**Курс:** Шаблоны проектирования приложений

**Тема:** Модуль 04 Принципы проектирования

*Цель задания:*

**Произведите корректную (правильную) по вашему мнению реализацию с применением принципа Single-Responsibility Principle (SRP):**

В этом примере класс **Order** отвечает за несколько вещей: хранение данных о заказе, расчет стоимости заказа с учетом скидок, обработку платежа и отправку уведомления пользователю.

public class Order {

private String productName;

private int quantity;

private double price;

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

// Метод для рассчета общей стоимости с учетом скидок

public double calculateTotalPrice() {

return quantity \* price \* 0.9;

}

// Метод для обработки платежа

public void processPayment(String paymentDetails) {

System.out.println("Payment processed using: " + paymentDetails);

}

// Метод для отправки уведомления по email

public void sendConfirmationEmail(String email) {

System.out.println("Confirmation email sent to: " + email);

}

}

**Проблемы:**

* Класс Order нарушает принцип SRP, так как он отвечает за несколько вещей: расчет цены, обработку платежа и отправку уведомлений. Это усложняет код и делает его менее гибким.

В этом примере Вам необходимо разделить ответственность между несколькими классами. Класс Order должен отвечает только за хранение данных о заказе. Другие задачи, такие как расчет цены, обработка платежа и отправка уведомлений, должны быть делегированы другим классам.

**Произведите корректную (правильную) по вашему мнению реализацию с применением принципа Open-Closed Principle, OCP:**

Расчет зарплаты сотрудников

В этом примере класс EmployeeSalaryCalculator нарушает принцип OCP, так как для добавления новой логики расчета зарплаты приходится изменять код метода CalculateSalary.

public class Employee {

private String name;

private double baseSalary;

private String employeeType; // "Permanent", "Contract", "Intern"

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getBaseSalary() {

return baseSalary;

}

public void setBaseSalary(double baseSalary) {

this.baseSalary = baseSalary;

}

public String getEmployeeType() {

return employeeType;

}

public void setEmployeeType(String employeeType) {

this.employeeType = employeeType;

}

}

public class EmployeeSalaryCalculator {

public double calculateSalary(Employee employee) {

switch (employee.getEmployeeType()) {

case "Permanent":

return employee.getBaseSalary() \* 1.2; // Постоянный сотрудник получает 20% бонус

case "Contract":

return employee.getBaseSalary() \* 1.1; // Контрактник получает 10% бонус

case "Intern":

return employee.getBaseSalary() \* 0.8; // Стажер получает 80% от базовой зарплаты

default:

throw new UnsupportedOperationException("Тип сотрудника не поддерживается");

}

}

}**Проблемы:**

* Если нужно добавить новый тип сотрудника, например, "Freelancer", придется изменить метод CalculateSalary. Это нарушает принцип OCP, так как мы изменяем уже существующий код, что может привести к ошибкам.

**Произведите корректную (правильную) по вашему мнению реализацию с применением принципа** **Interface Segregation Principle, ISP:**

Работа с принтерами

В этом примере интерфейс IPrinter содержит методы для различных типов принтеров: обычного принтера, сканера и факса. Но что если какой-то принтер поддерживает только печать и сканирование, но не поддерживает отправку факсов? Такой принтер будет вынужден реализовывать метод Fax, который ему не нужен.

// Интерфейс IPrinter

public interface IPrinter {

void print(String content);

void scan(String content);

void fax(String content);

}

// Класс AllInOnePrinter, реализующий все функции интерфейса IPrinter

public class AllInOnePrinter implements IPrinter {

@Override

public void print(String content) {

System.out.println("Printing: " + content);

}

@Override

public void scan(String content) {

System.out.println("Scanning: " + content);

}

@Override

public void fax(String content) {

System.out.println("Faxing: " + content);

}

}

// Класс BasicPrinter, реализующий только функцию печати

public class BasicPrinter implements IPrinter {

@Override

public void print(String content) {

System.out.println("Printing: " + content);

}

@Override

public void scan(String content) {

throw new UnsupportedOperationException();

}

@Override

public void fax(String content) {

throw new UnsupportedOperationException();

}

}**Проблемы:**

* Класс BasicPrinter вынужден реализовывать методы Scan и Fax, даже если он их не поддерживает. Это нарушает принцип ISP, так как он зависит от методов, которые ему не нужны.

Вам необходимо разделить IPrinter на несколько более мелких интерфейсов, каждый из которых описывает отдельную функциональность.

**Произведите корректную (правильную) по вашему мнению реализацию с применением принципа** **Dependency-Inversion Principle, DIP:**

Система уведомлений

В этом примере класс NotificationService напрямую зависит от конкретных классов EmailSender и SmsSender. Если нужно добавить новый тип уведомления, например, через мессенджер, придётся изменить класс NotificationService.

public class EmailSender {

public void sendEmail(String message) {

System.out.println("Email sent: " + message);

}

}

public class SmsSender {

public void sendSms(String message) {

System.out.println("SMS sent: " + message);

}

}

public class NotificationService {

private EmailSender emailSender = new EmailSender();

private SmsSender smsSender = new SmsSender();

public void sendNotification(String message) {

emailSender.sendEmail(message);

smsSender.sendSms(message);

}

}

**Проблемы:**

* Класс NotificationService жестко связан с конкретными реализациями EmailSender и SmsSender. Если потребуется изменить способ отправки уведомлений или добавить новый способ, придется изменять код NotificationService.