**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 :-)**



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?

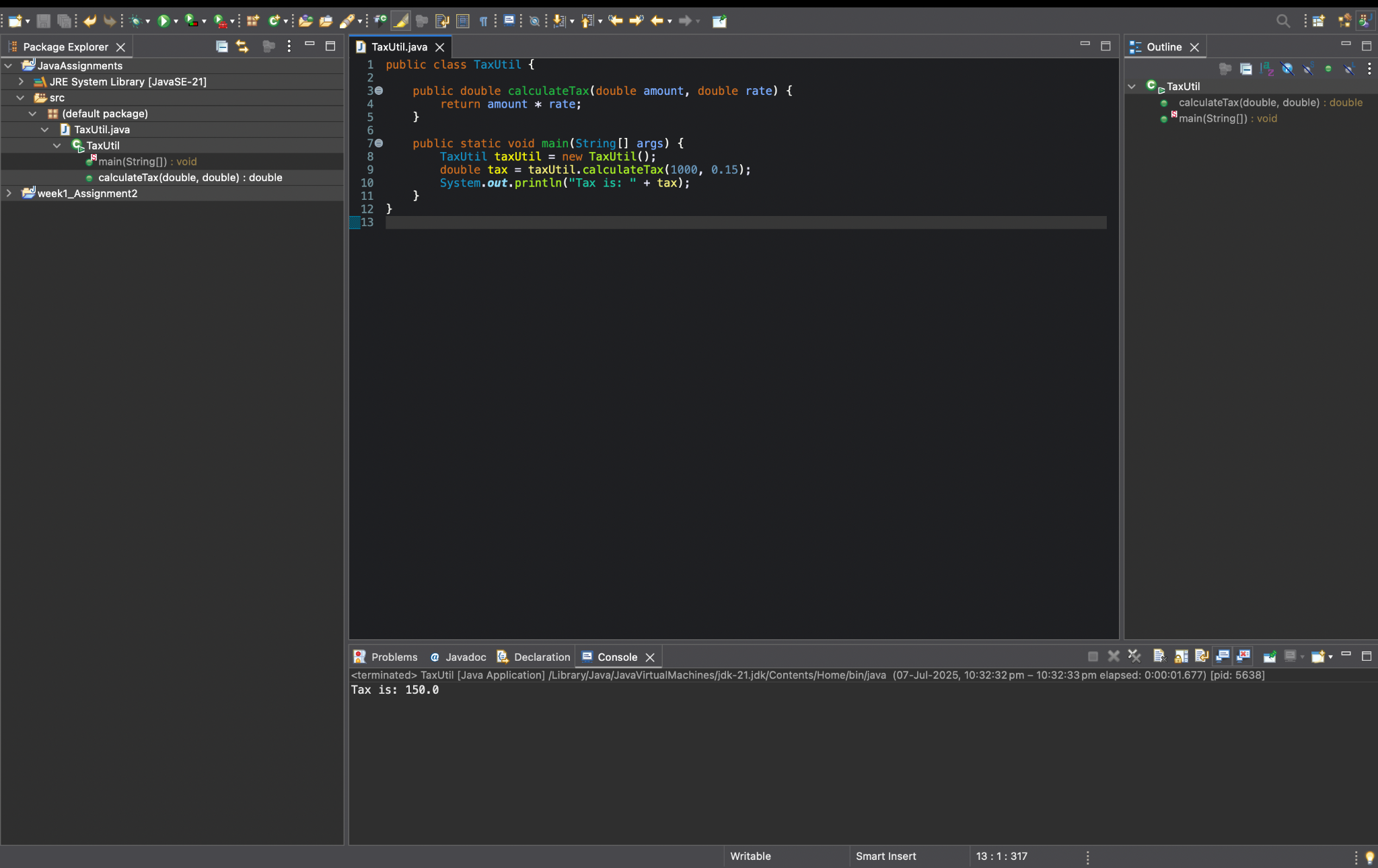
If you claim the method is NOT a pure function, please suggest a way to make it pure.

* Because it **depends on an instance variable rate**, which makes it rely on external state. A pure function should always give the same output for the same input **without depending on or modifying external data**.
* **Make it pure like this:**

public double calculateTax(double amount, double rate) {

return amount \* rate;

}

* it depends only on its parameters.
* 

2)

What will be the output for following code?

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)

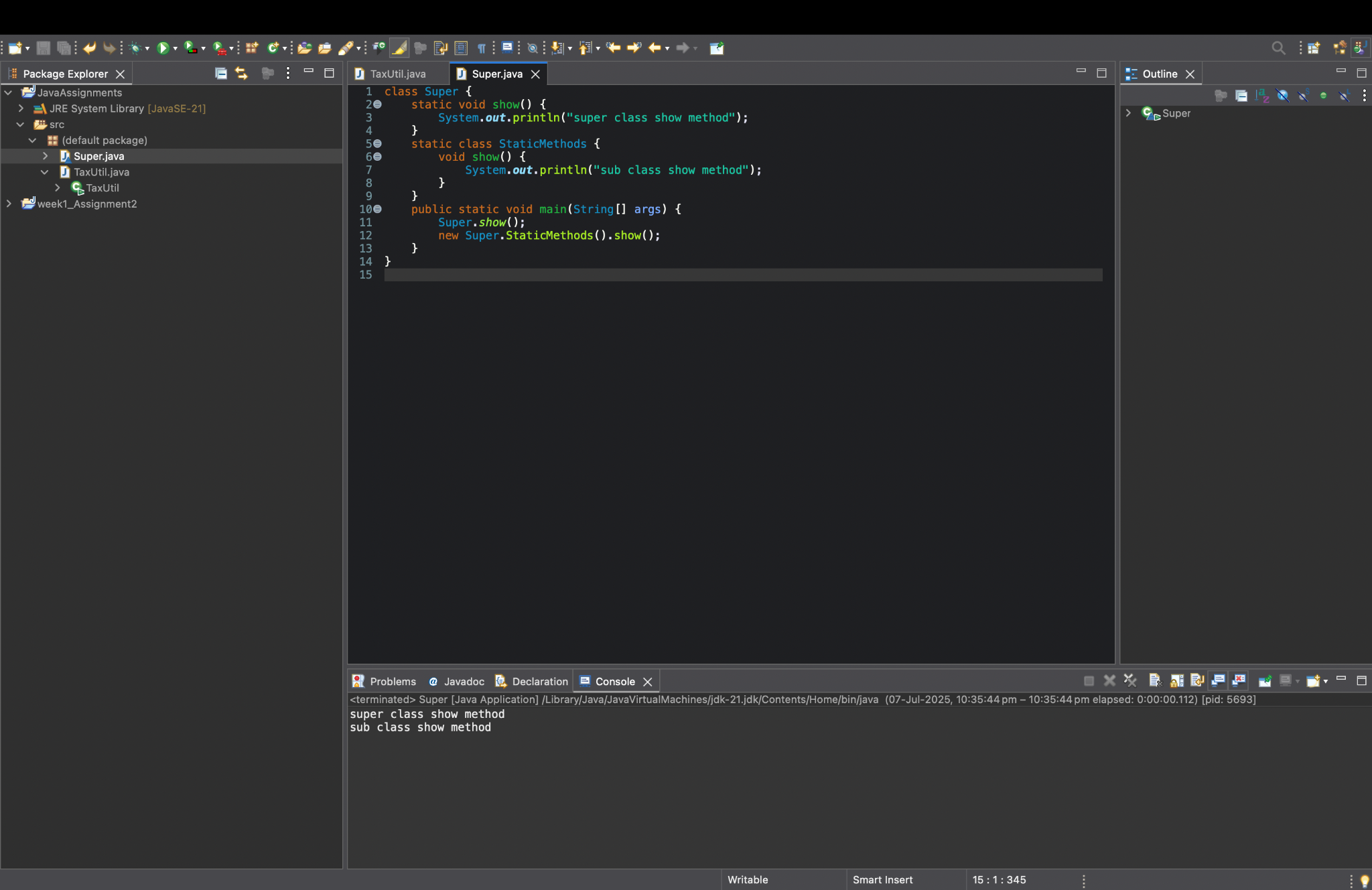
{

Super.show();

new Super.StaticMethods().show();

}

}



* Super.show(); calls the **static method** → prints super class show method.
* new Super.StaticMethods().show(); creates an object of the **static nested class** and calls its **instance method** → prints sub class show method.
* Static methods belong to the class, not objects.
* Static nested classes don’t need an outer class instance.

3)

What will be the output for the following code?

class Super

{

int num=20;

public void display()

{

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

}

}

public class ThisUse extends Super

{

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);

}

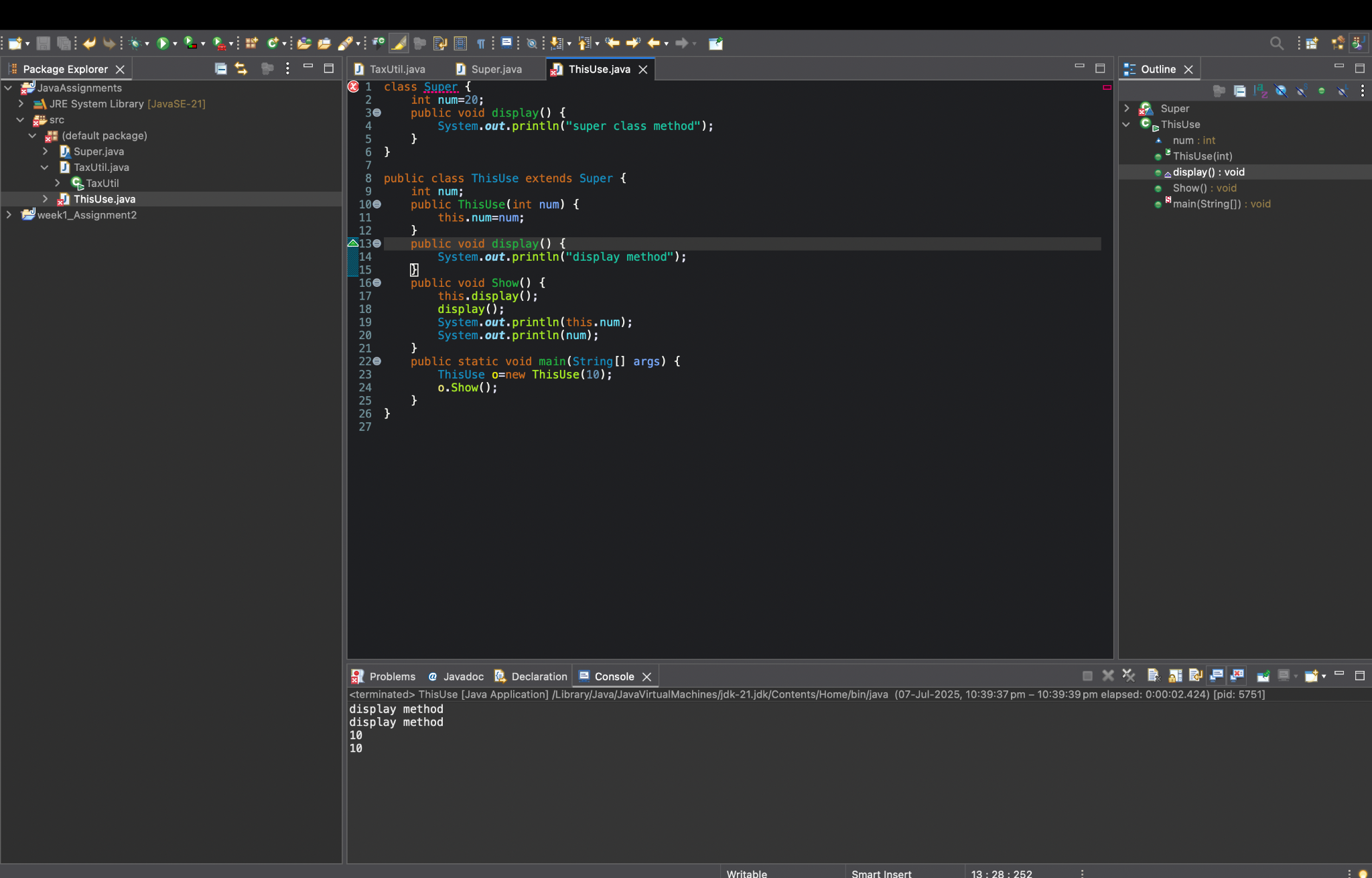
public static void main(String[]args)

{

ThisUse o=new ThisUse(10);

o.show();

}

}

this.display() and display() both call the overridden display() in ThisUse.  
 this.num and num print 10 from the subclass. Super’s num (20) is hidden.

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

* A **singleton pattern** ensures a class has **only one instance** and provides a global point of access to it.
* **Example:**

public class Singleton {

private static Singleton instance;

private Singleton() {} // private constructor

public static Singleton getInstance() {

if (instance == null) {

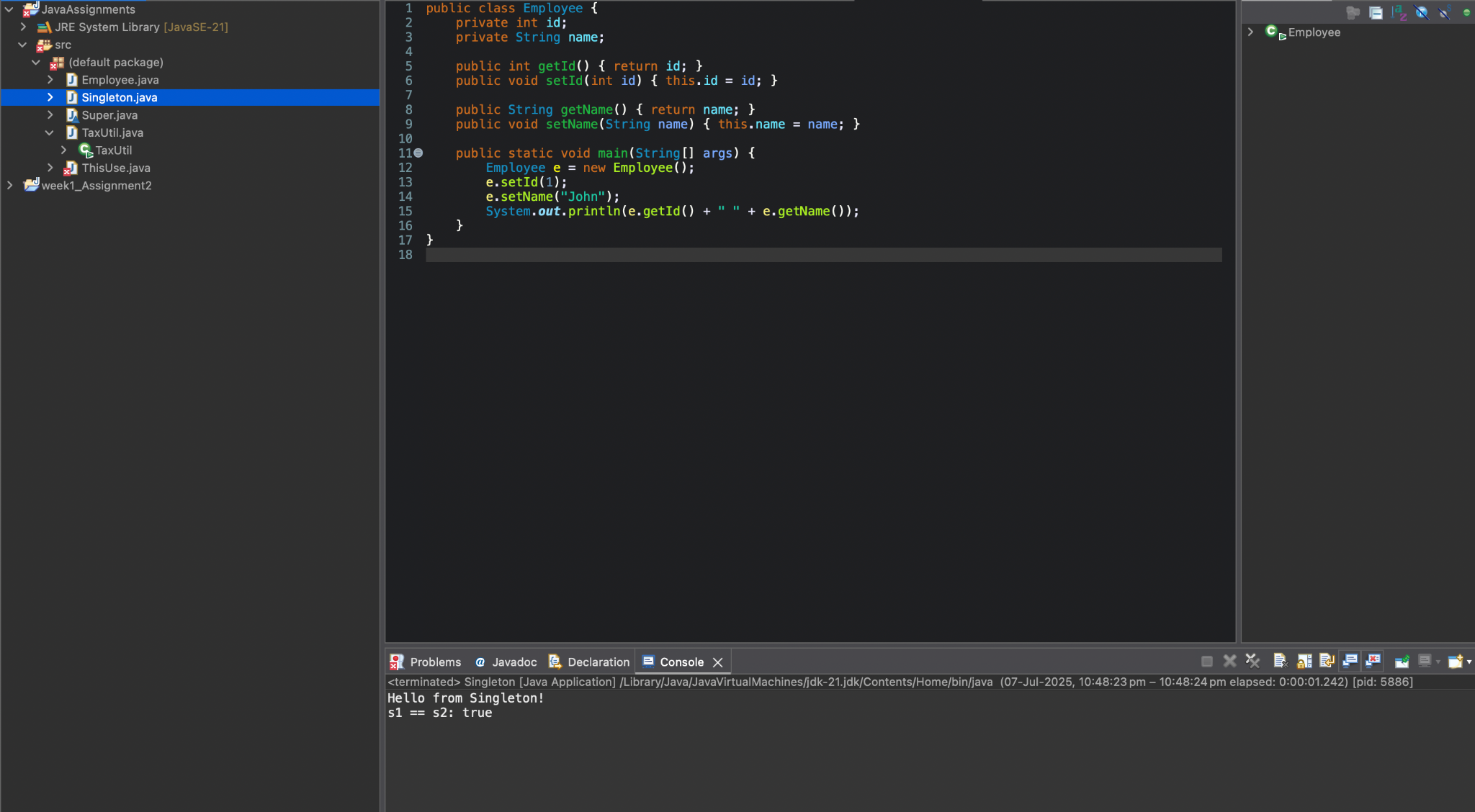
instance = new Singleton();

}

return instance;

}

}

* We can call Singleton.getInstance() to always get the **same object**
* ****

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

**Encapsulation** means keeping data (fields) private and exposing them via **getters/setters**.

* **Example:**

public class Employee {

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

* Fields are private.
* Access is only through public methods.  
  

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;

}

* **Employee class**

public class Employee {

private int id;

private String name;

private String department;

// constructors, getters & setters

}

* **EmployeeCRUD class**

import java.util.\*;

public class EmployeeCRUD {

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

public void addEmployee(Employee e) {

employees.add(e);

}

public void viewEmployees() {

for(Employee e : employees) {

System.out.println(e.getId() + " " + e.getName() + " " + e.getDepartment());

}

}

public void updateEmployee(int id, String newName, String newDept) {

for(Employee e : employees) {

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

e.setName(newName);

e.setDepartment(newDept);

}

}

}

public void deleteEmployee(int id) {

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

}

}



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;

}

What actually happens?

* It connects to an **H2 [file based] DB**, creates an Employee table.
* **Inserts** a record, then **reads** & prints it.
* **Updates** the department, then **deletes** the record.
* Shows how to perform full **CRUD operations using JDBC**.

import java.sql.\*;

public class EmployeeJDBC {

private int id;

private String name;

private String department;

// Constructors

public EmployeeJDBC() {}

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

this.id = id;

this.name = name;

this.department = department;

}

// Getters & setters

public int getId() { return id; }

public String getName() { return name; }

public String getDepartment() { return department; }

public void setId(int id) { this.id = id; }

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

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

// Database connection

private static final String URL = "jdbc:h2:~/testdb";

private static final String USER = "";

private static final String PASSWORD = "";

public void createTable() throws SQLException {

String sql = "CREATE TABLE Employee (id INT PRIMARY KEY, name VARCHAR(100), department VARCHAR(100))";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

Statement stmt = conn.createStatement()) {

stmt.execute(sql);

System.out.println("Table created");

}

}

public void addEmployee(EmployeeJDBC emp) throws SQLException {

String sql = "INSERT INTO Employee (id, name, department) VALUES (?, ?, ?)";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, emp.getId());

pstmt.setString(2, emp.getName());

pstmt.setString(3, emp.getDepartment());

pstmt.executeUpdate();

System.out.println("Employee added");

}

}

public void getEmployee(int id) throws SQLException {

String sql = "SELECT \* FROM Employee WHERE id = ?";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, id);

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

System.out.println("ID: " + rs.getInt("id"));

System.out.println("Name: " + rs.getString("name"));

System.out.println("Department: " + rs.getString("department"));

} else {

System.out.println("Employee not found");

}

}

}

public void updateEmployee(EmployeeJDBC emp) throws SQLException {

String sql = "UPDATE Employee SET name = ?, department = ? WHERE id = ?";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setString(1, emp.getName());

pstmt.setString(2, emp.getDepartment());

pstmt.setInt(3, emp.getId());

int updated = pstmt.executeUpdate();

System.out.println(updated + " row(s) updated");

}

}

public void deleteEmployee(int id) throws SQLException {

String sql = "DELETE FROM Employee WHERE id = ?";

try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

PreparedStatement pstmt = conn.prepareStatement(sql)) {

pstmt.setInt(1, id);

int deleted = pstmt.executeUpdate();

System.out.println(deleted + " row(s) deleted");

}

}

public static void main(String[] args) {

EmployeeJDBC db = new EmployeeJDBC();

try {

db.createTable();

EmployeeJDBC e1 = new EmployeeJDBC(1, "Alia", "IT");

db.addEmployee(e1);

db.getEmployee(1);

e1.setName("Sanjana S");

e1.setDepartment("HR");

db.updateEmployee(e1);

db.getEmployee(1);

db.deleteEmployee(1);

db.getEmployee(1);

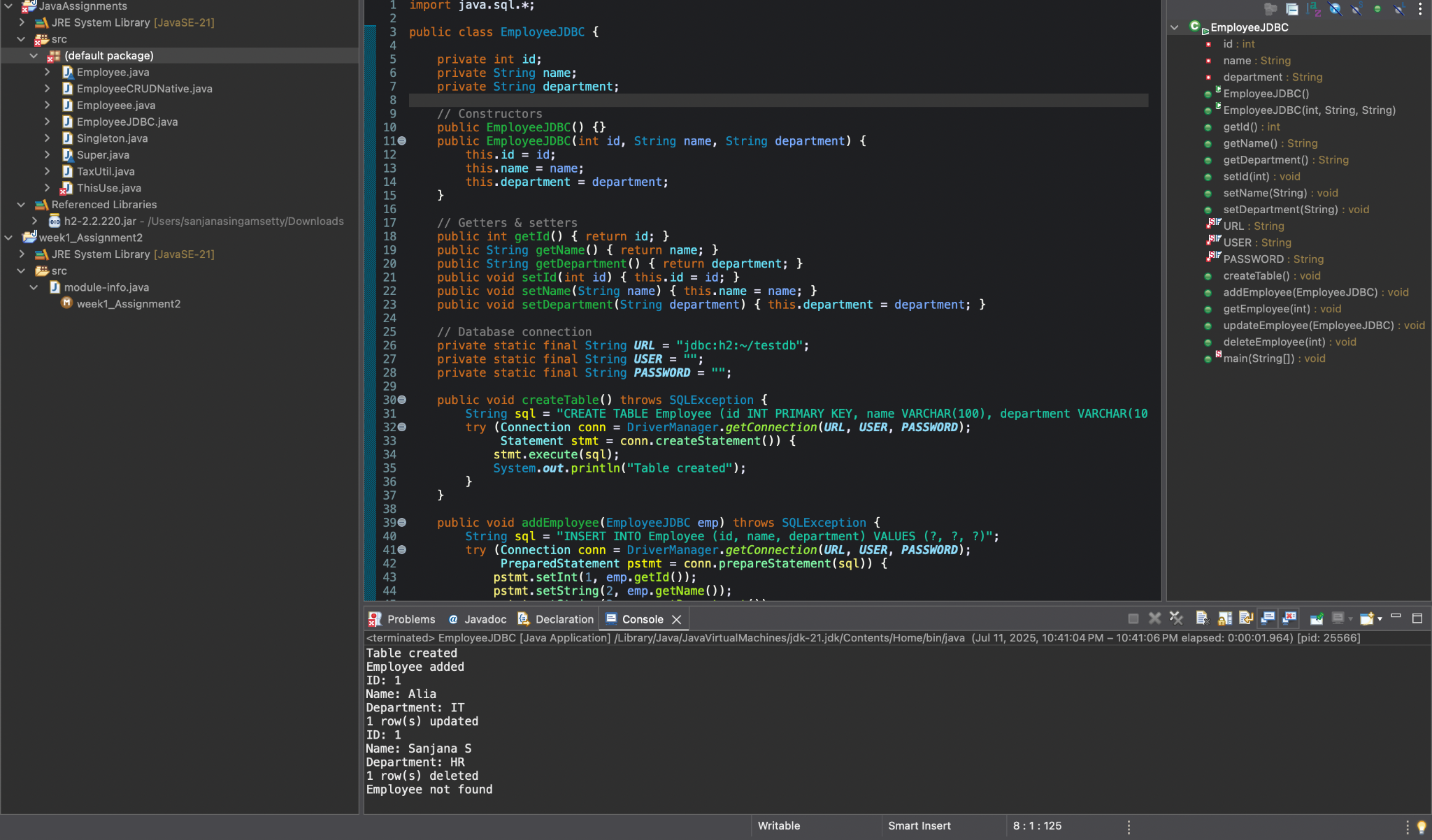
} catch (SQLException ex) {

ex.printStackTrace();

}

}

}



* This program uses **H2’s in-memory database** via JDBC to quickly create, store, and manipulate data entirely in RAM, perfect for testing without creating files.
* Here i did not create an in-memory database — I created a file-based database..