

LAB 4

Q.1. Method Overloading: Write a class Calculator with overloaded methods add(). Implement add() methods that take:

- Two integers
- Two double values
- Three integers
- A variable number of integers

Program:

```
package LAB4;

public class Calculator { //creating the Calculator Class
    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add two double values
    public double add(double a, double b) {
        return a + b;
    }

    // Method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Method to add a variable number of integers
    public int add(int... numbers) {
        int sum = 0;
        for (int num : numbers) {
            sum += num;
        }
        return sum;
    }

    // Example usage
    public static void main(String[] args) {
        Calculator calc = new Calculator(); //creating the object of
        the class

        System.out.println("Sum of two integers: " + calc.add(5,
        10)); //printing the statements
    }
}
```

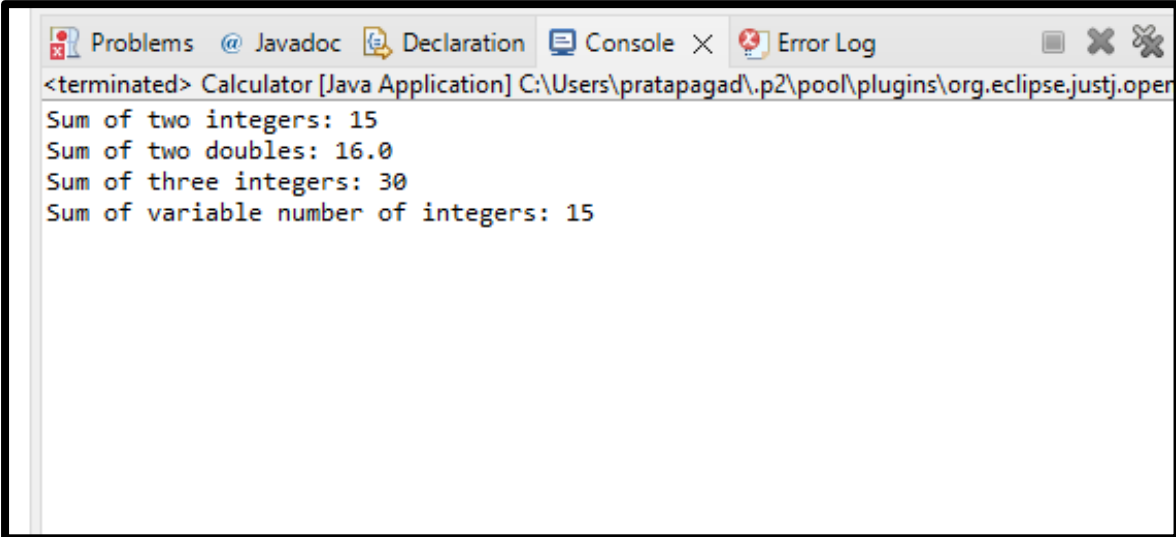
Student's ID: AF0402433

Trainer's Name: Manali Ma'am

Student's Name: Patel Abubakar Siddique Mehboob

```
        System.out.println("Sum of two doubles: " + calc.add(5.5,
10.5));
        System.out.println("Sum of three integers: " + calc.add(5,
10, 15));
        System.out.println("Sum of variable number of integers: " +
calc.add(1, 2, 3, 4, 5));
    }
}
```

Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Error Log. The console output displays the results of the Java application: "Sum of two integers: 15", "Sum of two doubles: 16.0", "Sum of three integers: 30", and "Sum of variable number of integers: 15". The window title is "<terminated> Calculator [Java Application] C:\Users\pratapagad\.p2\pool\plugins\org.eclipse.justj.open".

```
<terminated> Calculator [Java Application] C:\Users\pratapagad\.p2\pool\plugins\org.eclipse.justj.open
Sum of two integers: 15
Sum of two doubles: 16.0
Sum of three integers: 30
Sum of variable number of integers: 15
```

Q.2. Super Keyword: Create a class Person with a constructor that accepts and sets name and age.

- Create a subclass Student that adds a grade property and initializes name and age using the super keyword in its constructor.
- Demonstrate the creation of Student objects and the usage of super to call the parent class constructor.

Program:

```
package LAB4;
```

```
class Person { //creating the Person class
    String name; //declaring the variables
    int age;
```

```
    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
class Student extends Person { //creating the Student class which
    inherit to person class
    int grade;
```

```
    // Constructor using super keyword to call the parent class
    constructor
    public Student(String name, int age, int grade) {
        super(name, age); // Calling the constructor of the Person
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        // Creating a Student object
        Student student1 = new Student("Abubakar", 21, 50);

        // Accessing properties of the Student object
        System.out.println("Student name: " +
student1.name); //printing the statements
        System.out.println("Student age: " + student1.age);
        System.out.println("Student grade: " + student1.grade);
    }
}
```

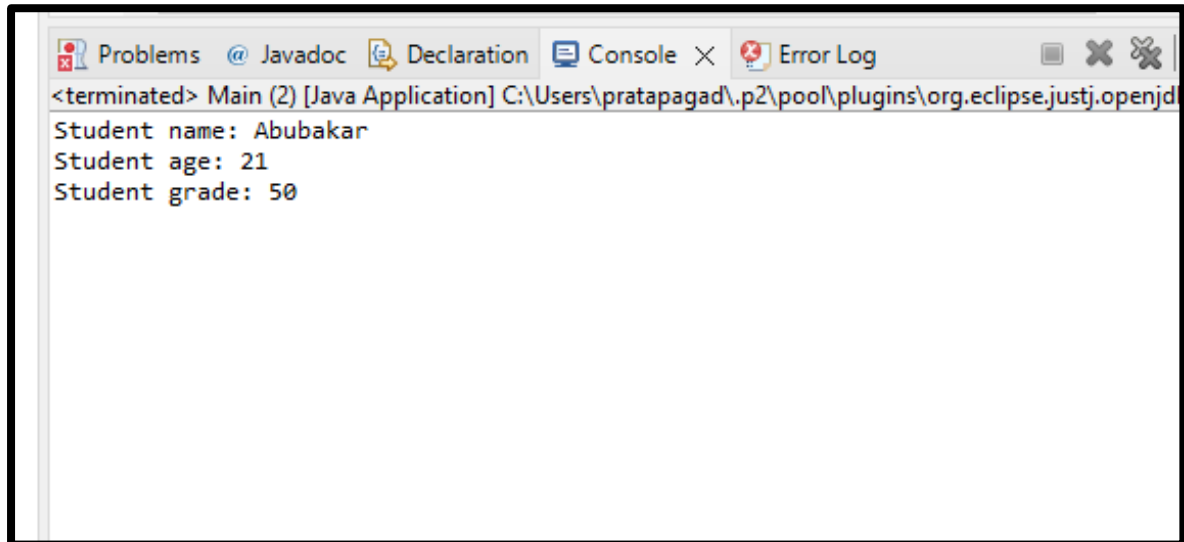
Student's ID: AF0402433

Trainer's Name: Manali Ma'am

Student's Name: Patel Abubakar Siddique Mehboob

```
}  
}
```

Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Error Log. The Console tab is active, displaying the output of a Java application. The output text is as follows:

```
<terminated> Main (2) [Java Application] C:\Users\pratapagad\p2\pool\plugins\org.eclipse.justj.openjdk  
Student name: Abubakar  
Student age: 21  
Student grade: 50
```

Q.3. Super Keyword: Create a base class Shape with a method draw() that prints "Drawing Shape".

- Create a subclass Circle that overrides draw() to print "Drawing Circle".
- Inside the draw() method of Circle, call the draw() method of the Shape class using super.draw().
- Write a main method to demonstrate calling draw() on a Circle object.

Program:

```
package LAB4;

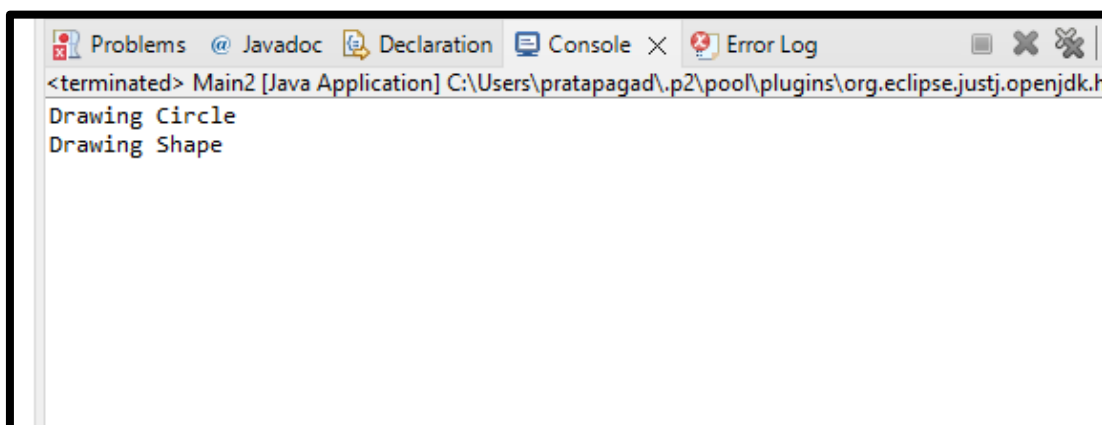
class Shape {
    public void draw() {
        System.out.println("Drawing Shape");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing Circle");
        super.draw(); // Calling draw() method of the Shape class
    }
}

public class Main2 {
    public static void main(String[] args) {
        // Creating a Circle object
        Circle circle = new Circle();

        // Calling draw() method on the Circle object
        circle.draw();
    }
}
```

Output:



Q.4. Create a base class BankAccount with a method deposit(amount) and a constructor that sets the initial balance.

- Create a subclass SavingsAccount that overrides deposit(amount) to add interest before depositing. Use the super keyword to call the deposit method of the base class.

- Write a main method to demonstrate creating a SavingsAccount and depositing an amount to see the effect of interest.

Program:

```
package LAB4;
```

```
class BankAccount {  
    protected double balance;  
  
    // Constructor to set initial balance  
    public BankAccount(double initialBalance) {  
        this.balance = initialBalance;  
    }  
  
    // Method to deposit amount  
    public void deposit(double amount) {  
        balance += amount;  
    }  
}
```

```
class SavingsAccount extends BankAccount {  
    private double interestRate;  
  
    // Constructor to set initial balance and interest rate  
    public SavingsAccount(double initialBalance, double  
interestRate) {  
        super(initialBalance);  
        this.interestRate = interestRate;  
    }  
  
    // Override deposit method to add interest before depositing  
    @Override  
    public void deposit(double amount) {  
        double interest = balance * (interestRate / 100); //  
Calculate interest  
        balance += interest; // Add interest to balance  
        super.deposit(amount); // Call deposit method of the base  
class
```

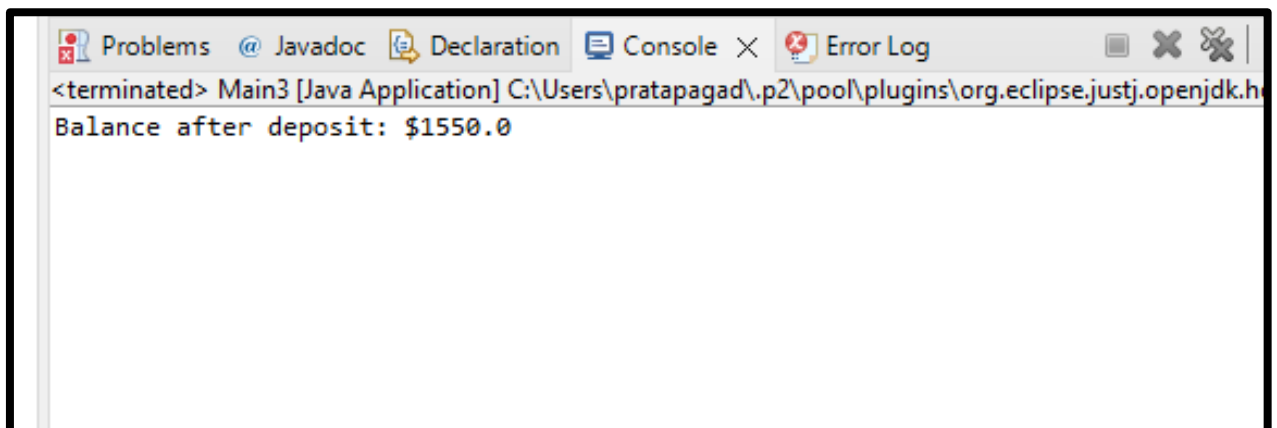
Student's ID: AF0402433

Trainer's Name: Manali Ma'am

Student's Name: Patel Abubakar Siddique Mehboob

```
    }  
}  
  
public class Main3 {  
    public static void main(String[] args) {  
        // Create a SavingsAccount with initial balance and interest  
        rate  
        SavingsAccount savingsAccount = new SavingsAccount(1000, 5);  
        // Initial balance: $1000, Interest rate: 5%  
  
        // Deposit an amount and observe the effect of interest  
        savingsAccount.deposit(500); // Deposit $500  
        System.out.println("Balance after deposit: $" +  
savingsAccount.balance); // Print updated balance  
    }  
}
```

Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Error Log. The Console output displays the message: <terminated> Main3 [Java Application] C:\Users\pratapagad\.p2\pool\plugins\org.eclipse.justj.openjdk.h Balance after deposit: \$1550.0. The text is color-coded, with the class name and path in blue and the output message in black.

Q.5. Define a class Employee with properties name and salary and a method displayDetails().

- Create a subclass Manager that adds a property department and overrides displayDetails() to include department details. Use the super keyword to call the displayDetails() method of Employee within Manager.
- In the main method, create objects of Employee and Manager and call displayDetails() to show the details.

Program:

```
package LAB4;
```

```
class Employee {
    protected String name;
    protected double salary;

    // Constructor
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Method to display employee details
    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}

class Manager extends Employee {
    private String department;

    // Constructor
    public Manager(String name, double salary, String department) {
        super(name, salary); // Call constructor of the base class
        this.department = department;
    }

    // Override displayDetails() method to include department
    details
    @Override
    public void displayDetails() {
        super.displayDetails(); // Call displayDetails() method of
the base class
    }
}
```


Student's ID: AF0402433

Trainer's Name: Manali Ma'am

Student's Name: Patel Abubakar Siddique Mehboob

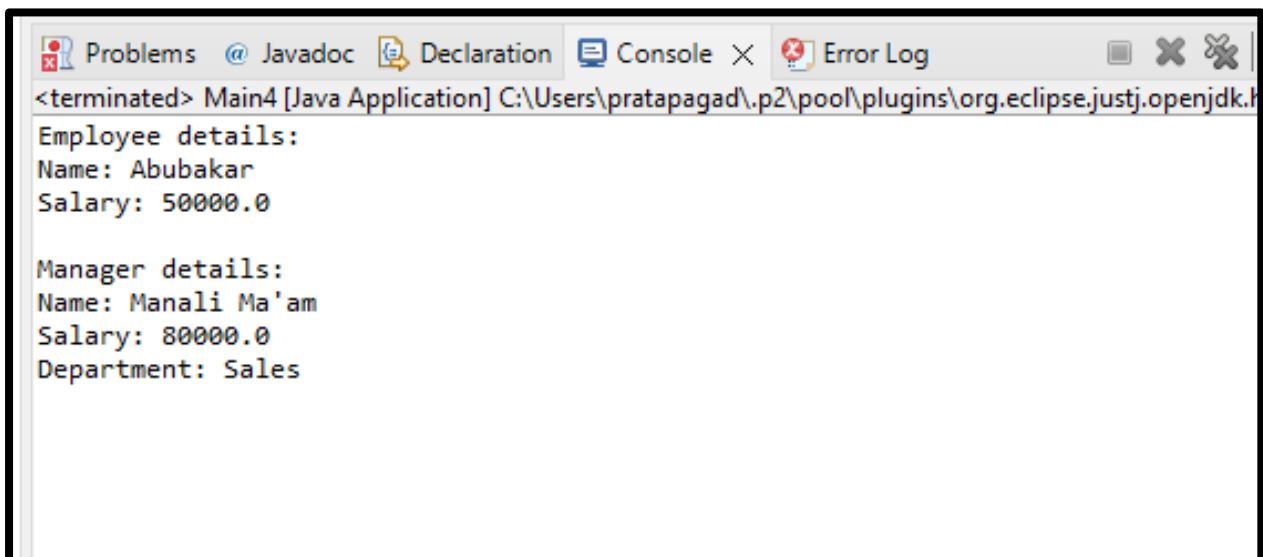
```
        System.out.println("Department: " + department);
    }
}

public class Main4 {
    public static void main(String[] args) {
        // Create an Employee object
        Employee emp = new Employee("Abubakar", 50000);
        System.out.println("Employee details:");
        emp.displayDetails(); // Display employee details

        System.out.println(); // Empty line for separation

        // Create a Manager object
        Manager manager = new Manager("Manali Ma'am", 80000,
        "Sales");
        System.out.println("Manager details:");
        manager.displayDetails(); // Display manager details
    }
}
```

Output:



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Error Log. The Console tab is active, displaying the output of the Java application. The output is as follows:

```
<terminated> Main4 [Java Application] C:\Users\pratapagad\.p2\pool\plugins\org.eclipse.justj.openjdk.b
Employee details:
Name: Abubakar
Salary: 50000.0

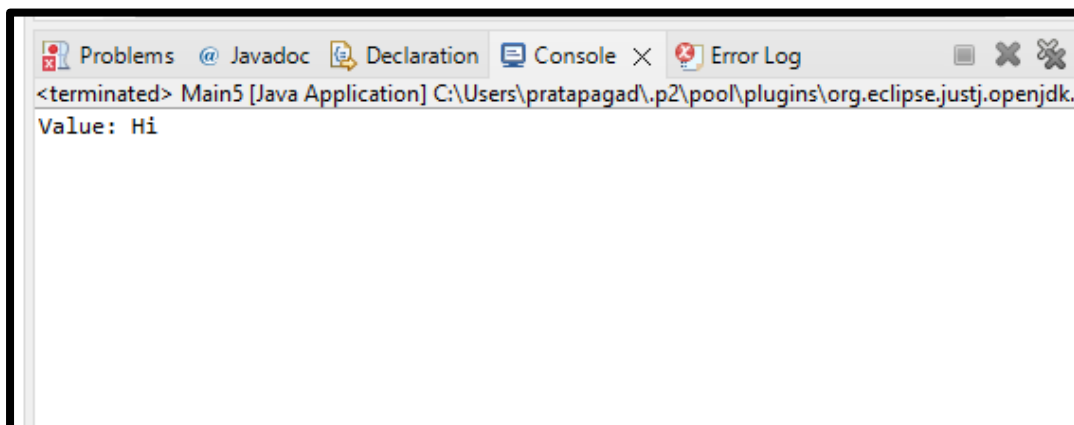
Manager details:
Name: Manali Ma'am
Salary: 80000.0
Department: Sales
```

Q.6. Write the same programme for the class ImmutableExample, to achieve object value 'Hi'.

Program:

```
package LAB4;
final class ImmutableExample {
    private final String value; // declaring the string
    // Private constructor to prevent external instantiation
    private ImmutableExample(String value) {
        this.value = value;
    }
    // Static factory method to create instances of ImmutableExample
    public static ImmutableExample createInstance() {
        // Return a new instance with the value "Hi"
        return new ImmutableExample("Hi");
    }
    // Getter method to access the value
    public String getValue() {
        return value;
    }
}
public class Main5 {
    public static void main(String[] args) {
        // Create an instance of ImmutableExample
        ImmutableExample immutableObj = ImmutableExample.createInstance();
        // Access and print the value
        System.out.println("Value: " + immutableObj.getValue()); // printing
the statement
    }
}
```

Output:



Q.7. Write the same programme for the class MutableExample, to output the object values 'hello 2' and 'hello3'.z

Program:

```
package LAB4;
class MutableExample { //declaring the class
    private String value; //creating the string
    // Constructor
    public MutableExample(String value) { //constructor
        this.value = value;
    }

    // Method to set the value
    public void setValue(String value) {
        this.value = value;
    }
    // Method to append a number to the value
    public void appendNumber(int number) {
        this.value += " " + number;
    }
    // Getter method to access the value
    public String getValue() {
        return value;
    }
}
public class Main6 {
    public static void main(String[] args) {
        // Create an instance of MutableExample with initial value
        "hello 2"
        MutableExample mutableObj = new MutableExample("hello 2");

        // Print the initial value
        System.out.println("Initial Value: " +
mutableObj.getValue());

        // Append number 3 to the value
        mutableObj.appendNumber(3);

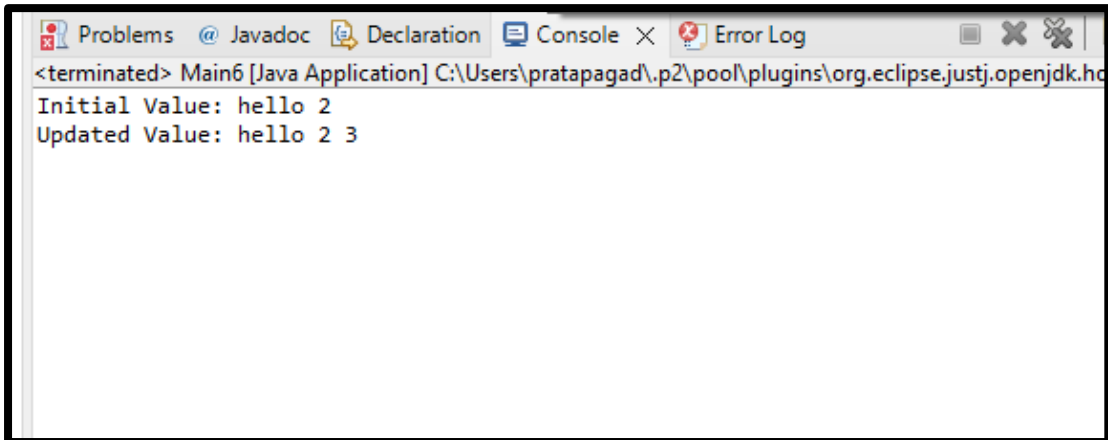
        // Print the updated value
        System.out.println("Updated Value: " +
mutableObj.getValue());
    }
}
```

Student's ID: AF0402433

Trainer's Name: Manali Ma'am

Student's Name: Patel Abubakar Siddique Mehboob

Output:

A screenshot of the Eclipse IDE's Console window. The window has a title bar with tabs for 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Error Log'. The 'Console' tab is active. The text in the console reads: '<terminated> Main6 [Java Application] C:\Users\pratapagad\p2\pool\plugins\org.eclipse.justj.openjdk.h...', 'Initial Value: hello 2', and 'Updated Value: hello 2 3'.

```
<terminated> Main6 [Java Application] C:\Users\pratapagad\p2\pool\plugins\org.eclipse.justj.openjdk.h...  
Initial Value: hello 2  
Updated Value: hello 2 3
```

Q.8. Write a java class to implement any 10 string methods:

- replace • contains • replaceAll • indexOf • substring • Equals • lastIndexOf • startsWith
- endsWith • EqualsIgnoreCase • toLowerCase • toUpperCase • isEmpty
- Length • split

Program:

```
package LAB4;
```

```
public class StringMethodsImplementation { //creating the class
```

```
    public static void main(String[] args) {
        // Example string
        String str = "Hello, World!";

        // Replace method
        System.out.println("Replace Method: " + str.replace('l',
        'x')); //printing the statements

        // Contains method
        System.out.println("Contains Method: " +
        str.contains("World"));

        // ReplaceAll method
        System.out.println("ReplaceAll Method: " +
        str.replaceAll("[aeiou]", "*"));

        // IndexOf method
        System.out.println("IndexOf Method: " + str.indexOf('o'));

        // Substring method
        System.out.println("Substring Method: " + str.substring(7));

        // Equals method
        System.out.println("Equals Method: " + str.equals("Hello,
        World!"));

        // LastIndexOf method
        System.out.println("LastIndexOf Method: " +
        str.lastIndexOf('o'));

        // StartsWith method
        System.out.println("StartsWith Method: " +
        str.startsWith("Hello"));

        // EndsWith method
```

```
System.out.println("EndsWith Method: " +
str.endsWith("World!"));

// EqualsIgnoreCase method
System.out.println("EqualsIgnoreCase Method: " +
str.equalsIgnoreCase("hello, world!"));

// ToLowerCase method
System.out.println("ToLowerCase Method: " +
str.toLowerCase());

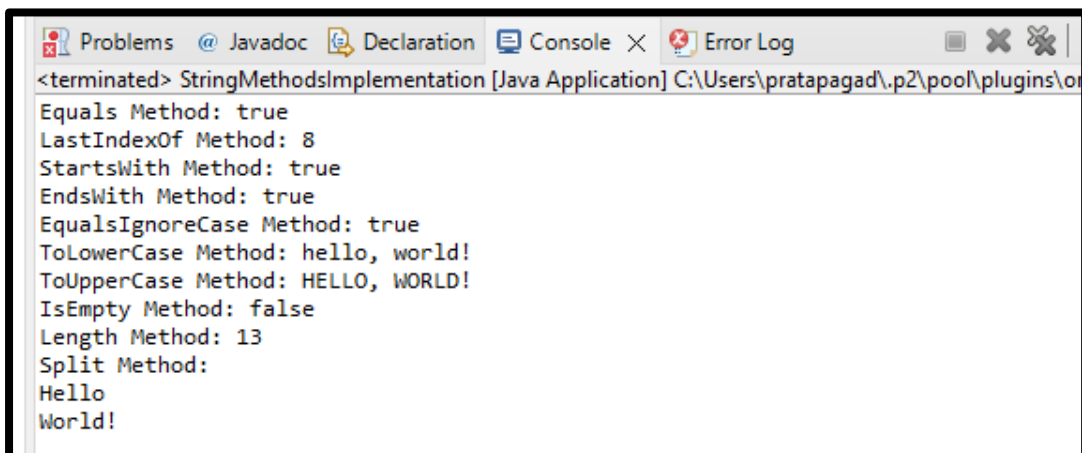
// ToUpperCase method
System.out.println("ToUpperCase Method: " +
str.toUpperCase());

// IsEmpty method
System.out.println("IsEmpty Method: " + str.isEmpty());

// Length method
System.out.println("Length Method: " + str.length());

// Split method
String[] splitArray = str.split(",");
System.out.println("Split Method: ");
for (String s : splitArray) {
    System.out.println(s.trim());
}
}
```

Output:



```
<terminated> StringMethodsImplementation [Java Application] C:\Users\pratapagad\.p2\pool\plugins\or
Equals Method: true
LastIndexOf Method: 8
StartsWith Method: true
EndsWith Method: true
EqualsIgnoreCase Method: true
ToLowerCase Method: hello, world!
ToUpperCase Method: HELLO, WORLD!
IsEmpty Method: false
Length Method: 13
Split Method:
Hello
World!
```

Q.8. Create a JavaBean class Car with properties make, model, year, and color. Implement the required no-argument constructor, getter and setter methods for each property. Write a main class to create an instance of Car, set its properties, and print the car details.

Program:

```
package LAB3;
public class Car_demo {
    private String make;
    private String model;
    private int year;
    private String color;

    public Car() {
        // Required no-argument constructor
    }

    // Getter and setter methods for make
    public String getMake() {
        return make;
    }

    public void setMake(String make) {
        this.make = make;
    }

    // Getter and setter methods for model
    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    // Getter and setter methods for year
    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Getter and setter methods for color
    public String getColor() {
        return color;
    }

    public void setColor(String color) {
```

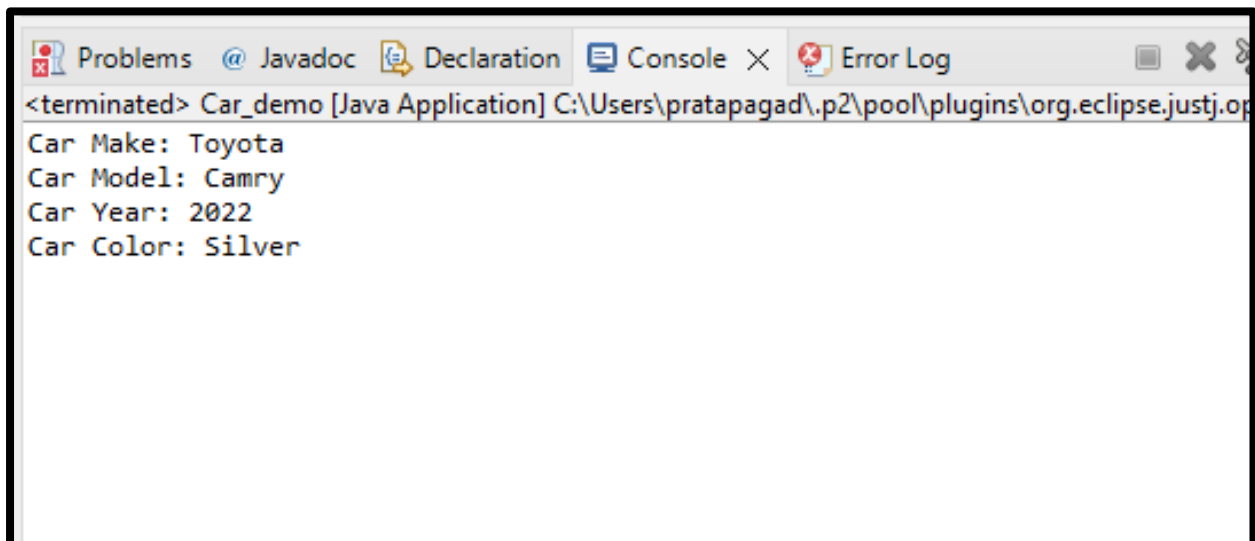
```
        this.color = color;
    }

    public static void main(String[] args) {
        // Create an instance of Car
        Car_demo car = new Car_demo();

        // Set properties
        car.setMake("Toyota");
        car.setModel("Camry");
        car.setYear(2022);
        car.setColor("Silver");

        // Print out the car details
        System.out.println("Car Make: " + car.getMake());
        System.out.println("Car Model: " + car.getModel());
        System.out.println("Car Year: " + car.getYear());
        System.out.println("Car Color: " + car.getColor());
    }
}
```

Output:

A screenshot of the Eclipse IDE's Console window. The window has a title bar with icons for Problems, Javadoc, Declaration, Console, and Error Log. The Console tab is active, showing the output of a Java application named 'Car_demo'. The output consists of four lines: 'Car Make: Toyota', 'Car Model: Camry', 'Car Year: 2022', and 'Car Color: Silver'. The text is displayed in a monospaced font with standard color coding (black text on a white background).

```
<terminated> Car_demo [Java Application] C:\Users\pratapagad\.p2\pool\plugins\org.eclipse.justj.op
Car Make: Toyota
Car Model: Camry
Car Year: 2022
Car Color: Silver
```