**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 :-)**

1. **Given:**

**public class TaxUtil {**

**double rate = 0.15;**

**public double calculateTax(double amount) {**

**return amount \* rate;**

**}**

**}**

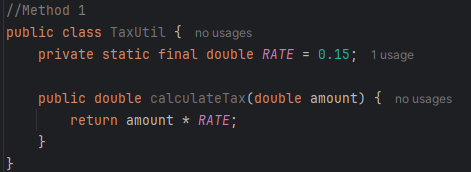
**Would you consider the method calculateTax() a 'pure function'? Why or why not? If you claim the method is NOT a pure function, please suggest a way to make it pure.**

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/TaxUtil.java>

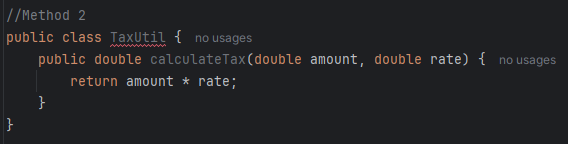
No, the method calculateTax() is not a pure function since it is dependent on external mutable variable “rate”.

We can make it pure in two ways:

1. Making rate as a STATIC & final variable:



1. Passing the “rate” as a parameter of calculateTax() function:



**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();**

**}**

**}**

Output:

super class show method

sub class show method

Explanation:

* *Super.show()* calls the static method of the outer class.
* *new Super.StaticMethods().show()* calls the non-static method of the static inner class.

**Github link :** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/Super.java>

**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();**

**}**

**}**

Output:

display method

display method

10

10

Explanation:

* this.display() and display() both call the overridden method.
* this.num and num both refer to the child class variable num, which is 10.

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/ThisUse.java>

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

The Singleton pattern ensures that only one instance of a class is created and provides a global point of access to that instance.

Example: In a School, there is only one Principal, whichever class you are in, you talk and listen to the same one principal

public class Principal {

private static Principal *Principal*;

private Principal() {

System.*out*.println("One Principal");

}

public static Principal getPrincipal() {

if (*Principal* == null) {

*Principal* = new Principal(); // only one will ever be created

}

return *Principal*;

}

public void giveAnnouncement() {

System.*out*.println("Do your work!");

}

}

Now, whenever anyone tries to create any instance of this class, the same instance will be created.

**Github link:** <https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/Principal.java>

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

Encapsulation is hiding internal data using private access. We can chose to expose it to other classes through getters/setters.

Example:

public class Student {

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;

}

}

**Github link:**

<https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/Student.java>

**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;**

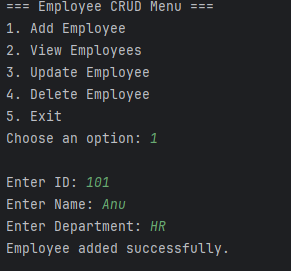
**}**

**Github Link:**

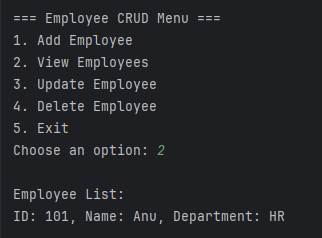
<https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/EmployeeCRUD.java>

This Java program allows the user to add, view, update, and delete employees using a simple console menu-driven interface. Performing CRUD operations.

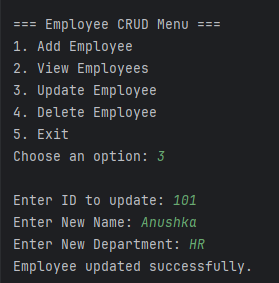
**Creating:**

****

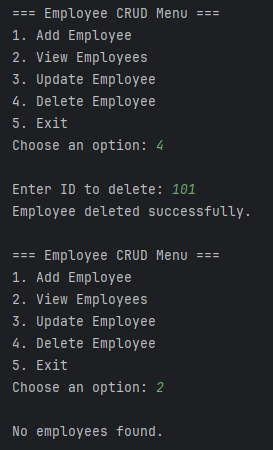
**Reading:**

****

**Updating:**

****

**Deleting:**

****

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

**class Employee{**

**private int id;**

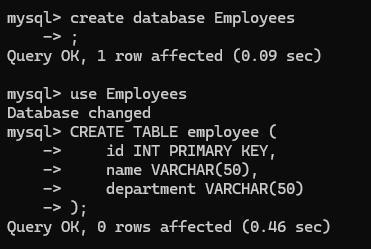
**private String name;**

**private String department;**

**}**

**Github link:**

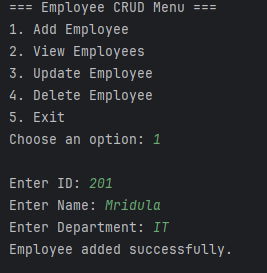
<https://github.com/ANUSHKA1509/rg-assignments/blob/feature-java/src/EmployeeJDBC.java>

Here, first we create a MySQL db named Employees and create a table called Employees to store their id, name and department values.  
  


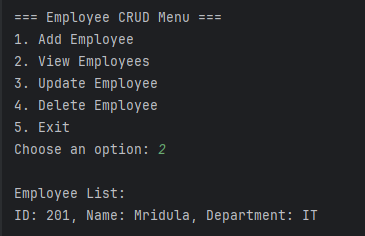
Next, this program uses JDBC to connect Java with a MySQL database.

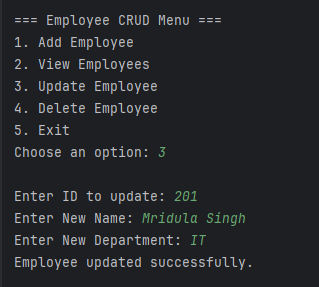
It uses PreparedStatement and ResultSet to insert, update, view, and delete employee records through a console menu.

**Creating:**

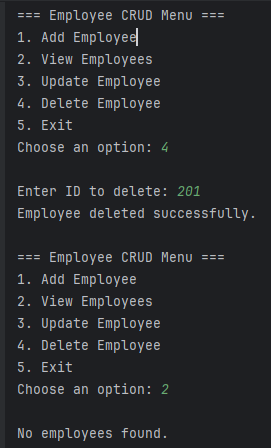
****

**Reading:**

**  
  
Updating:**

****

**Deleting:**

****