

# Программирование на платформе .NET

Введение, основы C#

Юрий Литвинов

yurii.litvinov@gmail.com

07.09.2017

# О курсе

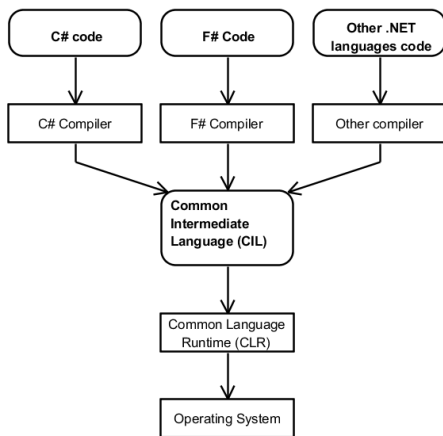
- ▶ Рассказ про основные языки для .NET — C# и (немного) F#
- ▶ Немного про саму платформу
- ▶ Будет также про основные библиотеки и технологии (WinForms, WPF, MVC, EF и т.д.)
- ▶ Лекции (каждую неделю), семинары (не каждую неделю), домашние задания
- ▶ Оценка: домашние задания (70%), экзамен в конце курса (30%)

# Язык C#

- ▶ Объектно-ориентированный язык общего назначения с сильной типизацией
- ▶ Основной язык программирования для платформы .NET
- ▶ Первая версия — 2002 год, актуальная — 07.03.2017, C# 7.0
- ▶ Международный стандарт (ISO/IEC 23270:2006)
- ▶ В основном для прикладного ПО
- ▶ 5-е место в индексе TIOBE
- ▶ Работает под Windows (.NET, .NET Core) и Linux/Mac OS (Mono, .NET Core)
- ▶ Средства разработки:
  - ▶ Rider (<https://www.jetbrains.com/rider/>)
  - ▶ Microsoft Visual Studio (<https://www.visualstudio.com/>)
  - ▶ MonoDevelop (<http://www.monodevelop.com/>)
  - ▶ Visual Studio Code (<https://code.visualstudio.com/>)

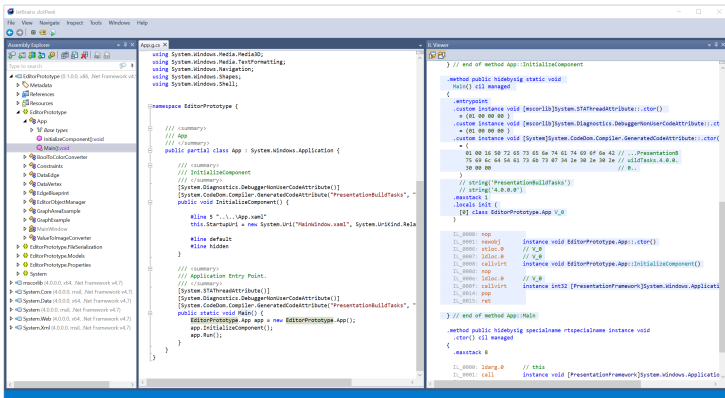
# Common Language Infrastructure

- ▶ Компиляция в байткод виртуальной машины (Common Intermediate Language, CIL)
- ▶ Виртуальная машина и набор библиотек (Common Language Runtime) реализуется для каждой платформы (ОС)
- ▶ Машина интерпретирует байт-код или компилирует его «на лету» в машинные коды
- ▶ Прежде всего для облегчения разработки компиляторов и обеспечения взаимодействия языков



# Байт-код

## Дизассемблер JetBrains dotPeek:



Ещё есть ildasm.exe

# Технические подробности C#

Как обычно, Hello, World

```
using System;
```

```
namespace HelloWorld
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Goodbye, cruel world!");
```

```
        }
```

```
    }
```

```
}
```

# Циклы

```
for (int i = 0; i < 300; ++i)
{
    Console.WriteLine("Hello, world!");
}
```

или

```
for (var i = 0; i < 300; ++i)
{
    Console.WriteLine("Hello, world!");
}
```

# Методы

```
private static int Factorial(int n)
{
    if (n <= 1)
    {
        return 1;
    }

    return n * Factorial(n - 1);
}
```

или так:

```
private static int Factorial(int n)
    => n <= 1 ? 1 : n * Factorial(n - 1);
```



# Стайлгайд

- ▶ C# Coding Conventions
- ▶ <https://stylecop.codeplex.com/>
  - ▶ <https://github.com/DotNetAnalyzers/StyleCopAnalyzers>
- ▶ Code Analysis for Managed Code (бывший FxCop)

# Элементарные типы

- ▶ Всё — объект, даже **int** наследуется от Object
- ▶ Типы стандартизованы: размер и множество значений одинаковы во всех реализациях
- ▶ Каждому типу соответствует библиотечный класс
  - ▶ Например, **int** — System.Int32
- ▶ У каждого типа есть значение по умолчанию
  - ▶ Им переменные и поля инициализируются при создании
  - ▶ Ключевое слово **default**, особо полезно в генериках

# Методы у типов

```
var inputString = Console.ReadLine();  
int number = int.Parse(inputString);
```

— это то же самое, что

```
var inputString = Console.ReadLine();  
int number = Int32.Parse(inputString);
```

# Массивы

```
int[] a = new int[10];
```

или

```
var a = new int[10];
```

Пример:

```
for (var i = 0; i < a.Length; ++i)
{
    a[i] = i;
}
```

Двумерные массивы:

```
int[,] numbers = new int[3, 3];
```

```
numbers[1, 2] = 2;
```

```
int[,] numbers2 = new int[3, 3] { {2, 3, 2}, {1, 2, 6}, {2, 4, 5} };
```

# Перечисления

Объявление:

```
enum SomeEnum  
{  
    red,  
    green,  
    blue  
}
```

Использование:

```
SomeEnum a = SomeEnum.blue;
```

(ну или через **var**: **var** a = SomeEnum.blue;)

# Структуры

## **struct Point**

```
{  
    public int x;  
    public int y;  
}
```

- ▶ Тип-значение
- ▶ Может содержать поля, методы, конструкторы и т.д.
- ▶ Не может иметь конструктор без параметров
- ▶ Может быть создан без вызова конструктора
- ▶ Может реализовывать интерфейсы
  - ▶ При касте к интерфейсу выполняется боксинг, опасайтесь
- ▶ Не может наследоваться

# Ссылочные типы и типы-значения

## Ссылочные типы:

- ▶ Классы
- ▶ Строки
- ▶ Массивы
- ▶ Исключения
- ▶ Делегаты

## Типы-значения:

- ▶ Примитивные типы
- ▶ Перечисления
- ▶ Структуры

# Пример

```
static void Add(string s1, string s2, string s3)
{
    s3 = s1 + s2;
}
```



# Передача параметров по ссылке

```
static void Add(string s1, string s2, ref string s3)
```

```
{  
    s3 = s1 + s2;  
}
```

```
private static void Main(string[] args)
```

```
{  
    string s1 = "a";  
    string s2 = "b";  
    string s3 = "c";  
    Add(s1, s2, ref s3);  
}
```

# Out-параметры

```
static void Add(string s1, string s2, out string s3)
```

```
{  
    s3 = s1 + s2;  
}
```

```
private static void Main(string[] args)
```

```
{  
    string s1 = "a";  
    string s2 = "b";  
    Add(s1, s2, out string s3);  
    Console.WriteLine(s3);  
}
```

# Кортежи и деконструкция

```
static (int prev, int cur) Fibonacci(int n)
{
    var (prevPrev, prev) = n <= 2 ? (0, 1) : Fibonacci(n - 1);
    return (prev, prevPrev + prev);
}
```

```
private static void Main(string[] args)
{
    var (_, result) = Fibonacci(7);
    Console.WriteLine(result);
}
```

# Конструкторы

```
class Circle
{
    public Circle(int x, int y, int r)
    {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    private int x;
    private int y;
    private int r;
}
```

# Перегрузка конструкторов, chaining

## class Circle

```
{  
    public Circle(int x, int y, int r)  
    {  
        this.x = x;  
        this.y = y;  
        this.r = r;  
    }  
  
    public Circle(int r)  
        : this(0, 0, r)  
    {  
    }  
  
    private int x;  
    private int y;  
    private int r;  
}
```

# Статические конструкторы

```
public class Bus
```

```
{  
    static Bus()  
    {  
        System.Console.WriteLine("The static constructor invoked.");  
    }  
}
```

```
public static void Drive()
```

```
{  
    System.Console.WriteLine("The Drive method invoked.");  
}  
}
```

```
class TestBus
```

```
{  
    static void Main()  
    {  
        Bus.Drive();  
    }  
}
```

# Наследование (1)

```
class Shape
```

```
{
```

```
    public Shape()
```

```
    {
```

```
    }
```

```
    public Shape(int x, int y)
```

```
    {
```

```
        this.x = x;
```

```
        this.y = y;
```

```
    }
```

```
    protected int x;
```

```
    protected int y;
```

```
}
```

## Наследование (2)

```
class Circle : Shape
{
    Circle(int x, int y, int r)
    {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    Circle(int r)
        : base(0, 0)
    {
    }

    private int r;
}
```



# Интерфейсы

```
interface IDrawable  
{  
    void Draw();  
}
```

# Реализация интерфейса

```
class Shape : IDrawable
{
    public void Draw()
    {
        Console.WriteLine("Drawing Shape");
    }

    protected int x;
    protected int y;
}
```

# Явная реализация интерфейса

```
class Shape : IDrawable
{
    void IDrawable.Draw()
    {
        Console.WriteLine("Drawing Shape");
    }

    protected int x;
    protected int y;
}
```

# Абстрактные классы

```
abstract class Shape
```

```
{  
    public Shape()  
    {  
    }  
  
    public abstract void Draw();  
  
    protected int x;  
    protected int y;  
}
```

# Виртуальные методы (1)

```
class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine(
            "Drawing Shape with coords ({0}, {1})", x, y);
    }

    protected int x;
    protected int y;
}
```

## Виртуальные методы (2)

**class Circle** : Shape

```
{  
    public Circle(int x, int y, int r)  
    {  
        this.x = x;  
        this.y = y;  
        this.r = r;  
    }  
}
```

```
public override void Draw()  
{  
    Console.WriteLine("Drawing Circle with radius {0}", r);  
}
```

```
private int r;  
}
```

# Виртуальные методы (3)

```
class Rectangle : Shape
{
    public Rectangle(int x, int y, int width, int height)
    {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    public override void Draw()
    {
        base.Draw();
        Console.WriteLine(
            "Drawing Rectangle with width={0} and height={1}", width, height);
    }
    protected int width;
    protected int height;
}
```

## Виртуальные методы (4)

```
private static void Main(string[] args)
{
    var circle = new Circle(0, 0, 10);
    var rectangle = new Rectangle(0, 0, 10, 10);
    var list = new System.Collections.Generic.List<Shape>();
    list.Add(circle);
    list.Add(rectangle);
    foreach (var shape in list)
    {
        shape.Draw();
    }
}
```



# Абстрактные методы

```
abstract class Shape
{
    public abstract void Draw();

    protected int x;
    protected int y;
}
```

# Переведение методов, new (1)

```
class Shape
```

```
{
```

```
    public virtual void Draw()
```

```
    {
```

```
        Console.WriteLine(
```

```
            "Drawing Shape with coords ({0}, {1})", x, y);
```

```
    }
```

```
    protected int x;
```

```
    protected int y;
```

```
}
```

## Переведение методов, new (2)

```
class Circle : Shape
{
    public Circle(int x, int y, int r)
    {
        this.x = x;
        this.y = y;
        this.r = r;
    }

    public new void Draw()
    {
        Console.WriteLine("Drawing Circle with radius {0}", r);
    }

    private int r;
}
```

## Переведение методов, new (3)

```
Circle circle1 = new Circle(10, 10, 3);  
Shape circle2 = new Circle(10, 10, 3);  
var circle3 = new Circle(10, 10, 3);
```

```
circle1.Draw();  
circle2.Draw();  
circle3.Draw();
```

# Модификаторы видимости

- ▶ **public** — применяется к типам и членам, доступ без ограничений
- ▶ **protected** — применяется только к членам, доступ в типе и потомках
- ▶ **internal** — применяется к типам и членам, доступ внутри сборки
- ▶ **protected internal** — применяется к типам и членам, доступ внутри сборки **или** в потомках
- ▶ **private** — применяется к типам и членам, доступ только внутри типа
- ▶ По умолчанию для типов **internal**, для членов — **private**

## Другие модификаторы

- ▶ **partial** — «частичный» класс, декларирует, что определение класса разбито на несколько файлов
  - ▶ Для интеграции сгенерированного и рукописного кода, не используйте без нужды
- ▶ **sealed** — запрещение наследования от класса
- ▶ **static** — не может быть инстанцирован, может содержать только **static**-методы

# Вложенные классы

```
class Circle
{
    private readonly Point pos;
    private readonly int r;

    private class Point
    {
        public int x;
        public int y;
    }

    public Circle(int x, int y, int r)
    {
        pos = new Point {x = 10, y = 10};
        this.r = r;
    }

    public void Draw() =>
        Console.WriteLine($"{pos.x}, {pos.y}), radius {r}");
}
```

# Преобразования типов

```
Shape shape = new Circle();  
Circle circle = (Circle)shape;
```

```
Shape shape = new Circle();  
Circle circle = shape as Circle;
```

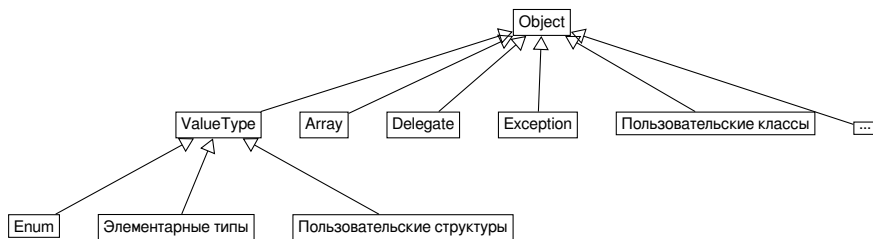
```
Shape shape = new Circle();  
if (shape is Circle)  
{  
    Circle circle = (Circle)shape;  
}
```



# Преобразование с сопоставлением шаблонов

```
switch (shape)
{
    case Circle c:
        WriteLine($"circle with radius {c.Radius}");
        break;
    case Rectangle s when (s.Length == s.Height):
        WriteLine($" {s.Length} x {s.Height} square");
        break;
    case Rectangle r:
        WriteLine($" {r.Length} x {r.Height} rectangle");
        break;
    default:
        WriteLine("<unknown shape>");
        break;
    case null:
        throw new ArgumentException(nameof(shape));
}
```

# Иерархия основных классов

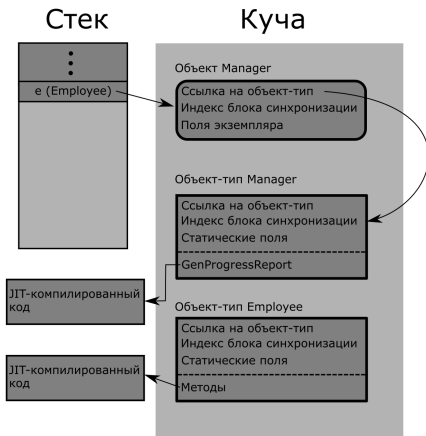


# Методы System.Object

- ▶ Equals — виртуальный
- ▶ GetHashCode — виртуальный
- ▶ ToString — виртуальный
- ▶ GetType — не виртуальный
- ▶ MemberwiseClone — не виртуальный защищённый
  - ▶ Создаёт объект, не вызывая конструктор
- ▶ Finalize — виртуальный защищённый

# Объекты в памяти

```
void Example()  
{  
    Employee e =  
        new Manager();  
    e.GenProgressReport();  
}
```



# Литература

- ▶ **Jeffrey Richter.** CLR via C# — must read про C#
- ▶ <https://blogs.msdn.microsoft.com/dotnet> — официальный блог о дотнете
- ▶ **Don Syme et al.** Expert F# — годная книга по F# от автора языка
- ▶ <https://fsharpforfunandprofit.com/> — набор статей про F#, где всё объясняется по-человечески
- ▶ <http://www.slideshare.net/ScottWlaschin/fp-patterns-buildstuffit> — лучшее объяснение про монады и вообще отличное введение в F#
- ▶ <http://tomasp.net/blog> — ещё один годный блог про C# и F#
- ▶ <https://habrahabr.ru/post/172627/> — статья про особенности строк в .NET
- ▶ **Adam Nathan.** WPF 4.5 Unleashed — годная книжка по WPF