**Лабораторная работа №74-75**

**«Использование структурных шаблонов»**

**Цель работы:** Научиться использовать структурные шаблоны.

**Образовательные результаты:**

Студент должен

уметь:

- создавать программу по разработанному алгоритму как отдельный модуль;

- уметь выполнять оптимизацию и рефакторинг программного кода;

- создавать программный код обработчиков событий.

знать:

- анализ и проектирование программных решений;  
- основные методы отладки программного обеспечения;  
- технологии тестирования программных продуктов.

**Краткие теоретические и учебно-методические материалы**

**по теме лабораторной работы**

**Задания для практического занятия:**

Используя структурный шаблон, выполните следующие задания:

1)Ваше приложение работает с разными платежными системами (Stripe, PayPal, Crypto), у каждой свой API (разные методы, параметры, форматы ответов). Сделать единый интерфейс для работы с платежами (processPayment(amount)), чтобы была поддержка новых платежных систем не ломала существующий код, клиентский код не зависел от конкретных реализаций, можно было добавлять логирование или валидацию для всех платежей.

using System;

public interface IPaymentProcessor

{

void ProcessPayment(decimal amount);

}

public class StripePaymentProcessor : IPaymentProcessor

{

public void ProcessPayment(decimal amount)

{

Console.WriteLine($"Processing payment of {amount} with Stripe.");I

}

}

public class PayPalPaymentProcessor : IPaymentProcessor

{

public void ProcessPayment(decimal amount)

{

Console.WriteLine($"Processing payment of {amount} with PayPal.");I

}

}

public class CryptoPaymentProcessor : IPaymentProcessor

{

public void ProcessPayment(decimal amount)

{

Console.WriteLine($"Processing payment of {amount} with Crypto.");

}

}

public class PaymentService

{

private readonly IPaymentProcessor \_paymentProcessor;

public PaymentService(IPaymentProcessor paymentProcessor)

{

\_paymentProcessor = paymentProcessor;

}

public void ExecutePayment(decimal amount)

{

// Валидация

if (amount <= 0)

{

throw new ArgumentException("Amount must be greater than zero.");

}

// Логирование

Console.WriteLine($"Initiating payment of {amount}.");

// Обработка платежа

\_paymentProcessor.ProcessPayment(amount);

// Логирование завершения

Console.WriteLine($"Payment of {amount} processed successfully.");

}

}

class Program

{

static void Main(string[] args)

{

// использования Stripe

IPaymentProcessor stripeProcessor = new StripePaymentProcessor();

PaymentService stripePaymentService = new PaymentService(stripeProcessor);

stripePaymentService.ExecutePayment(100.50m);

// использования PayPal

IPaymentProcessor paypalProcessor = new PayPalPaymentProcessor();

PaymentService paypalPaymentService = new PaymentService(paypalProcessor);

paypalPaymentService.ExecutePayment(200.75m);

// использования Crypto

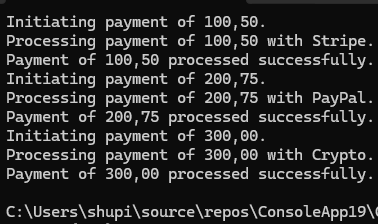
IPaymentProcessor cryptoProcessor = new CryptoPaymentProcessor();

PaymentService cryptoPaymentService = new PaymentService(cryptoProcessor);

cryptoPaymentService.ExecutePayment(300.00m);

}

}



**Контрольные вопросы**

1. В чем суть паттерна Адаптер? Приведите пример из реальной жизни.

Суть паттерна Адаптер

Паттерн Адаптер (Adapter) предназначен для обеспечения совместимости между двумя несовместимыми интерфейсами. Зарядное устройство для мобильного телефона. Оно может быть предназначено для одного типа разъема, но если у вас телефон с другим разъемом, вы можете использовать адаптер, который позволит подключить ваше устройство к зарядному устройству.

1. Чем Адаптер класса отличается от Адаптера объекта?

Адаптер класса (Class Adapter) использует наследование для адаптации интерфейсов. Это значит, что один класс будет наследоваться от другого, позволяя использовать его методы.

Адаптер объекта (Object Adapter) использует композицию. Он содержит ссылку на объект, который необходимо адаптировать, и переопределяет методы для предоставления совместимого интерфейса.

1. Когда лучше использовать Адаптер, а когда — Мост (Bridge)?

Адаптер следует использовать, когда вам нужно сделать несовместимые интерфейсы совместимыми и обеспечить возможность взаимодействия между существующими классами.

Мост (Bridge) лучше использовать, когда нужно отделить абстракцию от реализации так, чтобы они могли изменяться независимо.

1. Может ли Адаптер добавлять новую функциональность, или он только обеспечивает совместимость?

Адаптер в первую очередь предназначен для обеспечения совместимости и не добавляет новой функциональности. Однако, если это необходимо, его можно модернизировать таким образом, чтобы адаптер также обеспечивал дополнительное поведение

1. Как паттерн Компоновщик реализует древовидные структуры?

Он реализует иерархию, которая позволяет клиентам работать с объектами и группами объектов одинаково, не заботясь о том, являются ли они листьями или узлами.

1. Какие операции обычно определяют в интерфейсе компонента?

Add для добавления подкомпонентов

Remove для удаления подкомпонентов

Display или Operation для выполнения действий над компонентами.

1. В каких реальных системах (кроме файловой системы) можно применить Компоновщик?

Графические интерфейсы

Организационные структуры

Конструкторы документов.

1. Как Декоратор расширяет функциональность объекта без создания подклассов?

Паттерн Декоратор (Decorator) позволяет динамически добавлять дополнительное поведение или функциональность существующему объекту, оборачивая его в класс-декоратор.

1. Чем Декоратор отличается от Стратегии (Strategy)?

Декоратор используется для добавления или изменения поведения объекта без изменения его структуры, позволяя динамически модифицировать объект.

Стратегия используется для определения одного из семейства алгоритмов, инкапсуляции их и обеспечения возможности их замены.

1. Можно ли накладывать несколько декораторов на один объект? Если да, как это работает?

Да, можно накладывать несколько декораторов на один объект. Это работает за счет того, что каждый декоратор имеет ссылку на объект, который он декорирует.

1. Почему декораторы должны реализовывать тот же интерфейс, что и декорируемый объект?

чтобы обеспечить возможность замены одного объекта другим без изменения кода клиента.

1. В чем разница между Фасадом и Адаптером?

Фасад (Facade) предоставляет упрощенный интерфейс для более сложной системы или набора интерфейсов, облегчая клиенту взаимодействие с ними.

-Адаптер (Adapter) применяется для обеспечения совместимости между несовместимыми интерфейсами, адаптируя один интерфейс для работы с другим.

1. Какие типы прокси вы знаете? (Virtual, Remote, Protection и др.)

Virtual Proxy

Remote Proxy

-Protection Proxy

1. Чем Прокси отличается от Декоратора?

Декоратор (Decorator) изменяет или расширяет поведение объекта, позволяя динамически добавлять новые функции, тогда как прокси в основном служит для управления доступом и вызовами к объекту.

1. Вам нужно добавить логирование вызовов методов объекта. Какой паттерн вы выберете и почему?

подойдет Декоратор, так как он позволяет добавлять функциональность к объекту без изменения его структуры.

1. Как бы вы организовали кеширование результатов тяжелых вычислений?

Для кеширования результатов тяжелых вычислений можно использовать Паттерн Фабрика в сочетании с Паттерном Прокси. При вызове метода можно сначала проверять наличие результата в кэше.

1. У вас есть класс, который зависит от устаревшего API. Как интегрировать его в новую систему?

Для интеграции устаревшего API в новую систему можно использовать Паттерн Адаптер. Создание адаптера позволит преобразовывать вызовы из новой системы в формат, необходимый для устаревшего API, тем самым скрывая его сложность и несовместимость от других частей системы.

1. Как реализовать поддержку нескольких тем оформления (тёмная/светлая) без дублирования кода?

Для организации поддержки нескольких тем оформления без дублирования кода можно использовать Паттерн Стратегия.

1. Какой паттерн поможет упростить работу с микросервисами, если их API слишком сложный?

можно использовать Паттерн Фасад. Фасад создаст упрощенный интерфейс, который будет скрывать сложность взаимодействия с микросервисами, предоставляя клиентам более простой и понятный доступ к необходимым данным и операциям.

Вывод: Научился использовать структурные шаблоны.