

# Объектно-ориентированное программирование на C#

## Введение

Юрий Литвинов  
yurii.litvinov@gmail.com

17.02.2022

# Про что этот курс

- ▶ Объектно-ориентированное программирование и близкие вещи
  - ▶ Абстракция-инкапсуляция-наследование-полиморфизм
  - ▶ Исключения
  - ▶ Генерики
  - ▶ События
  - ▶ Пользовательские интерфейсы
- ▶ Технология разработки ПО
  - ▶ Юнит-тесты
  - ▶ CI
  - ▶ Визуальное моделирование
- ▶ Язык — C#
- ▶ Не будет новых алгоритмов, будут новые технологии

# Организационное

- ▶ Пары раз в неделю
- ▶ Отчётность
  - ▶ Домашки, много в начале
    - ▶ Сдавать через GitHub, пуллреквестом в свой репозиторий
    - ▶ Blackboard, MS Teams, Telegram
    - ▶ Дедлайны: назначаете сами, но не больше полутора месяцев
    - ▶ Надо всё сдать до 21 мая
  - ▶ Контрольная в середине семестра (одна)
  - ▶ Зачётная работа в конце семестра
  - ▶ Доклады (+10 баллов)

# Критерии оценивания

- ▶ Шкала оценивания ECTS
- ▶ Итоговый балл за домашки:  $MAX(0, (\frac{n}{N}-0.6)) * 2.5 * 100$ 
  - ▶ Если сделано меньше 60% — это 0, если 80% — 50 баллов
- ▶ Три «бесплатные» попытки сдачи
- ▶ Отдельный дедлайн на исправления (тоже выбираете сами)
- ▶ Итоговый балл за контрольные:  $\frac{n}{N} * 100$ , их можно переписывать
- ▶ Зачёт теперь учитывается отдельно, балл считается так же
- ▶ В качестве итогового берётся **минимум** из этих баллов

# Ориентировочно

- ▶ Всего около 67 баллов за д/з и по 10 за к/р и зачёт
  - ▶ Будет немного бонусных баллов к домашке
- ▶ А — 65 баллов за домашки, 9 за контрольную и зачёт
- ▶ В — 62 баллов за домашки, 8 за контрольную и зачёт
- ▶ С — 59 баллов за домашки, 7 за контрольную и зачёт
- ▶ D — 57 баллов за домашки, 6 за контрольную и зачёт
- ▶ E — 54 баллов за домашки, 5 за контрольную и зачёт

# Литература

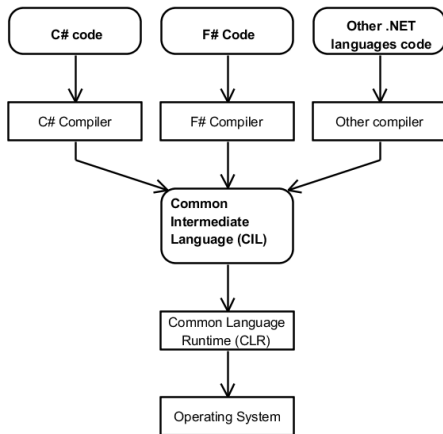
- ▶ **Джепикс Троелсен** Язык программирования C# 7 и платформы .NET и .NET Core — C# для самых маленьких
- ▶ **Джозеф Албахари, Бен Албахари** C# 7.0. Справочник. Полное описание языка
- ▶ **Jeffrey Richter.** CLR via C# — must read про C#
  - ▶ Или её русское издание “CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#”
- ▶ <https://ulearn.me/> — набор открытых онлайн-курсов от «Контур» и УрГУ
  - ▶ Очень рекомендую все три курса, если наш покажется сложным — мы за один семестр проходим почти всё, что они за три
- ▶ <https://blogs.msdn.microsoft.com/dotnet> — официальный блог о .NET

# Язык C#

- ▶ Объектно-ориентированный язык общего назначения с сильной типизацией
- ▶ Основной язык программирования для платформы .NET
- ▶ Первая версия — 2002 год, актуальная — 10.11.2020, C# 9.0
- ▶ В основном для прикладного ПО
- ▶ 5-е место в индексе TIOBE на февраль 2021 (после C, Java, Python, C++)
  - ▶ <https://www.tiobe.com/tiobe-index/>
- ▶ Работает под Windows и Linux/Mac OS
- ▶ Средства разработки
  - ▶ Microsoft Visual Studio (<https://www.visualstudio.com>)
  - ▶ Rider (<https://www.jetbrains.com/rider/>)
  - ▶ Visual Studio Code (<https://code.visualstudio.com/>)
  - ▶ да любой удобный текстовый редактор

# Common Language Infrastructure

- ▶ Компиляция не в машинные коды, а в байткод виртуальной машины (Common Intermediate Language, CIL)
- ▶ Виртуальная машина и набор библиотек (Common Language Runtime) реализуется для каждой платформы (ОС), на которой хотим запускать байт-код
- ▶ Машина интерпретирует байт-код или компилирует его “на лету” в машинные коды



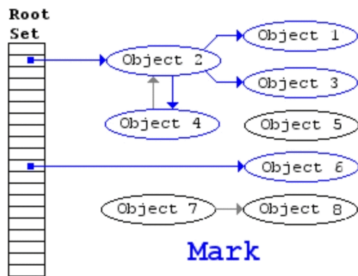


# Зачем так

- ▶ Единый байткод с механизмом оптимизации, интерпретации и генерации машинного кода позволяет экономить время на разработку компиляторов
  - ▶ Поэтому под .NET можно писать на C#, F#, Visual Basic, J# (зачем бы он ни был нужен), куче сторонних языков (например, Delphi)
- ▶ Совместимость кода, написанного на разных языках, возможность из кода на F# без усилий использовать библиотеки на C# и наоборот
- ▶ Виртуальная машина знает всё о выполнении программы и многое может проверять (обращения к памяти, границы массивов и т.д.)
- ▶ JIT-компиляция
- ▶ Такая схема использовалась для Паскаля и Лиспа аж в 1970-х

# Сборка мусора

- ▶ Виртуальная машина сама следит за используемой памятью и освобождает её, когда она перестаёт быть нужной
- ▶ Корневое множество (глобальные переменные и стек вызовов), достижимое множество
- ▶ Вызывается, “когда захочет”
- ▶ Относительно вычислительно сложно
- ▶ Устроено обычно довольно хитро
  - ▶ Поколения, Large Object Heap и т.д.



# Технические детали C#

Как обычно, сначала Hello, world

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

## Или «полная форма»

```
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("Мат-мех лучше всех!");
}
```

или

```
for (var i = 0; i < 300; ++i)
{
    Console.WriteLine("Мат-мех не для всех!");
}
```

# Функции

```
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);
```

# Использование

```
using System;
```

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

```
Console.WriteLine(Factorial(4));
```

## Или «полная» форма

```
namespace HelloWorld
```

```
{
```

```
    class Program
```

```
    {
```

```
        private static int Factorial(int n)
```

```
        {
```

```
            return n <= 1 ? 1 : n * Factorial(n - 1);
```

```
        }
```

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

```
        {
```

```
            System.Console.WriteLine(Factorial(4));
```

```
        }
```

```
    }
```

```
}
```



# Стайлгайд

- ▶ C# Coding Conventions
  - ▶ <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>
- ▶ <https://github.com/DotNetAnalyzers/StyleCopAnalyzers>
- ▶ Code Analysis for Managed Code (бывший FxCop)
- ▶ Здравый смысл

# 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

# 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} };
```

Массивы знают свою длину (см. свойство `Length` и метод `GetLength()`)

# 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;
}
```