**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.

**Solution:**The calculateTax method is not a pure function because it depends on the instance variable **rate**, making it non-deterministic if the rate changes which can lead to different outputs for the same input. To make it pure, pass rate as an argument to the method, ensuring it only relies on its input parameters and not on any external or instance variables. This guarantees that it always produces the same output for the same input and has no side effects.

**Making calculateTax a Pure Function**  
public class TaxUtil {

// No instance variable for rate

public double calculateTax(double amount, double rate) {

return amount \* rate;

}

}

**Github Link for Code:** [**Q1 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q1)

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();

}

}

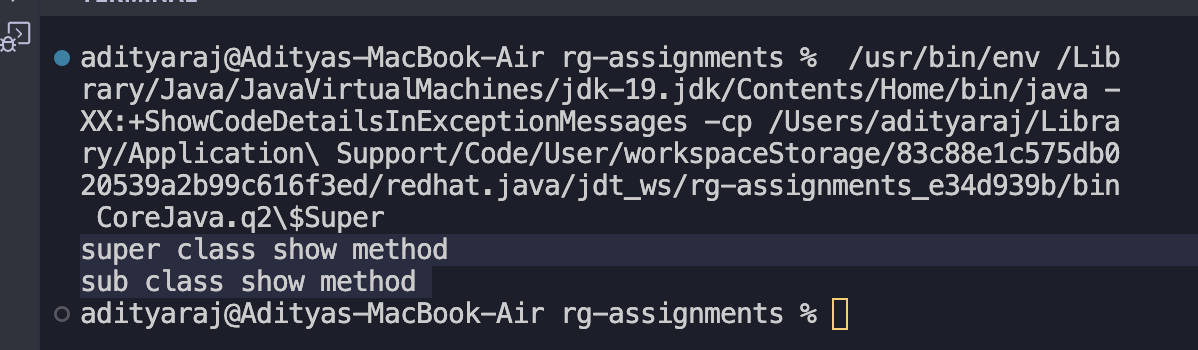
**OUTPUT**super class show method

sub class show method

**Github Link for Code:** [**Q2 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q2)

**main Method**:

* Calls the static method show of the Super class: Super.show(),which prints: super class show method;
* Creates an instance of the StaticMethods class and calls its show method: new Super.StaticMethods().show(), which prints sub class show method;



3) What will be the output for 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)

{

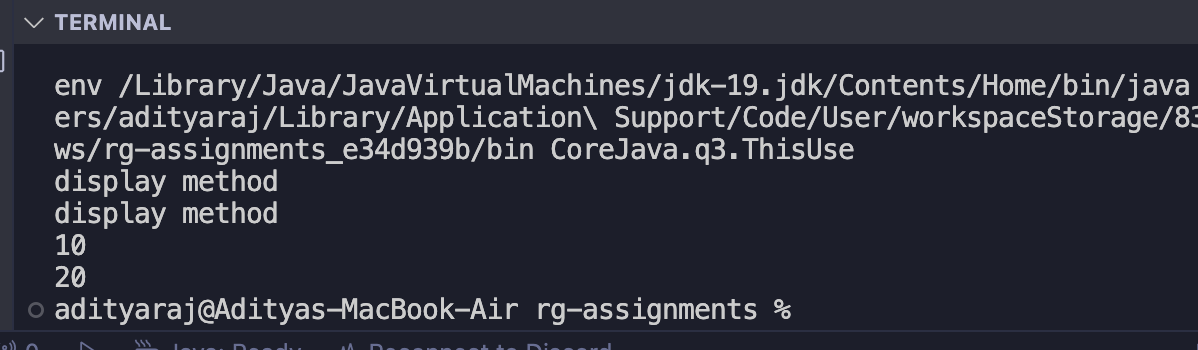
ThisUse o=new ThisUse(10);

o.show();

}

}

**OUTPUT:**

****

**display method**

**display method**

**10**

**20**

**Github Link for Code:** [**Q3 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q3)

**main Method**:

* this.display(); calls the display method defined in ThisUse (overriding the display method in Super Class). So it prints: **display method**
* display(); also calls the display method defined in ThisUse (same as the previous call). It prints: **display method**
* this.num refers to the num variable in ThisUse and outputs 10.
* super.num refers to the num variable inherited from the Super Class and outputs 20.

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

Solution:

The Singleton Design Pattern is a design pattern used in software engineering to ensure that a class has only one instance throughout the application and provides a global point of access to that instance. It is often used when a single instance of a class is needed to coordinate actions across the system.

**Github Link for Code:** [**Q4 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q4)

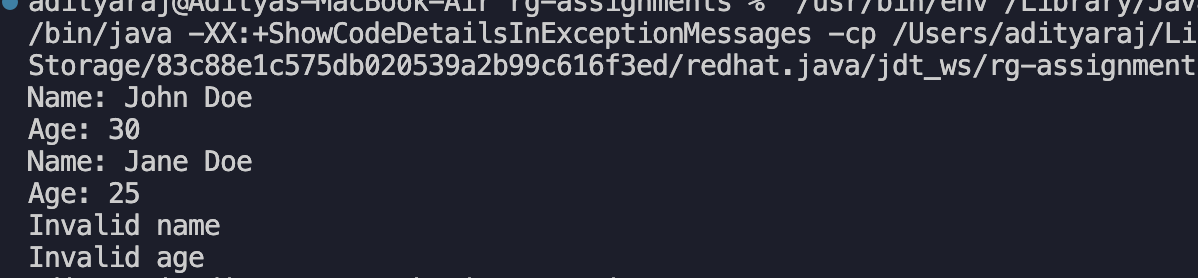
5) How do we make sure a class is encapsulated? Explain with a coding example.

**Solution:**

To make sure that our class is encapsulated by using these key concepts:

* **Private Variables**: Make the variables private to restrict direct access from outside the class.
* **Public Getters and Setters**: Provide public methods to access and modify private variables.
* **Controlled Access**: Ensure that any operation on the data is controlled and validated through these methods.

**Github Link for Code:** [**Q5 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q5)



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

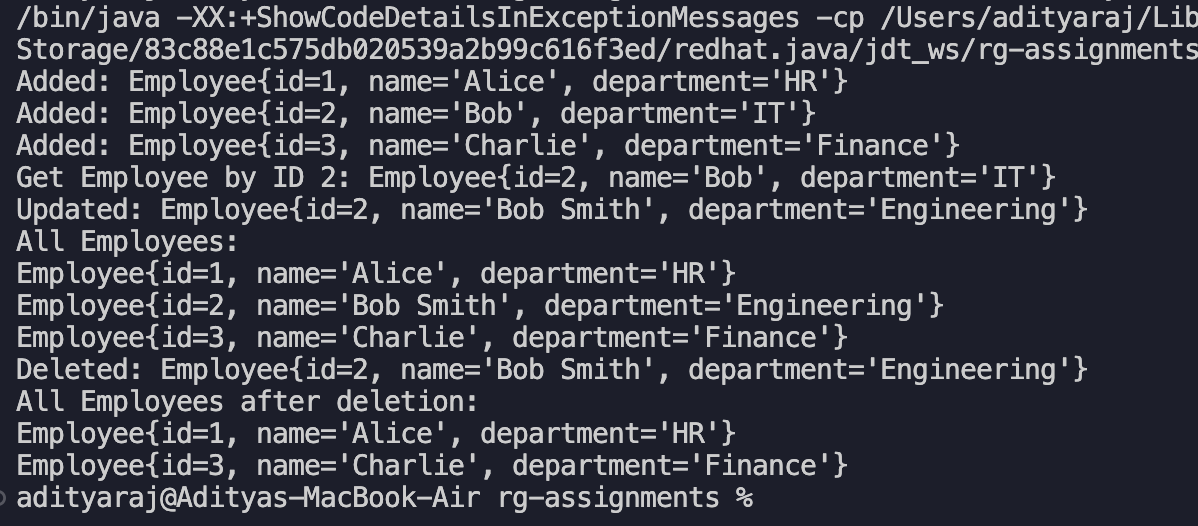
class Employee{

private int id;

private String name;

private String department;

}



**Github Link for Code:** [**Q6 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q6)

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

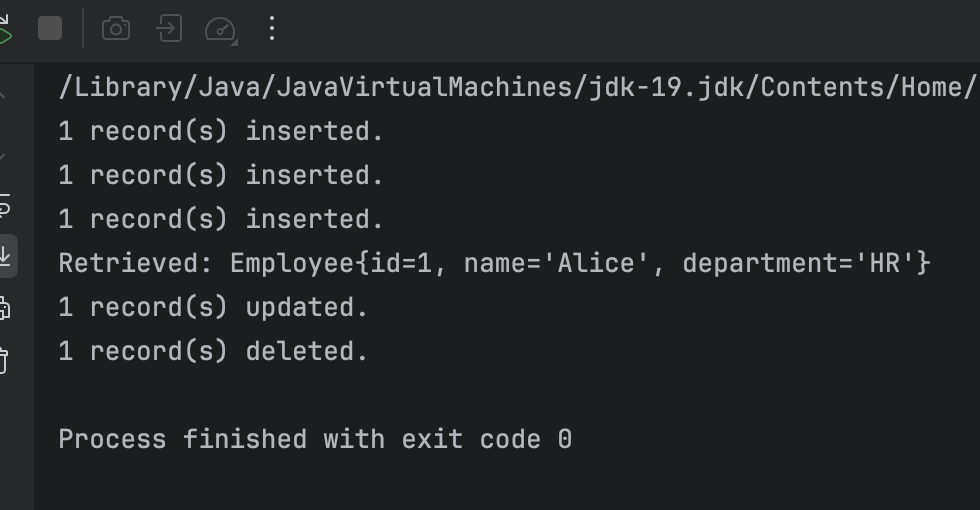
class Employee{

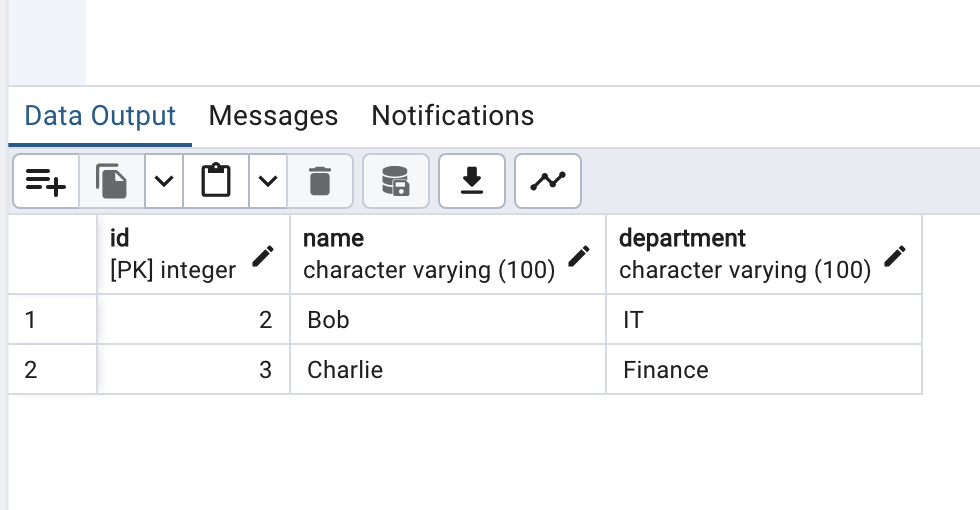
private int id;

private String name;

private String department;

}





**Github Link for Code:** [**Q7 Code Link**](https://github.com/Raj160201/rg-assignments/tree/feature-java/CoreJava/q7)