ЯЗЫКИ ПРИКЛАДНОГО ПРОГРАММИРОВАНИЯ

Лекция 4

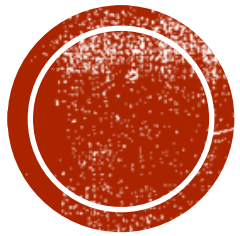
```
public class GameObject2D
{
    private int _x;
    private int _y;

    public GameObject2D(int x, int y)
    {
        _x = x;
        _y = y;
    }

    public void ChangePosition(int newX, int newY)
    {
        PositionsHistory.Add((x:_x, y: _y));
        _x = newX;
        _y = newY;
    }

    public readonly List<(int x, int y)> PositionsHistory = new();

    public (int x, int y) GetPosition => (_x, _y);
}
```

КОРТЕЖИ

- Предоставляют способ работы с набором значений

ЛОКАЛЬНЫЕ ФУНКЦИИ

```
public class LocalFunctionExample
{
    public void Method(int x)
    {
        int temp = 42;

        Console.WriteLine(InnerMethod());
        Console.WriteLine(InnerMethod());
        Console.WriteLine(InnerMethod());

        int InnerMethod()
        {
            temp += x;
            return temp;
        }
    }
}
```



REF, IN, OUT

REF

```
public static void RefExample(ref int param)
{
    param++;
    Console.WriteLine(param);
}

public static void RefExampleCall()
{
    int variable = 42;
    RefExample(ref variable);
    Console.WriteLine(variable);

    // Console:
    // 43
    // 43
}
```

- Используется для передачи параметров по ссылке
- Модификатор ref указывается как перед параметром при объявлении метода, так и при вызове метода перед аргументом, который передается параметру!

IN

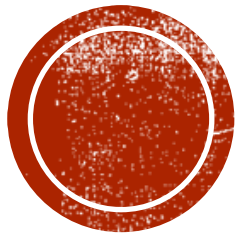
```
public static void InExample(in int param)
{
    // param - не может быть измененен

    Console.WriteLine(param);
}
```

- Модификатор in указывает, что данный параметр будет передаваться в метод по ссылке, однако внутри метода его значение параметра нельзя будет изменить.

OUT

```
public static void OutExample(out int param)
{
    param = 42;
    Console.WriteLine(param);
}

public static void OutExampleCall()
{
    OutExample(out var variable);
    Console.WriteLine(variable);

    int otherVariable;
    OutExample(out otherVariable);
}
```

- Модификатор для возврата значения по ссылке из метода.
- В теле метода обязательно должно быть присвоено какое-либо значение.
- Позволяют вернуть несколько значений из метода.



ДЕЛЕГАТЫ

ДЕЛЕГАТЫ

- Делегат – это специальный объект указывающий на метод и позволяющий вызвать его.
- Для объявления используется ключевое слово **delegate**, далее указывается возвращаемый тип, название и параметры.
- Метод соответствует делегату, если они имеют одинаковый возвращаемый тип и тот же набор параметров, **in**, **out**, **ref** также учитываются.
- Делегат может быть обобщенным (**generic**).

ДЕЛЕГАТЫ

```
public static class OtherClass
{
    public static void Hello() => Console.WriteLine("Hey bro!");
    public static double Sum(double f, double s) => f + s;
}

internal delegate void ActionDelegate();

internal delegate R BinaryOperation<T, R>(T first, T second);

public class DelegatesExample
{
    private static void WriteHelloWorld() => Console.WriteLine("Hello World");

    public static void SimpleDelegateExample()
    {
        ActionDelegate ad = WriteHelloWorld;
        ActionDelegate delegateFromOtherClass = OtherClass.Hello;
        BinaryOperation<double, double> binaryOperation = OtherClass.Sum;

        ad(); // Hello World
        delegateFromOtherClass(); // Hey bro!
        Console.WriteLine(binaryOperation(10, 20)); // 30
    }
}
```

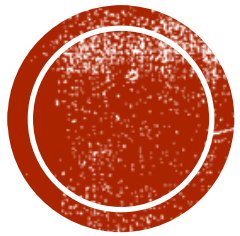
```
public class DelegatesExample2
{
    public static void SomeOtherExample()
    {
        ActionDelegate ad = WriteHelloWorld;
        ad();
        // Hello World

        ad += OtherClass.Hello;
        ad += OtherClass.Hello;
        ad();
        // Hello World
        // Hey bro!
        // Hey bro!

        ad -= OtherClass.Hello;
        ad -= OtherClass.Hello;
        ad?.Invoke();
        // Hello World

        ad -= WriteHelloWorld;
        ad?.Invoke();
        // No console
    }
}
```

ДЕЛЕГАТЫ

- Делегат может указывать на множество методов сразу. Все методы в делегате попадают в список вызова (invocation list).
- При вызове делегата все методы из этого списка последовательно вызываются.
- Для добавления методов в делегат применяется операция +=, для удаления -=

ДЕЛЕГАТЫ

- При удалении методов из делегата фактически будет создаваться новый делегат, который в списке вызова методов будет содержать на один метод меньше.
- При удалении метода не добавленного в делегат, результат - null.
- Если делегат содержит несколько ссылок на один и тот же метод, то операция -= начинает поиск с конца списка вызова делегата и удаляет только первое найденное вхождение.
- Если подобного метода в списке вызова делегата нет, то операция -= не имеет никакого эффекта.



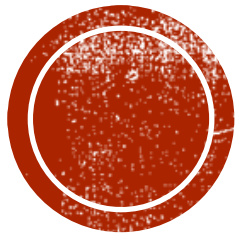
АНОНИМНЫЕ МЕТОДЫ

- Анонимные методы используются для создания экземпляров делегатов, они не могут существовать сами по себе, используется:
  - Для инициализации экземпляра делегата
  - в качестве аргумента для параметра метода (делегата)

```
public class AnonymousMethodsExamples
{
    private delegate int BinaryOperation(int first, int second);

    private static void CallDelegate(BinaryOperation op)
    {
        Console.WriteLine(op(5, 6));
    }

    public static void Exec()
    {
        BinaryOperation operation = delegate (int x, int y)
        {
            return x + y;
        };

        CallDelegate(delegate(int x, int y)
        {
            return x * y;
        });
    }
}
```

АНОНИМНЫЕ МЕТОДЫ

- Если анонимный метод использует параметры, то они должны соответствовать параметрам делегата.
- Если для анонимного метода не требуется параметров, то скобки с параметрами опускаются.
- Параметры можно опустить если их не планируется использовать, однако, параметры не могут быть опущены, если один или несколько параметров определены с модификатором out.

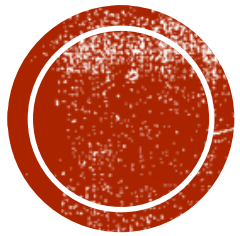


ЛЯМБДА-ВЫРАЖЕНИЯ

```
public class LambdasExample
{
    private delegate int BinaryOperation(int first, int second);

    private static void CallDelegate(BinaryOperation op)
    {
        Console.WriteLine(op(5, 6));
    }

    public static void Exec()
    {
        BinaryOperation operation = (f, s) => f + s;
        CallDelegate((f, s) => f * s);

        var f = (int x) => x * x;
        Console.WriteLine(f(10));
    }

    private static double S(double radius) => Math.PI * 2 * radius;
}
```

ЛЯМБДА ВЫРАЖЕНИЯ

- «Синтаксический сахар» для записи анонимных методов, более емкий и лаконичный.
- Тип данных – делегат.



ACTION, PREDICATE, FUNC

Action

```
public class DefaultDelegatesExample
{
    private Action _simpleAction = () => Console.WriteLine("Fail");
    private Action<int> _simpleActionWithParameter = (param) => Console.WriteLine(param);

    public delegate void Action();
    public delegate void Action<in T>(T obj);
    public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2);
    // ...
}
```

- Встроенный делегат
- Действие, которое ничего не возвращает
- Может принимать до 16 параметров

PREDICATE

- Встроенный делегат
- Принимает один параметр и возвращает **bool**

```
public Predicate<int> _predicate = (x) => x > 0;
```

Func

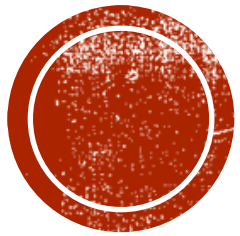
```
t> _func = () => 42;
t, int, int> _funcWithParam

 TResult Func<out TResult>(
 TResult Func<out TResult,
 TResult Func<in T1, in T2,
```

- Встроенный делегат
- Действие, которое имеет результат (функция/метод)
- Может принимать до 16 параметров

Action

```
private Func<int> _func = () => 42;
private Func<int, int, int> _funcWithParameters = (p1, p2) => p1 * p2;

public delegate TResult Func<out TResult>();
public delegate TResult Func<out TResult, in T>(T obj);
public delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2);
// ...
```

- Встроенный делегат
- Действие, которое имеет результат (функция/метод)
- Может принимать до 16 параметров



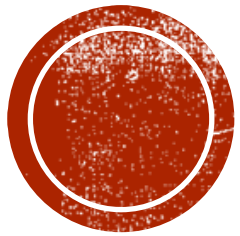
EVENT

```
public class Account
{
    public Account(int startBalance)
    {
        _balance = startBalance;
    }

    public event Action<int>? BalanceUpdated;

    private int _balance;

    public int Balance
    {
        get => _balance;
        set
        {
            _balance = value;
            BalanceUpdated?.Invoke(_balance);
        }
    }
}

public static class Caller
{
    public static void Call()
    {
        var account = new Account(100);
        account.BalanceUpdated += newBalance => Console.WriteLine($"Текущий баланс: {newBalance}");

        account.Balance -= 10;
        account.Balance -= 20;
        account.Balance -= 30;
    }
}
```

СОБЫТИЯ

- Позволяют перейти от линейного выполнения программы к обработке событий.

- События объявляются с помощью ключевого слова event, после которого указывается тип делегата, который представляет событие.
- С событием связывается один или несколько обработчиков.
- Для добавления обработчика события применяется операция +=:
- Обработчик может быть не определен, поэтому при вызове события лучше его всегда проверять на null.
- **add/remove accessors**



ЗАМЫКАНИЯ

ЗАМЫКАНИЕ

- Замыкания – функция имеющая доступ и использующая переменные из внешнего контекста выполнения/окружения.
- Замыкание состоит из:
  - Внешнего метода, который определяет область видимости и в которой определены переменные и параметры
  - переменные и параметры (лексическое окружение), которые определены во внешней функции
  - вложенная функция

```
public class ClosureExample
{
    public static void LambdaExample()
    {
        var outerFn = () =>
        {
            var x = 10;
            return () => Console.WriteLine(++x);
        };

        var fn = outerFn();

        fn();   // 11
        fn();   // 12
        fn();   // 13
    }

    delegate int Operation(int n);

    public static void InnerFunctionExample()
    {
        var fn = Multiply(5);

        Console.WriteLine(fn(5));   // 25
        Console.WriteLine(fn(6));   // 30
        Console.WriteLine(fn(7));   // 35

        Operation Multiply(int n)
        {
            int Inner(int m)
            {
                return n * m;
            }
            return Inner;
        }
    }
}
```

ЗАМЫКАНИЕ НА ПРАКТИКЕ

- На практике замыкание можно получить:
  - При помощи лямбда-выражения
  - При помощи локальной функции