Объектно-ориентированное программирование на C#

Введение

Юрий Литвинов
yurii.litvinov@gmail.com

15.02.2019г

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
  - Code Review
  - Визуальное моделирование
- Язык — C#
- Не будет новых алгоритмов, будут новые технологии

Организационное

- Пары раз в неделю
- Отчётность
  - Домашки
    - Сдавать через GitHub, пуллреквестом в свой репозиторий
    - HwProj
  - Контрольная в середине семестра (одна)
  - Зачётная работа в конце семестра
  - Доклады (-1 домашка)

Литература

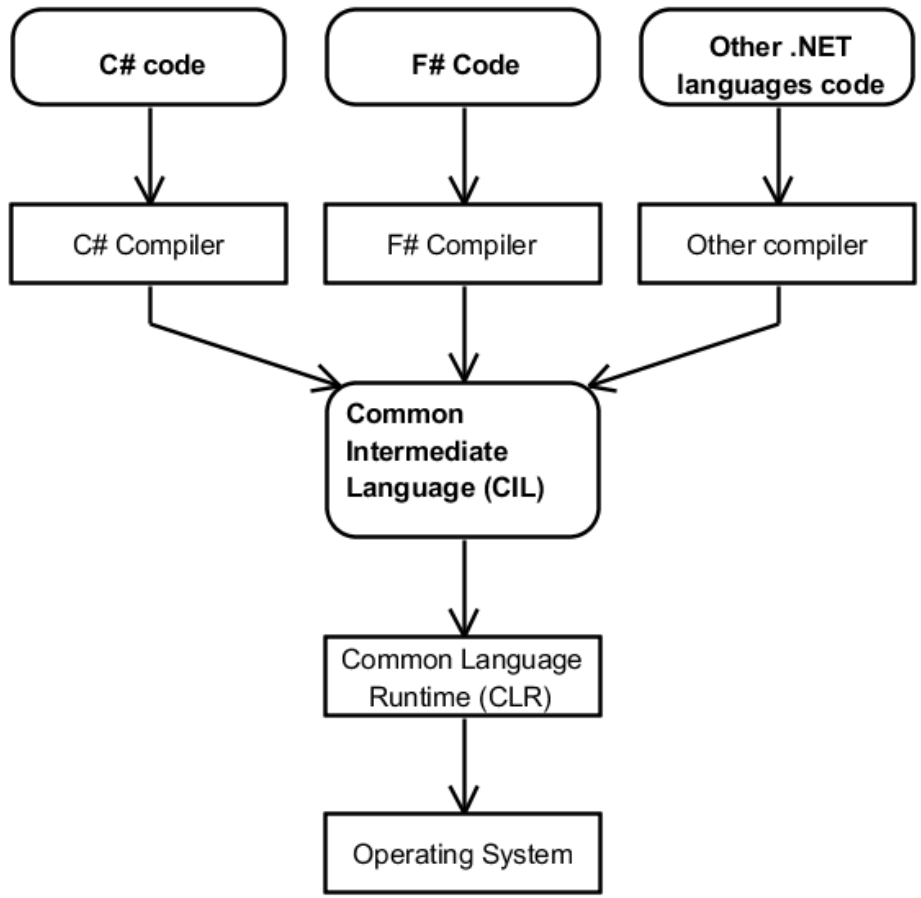
- **Jeffrey Richter.** CLR via C# — must read про C#
  - Или её русское издание “CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#”
- https://blogs.msdn.microsoft.com/dotnet — официальный блог о дотнете

Язык C#

- Объектно-ориентированный язык общего назначения с сильной типизацией
- Основной язык программирования для платформы .NET
- Первая версия — 2002 год, актуальная — 07.05.2018, C# 7.3
- В основном для прикладного ПО
- 7-е место в индексе TIOBE на январь 2019
  - https://www.tiobe.com/tiobe-index/
- Работает под Windows (.NET Framework, .NET Core) и Linux/Mac OS (.NET Core, Mono)
- Средства разработки
  - Microsoft Visual Studio (https://www.visualstudio.com)
  - Rider (https://www.jetbrains.com/rider/)
  - Visual Studio Code (https://code.visualstudio.com/)
  - MonoDevelop (http://www.monodevelop.com/)
  - Atom, Sublime, ...

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

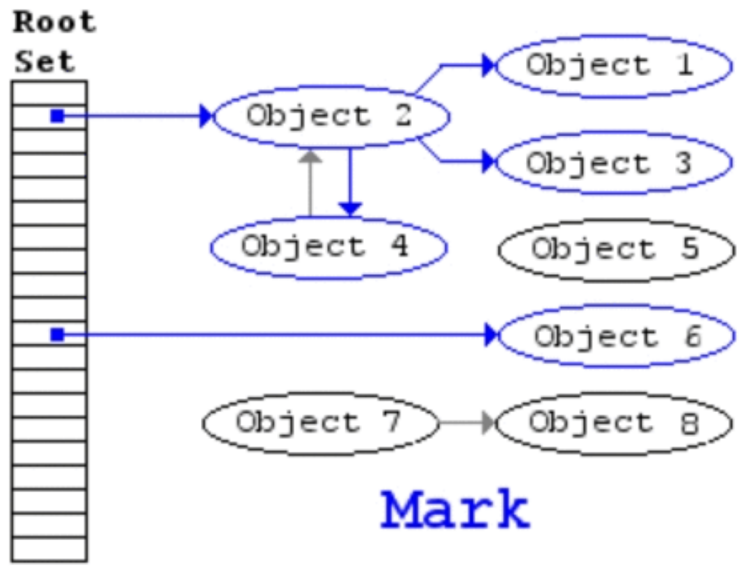


Зачем так

- 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

```
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!");
}
```

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

```
namespace HelloWorld
{
    class Program
    {
        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} };
```

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;
}
```