Блок 4. Расширенный C#

4.3 — Делегаты и события

План занятия

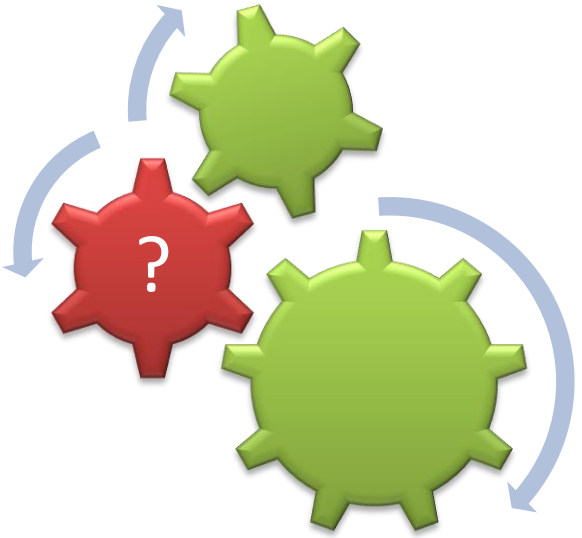
- Делегаты
- Основы многопоточности
- События

Понятие делегата

- Делегат — это структура данных, описывающая сигнатуру какого-либо метода;
- Делегат указывает на набор и способ передачи принимаемых методом аргументов и тип возвращаемого значения;
- Экземпляр делегата представляет собой ссылку на конкретный метод (конкретного экземпляра, либо статический), соответствующий делегату.

Зачем всё это нужно?

- Делегат представляет своего рода функциональный класс, описывающий не состояние, а поведение;
- Экземпляры делегатов дают возможность оперировать методами как объектами:
  - хранение в переменных;
  - передача в качестве аргументов.



Объявление делегата

[<спецификатор доступа>] **delegate** <тип результата>
([<перечень параметров>] **)**

```
namespace DelegatesExample1
{
    //Делегат, объявленный в пространстве имён
    public delegate double Function(double x);

    internal class Program
    {
        // Делегат, объявленный внутри класса
        private delegate void ArrayProcessor(double[] arr);
    }
}
```

Создание и использование делегатов

```
delegate double Function(double x);

static double Square(double x)
{
    return x * x;
}

static void Main(string[] args)
{
    // Создание экземпляра делегата
    Function func1 = new Function(Square);
    Function func2 = Square;
    Function func3 = null;
    func3 += Square;

    // Вызов метода делегата
    double y1 = func1.Invoke(6.28);
    double y2 = func1(6.28);
}
```

Создание и использование делегатов

- Делегаты строго типизированы.

```
delegate double Function(double x);

static double Square(double x)
{ return x * x; }

static int Round(double x)
{ return (int)x; }

static double Sqrt(int x)
{ return Math.Sqrt(x); }

static void Main(string[] args)
{
    Function func1 = new Function(Square);
    Function func2 = new Function(Round);
    Function func3 = new Function(Sqrt);
}
```

Сценарии использования делегатов

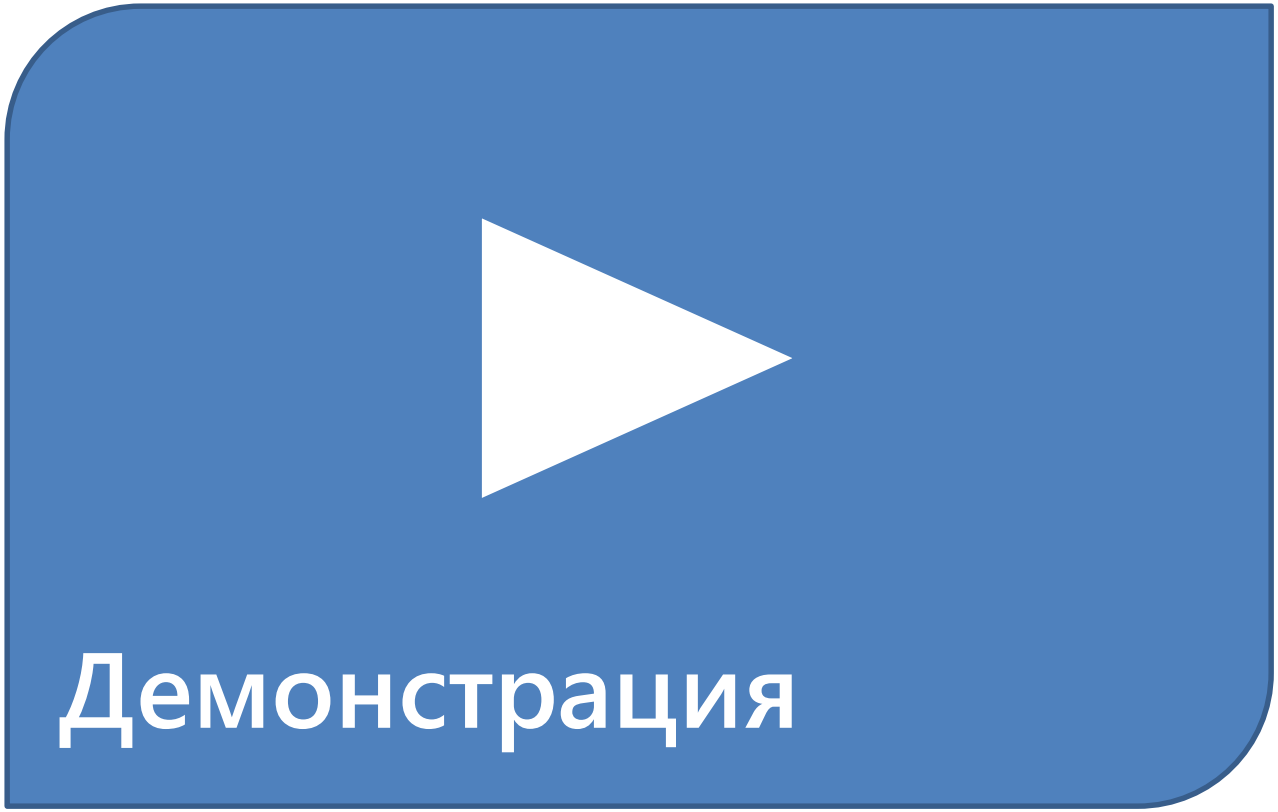
- Передача подзадачи в задачу.

```
public double EvalIntegral(
    double a,
    double b,
    double eps,
    Function func)
{
    Вычисление интеграла методом трапеций
}
```

- Обратный вызов

```
delegate void Callback();

public void DoManyWork(Callback callback)
{
    // Выполнение длительной задачи,
    // вызывающей метод делегата callback по завершению
}
```


Демонстрация

Многопоточность: мини-экскурс

- Потоки позволяют выполнять в рамках одного процесса несколько параллельно работающих алгоритмов.
- Потоки используют общие ресурсы приложения.
- Для реализации многопоточности применяются классы из пространств имён System.Threading и System.Tasks.

Многопоточность: мини-экскурс

- Порядок вызова метода в отдельном потоке:
  1. Создать экземпляр потока (класс Thread) с указанием на вызываемый метод.
  2. Вызвать метод потока Start().

- Вызываемый метод должен соответствовать одному из стандартных делегатов:
  - delegate void ThreadStart()
  - delegate void ParameterizedThreadStart(object)

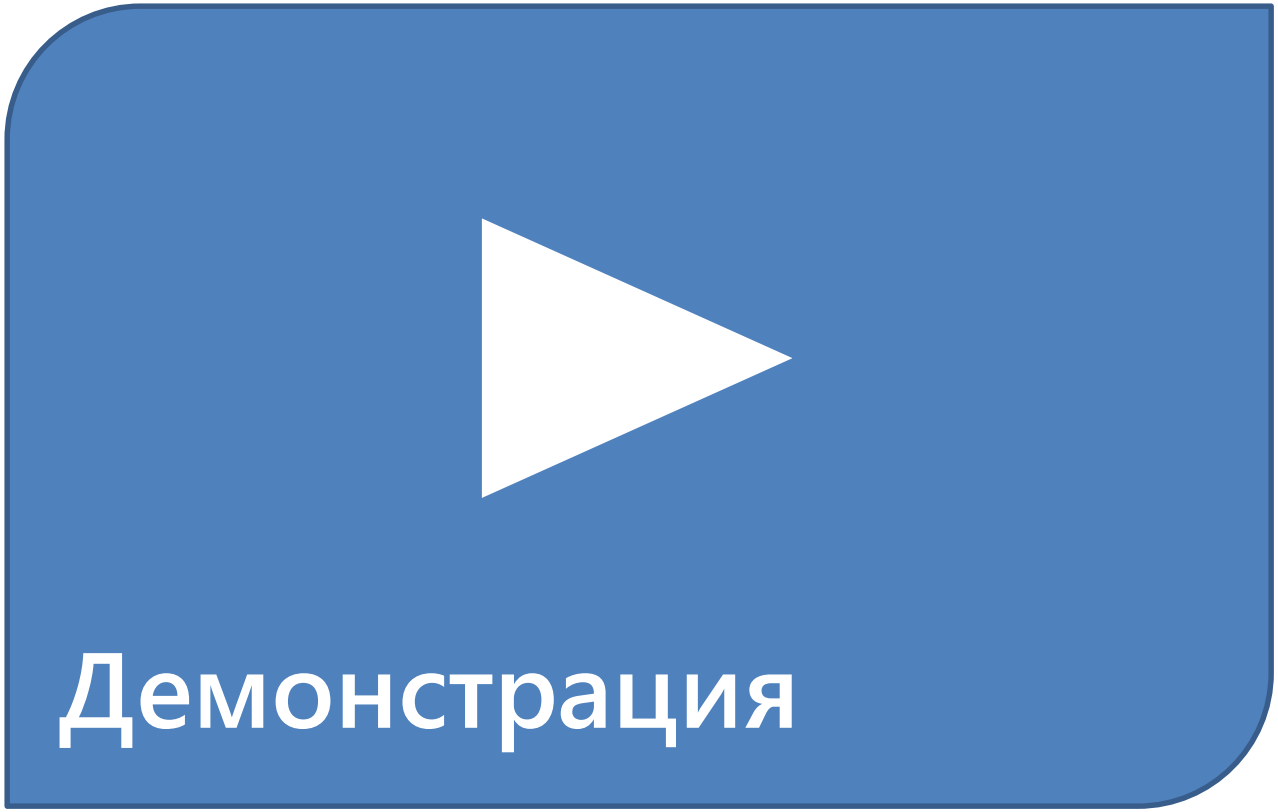
- Поток завершается при завершении вызванного метода.

Многопоточность: мини-экскурс

```
static void Run()
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
        Thread.Sleep(300);
    }
}

static void Main(string[] args)
{
    Thread th1 = new Thread(Run);
    Thread th2 = new Thread(Run);

    th1.Start();
    th2.Start();
}
```


Демонстрация

Анонимные функции

- Анонимная функция — это inline-оператор или выражение, которое можно использовать в качестве объекта делегата;
- Существует две формы записи анонимных функций:
  - Анонимные методы

```
Function PlusOne = delegate(double x) { return x + 1; };
```

  - Лямбда-выражения

```
Function PlusOne = (x) => x + 1;
```

Делегаты и экземплярные методы

```
class Person
{
    public string Name { get; set; }

    public void Greet(string anotherPerson)
    {
        Console.WriteLine("'Hello, {0}!', {1} said.", anotherPerson, Name);
    }
}

public class Test
{
    delegate void Message(string name);

    static void Main(string[] args)
    {
        Person john = new Person { Name = "John" };
        Message greetByJohn = new Message(john.Greet);
        greetByJohn("Bill");
    }
}
```

```
'Hello, Bill!', John said.
Press any key to continue . . .
```

MulticastDelegate

- Представляет групповой делегат, то есть делегат, содержащий связный список делегатов, называемый списком вызовов.
- При активации группового делегата делегаты списка вызовов вызываются последовательно в порядке, соответствующем порядку добавления.
- Если при выполнении этого списка происходит ошибка, выбрасывается исключение.

Добавление делегатов в список вызова

- Метод Delegate.Combine

```
Person john = new Person { Name = "John" };
Person mary = new Person { Name = "Mary" };
Person hugo = new Person { Name = "Hugo" };

Message greetByJohn = new Message(john.Greet);
Message greetByMary = new Message(mary.Greet);
Message greetByHugo = new Message(hugo.Greet);

Message greetByUs = (Message)Delegate.Combine(
    greetByJohn,
    greetByMary,
    greetByHugo);

greetByUs("Bill");
```

```
'Hello, Bill!', John said.
'Hello, Bill!', Mary said.
'Hello, Bill!', Hugo said.
Press any key to continue . . .
```

- Оператор +

```
Message greetByUs = greetByJohn + greetByMary;
greetByUs += hugo.Greet;
```

Удаление делегатов из списка вызова

- Метод Delegate.Remove

```
greetByUs = (Message)Delegate.Remove(greetByUs, greetByJohn);
greetByUs("Bob");
```

```
'Hello, Bob!', Mary said.
'Hello, Bob!', Hugo said.
Press any key to continue . . .
```

- Оператор –

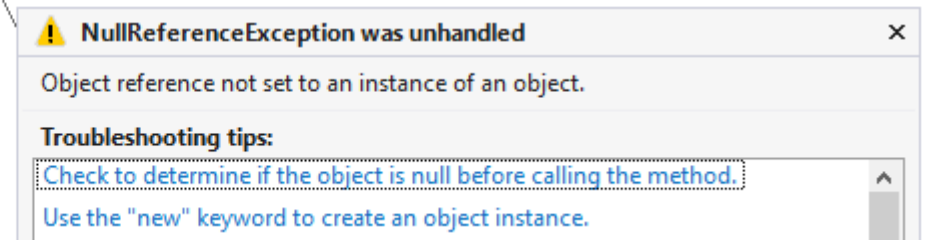
```
greetByUs -= mary.Greet;
greetByUs("Bob");
```

```
'Hello, Bob!', Hugo said.
Press any key to continue . . .
```

Проблема пустого делегата

- Важно помнить, что экземпляр делегата, не содержащий ссылок на методы, равен **null**.

```
greetByUs = new Message(john.Greet);
greetByUs -= john.Greet;
greetByUs("George");
```



- При вызове **ВСЕГДА** нужно проверять экземпляр делегата на наличие методов в списке вызовов.

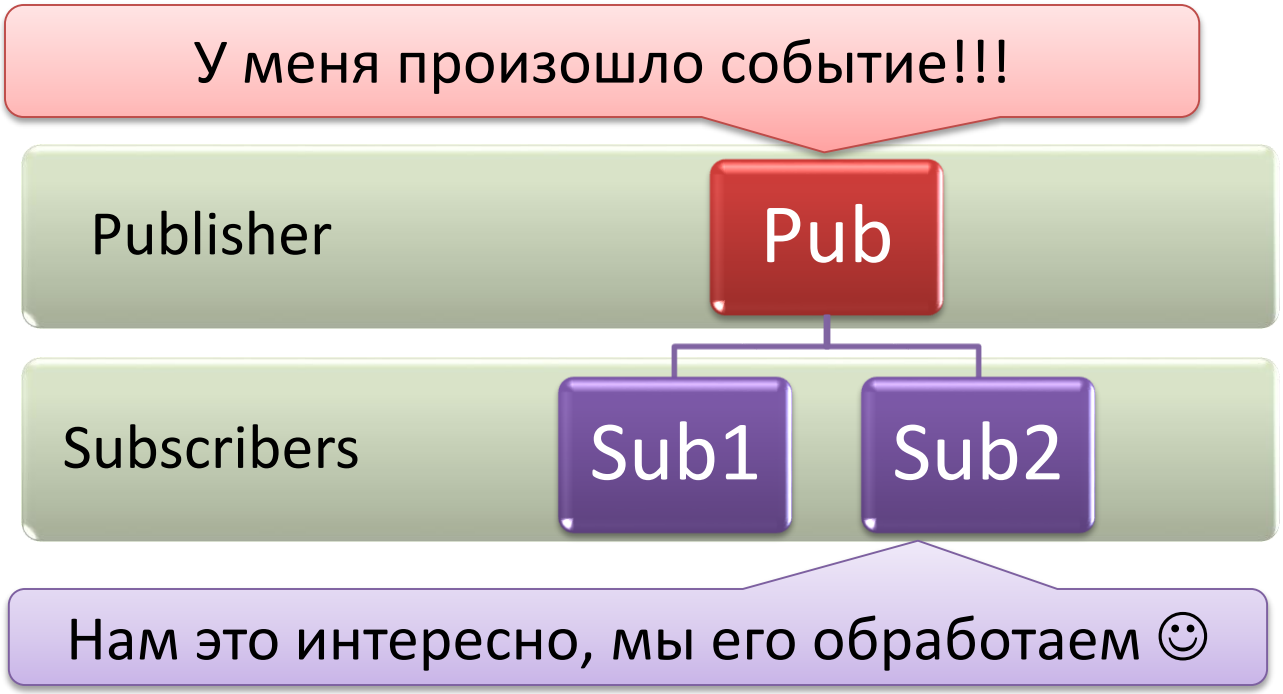
```
if (greetByUs != null)
{
    greetByUs("George");
}
```

Стандартные делегаты

- Func<out TResult>, Func<in T, out TResult>, Func<in T1, in T2, out TResult>, …
  Принимает: от 0 до 16 аргументов.
  Возвращает: **TResult**.
- Action, Action<in T>, Action<in T1, in T2>, …
  Принимает: от 0 до 16 аргументов.
  Возвращает: **void**.
- Predicate<in T>
  Принимает: аргумент типа **T**.
  Возвращает: **bool**.

События

- События позволяют издателю (Publisher) уведомлять подписчиков (Subscribers) о возникновении каких-либо ситуаций.



Свойства событий

- Издатель определяет момент вызова события, подписчики определяют предпринятое ответное действие;
- У события может быть несколько подписчиков. Подписчик может обрабатывать несколько событий от нескольких издателей;
- События, не имеющие подписчиков, никогда не возникают;
- Если событие имеет несколько подписчиков, то при его возникновении происходит синхронный вызов обработчиков событий.

Объявление и генерация события

[<спецификатор>] **event** <делегат> <имя события>;

```
public event EventHandler<EventArgs> Came;

protected virtual void OnCame()
{
    if (Came != null)
    {
        Came(this, EventArgs.Empty);
    }
}
```

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

Подписка на событие

- Подписка обработчика на событие синтаксически аналогична добавлению метода обработчика в список вызовов группового делегата.

```
hugo.Came += hugo_Came;
```

- Обработчик представляет собой метод, соответствующий сигнатуре делегата события.

```
static void hugo_Came(object sender, EventArgs e)
{
    Console.WriteLine("Hugo came");
}
```

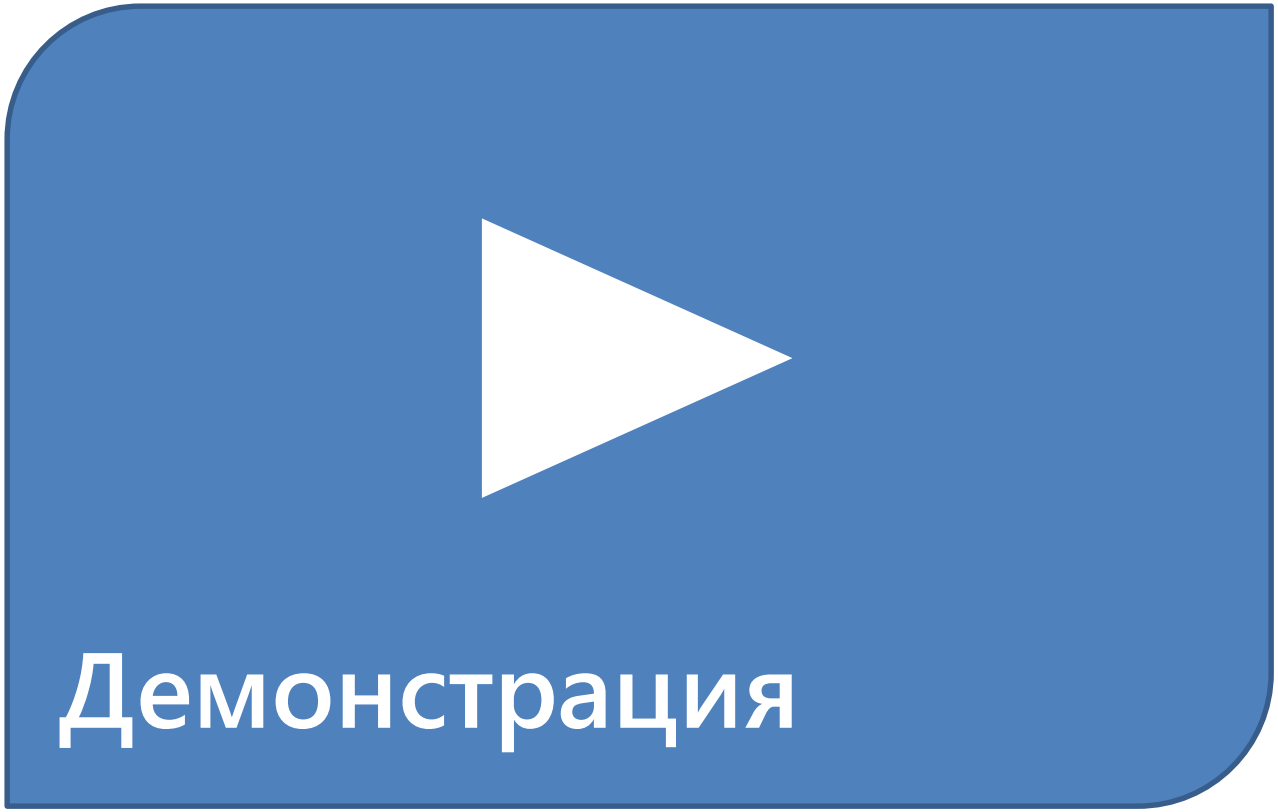
Отличия событий от публичных объектов делегатов

- Возможность управления подпиской:

```
public event EventHandler<EventArgs> Came
{
    add { /* manual subscription */ }
    remove { /* manual unsubscription */ }
}
```

- Возможность размещения событий в интерфейсах:

```
interface IEmployee
{
    event EventHandler<EventArgs> Came;
}
```


Демонстрация

Что почитать

1. Андерс Хейлсберг – Язык программирования C# (четвёртое издание)
2. http://rsdn.ru/article/dotnet/delegat.xml

Спасибо за внимание!

Контактная информация:
Дмитрий Вересkун
Инструктор
EPAM Systems, Inc.
Адрес: Саратов, Рахова, 181
Email: Dmitry_Vereskun@epam.com
http://www.epam.com