ЛЕКЦИЯ 6

- Модуль 3
- 23.01.2024
- Стандартный шаблон событий

ЦЕЛИ ЛЕКЦИИ

- Познакомиться с реализацией стандартного шаблона событий C#

ОБЩИЙ ВИД .NET-СОВМЕСТИМОГО ОБРАБОТЧИКА СОБЫТИЯ

Идентификатор обработчика события

```
[модификатор доступа] void HandlerId(object source, EventArgs e)
{
    // тело обработчика
}
```

Класс, содержащий данные о событии

Такой вариант подготовки обработчика и делегата является устаревшим, но отражает часть исторического развития шаблона обработки событий .NET

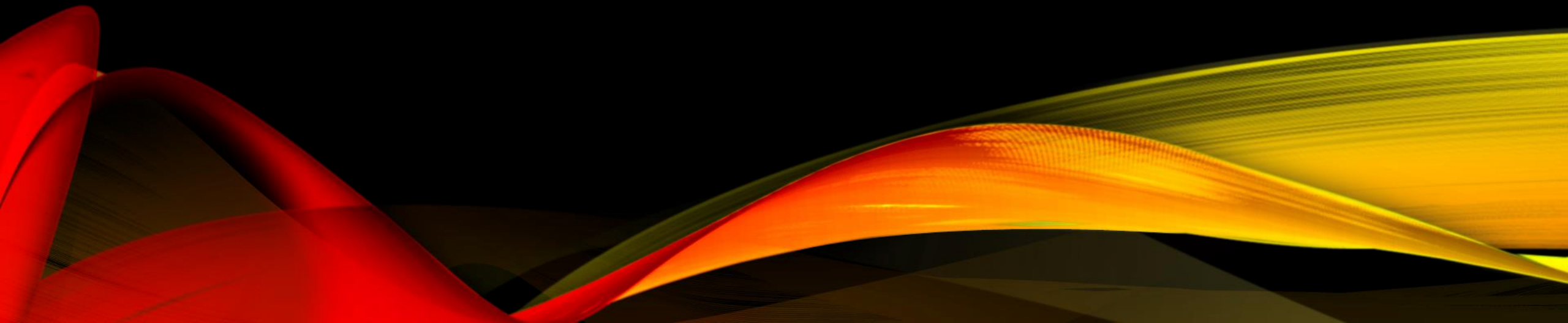
ПРИМЕР .NET-СОВМЕСТИМОГО ОБРАБОТЧИКА

```
/ Делегат для .NET совместимого обработчика событий.

public delegate void OnIncrementDel(object source, EventArgs e);

public class Number {
    public int Value { get; private set; }

    // Событие инкремента значения.
    public event OnIncrementDel OnNumberIncrement;

    public void SetValue(int newValue)  {
        if (Value != newValue)  {
            Value = newValue;
            // Если ничего не передаём с событием, то отправляем константу
EventArgs.Empty.
            OnNumberIncrement(this, EventArgs.Empty);
        }
    }
 }
```

```
Number n = new Number();
// Обработчик выполнен лямбда-выражением.
n.OnNumberIncrement += (object source, EventArgs e) =>
Console.WriteLine("changed!");
for(int i = -5; i < 5; i++)
{
    n.SetValue(i);
}
```


Исходный код: https://replit.com/@olgamaksimenkova/DotNetCompEvents

СТАНДАРТНЫЙ ШАБЛОН ОБРАБОТКИ СОБЫТИЙ .NET



СТАНДАРТНЫЙ ШАБЛОН ГЕНЕРАЦИИ СОБЫТИЙ .NET

Библиотечный делегат-тип в качестве основы для пользовательского кода:

```
public delegate void EventHandler<TEventArgs>(object sender, TEventArgs e);
```

Предполагается, что при возникновении события передаётся:

- Ссылка на издателя – `sender` (может быть `null`, если событие статическое)
- Типизированный объект `EventArgs` или любой из его наследников для передачи данных о событии (которые могут быть `EventArgs.Empty`, если необходимость в передаче данных отсутствует)

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-publish-events-that-conform-to-net-framework-guidelines

ПРИМЕР: КЛАСС-ТАЙМЕР (1)

```
using System;
using System.Timers;

public class MyTimerClass {
    public event EventHandler<ElapsedEventArgs> Elapsed; // Событие нужного типа.
    private Timer _measureTimer; // Объект-таймер для подсчёта времени.
    private void OnOneSecond(object obj, ElapsedEventArgs e)
                => Elapsed?.Invoke(this, e);

public MyTimerClass() {
        _measureTimer = new Timer();
        // Подписка на таймер, его запуск на 1 секунду:
        _measureTimer.Elapsed += OnOneSecond;
        _measureTimer.Interval = 1000;
        _measureTimer.Enabled = true;
    }
}
```

Класс System.Timers.Timer использует свой тип параметров события

Событие вызовется при срабатывании библиотечного таймера

ПРИМЕР: КЛАСС-ТАЙМЕР (2)

```
using System;

public class ClassA
{
    public void TimerHandlerA(object sender, ElapsedEventArgs elapsedArgs)
        => Console.WriteLine($"Handler A: timer event received, time:" +
                             $" {elapsedArgs.SignalTime}");
}

public class ClassB
{
    public static void TimerHandlerB(object sender, EventArgs elapsedArgs)
        => Console.WriteLine($"Handler B: timer event received, time:" +
                             $" {elapsedArgs.SignalTime}");
}
```

Сигнатура экземплярного метода соответствует делегату Elapsed

Сигнатура статического метода соответствует делегату Elapsed

ПРИМЕР: КЛАСС-ТАЙМЕР (3)

```
class Program
{
    static void Main()
    {
        MyTimerClass timer = new MyTimerClass();
        ClassA subscriberA = new ClassA();
        timer.Elapsed += subscriberA.TimerHandlerA;
        timer.Elapsed += ClassB.TimerHandlerB;
        // Таймер срабатывает каждую секунду.
        System.Threading.Thread.Sleep(2000);
        timer.Elapsed -= ClassB.TimerHandlerB;
        System.Threading.Thread.Sleep(1000);
    }
}
```

Вариант вывода:

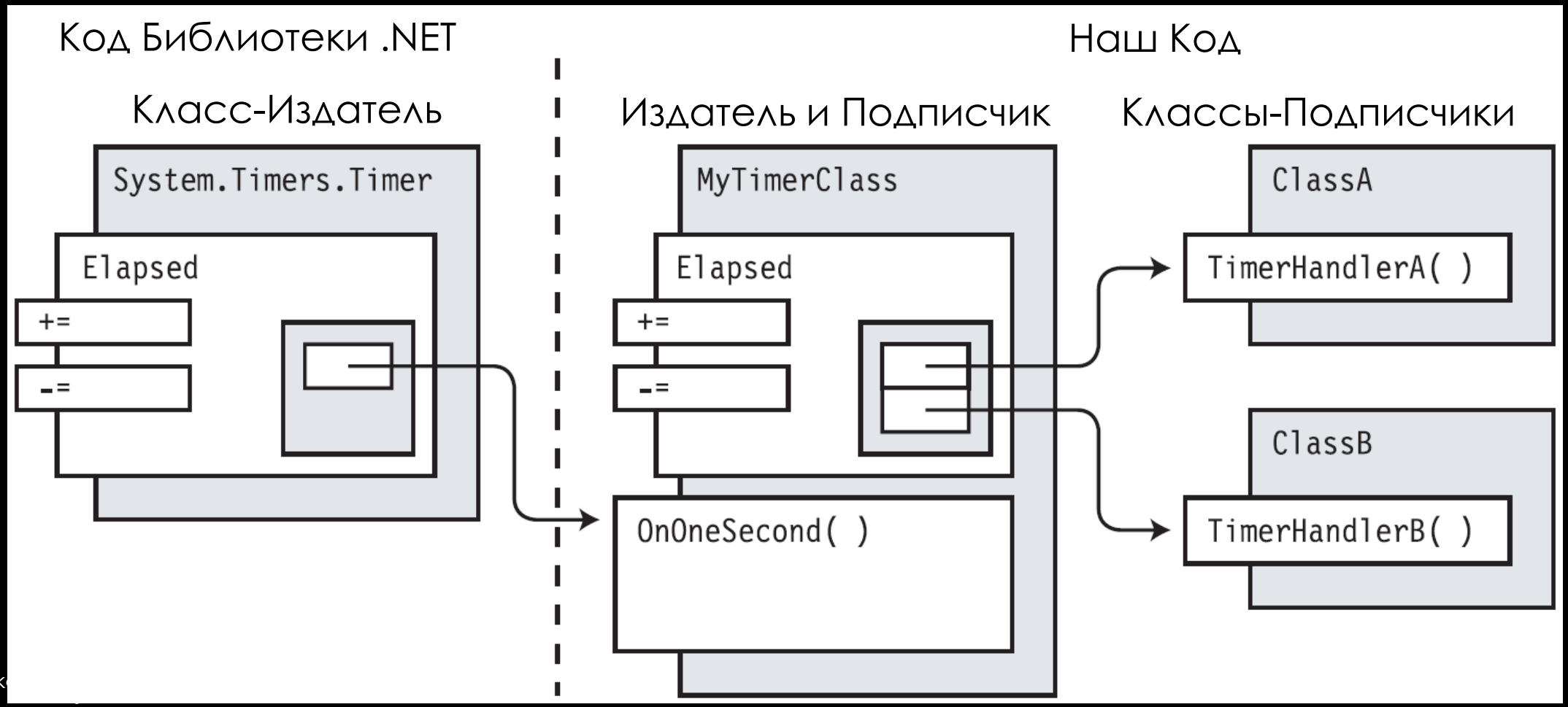
```
Handler A: timer event received, time: 17-Jan-22 13:05:01
Handler B: timer event received, time: 17-Jan-22 13:05:01
Handler A: timer event received, time: 17-Jan-22 13:05:02
Handler B: timer event received, time: 17-Jan-22 13:05:02
Handler A: timer event received, time: 17-Jan-22 13:05:03
```

Остановка выполнения основной программы на 2 секунды для демонстрации работы таймера

Отписываем статический метод класса B

Исходный код (https://replit.com/@olgamaksimenkova/StandardEventPatternTimer)

ИЛЛЮСТРАЦИЯ К ПРИМЕРУ С ТАЙМЕРОМ



ПРОЕКТИРОВАНИЕ ТИПОВ ДЛЯ ШАБЛОНА СОБЫТИЙ .NET

Для реализации стандартного шаблона генерации событий в .NET можно использовать 2 подхода:

- (Устаревший вариант) Объявить пользовательский делегат-тип с явно указанным наследником EventArgs в качестве второго параметра, добавить событие этого типа в нужный класс:

```
public delegate void MyTimerEventHandler(object sender, MyTimerEventArgs e);
// …
public event MyTimerEventHandler MyTimerEvent;
```

- Использовать событие обобщённого делегат-типа EventHandler<T>:

```
public event EventHandler<MyTimerEventHandler> MyTimerEvent;
```

АЛГОРИТМ ИСПОЛЬЗОВАНИЯ ШАБЛОНА СОБЫТИЙ .NET

1) Объявить класс для передачи данных о событии – наследник EventArgs
2) Выбрать делегат-тип для события:
   a. Объявить делегат-тип с наследником EventArgs из п. 1 в качестве второго параметра
   b. Использовать делегат-тип EventHandler<T>
3) Объявить событие делегат-типа из п. 2 внутри типа-издателя
4) Добавить в тип-издатель метод генерации события (как правило, **с именем On<ИмяСобытия>**) и организовать передачу аргументов событию в нём.
   - объявление такого метода как **protected virtual** позволяет вызывать событие в наследниках и переопределять логику вызова
5) Добавить в тип-издатель код, ответственный за генерацию события и вызов метода из п. 4
6) Объявить в типе-подписчике метод обработчик события, соответствующий сигнатуре делегата из п. 2
7) Добавить подписчика с помощью += к событию из п. 3

ПРИМЕР: БАНКОВСКИЙ СЧЁТ (1)

```
using System;

// Класс для передачи данных об изменении на счете:
public class BankEventArgs : EventArgs
{
    public decimal PreviousBalance { get; init; }
    public decimal NewBalance { get; init; }

    public BankEventArgs(decimal oldBalance, decimal newBalance)
        => (PreviousBalance, NewBalance) = (oldBalance, newBalance);
}
```

Наследование от EventArgs в соответствии шаблону.

ПРИМЕР: БАНКОВСКИЙ СЧЁТ (2, ОПЦИОНАЛЬНО)

```
// Делегат, соответствующий типу информации о событии:
public delegate void BankAccountEventHandler(object sender, BankEventArgs e);
```

Используется пользовательский тип параметров события.

ПРИМЕР: БАНКОВСКИЙ СЧЁТ

```
public class BankAccount {
    public string Owner { get; private set; }
    public decimal Balance { get; private set; }
    public event EventHandler<BankEventArgs> AccountBalanceChanged;

    public BankAccount(string ownerName, decimal initialBalance)
        => (Owner, Balance) = (ownerName, initialBalance);
    protected virtual void OnAccountBalanceChanged(object sender, BankEventArgs args)
        => AccountBalanceChanged?.Invoke(sender, args);

    public void AddToAccount(decimal value) {
        decimal oldBalance = Balance;
        Balance += value;
        OnAccountBalanceChanged(this, new(oldBalance, Balance));
    }
    public void RemoveFromAccount(decimal value) => AddToAccount(-value);
}
```

Шаг 3: делегат-тип в соответствии с пунктом 2.b).

Шаг 4: Метод On<Имя_Собятия>, контролирует вызов события.

Шаг 5: Вызов генерации события.

ПРИМЕР: БАНКОВСКИЙ СЧЁТ. ШАГ 6

```
// Класс человек - владелец банковского счёта:
public class Person
{
    public string Name { get; private set; }
    public Person(string name) => Name = name;

    public void OnAccountBalanceChangedEventHandler(object sender,
                                BankEventArgs args)
    {
        Console.WriteLine($"{Name}: Old balance - {args.PreviousBalance:F3}," +
                          $" New balance - {args.NewBalance:F3}");
    }
}
```

Метод-обработчик соответствует сигнатуре делегат-типа EventHandler<BankEventArgs>.

ПРИМЕР: БАНКОВСКИЙ СЧЁТ. ШАГ 7

```
class Program
{
    static void Main()
    {
        Person person = new Person("Victor");
        BankAccount account = new BankAccount(person.Name, 50_000);
        account.AccountBalanceChanged += person.OnAccountBalanceChangedEventHandler;
        account.AddToAccount(2_000);
        account.RemoveFromAccount(25_000);
    }
}
```

Подписка человека на событие изменения банковского счёта.

Вывод:

Victor: Old balance - 50000.000, New balance - 52000.000
Victor: Old balance - 52000.000, New balance - 27000.000

Исходный код (https://replit.com/@olgamaksimenkova/StandardEventPatternBanking)

ВНУТРЕННЕЕ УСТРОЙСТВО СОБЫТИЙ

Компилятор преобразует все события в несколько связных конструкций:

- Закрытое поле указанного делегат-типа
- Метод доступа add() – для добавления подписчиков делегата
- Метод доступа remove() – для удаления подписчиков делегата

C# допускает явное определение add/remove, хотя, как правило, необходимость в этом возникает редко – для абстрактных событий часто достаточно реализации со стороны компилятора

- Однако, необходимость может возникнуть для интерфейсов и виртуальных событий (далее – подробнее об этом)

Для генерируемых компилятором add/remove организуются дополнительные меры по обеспечению потокобезопасности

ПРИМЕР: ЯВНАЯ РЕАЛИЗАЦИЯ МЕТОДОВ ДОСТУПА

```
public class MyStringEventArgs : EventArgs
{
    public string MyString { get; set; }
    public MyStringEventArgs(string str) { MyString = str;}
}
```

```
public class EventPublisher {
    private EventHandler<MyStringEventArgs> _SmthHappened;

    public event EventHandler<MyStringEventArgs> SmthHappened {
        add {
            _SmthHappened += value;
            Console.WriteLine("_SmthHappened += value;");
        }
        remove {
            _SmthHappened -= value;
            Console.WriteLine("_SmthHappened -= value;");
        }
    }
    // Остальной код класса-издателя…
}
```

```
class EventDemo
{
    public event EventHandler Event;
}
```

Преобразуется в...

```
class EventDemo {
    private EventHandler _event;
    private object _objectLock = new object();
    public event EventHandler Event
    {
        add
        {
            lock (_objectLock)
            {
                _event += value;
            }
        }
        remove
        {
            lock(_objectLock)
            {
                _event -= value;
            }
        }
    }
}
```

Упомянутая ранее поддержка многопоточности.

ПРОБЛЕМА ИСПОЛЬЗОВАНИЯ ВИРТУАЛЬНЫХ СОБЫТИЙ

Документация Microsoft явно рекомендует не использовать виртуальные события

- компилятор неявно генерирует скрытое поле делегат-типа и для родителя (с virtual), и для наследника (с override).
  - Это приводит к разному поведению при обращении к событию из методов родителя/наследника (т. к. фактически сгенерированные поля делегат-типов не связаны друг с другом).
- Единственное решение в данном случае – явная реализацию методов доступа add/remove как в классе, объявляющем событие, так и в его наследниках.

Подробно о проблеме:
https://habr.com/ru/company/pvs-studio/blog/315600/ - на русском
https://pvs-studio.com/en/blog/posts/csharp/0453/ - на английском

ССЫЛКИ С ИСТОЧНИКАМИ ПО СОБЫТИЯМ

- Тепляков С. Паттерны проектирования на платформе .NET. – СПб.: Питер, 2015. – 320 с.
- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
- Обзорная информация по событиям: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/
- Ключевое слово event, допустимые модификаторы: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/event
- Подписка на события; методы доступа add и remove, их явная реализация: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-subscribe-to-and-unsubscribe-from-events https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/add https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/remove https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-implement-custom-event-accessors
- Стандартный шаблон событий .NET:
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-publish-events-that-conform-to-net-framework-guidelines
- Вызов событий базового типа из типов наследников: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-raise-base-class-events-in-derived-classes
- Проблема использования виртуальных событий: https://pvs-studio.com/en/blog/posts/csharp/0453/
- Реализация интерфейсов с событиями: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/events/how-to-implement-interface-events