Program.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

using Lab6Lib; // Подключаем пространство имен библиотеки Lab6Lib

namespace Patterns06 // Определяем пространство имен Patterns06

{

class Program // Определяем класс Program

{

static void Main(string[] args) // Метод Main - точка входа в программу

{

// Создаем три экземпляра Publisher с уникальными именами событий

Publisher event1 = new Publisher("EVENT-1");

Publisher event2 = new Publisher("EVENT-2");

Publisher event3 = new Publisher("EVENT-3");

// Создаем экземпляры подписчиков

SubscriberA subscriberA = new SubscriberA();

SubscriberB subscriberB = new SubscriberB();

SubscriberC subscriberC = new SubscriberC();

// Тест 1: Подписка и уведомление

Console.WriteLine("--------------------Тест 1 ------------------");

event1.subscribe(subscriberA); // Подписываем A на EVENT-1

event2.subscribe(subscriberA); // Подписываем A на EVENT-2

event3.subscribe(subscriberA); // Подписываем A на EVENT-3

event1.nonify(); // Уведомляем подписчиков о событии 1

event2.nonify(); // Уведомляем подписчиков о событии 2

event3.nonify(); // Уведомляем подписчиков о событии 3

// Тест 2: Изменение подписок

Console.WriteLine("--------------------Тест 2 ------------------");

event1.subscribe(subscriberB); // Подписываем B на EVENT-1

event2.unsubscribe(subscriberA); // Отписываем A от EVENT-2

event1.nonify(); // Уведомляем подписчиков о событии 1

event2.nonify(); // Уведомляем подписчиков о событии 2

event3.nonify(); // Уведомляем подписчиков о событии 3

// Тест 3: Новые подписки

Console.WriteLine("--------------------Тест 3 ------------------");

event2.subscribe(subscriberB); // Подписываем B на EVENT-2

event3.subscribe(subscriberC); // Подписываем C на EVENT-3

event1.nonify(); // Уведомляем подписчиков о событии 1

event2.nonify(); // Уведомляем подписчиков о событии 2

event3.nonify(); // Уведомляем подписчиков о событии 3

// Тест 4: Повторное уведомление

Console.WriteLine("--------------------Тест 4 ------------------");

event2.subscribe(subscriberB); // Подписываем B на EVENT-2 (дублируется)

event3.subscribe(subscriberC); // Подписываем C на EVENT-3 (дублируется)

event1.nonify(); // Уведомляем подписчиков о событии 1

event2.nonify(); // Уведомляем подписчиков о событии 2

event3.nonify(); // Уведомляем подписчиков о событии 3

// Тест 5: Отписка всех подписчиков

Console.WriteLine("--------------------Тест 5 ------------------");

event1.unsubscribe(subscriberA); // Отписываем A от EVENT-1

event1.unsubscribe(subscriberB); // Отписываем B от EVENT-1

event2.unsubscribe(subscriberB); // Отписываем B от EVENT-2

event3.unsubscribe(subscriberA); // Отписываем A от EVENT-3

event3.unsubscribe(subscriberC); // Отписываем C от EVENT-3

event1.nonify(); // Уведомляем подписчиков о событии 1

event2.nonify(); // Уведомляем подписчиков о событии 2

event3.nonify(); // Уведомляем подписчиков о событии 3

}

}

// Класс для подписчика A, реализующий интерфейс ISubscriber

public class SubscriberA : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

Console.WriteLine(string.Format("Subscriber:{0}, Event:{1}", "A", eventname)); // Выводим сообщение о получении уведомления

}

}

// Класс для подписчика B, реализующий интерфейс ISubscriber

public class SubscriberB : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

Console.WriteLine(string.Format("Subscriber:{0}, Event:{1}", "B", eventname)); // Выводим сообщение о получении уведомления

}

}

// Класс для подписчика C, реализующий интерфейс ISubscriber

public class SubscriberC : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

Console.WriteLine(string.Format("Subscriber:{0}, Event:{1}", "C", eventname)); // Выводим сообщение о получении уведомления

}

}

}

Class1.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lab6Lib // Определяем пространство имен Lab6Lib

{

public class Class1 // Определяем класс Class1

{

// Этот класс пока что пустой и не содержит методов или свойств

}

}

ISubscriber.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lab6Lib // Определяем пространство имен Lab6Lib

{

public interface ISubscriber // Определяем интерфейс ISubscriber

{

// Метод update, который будет вызываться для уведомления подписчика о событии

void update(string eventname);

}

}

Publisher.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lab6Lib // Определяем пространство имен Lab6Lib

{

public class Publisher // Определяем класс Publisher

{

List<ISubscriber> list = new List<ISubscriber>(); // Список подписчиков

private string \_eventname; // Переменная для хранения имени события

// Конструктор класса, принимающий имя события

public Publisher(string eventname)

{

\_eventname = eventname; // Инициализируем имя события

}

// Метод для подписки на события

public void subscribe(ISubscriber subscriber)

{

// Проверяем, есть ли уже подписчик в списке

foreach (var sub in list)

{

if (sub == subscriber)

{

return; // Если подписчик уже есть, выходим из метода

}

}

list.Add(subscriber); // Добавляем подписчика в список

}

// Метод для отписки от событий

public bool unsubscribe(ISubscriber subscriber)

{

// Удаляем подписчика из списка и возвращаем результат операции

if (list.Remove(subscriber)) return true;

else return false;

}

// Метод для уведомления всех подписчиков о событии

public int nonify()

{

// Уведомляем каждого подписчика и передаем имя события

foreach(var sub in list)

{

sub.update(\_eventname); // Вызываем метод update для каждого подписчика

}

return list.Count(); // Возвращаем количество подписчиков

}

}

}

SubscriberA.cs

using System; // Подключаем пространство имен для базовых классов

using System.Collections.Generic; // Подключаем пространство имен для работы с коллекциями

using System.Linq; // Подключаем пространство имен для LINQ

using System.Text; // Подключаем пространство имен для работы с текстом

using System.Threading.Tasks; // Подключаем пространство имен для работы с асинхронностью и задачами

namespace Lab6Lib // Определяем пространство имен Lab6Lib

{

// Класс для подписчика A, реализующий интерфейс ISubscriber

public class SubscriberA : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

// Выводим сообщение о подписчике и событии

Console.WriteLine(string.Format("Subscriber: {0}, Event: {1}", 'A', eventname));

}

}

// Класс для подписчика B, реализующий интерфейс ISubscriber

public class SubscriberB : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

// Выводим сообщение о подписчике и событии

Console.WriteLine(string.Format("Subscriber: {0}, Event: {1}", 'B', eventname));

}

}

// Класс для подписчика C, реализующий интерфейс ISubscriber

public class SubscriberC : ISubscriber

{

// Метод обновления, вызываемый при уведомлении о событии

public void update(string eventname)

{

// Выводим сообщение о подписчике и событии

Console.WriteLine(string.Format("Subscriber: {0}, Event: {1}", 'C', eventname));

}

}

}

Ответы на вопросы

### 5. Поясните суть паттерна Observer, что дает его применение.

Паттерн \*\*Observer\*\* (наблюдатель) — это поведенческий паттерн проектирования, который позволяет объектам (наблюдателям) подписываться на события, происходящие в другом объекте (субъекте). Когда состояние субъекта изменяется, все подписанные наблюдатели получают уведомления и могут реагировать на эти изменения.

\*\*Преимущества применения:\*\*

- \*\*Слабая связь\*\*: Наблюдатели и субъекты могут быть изменены независимо друг от друга, что упрощает поддержку и расширяемость кода.

- \*\*Переиспользование\*\*: Наблюдатели могут быть повторно использованы в различных контекстах, что уменьшает дублирование кода.

- \*\*Динамическое обновление\*\*: Наблюдатели могут быть добавлены или удалены в любое время, что позволяет гибко управлять поведением программы.

### 6. Что необходимо добавить в библиотеку Lab6Lib, чтобы добавить новое событие?

Чтобы добавить новое событие в библиотеку Lab6Lib, необходимо:

1. \*\*Создать новый класс или метод в классе `Publisher`\*\* для определения нового события. Это может включать добавление нового метода для подписки и уведомления подписчиков о конкретном событии.

2. \*\*Определить обработчик события\*\*, который будет выполнять необходимые действия при возникновении этого события.

3. \*\*Обновить интерфейс `ISubscriber`\*\*, если необходимо, добавив в него методы, специфичные для нового события.

### 7. Что необходимо добавить в библиотеку Lab6Lib, чтобы добавить нового подписчика?

Чтобы добавить нового подписчика в библиотеку Lab6Lib, необходимо:

1. \*\*Создать новый класс, реализующий интерфейс `ISubscriber`\*\*, который будет определять, как новый подписчик будет реагировать на события.

2. \*\*Реализовать метод `update`\*\* в этом классе, который будет содержать логику обработки уведомлений о событиях.

3. \*\*Добавить экземпляр нового подписчика\*\* в класс `Publisher`, используя метод подписки, чтобы он мог получать уведомления о событиях.

Если есть дополнительные вопросы или требуется больше информации, дай знать!