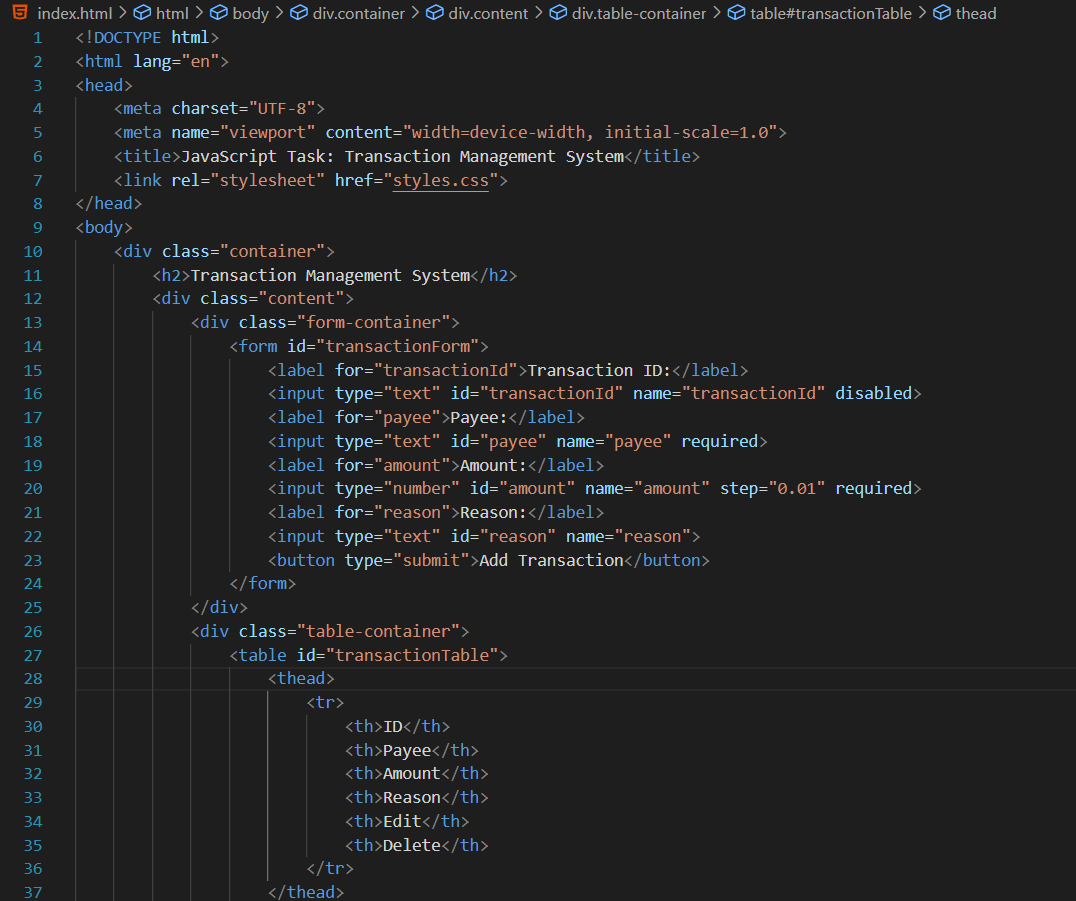
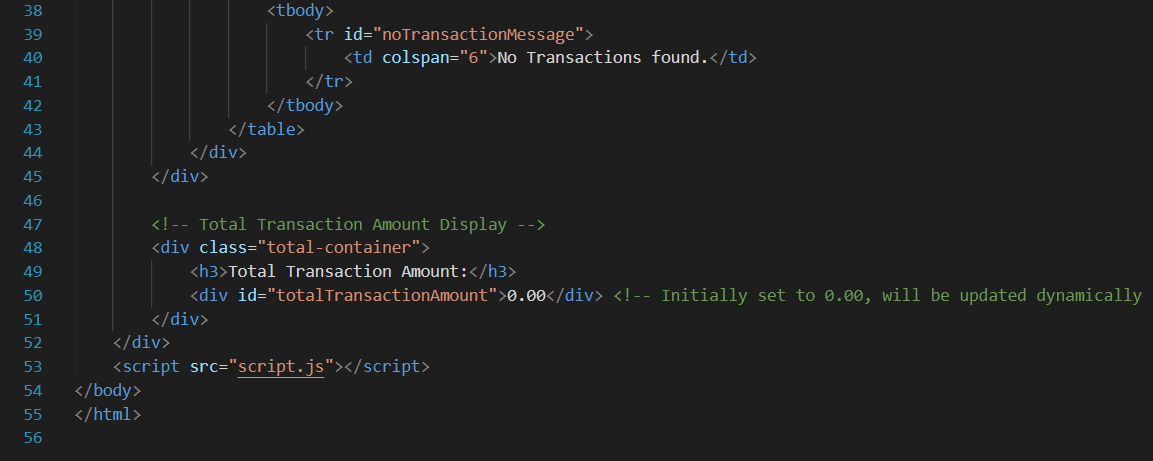
**JavaScript Tasks**

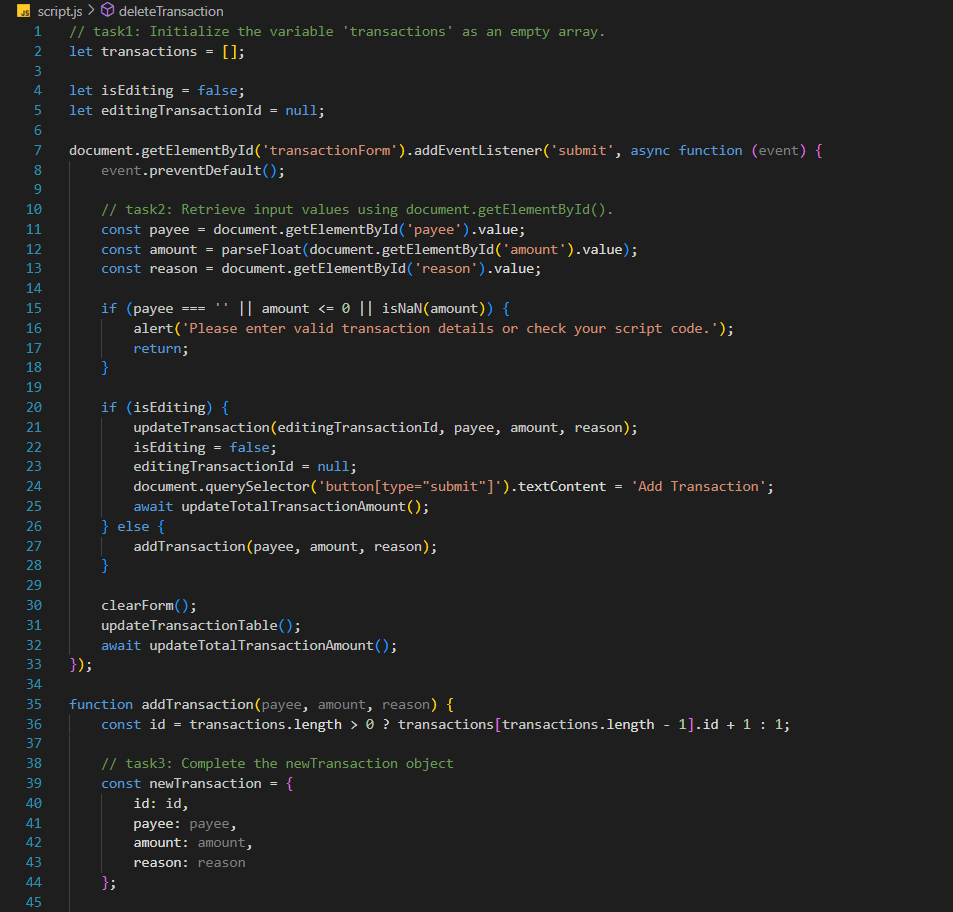
* **GitHub Link -** [**Devesh Jangir\_tasks2**](https://github.com/imdevesh01/rg-assignments/tree/task2JavaScriptAssignment/tasks2)
* **Files -**

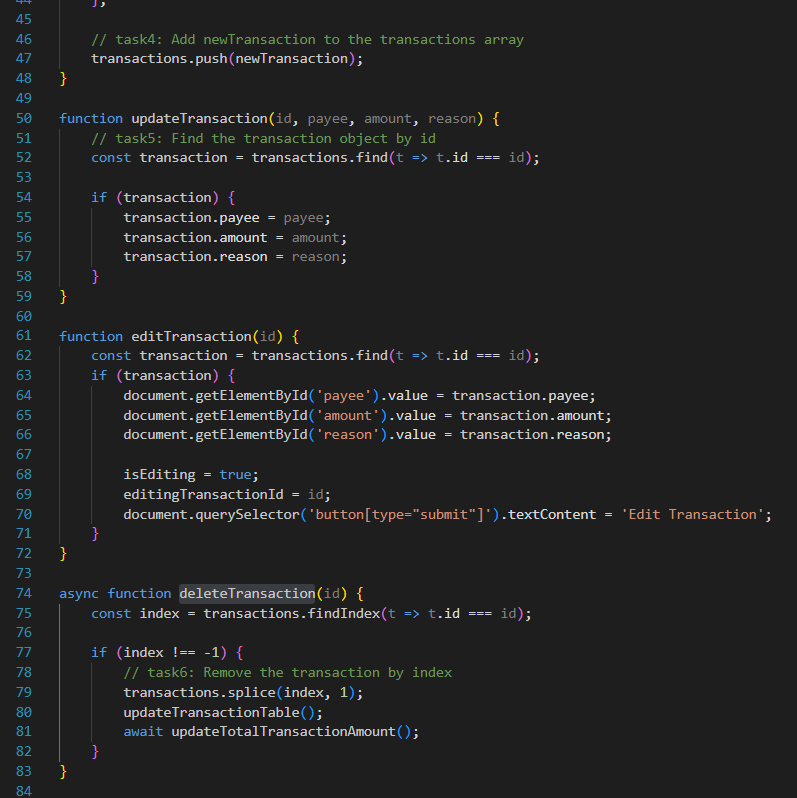
**Index.html**

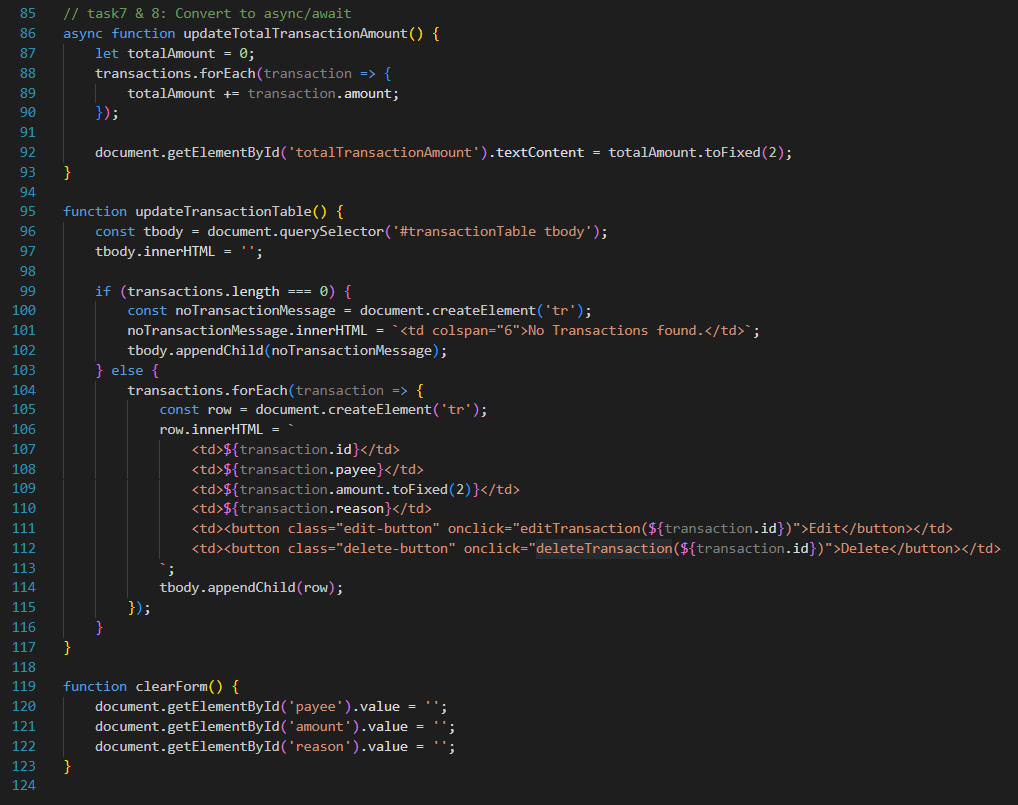
****

****

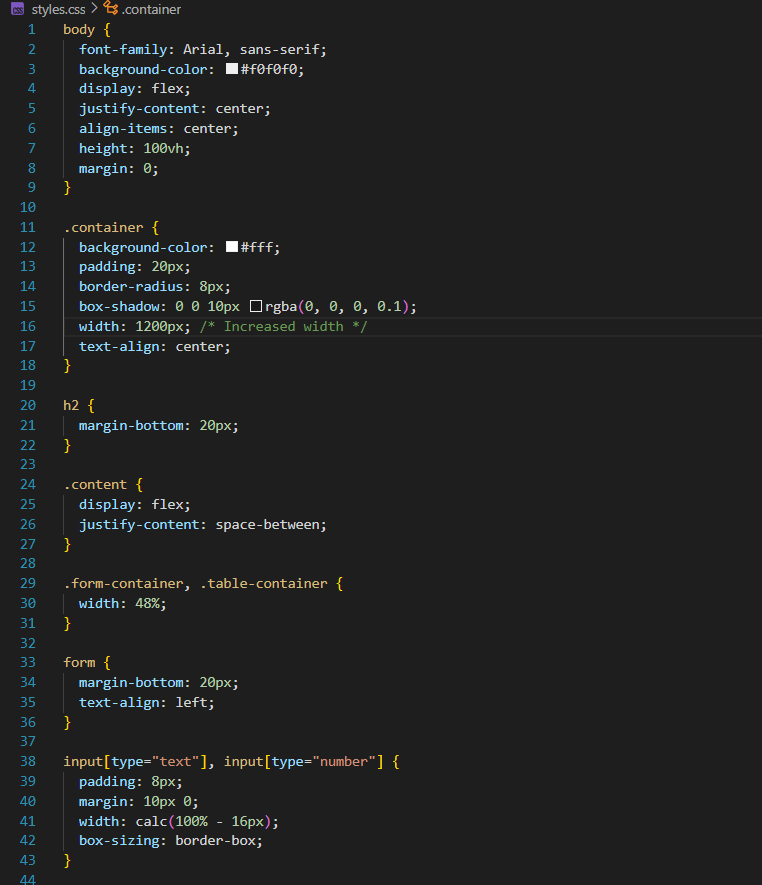
**Script.js**

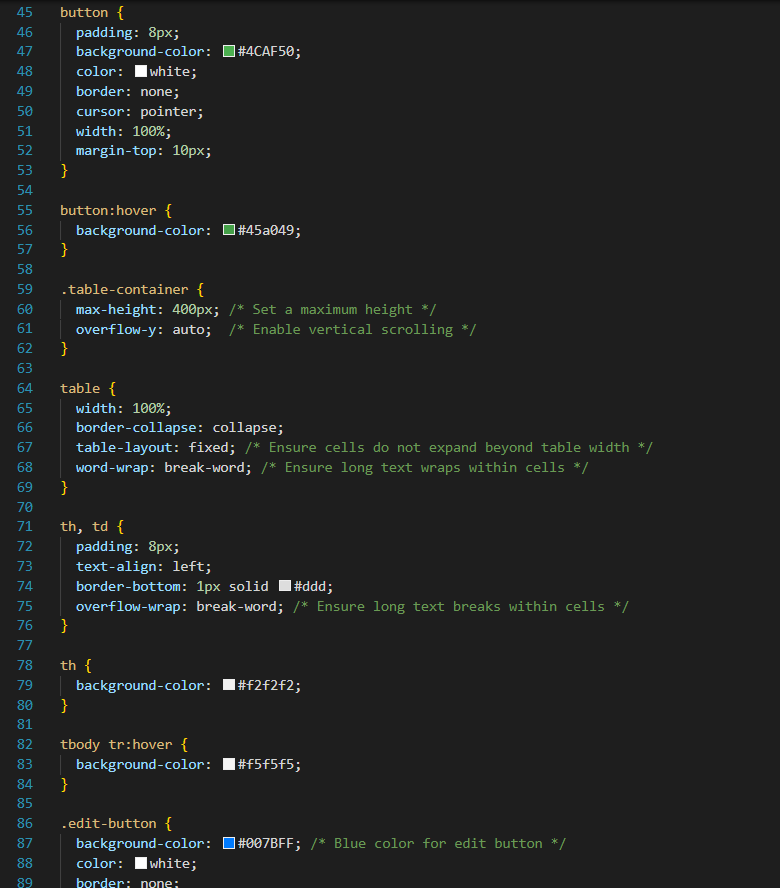
****

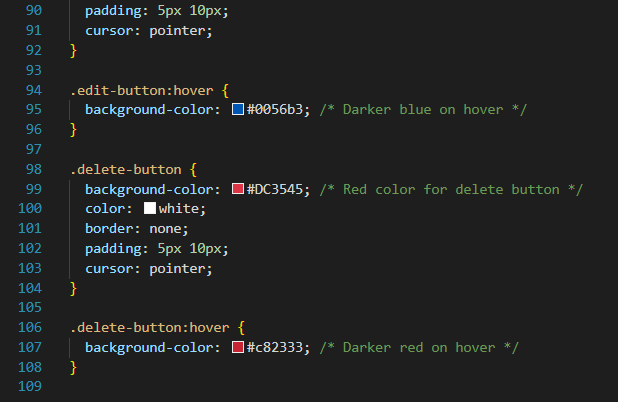
****

****

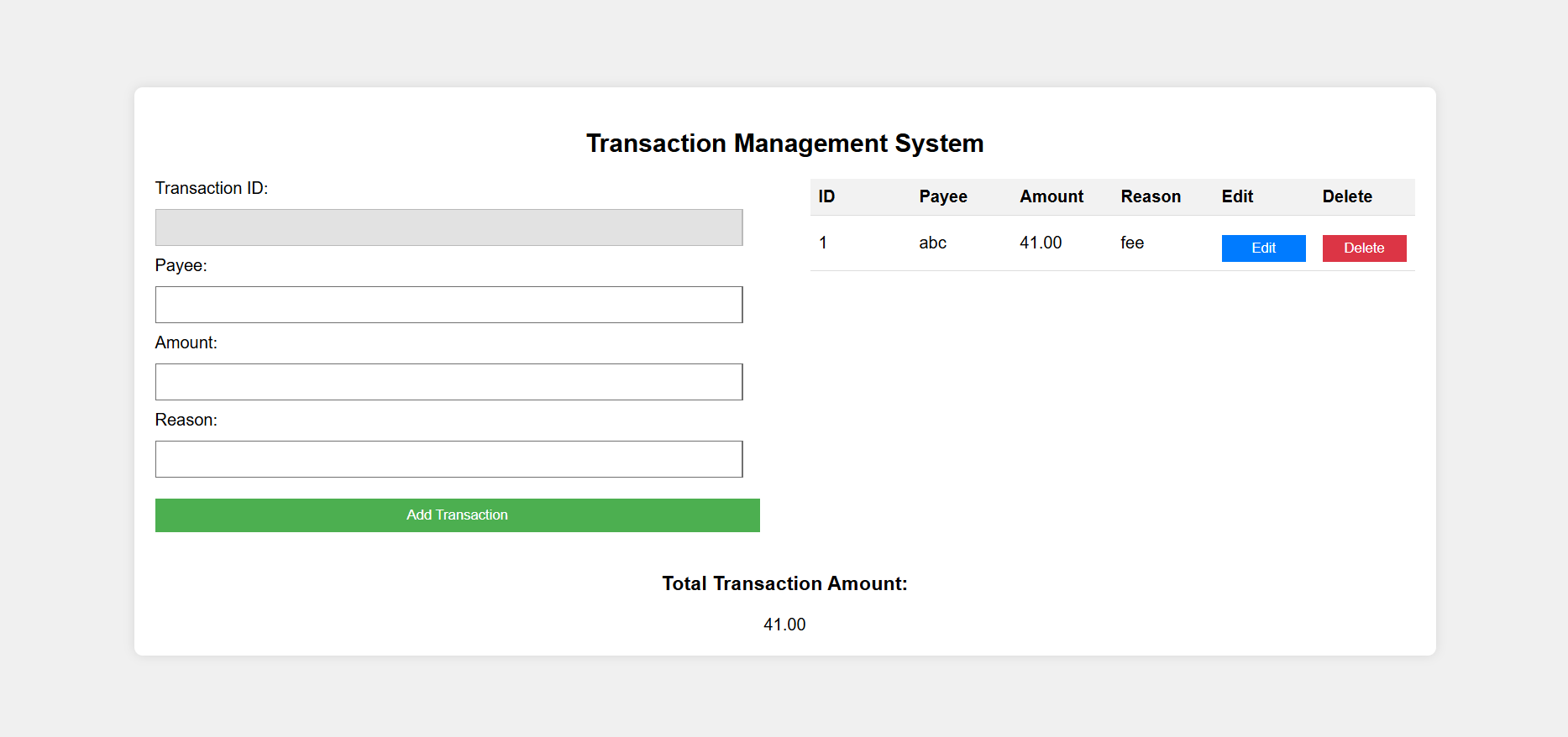
**Styles.css**

****

****

****

* **Code Result/Output -**

****

**Core Java Tasks**

**Please make sure to save/push all your code in the branch feature-java created in the previous week assignment as part of your github repo rg-assignments**

**Please share your output screenshots in the assignment document along with the github link for each question. Provide an explanation wherever possible as part of your response :-)**

**Complete GitHub Repository Link:** [**Repository(feature-java branch)**](https://github.com/imdevesh01/rg-assignments/tree/feature-java)

1) Given:

public class TaxUtil {

double rate = 0.15;

public double calculateTax(double amount) {

return amount \* rate;

}

}

Would you consider the method calculateTax() a 'pure function'? Why or why not?

If you claim the method is NOT a pure function, please suggest a way to make it pure.

**Ans.**

No, calculateTax() is not a pure function because it uses an instance variable rate which can be changed from outside. A pure function's output should only depend on its inputs and not on external or mutable state.

**Making it pure –**

public class TaxUtil {

public double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

Now calculateTax() depends only on its arguments — it's pure.

**GitHub Link:** [**TaxUtil.java**](https://github.com/imdevesh01/rg-assignments/blob/feature-java/TaxUtil.java)

2) What will be the output for following code?

**Ans.**

**GitHub Link:**  [**Super.Java**](https://github.com/imdevesh01/rg-assignments/blob/feature-java/Super.java)

class Super {

static void show() {

System.out.println("super class show method");

}

static class StaticMethods {

void show() {

System.out.println("sub class show method");

}

}

public static void main(String[] args) {

Super.show();

new Super.StaticMethods().show();

}

}

Execution--  
  
**1. Super.show();**

* This is a **static method call** on class Super.
* It prints:

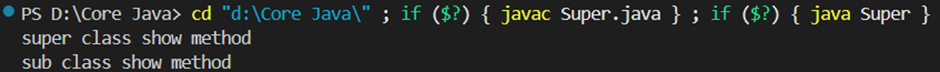
: super class show method

**2. new Super.StaticMethods().show();**

* Here, StaticMethods is a **static nested class** inside class Super.
* An object of StaticMethods is created using new Super.StaticMethods().
* Then the show() method (non-static) of the nested class is called.
* It prints:

: sub class show method

**Output –**



3) What will be the output for the following code?

**Ans.**

**GitHub Link:** [**ThisUse.java**](https://github.com/imdevesh01/rg-assignments/blob/feature-java/ThisUse.java)

class q3 {

int num = 20;

public void display() {

System.out.println("super class method");

}

}

public class ThisUse extends q3 {

int num;

public ThisUse(int num) {

this.num = num;

}

public void display() {

System.out.println("display method");

}

public void Show() {

this.display();

display();

System.out.println(this.num);

System.out.println(num);

}

public static void main(String[] args) {

ThisUse o = new ThisUse(10);

o.Show();

}

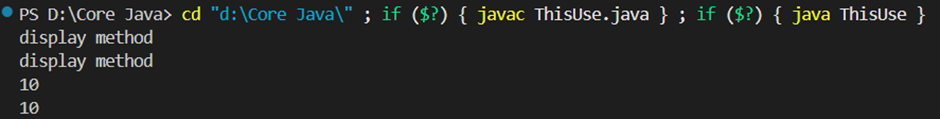
}

Execution--   
  
1. Constructor ThisUse(10) assigns this.num = 10.

2. Show() is called:

* this.display() → Calls overridden method in ThisUse: prints "display method".
* display() → Same as above (implicitly this.display()): prints "display method".
* System.out.println(this.num) → prints 10.
* System.out.println(num) → also prints 10.

**Output -**



4) What is the singleton design pattern? Explain with a coding example.

**Ans.** A **singleton** ensures that only one object of a class is created during runtime.

**GitHub Link:** [Singleton.java](https://github.com/imdevesh01/rg-assignments/blob/feature-java/Singleton.java)

**Code -**

public class Singleton {

private static Singleton instance;

private Singleton() {

// private constructor

}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton(); // lazy initialization

}

return instance;

}

}

**Use:**

public class Main {

public static void main(String[] args) {

Singleton obj1 = Singleton.getInstance();

Singleton obj2 = Singleton.getInstance();

System.out.println(obj1 == obj2); // true

}

}

5) How do we make sure a class is encapsulated? Explain with a coding example.

**Ans. Encapsulation** means hiding internal data and only exposing it via getters and setters.

**GitHub Link:** [**Student.java**](https://github.com/imdevesh01/rg-assignments/blob/feature-java/Student.java)

**Code -**

public class Student {

private int id;

private String name;

// Getters and Setters

public int getId() {

return id;

}

public void setId(int id) {

if (id > 0) this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

if (!name.isEmpty()) this.name = name;

}

}

6) Perform CRUD operation using ArrayList collection in an EmployeeCRUD class for the below Employee

class Employee{

private int id;

private String name;

private String department;

}

**Ans**.

**GitHub Link**: [EmployeeCRUD.java](https://github.com/imdevesh01/rg-assignments/blob/feature-java/EmployeeCRUD.java)

**Code -**

import java.util.\*;

class Employee {

private int id;

private String name;

private String department;

public Employee(int id, String name, String department) {

this.id = id;

this.name = name;

this.department = department;

}

public int getId() { return id; }

public String getName() { return name; }

public String getDepartment() { return department; }

public void setName(String name) { this.name = name; }

public void setDepartment(String department) { this.department = department; }

public String toString() {

return id + " - " + name + " - " + department;

}

}

public class EmployeeCRUD {

List<Employee> employees = new ArrayList<>();

public void create(Employee e) {

employees.add(e);

}

public void read() {

employees.forEach(System.out::println);

}

public void update(int id, String name, String dept) {

for (Employee e : employees) {

if (e.getId() == id) {

e.setName(name);

e.setDepartment(dept);

}

}

}

public void delete(int id) {

employees.removeIf(e -> e.getId() == id);

}

public static void main(String[] args) {

EmployeeCRUD crud = new EmployeeCRUD();

crud.create(new Employee(1, "Alice", "HR"));

crud.create(new Employee(2, "Bob", "IT"));

crud.read();

crud.update(2, "Bobby", "Engineering");

crud.read();

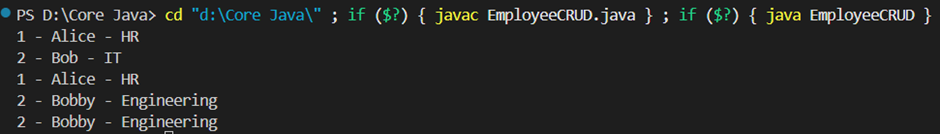
crud.delete(1);

crud.read();

}

}

**Output –**



7) Perform CRUD operation using JDBC in an EmployeeJDBC class for the below Employee

class Employee{

private int id;

private String name;

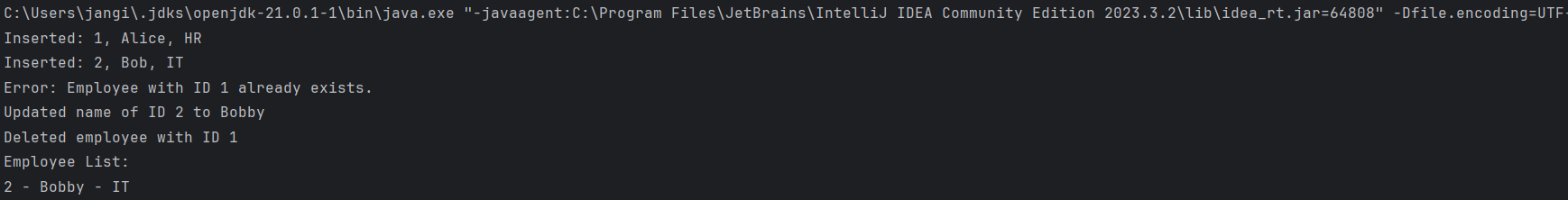
private String department;

}

**Ans:**

**GitHub Link:** [**EmployeeJDBC.java**](https://github.com/imdevesh01/rg-assignments/blob/feature-java/EmployeeJDBC.java)

**Code -**  
  
import java.sql.\*;  
  
public class EmployeeJDBC {  
 static final String *URL* = "jdbc:mysql://localhost:3306/testdb";  
 static final String *USER* = "root";  
 static final String *PASS* = "root";  
  
 public void create(int id, String name, String dept) {  
 try (Connection con = DriverManager.*getConnection*(*URL*, *USER*, *PASS*);  
 PreparedStatement ps = con.prepareStatement("INSERT INTO employees VALUES (?, ?, ?)")) {  
  
 ps.setInt(1, id);  
 ps.setString(2, name);  
 ps.setString(3, dept);  
 ps.executeUpdate();  
 System.*out*.println("Inserted: " + id + ", " + name + ", " + dept);  
  
 } catch (SQLIntegrityConstraintViolationException e) {  
 System.*out*.println("Error: Employee with ID " + id + " already exists.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void read() {  
 try (Connection con = DriverManager.*getConnection*(*URL*, *USER*, *PASS*);  
 Statement st = con.createStatement();  
 ResultSet rs = st.executeQuery("SELECT \* FROM employees")) {  
  
 System.*out*.println("Employee List:");  
 while (rs.next()) {  
 System.*out*.println(rs.getInt(1) + " - " + rs.getString(2) + " - " + rs.getString(3));  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void update(int id, String name) {  
 try (Connection con = DriverManager.*getConnection*(*URL*, *USER*, *PASS*);  
 PreparedStatement ps = con.prepareStatement("UPDATE employees SET name=? WHERE id=?")) {  
  
 ps.setString(1, name);  
 ps.setInt(2, id);  
 int rows = ps.executeUpdate();  
 if (rows > 0) {  
 System.*out*.println("Updated name of ID " + id + " to " + name);  
 } else {  
 System.*out*.println("No employee found with ID " + id);  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void delete(int id) {  
 try (Connection con = DriverManager.*getConnection*(*URL*, *USER*, *PASS*);  
 PreparedStatement ps = con.prepareStatement("DELETE FROM employees WHERE id=?")) {  
  
 ps.setInt(1, id);  
 int rows = ps.executeUpdate();  
 if (rows > 0) {  
 System.*out*.println("Deleted employee with ID " + id);  
 } else {  
 System.*out*.println("No employee found with ID " + id);  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Main method for demo  
 public static void main(String[] args) {  
 EmployeeJDBC db = new EmployeeJDBC();  
  
 db.create(1, "Alice", "HR");  
 db.create(2, "Bob", "IT");  
 db.create(1, "Charlie", "Sales");  
 db.update(2, "Bobby");  
 db.delete(1);  
 db.read();  
 }  
}

**Output -   
  
**