using constructor with name and species
    bird3.display();
  }
}
```

O/P

Bird's name: Unknown Bird
Bird's name: Eagle
Bird's name: Parrot the Macaw

5. **Write a program that creates an abstract class called dimension, create two subclasses, rectangle and triangle. Include appropriate methods for both the subclass that calculate and display the area of the rectangle and triangle.**

```
// Abstract class Dimension
abstract class Dimension {
  // Abstract method to calculate the area
```

```
    public abstract double calculateArea();

    // Abstract method to display the area
    public abstract void displayArea();
}

// Rectangle class that extends Dimension
class Rectangle extends Dimension {
    private double length;
    private double width;

    // Constructor to initialize length and width
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implementing the calculateArea method
    @Override
    public double calculateArea() {
        return length * width;
    }

    // Implementing the displayArea method
    @Override
    public void displayArea() {
        double area = calculateArea();
        System.out.println("Area of the Rectangle: " + area + " square units");
    }
}

// Triangle class that extends Dimension
class Triangle extends Dimension {
    private double base;
    private double height;

    // Constructor to initialize base and height
    public Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

    // Implementing the calculateArea method
    @Override
```

```
    public double calculateArea() {
        return 0.5 * base * height;
    }

    // Implementing the displayArea method
    @Override
    public void displayArea() {
        double area = calculateArea();
        System.out.println("Area of the Triangle: " + area + " square units");
    }
}

// Main class to test the program
public class Main {
    public static void main(String[] args) {
        // Create a Rectangle object
        Rectangle rect = new Rectangle(5, 10);
        rect.displayArea();  // Display the area of the rectangle

        // Create a Triangle object
        Triangle tri = new Triangle(6, 8);
        tri.displayArea();  // Display the area of the triangle
    }
}
```

Output:

```
Area of the Rectangle: 50.0 square units
Area of the Triangle: 24.0 square units
```

6. **Write a Java program to demonstrate the use of the Integer, Double, and Character wrapper classes. Convert a primitive value of type int, double, and char to their corresponding wrapper classes and print them.**

```
public class WrapperClassDemo {
    public static void main(String[] args) {
        // Primitive values
        int num = 42;
        double pi = 3.14159;
        char letter = 'A';

        // Convert primitive types to corresponding wrapper classes
        Integer intWrapper = Integer.valueOf(num);
        Double doubleWrapper = Double.valueOf(pi);
        Character charWrapper = Character.valueOf(letter);
```

```
        // Print the values of the wrapper classes
        System.out.println("Integer wrapper value: " + intWrapper);
        System.out.println("Double wrapper value: " + doubleWrapper);
        System.out.println("Character wrapper value: " + charWrapper);

        // Demonstrating autoboxing (automatic conversion from primitive to wrapper class)
        Integer autoBoxedInt = num;
        Double autoBoxedDouble = pi;
        Character autoBoxedChar = letter;

        // Print the auto-boxed values
        System.out.println("Auto-boxed Integer value: " + autoBoxedInt);
        System.out.println("Auto-boxed Double value: " + autoBoxedDouble);
        System.out.println("Auto-boxed Character value: " + autoBoxedChar);
    }
}
```

Output:

```
Integer wrapper value: 42
Double wrapper value: 3.14159
Character wrapper value: A
Auto-boxed Integer value: 42
Auto-boxed Double value: 3.14159
Auto-boxed Character value: A
```

7. **Write a Java program to demonstrate the use of a** non-static inner class**. Create an outer class with a method that instantiates the inner class and calls a method of the inner class.**

```
// Outer class
class OuterClass {
    private String outerMessage = "Hello from the Outer Class!";

    // Non-static inner class
    class InnerClass {
        // Method of the inner class
        public void displayMessage() {
            // Accessing outer class's member variable
            System.out.println("Message from the Inner Class: " + outerMessage);
        }
    }

    // Method in the outer class to instantiate the inner class and call its method
    public void invokeInnerClass() {
        // Creating an instance of the non-static inner class
        InnerClass inner = new InnerClass();
```

```
        // Calling the inner class method
        inner.displayMessage();
    }
}

// Main class to test the program
public class Main {
    public static void main(String[] args) {
        // Creating an instance of the outer class
        OuterClass outer = new OuterClass();

        // Calling the method that instantiates and calls the inner class method
        outer.invokeInnerClass();
    }
}
```

Output:

Message from the Inner Class: Hello from the Outer Class!

8. **Write a Java program where multiple interfaces are implemented by a single class. Define two interfaces, Animal and Vehicle, each with a method (makeSound() for Animal and drive() for Vehicle). Implement both interfaces in a class Robot and call both methods from the main function**.

```
// Interface Animal
interface Animal {
    void makeSound();  // Method to make sound
}

// Interface Vehicle
interface Vehicle {
    void drive();  // Method to drive
}

// Class Robot implements both Animal and Vehicle interfaces
class Robot implements Animal, Vehicle {

    // Implementing the makeSound() method of Animal interface
    @Override
    public void makeSound() {
        System.out.println("Robot makes sound: Beep Beep!");
    }

    // Implementing the drive() method of Vehicle interface
    @Override
    public void drive() {
        System.out.println("Robot is driving: Vroom Vroom!");
```

```
    }
}

// Main class to test the Robot class
public class Main {
    public static void main(String[] args) {
        // Creating an instance of the Robot class
        Robot robot = new Robot();

        // Calling both methods implemented from interfaces
        robot.makeSound();  // Method from Animal interface
        robot.drive();      // Method from Vehicle interface
    }
}
```

Output:

```
Robot makes sound: Beep Beep!
Robot is driving: Vroom Vroom!
```

9. **Write a Java program where one interface extends multiple other interfaces. For example, create an interface Worker that extends two interfaces, Driver and Cook, both having different methods. Implement the Worker interface in a class Person and call the methods from the main function.**

```
// Interface Driver
interface Driver {
    void drive();  // Method to drive
}

// Interface Cook
interface Cook {
    void cook();  // Method to cook
}

// Interface Worker extends both Driver and Cook
interface Worker extends Driver, Cook {
    // No additional methods are required here, just inheriting methods from Driver
and Cook
}

// Class Person implements Worker interface
class Person implements Worker {

    // Implementing the drive() method from Driver interface
    @Override
    public void drive() {
        System.out.println("Person is driving a car.");
    }

    // Implementing the cook() method from Cook interface
    @Override
    public void cook() {
        System.out.println("Person is cooking a meal.");
    }
```

```
}

// Main class to test the Person class
public class Main {
    public static void main(String[] args) {
        // Creating an instance of Person, which implements Worker
        Person person = new Person();

        // Calling the methods implemented from Driver and Cook interfaces
        person.drive();  // Method from Driver interface
        person.cook();   // Method from Cook interface
    }
}
```

Output:

```
Person is driving a car.
Person is cooking a meal.
```

10. **Write a program, which throws Arithmetic Exception. Write another class (in a different file) that handles the Exception**

//File 1: `ThrowException.java`

```
// ThrowException.java
public class ThrowException {
   public static void main(String[] args) {
      int numerator = 10;
      int denominator = 0; // Division by zero will cause ArithmeticException

      // Try to perform division and throw an ArithmeticException
      int result = numerator / denominator; // This line will throw ArithmeticException
      System.out.println("Result: " + result);
   }
}
```

File 2: `HandleException.java`

```
// HandleException.java

public class HandleException {

   public static void main(String[] args) {

      try {

         // Create an instance of ThrowException class

         ThrowException throwException = new ThrowException();

         // Call the main method of ThrowException to throw the exception

         throwException.main(args);
```

```
} catch (ArithmeticException e) {
    // Catch the ArithmeticException and handle it
    System.out.println("Error: Division by zero is not allowed.");
}   }}
```

Output:

Error: Division by zero is not allowed.

11. **Create a user defined Exception class which throws Exception when the user inputs the marks greater than 100 Catch it and again rethrow it.**

```
//File 1: InvalidMarksException.java (User-Defined Exception Class)
// InvalidMarksException.java
public class InvalidMarksException extends Exception {
    // Constructor to initialize the exception with a message
    public InvalidMarksException(String message) {
        super(message);
    }
}

//File 2: Main.java (Main Class that Handles Input and Exception)
import java.util.Scanner;

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

        System.out.print("Enter marks: ");
        int marks = scanner.nextInt();

        try {
            // Check if marks are greater than 100
            if (marks > 100) {
                // Throw the custom exception if marks are greater than 100
                throw new InvalidMarksException("Marks cannot be greater than 100.");
            } else {
                System.out.println("You entered valid marks: " + marks);
            }
        } catch (InvalidMarksException e) {
            System.out.println("Caught Exception: " + e.getMessage());
            // Rethrow the exception
            try {
```

```
            throw e; // Rethrow the exception
        } catch (InvalidMarksException rethrownException) {
            System.out.println("Rethrown Exception: " + rethrownException.getMessage());
        }
    } finally {
        scanner.close(); // Close the scanner resource
    }           }}
```

Output:

Case 1: Valid Marks (less than or equal to 100)
Enter marks: 85
You entered valid marks: 85
Case 2: Invalid Marks (greater than 100)
Enter marks: 120
Caught Exception: Marks cannot be greater than 100.
Rethrown Exception: Marks cannot be greater than 100.

12. Write a program to illustrate various String class methods.

```
public class StringMethodsDemo {
    public static void main(String[] args) {
        // Initializing a String
        String str = "  Hello, World! Java is fun!  ";

        // 1. length() - Returns the length of the string
        System.out.println("Length of the string: " + str.length());

        // 2. trim() - Removes leading and trailing whitespaces
        String trimmedStr = str.trim();
        System.out.println("Trimmed String: '" + trimmedStr + "'");

        // 3. toLowerCase() - Converts all characters to lowercase
        System.out.println("Lowercase String: " + str.toLowerCase());

        // 4. toUpperCase() - Converts all characters to uppercase
        System.out.println("Uppercase String: " + str.toUpperCase());

        // 5. charAt() - Returns the character at a specified index
        System.out.println("Character at index 7: " + str.charAt(7));

        // 6. substring() - Extracts a substring from the given string
        System.out.println("Substring from index 7 to 12: " + str.substring(7, 12));

        // 7. replace() - Replaces all occurrences of a character with another character
        System.out.println("String after replacing 'o' with '0': " + str.replace('o', '0'));
```

```
        // 8. contains() - Checks if a substring is present in the string
        System.out.println("Does the string contain 'World'? " + str.contains("World"));

        // 9. startsWith() - Checks if the string starts with a specific prefix
        System.out.println("Does the string start with '  Hello'? " + str.startsWith("  Hello"));

        // 10. endsWith() - Checks if the string ends with a specific suffix
        System.out.println("Does the string end with 'fun!'? " + str.endsWith("fun!"));

        // 11. indexOf() - Returns the index of the first occurrence of a character or substring
        System.out.println("Index of 'Java': " + str.indexOf("Java"));

        // 12. lastIndexOf() - Returns the index of the last occurrence of a character or substring
        System.out.println("Last index of 'o': " + str.lastIndexOf('o'));

        // 13. split() - Splits the string into an array based on a delimiter
        String[] words = str.split(" ");
        System.out.println("Words in the string:");
        for (String word : words) {
            System.out.println(word);
        }

        // 14. valueOf() - Converts other data types (like int, boolean, etc.) to a string
        int number = 123;
        boolean flag = true;
        System.out.println("String representation of int 123: " + String.valueOf(number));
        System.out.println("String representation of boolean true: " + String.valueOf(flag));

        // 15. compareTo() - Compares two strings lexicographically
        String anotherStr = "Hello, World! Java is fun!";
        System.out.println("Comparison with another string: " + str.compareTo(anotherStr));

        // 16. equals() - Checks if two strings are equal
        String str2 = "Hello, World! Java is fun!";
        System.out.println("Are the two strings equal? " + str.equals(str2));

        // 17. equalsIgnoreCase() - Checks if two strings are equal, ignoring case
        String str3 = "hello, world! java is fun!";
        System.out.println("Are the strings equal (ignoring case)? " + str.equalsIgnoreCase(str3));

        // 18. format() - Formats a string with specified placeholdersString formattedStr =
String.format("The number is: %d, and the message is: %s", 42, "Success");
        System.out.println("Formatted String: " + formattedStr);
    }}
```

Output:

```
Length of the string: 30
```

```
Trimmed String: 'Hello, World! Java is fun!'
Lowercase String:   hello, world! java is fun!
Uppercase String:   HELLO, WORLD! JAVA IS FUN!
Character at index 7: W
Substring from index 7 to 12: World
String after replacing 'o' with '0':  Hell0, W0rld! Java is fun!
Does the string contain 'World'? true
Does the string start with '  Hello'? true
Does the string end with 'fun!'? true
Index of 'Java': 19
Last index of 'o': 4
Words in the string:
Hello,
World!
Java
is
fun!
String representation of int 123: 123
String representation of boolean true: true
Comparison with another string: 0
Are the two strings equal? true
Are the strings equal (ignoring case)? true
Formatted String: The number is: 42, and the message is: Success
```

13. Write a Program to implement Bank Account Class which illustrates the concept of Thread Synchronization.

```
// BankAccount.java
class BankAccount {
   private double balance;

   // Constructor to initialize the account balance
   public BankAccount(double balance) {
      this.balance = balance;
   }

   // Synchronized method to withdraw money from the account
   public synchronized void withdraw(double amount) {
      if (balance >= amount) {
         System.out.println(Thread.currentThread().getName() + " is going to withdraw " +
amount);
         // Simulate a delay (e.g., to mimic network/database operations)
         try {
            Thread.sleep(1000);
         } catch (InterruptedException e) {
            System.out.println(e);
         }
```

```
            balance -= amount;
            System.out.println(Thread.currentThread().getName() + " completes withdrawal of " +
amount);
        } else {
            System.out.println(Thread.currentThread().getName() + " cannot withdraw " + amount +
" due to insufficient funds.");
        }
    }

    // Method to get the current balance
    public double getBalance() {
        return balance;
    }
}

// WithdrawalThread.java
class WithdrawalThread extends Thread {
    private BankAccount account;
    private double amount;

    // Constructor to initialize the thread with a bank account and amount to withdraw
    public WithdrawalThread(BankAccount account, double amount) {
        this.account = account;
        this.amount = amount;
    }

    // The run method that will be executed when the thread starts
    @Override
    public void run() {
        account.withdraw(amount);
    }
}

// Main.java
public class Main {
    public static void main(String[] args) {
        // Create a BankAccount object with an initial balance of 1000
        BankAccount account = new BankAccount(1000);

        // Create multiple threads simulating withdrawal operations
        Thread thread1 = new WithdrawalThread(account, 500);
        Thread thread2 = new WithdrawalThread(account, 300);
        Thread thread3 = new WithdrawalThread(account, 200);
        Thread thread4 = new WithdrawalThread(account, 100);

        // Start the threads
```

```
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
    }}
```

Output:

```
Thread-0 is going to withdraw 500.0
Thread-0 completes withdrawal of 500.0
Thread-1 is going to withdraw 300.0
Thread-1 completes withdrawal of 300.0
Thread-2 is going to withdraw 200.0
Thread-2 completes withdrawal of 200.0
Thread-3 is going to withdraw 100.0
Thread-3 completes withdrawal of 100.0
```

14. Write a Java program to illustrate collection classes.

```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.HashMap;
import java.util.Iterator;

public class CollectionDemo {
    public static void main(String[] args) {
        // 1. Using ArrayList - Dynamic Array
        ArrayList<String> arrayList = new ArrayList<>();

        // Adding elements to ArrayList
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Orange");
        arrayList.add("Mango");
        arrayList.add("Banana");  // Duplicate element

        System.out.println("ArrayList: " + arrayList);

        // Accessing elements in ArrayList
        System.out.println("First Element: " + arrayList.get(0));

        // Removing an element
        arrayList.remove("Banana");
        System.out.println("ArrayList after removing 'Banana': " + arrayList);

        // 2. Using HashSet - Set (No duplicates and no order)
        HashSet<String> hashSet = new HashSet<>();

        // Adding elements to HashSet
```

```
        hashSet.add("Apple");
        hashSet.add("Banana");
        hashSet.add("Orange");
        hashSet.add("Mango");
        hashSet.add("Banana");  // Duplicate, won't be added

        System.out.println("HashSet: " + hashSet);

        // 3. Using HashMap - Map (key-value pairs)
        HashMap<String, Integer> hashMap = new HashMap<>();

        // Adding key-value pairs to HashMap
        hashMap.put("Apple", 10);
        hashMap.put("Banana", 20);
        hashMap.put("Orange", 15);
        hashMap.put("Mango", 30);

        System.out.println("HashMap: " + hashMap);

        // Accessing a value using a key
        System.out.println("Price of Mango: " + hashMap.get("Mango"));

        // Iterating through keys in the HashMap
        System.out.println("Keys in HashMap:");
        for (String key : hashMap.keySet()) {
            System.out.println(key + ": " + hashMap.get(key));
        }

        // 4. Iterating over HashSet using Iterator
        System.out.println("Iterating over HashSet:");
        Iterator<String> iterator = hashSet.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }    }}
```

Output:

```
ArrayList: [Apple, Banana, Orange, Mango, Banana]
First Element: Apple
ArrayList after removing 'Banana': [Apple, Orange, Mango, Banana]
HashSet: [Apple, Banana, Orange, Mango]
HashMap: {Apple=10, Banana=20, Orange=15, Mango=30}
Price of Mango: 30
Keys in HashMap:
Apple: 10
Banana: 20
Orange: 15
Mango: 30
```

Iterating over HashSet:
Apple
Banana
Orange
Mango

15. **Write an applet that demonstrates the** life cycle of an applet**. Implement the init(), start(), stop(), and destroy() methods and display appropriate messages when each of these methods is called. Print the current state of the applet in each method.**

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

public class AppletLifeCycleDemo extends Applet {

    // This method is called when the applet is first loaded
    @Override
    public void init() {
        // Initialization code here
        System.out.println("init() method called - Applet initialized.");
    }

    // This method is called when the applet is started (after init or after stop)
    @Override
    public void start() {
        // Starting the applet (can be used for animations or other tasks)
        System.out.println("start() method called - Applet started.");
    }

    // This method is called when the applet is stopped (e.g., browser or window is closed)
    @Override
    public void stop() {
        // Applet stop tasks (e.g., suspending ongoing tasks or animations)
        System.out.println("stop() method called - Applet stopped.");
    }

    // This method is called when the applet is destroyed (e.g., when the browser is closed)
    @Override
    public void destroy() {
        // Cleanup code for the applet (e.g., releasing resources)
        System.out.println("destroy() method called - Applet destroyed.");
    }

    // This method is used to display the state of the applet
    @Override
    public void paint(Graphics g) {
        g.drawString("Applet Life Cycle Demo", 50, 50);
```

}}

Output in Console:

```
init() method called - Applet initialized.
start() method called - Applet started.
stop() method called - Applet stopped.
destroy() method called - Applet destroyed.
```