Промышленное программирование

Лекция 4

❶ Делегаты. Лямбда-выражения
❷ События
❸ LINQ

❶ **Делегаты. Лямбда-выражения**
❷ События
❸ LINQ

Делегаты

```
delegate bool Predicate(int value);

static bool IsEven(int value) => value % 2 == 0;

static bool IsOdd(int value) => value % 2 != 0;

static void Print(IEnumerable<int> data, Predicate p) {
    foreach (var value in data)
        if (p(value))
            Console.WriteLine(value);
}

static void Main() {
    var data = new[] {1, 2, 3};
    Print(data, IsEven);
    Print(data, IsOdd);
}
```

System.Action

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

static void SayHello(string name) => Console.WriteLine($"Hello, {name}");

static void SayHi(string name) => Console.WriteLine($"Hi, {name}");

static void Main()
{
    Action<string> action = new Random().Next() % 2 == 0
        ? SayHello
        : SayHi;
    action("Victor");
}
```

System.Func

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

static string SayHello(string name) => $"Hello, {name}";

static string SayHi(string name) => $"Hi, {name}";

static void Main()
{
    Func<string, string> action = new Random().Next() % 2 == 0
        ? SayHello
        : SayHi;
    Console.WriteLine(action("Victor"));
}
```

Локальные функции и лямбда-выражения

```
static void Main()
{
    var name = "Victor";

    Console.WriteLine(SayHello(name));
    Console.WriteLine(SayHi());

    static string SayHello(string name)
        => $"Hello, {name}";

    string SayHi()
        => $"Hi, {name}";
}
```

```
static void Main()
{
    Func<string, string> hello =
        (string name) =>
        {
            return $"Hello, {name}";
        };

    Func<string, string> hi =
        name => $"Hi, {name}";

    var action = new Random().Next() % 2 == 0
        ? hello
        : hi;

    Console.WriteLine(action("Victor"));
}
```

System.MulticastDelegate

```
static void Main()
{
    var func = () =>
    {
        Console.WriteLine("#1");
        return 1;
    };

    func += () =>
    {
        Console.WriteLine("#2");
        return 2;
    };

    Console.WriteLine(func());
}
```

❶ Делегаты. Лямбда-выражения
❷ **События**
❸ LINQ

Инфраструктура System.ComponentModel

```
public class PropertyChangedEventArgs : EventArgs
{
    public PropertyChangedEventArgs(string? propertyName);

    public virtual string? PropertyName { get; }
}


delegate void PropertyChangedEventHandler(object? sender, PropertyChangedEventArgs e)


interface INotifyPropertyChanged
{
    event PropertyChangedEventHandler? PropertyChanged;
}
```

Пример: издатель, первая версия

```
class Person : INotifyPropertyChanged {
    public string FirstName { get => _firstName; set => SetField(ref _firstName, value) }
    public string LastName { get => _lastName; set => SetField(ref _lastName, value); }

    public event PropertyChangedEventHandler? PropertyChanged {
        add => _propertyChanged += value;
        remove => _propertyChanged -= value;
    }

    private void SetField<T>(ref T field, T value, [CallerMemberName] string propertyName = "") {
        if (EqualityComparer<T>.Default.Equals(field, value))
            return;
        field = value;
        _propertyChanged?.Invoke(this, new(propertyName));
    }

    private PropertyChangedEventHandler? _propertyChanged;
    private string _firstName = string.Empty;
    private string _lastName = string.Empty;
}
```

```
// using System.ComponentModel;
// using System.Collections.Generic;
// using System.Runtime.CompilerServices;
```

Пример: издатель, вторая версия

```
class Person : INotifyPropertyChanged {
    public string FirstName { get => _firstName; set => SetField(ref _firstName, value) }
    public string LastName { get => _lastName; set => SetField(ref _lastName, value); }

    public event PropertyChangedEventHandler? PropertyChanged;

    private void SetField<T>(ref T field, T value, [CallerMemberName] string propertyName = "") {
        if (EqualityComparer<T>.Default.Equals(field, value))
            return;
        field = value;
        PropertyChanged?.Invoke(this, new(propertyName));
    }

    private string _firstName = string.Empty;
    private string _lastName = string.Empty;
}
```

Пример: издатель, третья версия

Для реализации **именно** INotifyPropertyChanged можно использовать библиотеку https://www.nuget.org/packages/ReactiveUI.Fody/

```
using ReactiveUI;
using ReactiveUI.Fody.Helpers;

class Person : ReactiveObject
{
    [Reactive]
    public string FirstName { get; set; } = string.Empty;

    [Reactive]
    public string LastName { get; set; } = string.Empty;
}
```

Пример: подписчик(и)

```
static void Main()
{
    var p = new Person
    {
        FirstName = "Severus",
        LastName = "Snape"
    };

    p.PropertyChanged += (sender, args) =>
    {
        Console.WriteLine($"Property {args.PropertyName} changed.");
    };

    p.FirstName = "Harry";
    p.LastName = "Potter";
}
```

! ! !

Один из фундаментальных паттернов
1. Observer
2. Издатель-подписчик
3. Наблюдатель
4. Слушатель

https://refactoring.guru/ru/design-patterns/observer

Ключевая конструкция для реализации реактивной парадигмы
Реактивное программирование – декларативная парадигма, ориентированная на потоки данных и распространение изменений
http://introtorx.com/

```
var expression = a + b;
```

❶ Делегаты. Лямбда-выражения
❷ События
❸ **LINQ**

Методы расширения

```
class Person {
    public string FirstName { get; init; }
    public string LastName { get; init; }
}

static class PersonExtensions {
    public static string GetFullName(this Person person)
        => $"{person.FirstName} {person.LastName}";
}

static class Program {
    static void Main() {
        var person = new Person { FirstName = "Severus", LastName = "Snape" };
        Console.WriteLine(PersonExtensions.GetFullName(person));
        Console.WriteLine(person.GetFullName());
    }
}
```

Методы-итераторы

```
static IEnumerable<int> GetItems()
{
    Console.WriteLine("#1");
    yield return 1;
    Console.WriteLine("#2");
    yield return 2;
    Console.WriteLine("#End");
}

static void Main()
{
    var items = GetItems();
    foreach (var value in items)
        Console.WriteLine(value);
}
```

LINQ

Методы расширения для System.Collections.Generic.IEnumerable<T>:
https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable

```
       IEnumerable<T> Empty<T>();
                 bool All<T>(..., Func<T, bool> predicate);
                 bool Any<T>(...);
                 bool Contains<T> (..., T value);
                  int Count<T> (...);
                  int Count<T> (..., Func<T, bool> predicate);
       IEnumerable<T> OfType<T> (...);
IOrderedEnumerable<T> OrderBy<T, TKey> (..., Func<T, TKey> keySelector);
IOrderedEnumerable<T> OrderByDescending<T, TKey> (..., Func<T, TKey> keySelector);
 IEnumerable<TResult> Select<T, TResult> (..., Func<T, TResult> selector);
                    T Single<T> (...);
                    T Single<T> (..., Func<T, bool> predicate);
       IEnumerable<T> Skip<T> (..., int count);
       IEnumerable<T> SkipLast<T> (..., int count);
IOrderedEnumerable<T> ThenBy<T, TKey> (... /* IOrdered */, Func<T, TKey> keySelector);
                  ... To[Array|Dictionary|HashSet|List](...);
       IEnumerable<T> Where<T> (..., Func<T, bool> predicate);
       IEnumerable<T> Where<T> (..., Func<T, int, bool> predicate);
```

LINQ: Method Syntax

```
static IEnumerable<string> Items {
    get {
        yield return "1";
        yield return "3";
        yield return "5";
        yield return "2";
    }
}

static void Main() {
    var result = Items
        .Skip(1)
        .Select(int.Parse)
        .Where(value => value % 2 == 1)
        .OrderByDescending(value => value);

    foreach (var value in result)
        Console.WriteLine(value);
}
```

LINQ: Query Syntax

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/query-keywords

```
var result = Items
    .Skip(1)
    .Select(int.Parse)
    .Where(value => value % 2 == 1)
    .OrderByDescending(value => value);

var result =
    from value in
        from stringValue in Items.Skip(1)
        select int.Parse(stringValue)
    where value % 2 == 1
    orderby value descending
    select value;
```

Заключение

1. Делегаты
    1. Идейно – синтаксический сахар
    2. Аналог указателя на функцию
    3. По умолчанию может хранить ссылки на несколько методов
    4. Для очевидных случаев есть Action<> и Func<>
2. События
    1. Идейно – синтаксический сахар
    2. События – основа паттерна «Наблюдатель»
    3. Паттерн «Наблюдатель» – основа реактивной парадигмы
3. Синтаксический сахар: методы расширения и методы-итераторы
4. Методы-итераторы – синтаксический сахар
5. LINQ: готовые методы-расширения для перечислимых типов