ОБРАЗОВАТЕЛЬНЫЙ ЦЕНТР
ПРОГРАММИРОВАНИЯ И ВЫСОКИХ
ТЕХНОЛОГИЙ

2020г.

ДЕЛЕГАТЫ И СОБЫТИЯ

AGENDA

- ✓ Понятие делегата
- ✓ Анонимная функция
- ✓ Анонимный тип
- ✓ Анонимный метод
- ✓ Лямбда-выражения
- ✓ Стандартные типы делегатов: Action, Predicate и Func
- ✓ События и делегат EventHandler

Делегаты

Делегат – это класс, содержащий данные о сигнатуре метода.

Делегаты представляют такие объекты, которые указывают на методы. То есть делегаты – это указатели на методы и с помощью делегатов мы можем вызвать данные методы.

Предоставляет возможность инкапсулировать метод.

Экземпляр делегата (**delegate instance**) – объект, который позволяет привязаться к конкретному методу, соответствующему определенной сигнатуре.

Объявление:

delegate возвращаемый_тип имя (список_параметров);

delegate – ключевое слово.

имя – имя делегата, по которому будем обращаться к нему.

возвращаемый_тип – тип значения, который возвращают методы, вызываемые делегатом.

список_параметров – список необходимых параметров для методов, вызываемых делегатом.

Делегаты

Событийная система управления строится на:

- Уведомлениях (**notifications**).
- Обратные вызовы (**callback**).

Реализация в языках программирования:

- C\C++ - указатели на функцию.
- C# - делегаты.

Делегаты в CRL реализованы в виде классов.

Базовый класс делегатов:

- **Singlecast delegate –** хранит ссылку на один метод.
  public abstract class Delegate: ICloneable, Iserializable
  Свойства:
  public object Target {get;} – если делегат ссылается на нестатический метод, то в данном поле храниться ссылка конкретный объект, иначе **null.**
  public MethodInfo Method {get;} – имя метода, на который ссылается делегат.
- **Multicast delegate –** может хранить ссылки нескольких методов.
  public abstract class MulticastDelegate: Delegate

Делегаты

Объявление multicast:

Тип возвращаемого значения – всегда **void.**

- ***public delegate void MyDelegate(int a, out int z)***

Добавление ссылок в делегат:

- **+** или **+=.**
- **Combine()** или **Combine([]).**

Удаление ссылок из делегата:

- **-** или **-=.**
- **Remove().**

В случае возвращения значения из метода вызываемого делегатом, то возвращается значение полученное последним методом.

Для хранения списка ссылок используется дополнительное поле **_prev**, которое ссылается на следующий элемент.

Делегаты

Создание объекта делегата при помощи оператора **new**:

```
MyDelegate myDel = new MyDelegate(Add);
var myDel = new MyDelegate(Add);
```

Начиная с **Framework 2.0** использовать **new** для создания экземпляра делегата не обязательно. При этом **var** использовать нельзя:

```
MyDelegate myDel = Add;
```

Можно вызвать метод с соответствующей сигнатурой и возвращаемым типом.

```
myDel = calc.Mul;
myDel(2, 3);
```

Определяет, что **myDel** – это экземпляр делегата.

Использует информацию из свойств **Target** и **Method**.

Вызывает необходимый метод посредством метода **Invoke**, позволяющий динамически вызвать метод, данные о котором хранятся в объекте (Получаем из **MethodInfo** возвращаемым **Method**).

Анонимный тип

Анонимные типы позволяют создать объект с некоторым набором свойств без определения класса. Анонимный тип определяется с помощью ключевого слова **var** и инициализатора объектов.

```
var user = new { Name = "Tom", Age = 34 };
Console.WriteLine(user.Name);
```

В отличие от реального класса, анонимный тип не может иметь приватные поля или свойства, а так же методы

После инициализации объекта в него нельзя добавить новые свойства.

Свойства только для чтения – как только объект был инициализирован, вы не можете изменить их значения.

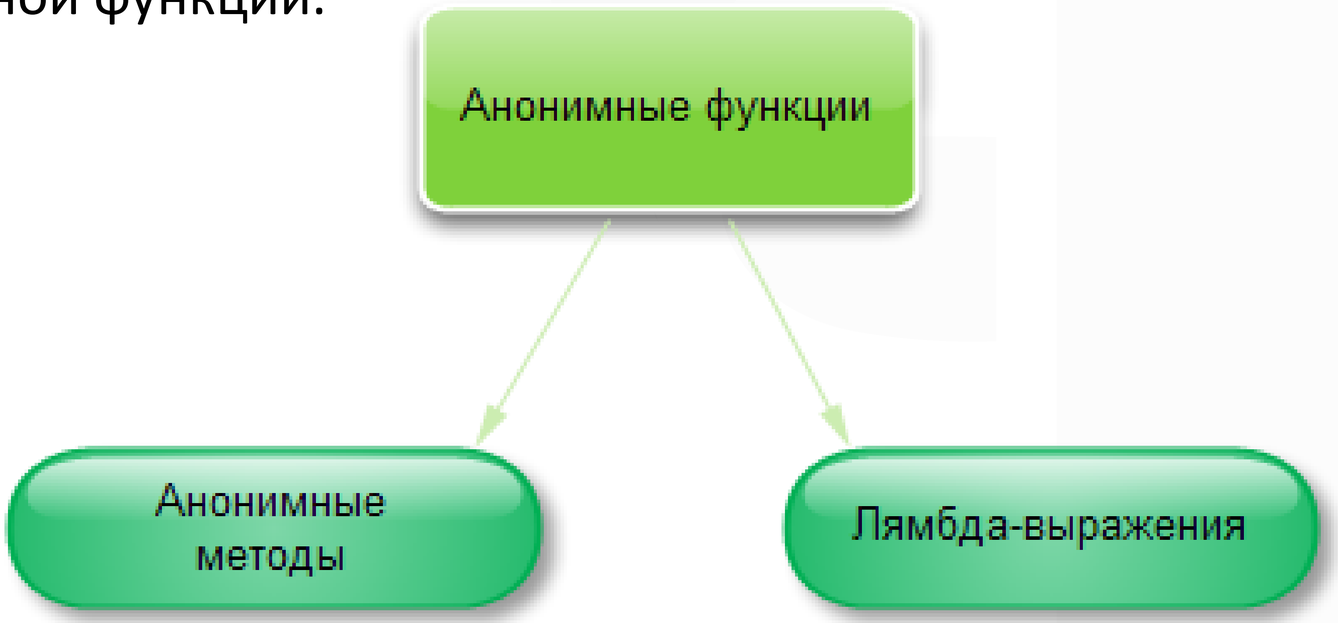
При этом во время компиляции компилятор сам будет создавать для него имя типа и использовать это имя при обращении к объекту.

Для исполняющей среды CLR анонимные типы будут также, как и классы, представлять ссылочный тип.

Анонимная функция

Анонимная функция – это функция, которая не имеет имени а содержит только блок программного кода, который она выполняет. Анонимные функции удобно использовать в объединении с делегатами. Анонимную функцию можно вызвать только с помощью **делегата**. Сама функция непосредственно не вызовется никогда.

Анонимные методы были внедрены в **C#** еще в версии **2.0**, а **лямбда-выражения** – в версии **3.0**. В целом лямбда-выражение совершенствует принцип действия анонимного метода и в настоящее время считается более предпочтительным для создания анонимной функции.



Анонимный метод

Анонимный метод — один из способов создания безымянного блока кода, связанного с конкретным экземпляром делегата. Для создания анонимного метода достаточно указать кодовый блок после ключевого слова **delegate**.

Анонимные методы тесно связаны с делегатами. Они используются для создания экземпляров делегатов.

```
delegate(параметры)
{
    // инструкции
}
```

Анонимный метод не может существовать сам по себе, он используется для инициализации экземпляра делегата.

Через переменную делегата можно вызвать данный анонимный метод.

Параметры анонимного метода, должны соответствовать параметрам делегата.

Анонимный метод

```
class Program
{
    delegate void MessageHandler(string message);
    static void Main(string[] args)
    {
        MessageHandler handler = delegate(string mes)
        {
            Console.WriteLine(mes);
        };
        handler("hello world!");

        Console.Read();
    }
}
```

Лямбда-выражения

Лямбда-выражения – это упрощенная запись анонимных методов.

Лямбда-выражения позволяют создать лаконичные методы, которые могут возвращать некоторое значение и которые можно передать в качестве параметров в другие методы.

(список_параметров) => выражение;

Нет необходимости указывать тип параметров, а при возвращении результата не надо использовать оператор **return.**

Лямбда-выражения можно использовать в качестве аргумента метода.

Лямбда-выражения

```
class Program
{
    delegate int Operation(int x, int y);
    static void Main(string[] args)
    {
        Operation operation = (x, y) => x + y;
        Console.WriteLine(operation(10, 20));       // 30
        Console.WriteLine(operation(40, 20));       // 60
        Console.Read();
    }
}
```

Стандартные типы делегатов

Делегат **Action** является обобщенным, принимает параметры и возвращает значение **void**. Данный делегат имеет ряд перегруженных версий. Каждая версия принимает разное число параметров: от **Action<in T1>** до **Action<in T1, in T2,….in T16>**. Таким образом можно передать до 16 значений в метод.

public delegate void Action<T>(T obj)

Делегат **Predicate<T>**, как правило, используется для сравнения, сопоставления некоторого объекта **T** определенному условию. В качестве выходного результата возвращается значение **true**, если условие соблюдено, и **false**, если не соблюдено.

public delegate bool Predicate<in T>(T obj);

Еще одним распространенным делегатом является **Func**. Он возвращает результат действия и может принимать параметры. Он также имеет различные формы: от **Func<out T>()**, где **T** – тип возвращаемого значения, до **Func<in T1, in T2,…in T16, out TResult>()**, то есть может принимать до 16 параметров.

public delegate TResult Func<out TResult>();

public delegate TResult Func<in T, out TResult>(T obj);

События

В **CRL** строятся на основе комбинированных делегатов.

Используется схема:

- **Издатель**. Класс, предоставляющий события другим классам, т.е. способен уведомлять их о переходе в определенное состояние.
- **Подписчик**. Класс (-ы), которые желают узнать о переходе класса в определенное состояние.

Подписчик обязан выполнить регистрацию на определенное событие.

- **Регистрация** – передача классу издателю делегата на метод, который необходимо вызвать при наступлении события. Передаваемый метод (**метод-обработчик**) находится в теле класса подписчика.

Метод-обработчик:

- Находится в теле класса-подписчика.
- Содержит код, который необходимо выполнить при наступлении определенного события.

События

Используется ключевое слово **event**:

- **event делагат_события имя_события;**
- **делегат_события** – делегат для поддержки события.
- **имя_события** – имя, по которому можно обратится к событию.

Определение делегата и события:

- ***delegate void ShowMessage(string message);***
- ***event ShowMessage Notify;***

Вызов события:

- ***Notify("Произошло действие");***
- ***if(Notify !=null) Notify("Произошло действие");***
- ***Notify?.Invoke("Произошло действие");***

Добавление и удаление обработчика события:

- ***Notify += обработчик события;*** // Добавление метода-обработчика
- ***Notify -= обработчик события;*** // Удаление метода-обработчика

События

Для совместимости программных компонентов со средой **.NET Framework** следует придерживаться рекомендаций, установленных корпорацией **Microsoft**. Эти рекомендации, сводятся к следующему требованию: у обработчиков событий должны быть два параметра. Первый из них – ссылка на объект, формирующий событие, второй – параметр типа **EventArgs**, содержащий любую дополнительную информацию о событии, которая требуется обработчику.

EventArgs служит в качестве базового класса, от которого получается производный класс, содержащий все необходимые поля. В классе **EventArgs** имеется одно поле **Empty** типа **static**, которое представляет собой объект типа **EventArgs** без данных.

В среде **.NET Framework** предоставляется встроенный обобщенный делегат под названием **EventHandler<TEventArgs>.** В данном случае тип **TEventArgs** обозначает тип аргумента, передаваемого параметру **EventArgs** события.

С целью упростить создание кода когда параметр типа **EventArgs** оказывается ненужным в среду **.NET Framework** внедрен необобщенный делегат типа **EventHandler**. Он может быть использован для объявления обработчиков событий, которым не требуется дополнительная информация о событиях.

Задание

Разработать класс **NewsProvider** предоставляющий услуги рассылки информации по категориям (новости, погода, спорт, происшествия, юмор).

Разработать класс **Client**, который подписывается на определенную категорию и при выходе новой новости в категории, подписчик получает услугу.