Web-разработка на C# и платформе Microsoft .NET

Делегаты и события

План занятия

- Делегаты: что, зачем и как
- Основы многопоточности
- События

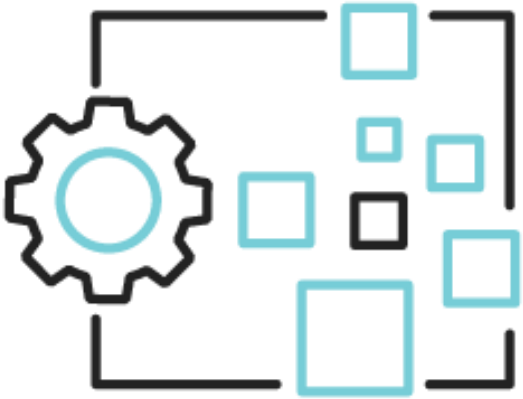
Что такое делегат?

- Делегат – это структура данных, указывающая на:
  - статический метод какого-либо класса;
  - экземпляр класса и его метод.

- Представляет собой ссылку на метод с заданным списком параметров и типом результата.

А зачем?

Делегаты дают возможность использовать методы как сущности, которые могут быть присвоены переменным и/или передаваться как параметры.



Обьявление делегата

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

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

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

Создание и использование делегатов

```
delegate double Function(double x);

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


Function func = new Function(Square);
double y = func(6.28);
```

Создание и использование делегатов

Делегаты строго типизированы.

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

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

static void Main(string[] args)
{
    Function func = new Function(Square);
    double y = func(6.28);

    Function func2 = new Function(Round);


    Function func3 = new Function(Sqrt);


    Function func4 = new Function(Math.Sqrt);
```

int Program.Round(double x)

Error:
'int DelegatesExample1.Program.Round(double)' has the wrong return type

delegate DelegatesExample1.Function

Error:
No overload for 'Sqrt' matches delegate 'DelegatesExample1.Function'

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
}
```

Многопоточность

- Потоки позволяют программе выполнять параллельную обработку, за счет чего появляется возможность одновременного выполнения нескольких операций.
- Пространство имён System.Threading упрощает использование потоков.
- Потоки используют одни и те же ресурсы приложения.

Многопоточность

- Порядок вызова метода в отдельном потоке:
  - Создание экземпляра класса Thread, с указанием на вызываемый метод.
  - Вызов метода Start.

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

- Поток завершается при завершении вызванного метода.

Многопоточность

```
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();
}
```

Анонимные функции

Анонимная функция – это оператор или выражение, которое можно использовать каждый раз, когда ожидается тип делегата. Ее можно использовать для инициализации именованного делегата или подставить вместо типа именованного делегата в качестве параметра метода.

Существует два типа анонимных функций:

Анонимные методы

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

Лямбда-выражения

```
Function PlusTwo = (x) => x + 2;
```

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 . . .
```

Multicast delegate

- Представляет групповой делегат, то есть делегат, содержащий связный список делегатов, называемый списком вызовов, состоящий из одного или нескольких элементов.

- При активации группового делегата делегаты в списке вызовов вызываются одновременно в том порядке, в каком они представлены.

- Если при выполнении этого списка происходит ошибка, выбрасывается исключение.

Добавление дегатов 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 вызовов

- Метод Delegate.Remove

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

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

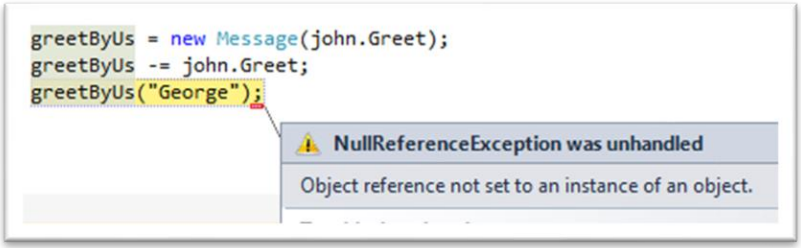
- Оператор -

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

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

Null и делегаты

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



При вызове делегата **ВСЕГДА** проверяй его на null!

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

События

События позволяют классу или объекту уведомлять другие классы или объекты о возникновении каких-либо ситуаций.
Класс, вызывающий событие, называется издателем, а классы, обрабатывающие его, – подписчиками.

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.
- В библиотеке классов .NET Framework в основе событий лежит делегат EventHandler и базовый класс EventArgs.

Объявление события

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

```
class Person
{
   public event EventHandler Came;

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

Подписка на событие

```
hugo.Came += new EventHandler(hugo_Came);


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

СПАСИБО ЗА ВНИМАНИЕ