SOLID Principles: Assignment — Open-Closed Principle (OCP) in a Payment Module

**Objective**: prove you can extend functionality without modifying shipped code.

#### **Starter code**

File src/main/java/legacy/PaymentProcessor.java

public class PaymentProcessor {

public void processPayment(String paymentType) {

if (paymentType.equals("CreditCard")) {

System.out.println("Processing credit card payment...");

} else if (paymentType.equals("PayPal")) {

System.out.println("Processing PayPal payment...");

} else {

System.out.println("Unsupported payment method.");

}

}

}

Clone or copy this file into your project; no other code is provided.

#### **Tasks**

1. **Analyse the violation**: list every reason the class might change and explain how each risk breaks OCP. Write your findings in analysis/ocp\_problems.md.
2. **Design for extension**: decide which architecture or design pattern best solves the problem (e.g., Strategy, Factory, Dependency-Injection). Sketch the class diagram in analysis/ocp\_design.png (hand-drawn or digital).
3. **Refactor**: implement your pattern inside src/main/java/clean/ so that adding a new payment method is possible without touching existing classes. Supply at least three payment types (CreditCard, PayPal, GooglePay).
4. **Add a new feature**: create ApplePayPayment without editing previously committed files; show it works in Main.java.
5. **Reflection**: in reflection.md answer  
   * What concrete steps made the code open for extension?
   * Which architecture pattern did you choose and why?
   * What trade-offs (complexity, files) came with the new design?

#### **Deliverables**

analysis/ocp\_problems.md

analysis/ocp\_design.png

src/main/java/clean/\*\* ← refactored code

src/main/java/legacy/\*\* ← original code (unchanged)

src/main/java/Main.java ← demo running four payment methods

reflection.md

README.md ← how to run

#### **Solution (reference implementation)**

/\* PaymentMethod.java \*/

public interface PaymentMethod {

void processPayment();

}

/\* CreditCardPayment.java \*/

public class CreditCardPayment implements PaymentMethod {

public void processPayment() { System.out.println("Processing credit card payment..."); }

}

/\* PayPalPayment.java \*/

public class PayPalPayment implements PaymentMethod {

public void processPayment() { System.out.println("Processing PayPal payment..."); }

}

/\* GooglePayPayment.java \*/

public class GooglePayPayment implements PaymentMethod {

public void processPayment() { System.out.println("Processing Google Pay payment..."); }

}

/\* ApplePayPayment.java \*/

public class ApplePayPayment implements PaymentMethod {

public void processPayment() { System.out.println("Processing Apple Pay payment..."); }

}

/\* PaymentProcessor.java (clean module) \*/

public class PaymentProcessor {

private final PaymentMethod paymentMethod;

public PaymentProcessor(PaymentMethod paymentMethod) { this.paymentMethod = paymentMethod; }

public void process() { paymentMethod.processPayment(); }

}

/\* Main.java \*/

public class Main {

public static void main(String[] args) {

new PaymentProcessor(new CreditCardPayment()).process();

new PaymentProcessor(new PayPalPayment()).process();

new PaymentProcessor(new GooglePayPayment()).process();

new PaymentProcessor(new ApplePayPayment()).process();

}

}

Pattern used: **Strategy** (each PaymentMethod is a strategy; PaymentProcessor is the context).