

# SOLID PRINCIPLES ASSIGNMENT

In the rolex project I have chosen four classes for showing solid principles.

- Cart
  - Bill
  - Payment
  - Product
- 
- **Single Responsibility Principle (SRP):**
    - CartServices, BillCalculator, BillGenerator: These classes have a single responsibility each. For example, handling cart operations, calculating the bill total, and generating bill details.
    - OnlinePayment and OfflinePayment handle online and offline payments, respectively.

```
package com.ilp.service;  
import com.ilp.entity.Bill;  
public class BillCalculator {  
    public double calculateTotal(Bill bill) {  
  
        return bill.getSubtotal() + 1000.0;  
    }  
}
```

Here the BillCalculator has just a single responsibility to calculate the total. For example if I want to generate the bill I need to create another class called BillGenerator to generate the bill.

```

package com.ilp.service;
import com.ilp.entity.Bill;
public class BillGenerator {
    public void generateBillDetails(Bill bill) {
        System.out.println("Bill generated");
    }
}

```

Therefore the BillCalculator just Calculates the subtotal and total whereas the BillGenerator generates the bill.

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

- PaymentMethod Interface, OnlinePayment, CreditCardPayment, UpiPayment: These classes and interfaces adhere to the Open/Closed Principle.
- New payment methods can be added without modifying existing code by creating classes that implement PaymentMethod interface.

```

package com.ilp.service;
import com.ilp.interfaces.PaymentMethod;
public class CreditCardPayment implements PaymentMethod {
    public void processPayment(double totalAmount) {
        System.out.println("Payment of $" + totalAmount + " made using Credit Card");
    }
}

```

```

package com.ilp.service;
import com.ilp.interfaces.PaymentMethod;
public class NetBanking implements PaymentMethod{
    public void processPayment(double totalAmount){

```

```
        System.out.println("Payment of $" + totalAmount + " made using Net  
Banking");  
    }  
}
```

Here CreditCardPayment and NetBanking Implements the interface PaymentMethod.

- **Liskov Substitution Principle (LSP):**

- OnlinePayment and OfflinePayment are subclasses of Payment, and they are used interchangeably with the parent class Payment. Similarly, UpiPayment is another implementation of PaymentMethod, and it is used in the Payment class.
- This usage adheres to the Liskov Substitution Principle because you can substitute instances of the derived classes (OnlinePayment, OfflinePayment, UpiPayment) for instances of the base class (Payment) without affecting the behavior of the program.

- **Interface Segregation Principle (ISP):**

- The interfaces (BillingOperations, CartOperations, PaymentMethod, PaymentOperations) are specific and not too broad. Each interface has methods related to its specific responsibility, adhering to the Interface Segregation Principle.

- **Dependency Inversion Principle (DIP):**

- The code appears to depend on abstractions (interfaces) rather than concrete implementations. For example, the Payment class depends on the PaymentOperations interface. This follows the Dependency Inversion Principle.

## Situations where this code can violate the solid principles:

### Single Responsibility Principle (SRP);

- Suppose the CartServices class not only handles cart operations but also contains methods for calculating the bill total and generating bill details. In this case, it would violate SRP, as the class is responsible for more than one thing.

```
package com.ilp.service;
import com.ilp.entity.Bill;
public class BillCalculator {
    public double calculateTotal(Bill bill) {

        return bill.getSubtotal() + 1000.0;
    }
    public void addToCart() {
        System.out.println("Product added to the cart");
    }
    public void removeFromCart() {
        System.out.println("Product removed from the cart");
    }
}
```

This is violating the SRP principle as the BillCalculator is responsible for both calculating the bill total, adding and removing products from the cart.

According to SRP a class should have only a single responsibility.

### Open/Closed Principle (OCP):

- If adding a new payment method involves modifying existing code in the Payment class instead of extending it, the Open/Closed Principle is violated.

This may occur if the existing code has conditional statements for each payment method.

**Liskov Substitution Principle (LSP):**

- LSP may be violated if a subclass fails to provide the expected behavior of the base class. For example, if OnlinePayment or OfflinePayment introduces behavior that is not consistent with the Payment class, substituting instances might lead to unexpected results.

**Interface Segregation Principle (ISP):**

- Suppose the BillingOperations interface includes methods not relevant to all implementing classes. For instance, if BillingOperations forces a class to implement a method related to cart operations, it violates ISP.

**Dependency Inversion Principle (DIP):**

- The Payment class directly instantiates the OnlinePayment class, forming a tight coupling between the high-level module (Payment) and the low-level module (OnlinePayment). If the high-level module directly depends on a low-level module rather than abstractions, DIP is violated.