Core Java Assignment
Harshit Gautam

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.

The given method is not a pure function because the method is depending on the mutable external instance variable “rate” which could be changed by any other method resulting in the situation where the output for the same input might be different - violating one of the rules of pure function “Always returns the same output for the same input”.

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();
}
}
```

Ans:

The outer show method is a static method so it can be called directly without instantiating an object. But, the inner show method is a non-static method so an object of the inner "StaticMethods" class should be created then show method should be called.

Creation of object => new Super.StaticMethods()
Calling of inner show() with the above created object => new Super.StaticMethods().show()

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();
}
}
```

Ans:

Both “this.display()” and “display()” are calling the same overridden method in the child class. Same with “this.num” and “num()”.

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

Ans:
Singleton is a creational design pattern that ensures that a class has only one instance, while providing a global access point to this instance.

```
public class Database {

   // Step 1: Create a private static instance (initially null)
    private static Database instance;

  // Step 2: Make the constructor private to prevent instantiation from outside
    private Database() {
        System.out.println("Database connection created.");
    }

    // Step 3: Provide a public method to get the instance
    public static Database getInstance() {
        if (instance == null) {
            instance = new Database();
        }
        return instance;
    }
    // Example business logic method
    public void query(String sql) {
        System.out.println("Executing query: " + sql);
    }
}

public class Application {
    public static void main(String[] args) {
        Database db1= Database.getInstance();
        db1.query("SELECT * FROM users");

        Database db2= Database.getInstance();
        db2.query("SELECT * FROM orders");

        if (db1 == db2) {
            System.out.println("Both foo and bar refer to the same Database instance.");
        }
    }
}
```

To make only a single instance we should prevent initialization from any exterior code so we made the constructor private in step2 - so that only internal code within the class can create an object.

Now to make this object available to the exterior code only through a public method we made it private and we made it static so that it belongs to the class itself but not to any specific object. This means there's only one copy of a static variable or method shared among all instances of the class.

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

Ans: A class is encapsulated to prevent direct access(hiding the internal state of an object) and requiring all interaction to be performed through an object's methods.

Encapsulation is achieved by:
=> Declaring variables as private.
=> Providing public getter and setter methods to access or modify the private fields.

```
public class BankAccount {
    // Step 1: Make fields private
    private                  String
    accountHolder;        private
    double balance;
    // Constructor
    public BankAccount(String accountHolder, double initialBalance) {
        this.accountHolder = accountHolder;
        if (initialBalance >= 0) {
            this.balance = initialBalance;
        }
    }

    // Step 2: Provide public getters and methods for safe access

    public String getAccountHolder() {
        return accountHolder;
    }

    public double getBalance() {
        return balance;
    }

    // Controlled method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    // Controlled method to withdraw money
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) { balance
            -= amount; System.out.println("Withdrawn:
            $" + amount);
        } else {
            System.out.println("Invalid or insufficient balance.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        BankAccount account = new BankAccount("John Doe", 500.0);
        System.out.println("Account Holder: " + account.getAccountHolder());
        System.out.println("Initial Balance: $" + account.getBalance());

        account.deposit(200);
        account.withdraw(100);
        account.withdraw(700); // Will be rejected due to insufficient balance

        System.out.println("Final Balance: $" + account.getBalance());
    }
}
```

Github link: https://github.com/durgavinay8/rg-assignments/blob/feature-

java/BankAccount.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;
}
```

Ans:

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;
}
```