NAME : VIPUL RAJ

UID : 23BCS10592

SECTION : KRG-3(B)

Q.1 :

SRP and OCP are two important principles of the Solid principles of Object-Oriented Programming. These principles help developers design software that is easy to understand, maintain, and extend.

- **SRP (Solid Responsibility Principle) :**

This principle states that a class should only have one responsibility. Furthermore, it should only have one reason to change.

This means a class should perform only one specific task. If a class performs multiple tasks, it becomes difficult to maintain and modify.

Example :

Violation Of SRP:

```
class Employee {
   void calculateSalary() { //Calculating the Salary
   }
   void saveEmployee() { //Adding New Employee
   }
   void generateReport(){   // report generation logic
   }
}
```

In the Above Example we can notice that the Employee class has more than one responsibility.

Following SRP :

```
class Employee {
   void calculateSalary() { // salary calculation logic
   }
}
class EmployeeRepository {
   void saveEmployee() { // database logic
   }
}
class EmployeeReport {
   void generateReport() { // report generation logic
   }
}
```

In the Above Example We are creating different class for different Responsibility which is a good example of SRP.

- **OCP (Open/Closed Principle):**

The second most important principle is open closed principle, it says:

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification. This means we should be able to add new functionality without changing existing code.

Example:

```
interface Discount {
   double calculate(double amount);
}
class StudentDiscount implements Discount {
   public double calculate(double amount) {
      return amount * 0.10;
   }
}
```

```
class SeniorDiscount implements Discount {
    public double calculate(double amount) {
        return amount * 0.20;
    }
}
```

In this example we can add new features without changing the existing ones.

The **Single Responsibility Principle** ensures that each class focuses on a single task, making the system easier to maintain.

The **Open/Closed Principle** allows software to grow by adding new features without changing existing code.

Together, SRP and OCP help in developing clean, flexible, and maintainable software systems.

Q.2 :

Sometime we Design a Software system where Single Responsibility Principle (SRP) and Open/Closed Principle (OCP) are frequently violated.

1. **Violations in Single Responsibility Principle (SRP):**

A violation of SRP occurs when a class performs multiple unrelated tasks.

Example:

```
class Employee {
    void calculateSalary() { //Calculating the Salary
    }
    void saveEmployee() { //Adding New Employee
    }
    void generateReport(){   // report generation logic
    }
}
```

In the Above Example we can notice that the Employee class has more than one responsibility.

The Employee class has three responsibilities:

1. Salary logic (calculating Salary)
2. Adding operations
3. Report logic (printing)

So we are Performing more than one operation into a single class which violates the rule of SRP.

Fix for SRP Violation

Each responsibility should be placed in a separate class.

Example:

```
class Employee {
   void calculateSalary() { // salary calculation logic
   }
}
class EmployeeRepository {
   void saveEmployee() { // database logic
   }
}
class EmployeeReport {
   void generateReport() { // report generation logic
   }
}
```

In the Above Example we can notice that we are creating more than one class and each class performing some certain task which is obeying the rule of SRP.

2. Violations in Open/Closed Principle (OCP)

An OCP violation happens when existing code must be modified to add new functionality.

Example:

```
class Payment {
   double processPayment(String mode, double amount) {
      if (mode.equals("Cash")) {
         return amount;
      } else if (mode.equals("Card")) {
         return amount + 50;
      }
      return amount;
   }
}
```

In the above Example we can see that in order to add any new method (e.g. UPI) we need to change the complete processPayment method which violates the rule of OCP.

Fix for OCP Violation :

Use abstraction (interfaces or abstract classes) and polymorphism.

Example:

```
interface Payment {
   double pay(double amount);
}

class CashPayment implements Payment {
   public double pay(double amount) {
      return amount;
   }
}

class CardPayment implements Payment {
   public double pay(double amount) {
      return amount + 50;
   }
}
```

In the above example we can add any other method with disturbing the previous ones which is obeying the rule of OCP