C#

- **C#** (произносится си шарп) — разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы **Microsoft.NET**.

Символ # в названии языка можно интерпретировать как две пары плюсов ++, намекающие на новый шаг в развитии языка по сравнению с C++

- Компилятор с **C#** входит в стандартную установку самой **.NET** (читается Dot Net), поэтому программы на нём можно создавать и компилировать даже без инструментальных средств вроде Visual Studio.

- Андерс Хейлсберг (Anders Hejlsberg; род. в декабре 1960, Копенгаген) — датский инженер-программист.
- В 1980 году написал свой первый компилятор языка Паскаль и продал его фирме **Borland**.

- Эта версия легла в основу **Turbo/Borland Pascal**, который развивался до 1995 года. До 1996 года Хейлсберг был главным проектировщиком фирмы **Borland**, где создал новое поколение компиляторов Паскаля — язык **Delphi**, компилятор которого работал уже под операционной системой Windows.
- В 1996 году он перешёл в **Microsoft**, где работал над такими проектами, как **J++** (версия Java) и **Microsoft Foundation Classes**. Позже возглавил группу по созданию и проектированию языка **C#**.

- ***C#*** относится к семье языков с С-подобным синтаксисом, из них его синтаксис наиболее близок к ***C++*** и ***Java***.

- Переняв многое от своих предшественников — языков ***C++***, ***Java***, ***Delphi***, ***Модула*** и ***Smalltalk*** — **C#**, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем

- Авторы **C#** стремились создать язык, сочетающий простоту и выразительность современных объектно-ориентированных языков (вроде Java) с богатством возможностей и мощью ***C++***. По словам Андерса Хейлсберга, **C#** позаимствовал большинство своих синтаксических конструкций из ***C++.*** Некоторые синтаксические конструкции ***C#*** унаследованы и от ***Visual Basic***.

История создания C#

- Проект **C#** был начат в декабре 1998 и получил кодовое название **COOL** (C-style Object Oriented Language).

- Версия 1.0 появилась вместе с платформой .NET в июне 2000 года, тогда же появилась и первая общедоступная бета-версия;

- **C# 1.0** окончательно вышел вместе с **Microsoft Visual Studio.NET** в феврале 2002 года.

C# 1.0	Февраль 2002	Managed code
C# 2.0	Сентябрь 2005	Iterators/Generics/Anonymous
C# 3.0	Август 2007	LINQ/Lambda
C# 4.0	Апрель 2010	Dynamic/PLIQ
C# 5.0	Август 2012	TAP (Task-based asynchronous pattern)

Версия 1.0

- Первая версия **C#** напоминала по своим возможностям Java 1.4, несколько их расширяя.

- Основным нововведением был управляемый код (Managed code). Теперь программистам не надо было думать об освобождении памяти. Платформа автоматически делала это за них.

Версия 2.0

Проект спецификации **C# 2.0** впервые был выложен Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), **C# 2.0** окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и .NET 2.0.

Новые возможности:

- Частичные типы (разделение реализации класса более чем на один файл).
- Обобщённые, или параметризованные типы (***generics***). В отличие от шаблонов ***C++***, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины.
- Новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова ***yield***.
- Анонимные методы, обеспечивающие функциональность замыкания.
- Поддержка 64-разрядных вычислений, что кроме всего прочего, позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных.

Версия 3.0

- В сентябре 2005 года вышли проект спецификации **C# 3.0** и бета-версия ***C#*** 3.0, устанавливаемая в виде дополнения к существующим Visual Studio 2005 и .NET 2.0. **C# 3.0** совместим с **C# 2.0** по генерируемому **MSIL**-коду; улучшения в языке — чисто синтаксические и реализуются на этапе компиляции.

Версия 4.0

- Версия ***C# 4.0***, которая была представлена в конце 2008 г.
- Новые возможности:
  - Возможность использования позднего связывания, для использования:
    - с языками с динамической типизацией (Python, Ruby)
    - с COM-объектами
    - отражения (reflection)
    - объектов с изменяемой структурой (DOM). Появляется ключевое слово dynamic.
  - Именованные и опциональные параметры
  - Новые возможности COM interop
  - Ковариантность и контравариантность
  - Контракты в коде (Code Contracts)

Версия 5.0

- Текущей является Версия C# 5.0, которая была представлена в апреле 2010 г..
- Два важных средства, внедренных в версии 4.0 и непосредственно связанных с программированием на C#, предоставляются не самим языком, а средой .NET Framework 4.0. Речь идет о поддержке параллельного программирования с помощью библиотеки распараллеливания задач (TPL) и параллельном варианте языка интегрированных запросов (PLINQ). Оба эти средства позволяют существенно усовершенствовать и упростить процесс создания программ, в которых применяется принцип параллелизма. И то и другое средство упрощает создание многопоточного кода, который масштабируется автоматически для использования нескольких процессоров, доступных на компьютере. В настоящее время широкое распространение получили компьютеры с многоядерными процессорами, и поэтому возможность распараллеливать выполнение кода среди всех доступных процессоров приобретает все большее значение практически для всех, кто программирует на C#.

Литература:

- А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд Язык программирования C#. Классика Computers Science. 4-е издание = C# Programming Language (Covering C# 4.0), 4th Ed. — СПб.: «Питер», 2012. — 784 с. — ISBN 978-5-459-00283-6
- Э. Стиллмен, Дж. Грин Изучаем C#. 2-е издание = Head First C#, 2ed. — СПб.: «Питер», 2012. — 704 с. — ISBN 978-5-4461-0105-4
- Эндрю Троелсен Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание = Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. — М.: «Вильямс», 2013. — 1312 с. — ISBN 978-5-8459-1814-7
- Джозеф Албахари, Бен Албахари C# 5.0. Справочник. Полное описание языка = C# 5.0 in a Nutshell: The Definitive Reference. — М.: «Вильямс», 2013. — 1008 с. — ISBN 978-5-8459-1819-2
- Герберт Шилдт. C# 4.0: полное руководство = C# 4.0 The Complete Reference. — М.: «Вильямс», 2010. — С. 1056. — ISBN 978-5-8459-1684-6
- Джон Скит. C#: программирование для профессионалов, 2-е издание = C# in Depth, 2nd Edition. — М.: «Вильямс», 2011. — 544 с. — ISBN 978-5-8459-1555-9
- Кристиан Нейгел, Карли Уотсон и др. Visual C# 2010: полный курс = Beginning Microsoft Visual C# 2010. — М.: Диалектика, 2010. — ISBN 978-5-8459-1699-0
- Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер. C# 4: Платформа .NET 4 для профессионалов = Professional C# 4 and .NET 4. — М.: Диалектика, 2010. — С. 1440. — ISBN 978-5-8459-1656-3

.NET Framework

- Программа на языке C# выполняется в среде .NET Framework
- Microsoft начала разрабатывать .NET Framework в конце 1990-х под именем «Next Generation Windows Services» (NGWS).

Версия	Номер версии	Дата выхода	Visual Studio	По умолчанию в Windows
1.0	1.0.3705.0	1 мая 2002 года	Visual Studio .NET	
1.1	1.1.4322.573	1 апреля 2003 года	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	11 июля 2005 года	Visual Studio 2005	Windows Vista, Windows 7, Windows Server 2008 R2
3.0	3.0.4506.30	6 ноября 2006 года	Visual Studio 2005 + расширения	Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2
3.5	3.5.21022.8	9 ноября 2007 года	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	12 апреля 2010 года	Visual Studio 2010	Windows 8, Windows Server 2012
4.5	4.5.50709.17929	15 августа 2012 года	Visual Studio 2012	Windows 8, Windows Server 2012

.NET Framework – интегрированный компонент Windows, содержит виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов:

- **FCL** (Framework Class Library) - библиотека классов;
- **CLR** (Common Language Runtime) - общеязыковая исполнительная среда

FCL (Framework Class Library)

- **.NET Framework** состоит прежде всего из огромной библиотеки программ, к которой можно обращаться из различных языков программирования с помощью различных технологий объектно-ориентированного программирования (**FCL** - Framework Class Library)

- Часть библиотеки **.NET Framework** посвящена описанию некоторых базисных **типов.** Тип — это способ представления данных; определение наиболее фундаментальных из них (например 32-разрядного целого со знаком) облегчает совместное использование языков программирования с помощью **.NET Framework**. Все вместе это называется **CTS** (Common Type System — единая система типов)

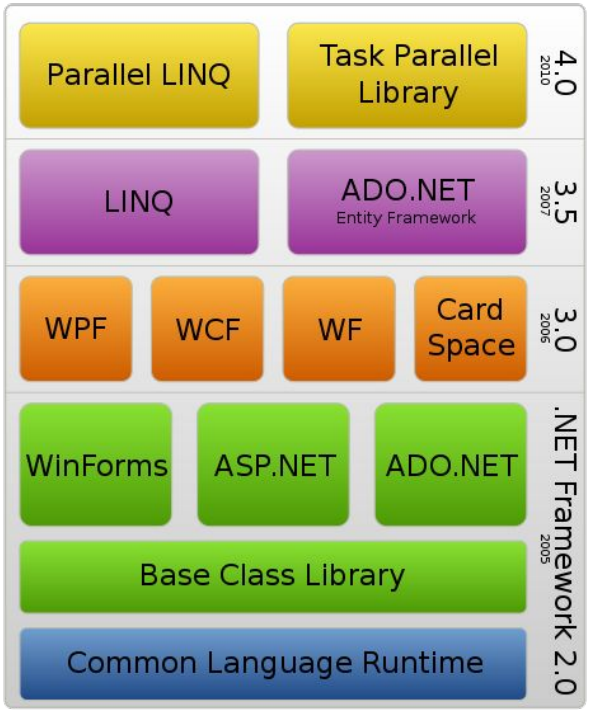
- Число классов библиотеки **FCL** велико (более 4 тысяч). Поэтому понадобился способ их структуризации. Логически классы с близкой функциональностью объединяются в группы, называемые пространством имен (**Namespace**).

- Основным пространством имен библиотеки **FCL** является пространство **System**, содержащее как классы, так и другие вложенные пространства имен. Так, уже упоминавшийся примитивный тип **Int32** непосредственно вложен в пространство имен **System** и его полное имя, включающее имя пространства - **System.Int32**.

- В пространство **System** вложен целый ряд других пространств имен.
  - Например, в пространстве **System.Collections** находятся классы и интерфейсы, поддерживающие работу с коллекциями объектов - списками, очередями, словарями.
  - В пространство **System.Collections**, в свою очередь, вложено пространство имен **Specialized**, содержащие классы со специализацией, например, коллекции, элементами которых являются только строки.
  - Пространство **System.Windows.Forms** содержит классы, используемые при создании Windows-приложений. Класс **Form** из этого пространства задает форму - окно, заполняемое элементами управления, графикой, обеспечивающее интерактивное взаимодействие с пользователем.

Первая программа на C#

```
using System;          - подключение пространства имен
namespace first {      - новое пространство имен
    class Program    {   - класс Program
        static void Main(string[ ] args)      - главная функция
     {
            Console.WriteLine(“HELLO, WORLD!!!");
     }
    }
}
```



The .NET Framework Stack

- В состав библиотеки классов .NET Framework, входят классы, которые используются при разработке Windows-приложений, Web-приложении, а также приложении с базами данных.
- В библиотеке классов .NET Framework имеются также классы, обеспечивающие взаимодействие с языком XML, с моделью компонентных объектов Microsoft (COM) и с любой платформой, поддерживающей интерфейс 32-разрядных Windows-приложении (Win32 API).

ADO.NET Entity Framework (EF) — объектно-ориентированная технология доступа к данным

ASP.NET — технология создания веб-приложений и веб-сервисов от компании Майкрософт.

Windows Forms — название интерфейса программирования приложений (API), отвечающего за графический интерфейс пользователя.

CLR (Common Language Runtime)

- Наиболее революционным изобретением **Framework.Net** явилось создание **исполнительной среды CLR** (Common Language Runtime — единая система выполнения программ). С ее появлением процесс написания и выполнения приложений становится принципиально другим.

- Компиляторы языков программирования, включенные в **Visual Studio.Net**, создают модули на промежуточном языке **IL** (Microsoft Intermediate Language)

- Фактически компиляторы создают так называемый управляемый модуль - переносимый исполняемый файл (Portable Executable или **PE**-файл). Этот файл содержит код на **IL** и метаданные - всю необходимую информацию как для **CLR**, так и конечных пользователей, работающих с приложением

- В зависимости от выбранного типа проекта, **PE**-файл может иметь уточнения **exe**, **dll**, **mod** или **mdl**.

Виртуальная машина

Файл, имеющий уточнение **exe**, хотя и является exe-файлом, но это не совсем обычный, исполняемый файл. При его запуске он распознается как специальный PE-файл и передается **CLR** для обработки.

Исполнительная среда начинает работать с кодом, в котором специфика исходного языка программирования исчезла. Код на **IL** начинает выполняться под управлением **CLR** (по этой причине **код** называется **управляемым**).

Исполнительную среду можно рассматривать как своеобразную виртуальную **IL**-машину. Эта машина транслирует "на лету" требуемые для исполнения участки кода в команды реального процессора, который в действительности и выполняет код

Microsoft использовала получивший широкое признание опыт виртуальной машины **Java**, улучшив процесс за счет того, что, в отличие от **Java**, промежуточный код не интерпретируется исполнительной средой, а компилируется с учетом всех особенностей текущей платформы. Благодаря этому создаются высокопроизводительные приложения.

В состав **CLR** входят трансляторы **JIT** (Just In Time Compiler), которые и выполняют трансляцию **IL** в командный код той машины, где установлена и функционирует исполнительная среда **CLR**

- Если у вас есть готовый **PE**-файл, то иногда полезно анализировать его **IL**-код и связанные с ним метаданные. В состав **Framework SDK** входит дизассемблер - **ildasm**, выполняющий дизассемблирование **PE**-файла и показывающий метаданные, а также **IL**-код с комментариями в наглядной форме.
- Программисты, предпочитающие работать на низком уровне, могут программировать на языке ассемблера **IL**. В этом случае в их распоряжении будет вся мощь библиотеки **FCL** и все возможности **CLR**.

Выполнение программы на C#

Исходный код, написанный на языке C#, компилируется в промежуточный язык (IL). Код IL и ресурсы, такие как растровые изображения и строки, хранятся на диске в исполняемом файле, называемом **сборкой**, с расширением EXE или DLL в большинстве случаев. Сборка содержит **манифест** со сведениями о типах сборки, версии, языке и региональных параметрах и требованиях безопасности.

Каждая программная единица в среде .NET, (сборка), помимо кода на языке IL обязательно содержит метаданные, описывающие как её в целом (манифест), так и каждый тип, содержащийся в ней, в отдельности (метаданные). На рисунке показаны составные части однофайловой сборки.

Файл начинается с **манифеста** и включает в себя описание всех классов, хранимых в **PE**-файле, их свойств, методов, всех аргументов этих методов - всю необходимую **CLR** информацию. Поэтому помимо **PE**-файла не требуется никаких дополнительных файлов и записей в реестр - вся нужная информация извлекается из самого файла.

- **.NET-приложения** содержат в себе **метаданные**, т.е. описание кода и данных, используемых приложением. Благодаря использованию метаданных возможно автоматическое преобразование данных в последовательную форму общеязыковой средой выполнения **CLR** при их сохранении.

Метаданные — это данные в двоичном формате с описанием программы, хранящиеся либо в переносимом исполняемом (PE) файле среды ***CLR***, либо в памяти

Когда код компилируется в PE-файл, метаданные помещаются в одну часть файла, а код преобразуется в ***IL*** и помещается в другую часть файла

В метаданных описываются все типы и члены, определенные или используемые в модуле или сборке

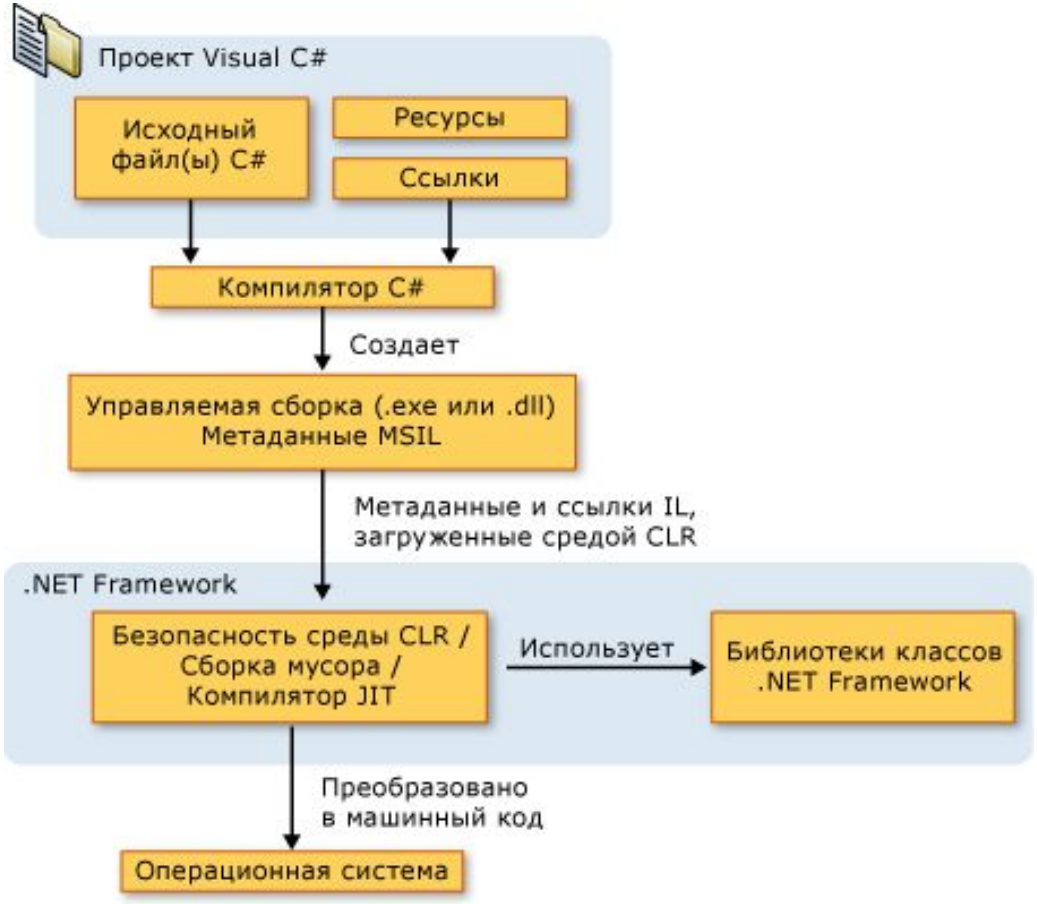
При исполнении кода среда выполнения загружает метаданные в память и обращается к ним для получения сведений о классах, членах, наследовании и других элементах кода

Среди классов библиотеки **FCL** имеется класс **Reflection**, методы которого позволяют извлекать необходимую информацию. При проектировании класса программист может создавать собственные атрибуты, добавляемые к метаданным **PE**-файла.

При выполнении программы на ***C#*** сборка загружается в среду ***CLR*** в зависимости от сведений в манифесте. Далее, если требования безопасности соблюдены, среда ***CLR*** выполняет JIT-компиляцию для преобразования кода IL в инструкции машинного кода.

Среда ***CLR*** также предоставляет другие службы, относящиеся к автоматическому сбору мусора, обработке исключений и управлению ресурсами. Код, выполняемый средой ***CLR***, иногда называют "управляемым кодом" в противопоставление "неуправляемому коду", который компилируется в машинный код, предназначенный для определенной системы.

Ниже показаны отношения во время компиляции и время выполнения между файлами с исходным кодом ***C#***, библиотеками классов .NET Framework, сборками и средой ***CLR***.



Сборщик мусора (Garbage Collector) и управление памятью

- Важной особенностью построения ***CLR*** является то, что исполнительная среда берет на себя часть функций, традиционно входящих в ведение разработчиков трансляторов, и облегчает тем самым их работу.

- Один из таких наиболее значимых компонентов ***CLR*** - сборщик мусора (**Garbage Collector**). Под сборкой мусора понимается освобождение памяти, занятой объектами, которые стали бесполезными и не используются в дальнейшей работе приложения.

- В ряде языков программирования (классическим примером является язык ***C/C++***) память освобождает сам программист, в явной форме отдавая команды как на создание, так и на удаление объекта. Неизбежные ошибки программиста при работе с памятью тяжелы по последствиям, и их крайне тяжело обнаружить.

- В ***CLR*** задача сборки мусора снята не только с программистов, но и с разработчиков трансляторов, она решается - исполнительной средой, ответственной за выполнение вычислений.

- Здесь же решаются и многие другие вопросы, связанные с использованием памяти, в частности, проверяются возможные нарушения связанные, например, с использованием указателей. Данные, удовлетворяющие требованиям ***CLR*** и допускающие сборку мусора, называются ***управляемыми данными***.

- Но, как же быть с языком ***C++*** и другими языками, где есть указатели, адресная арифметика, возможности удаления объектов программистом? Такие возможности сохранены и в языке ***C#***.

CLR позволяет работать как с управляемыми, так и с неуправляемыми данными. Однако использование неуправляемых данных регламентируется и не поощряется. Так, в ***C#*** модуль, использующий неуправляемые данные (указатели, адресную арифметику), должен быть помечен как небезопасный (***unsafe***), и эти данные должны быть четко зафиксированы.

Исключительные ситуации

- Что происходит, когда при вызове некоторой функции (процедуры) обнаруживается, что она не может нормальным образом выполнить свою работу?

- Возможны разные варианты обработки такой ситуации. Функция может возвращать код ошибки или специальное значение типа **HResult**, может **выбрасывать исключение**, тип которого характеризует возникшую ошибку.

- В **CLR** принято во всех таких ситуациях выбрасывать исключение. Косвенно это влияет и на язык программирования. Выбрасывание исключений наилучшим образом согласуется с исполнительной средой.

- В языке **C#** выбрасывание исключений, их дальнейший перехват и обработка - основной рекомендуемый способ обработки исключительных ситуаций.

События

- У ***CLR*** есть свое видение того, что представляет собой тип. Есть формальное описание общей системы типов ***CTS*** (Common Type System). В соответствии с этим описанием, каждый тип, помимо полей, методов и свойств, может содержать и события.

- При возникновении событий в процессе работы с тем или иным объектом данного типа посылаются сообщения, которые могут получать другие объекты.

- Механизм обмена сообщениями основан на ***делегатах*** - функциональном типе.

- В язык ***C#*** встроен механизм событий, полностью согласованный с возможностями ***CLR***.

Общие спецификации и совместимые модули

- Уже говорилось, что каркас **Framework.Net** облегчает межъязыковое взаимодействие. Для того чтобы классы, разработанные на разных языках, мирно уживались в рамках одного приложения они должны удовлетворять некоторым ограничениям.

- Эти ограничения задаются набором общеязыковых спецификаций - **CLS** (Common Language Specification). Класс, удовлетворяющий спецификациям **CLS**, называется **CLS**-совместимым. Он доступен для использования в других языках, классы которых могут быть клиентами или наследниками совместимого класса.

- Спецификации **CLS** точно определяют, каким набором встроенных типов можно пользоваться в совместимых модулях. Эти типы должны быть общедоступными для всех языков, использующих **Framework.Net**. В совместимых модулях должны использоваться управляемые данные и выполняться некоторые другие ограничения.

- Исполнительная среда **CLR** обладает мощными динамическими механизмами - сборки мусора, динамического связывания, обработки исключительных ситуаций и событий. Все эти механизмы и их реализация в **CLR** созданы на основании практики существующих языков программирования.

- Но и исполнительная среда влияет на языки, ориентированные на использование **CLR**.

- Поскольку язык **C#** создавался одновременно с созданием **CLR**, то, естественно, он стал языком, наиболее согласованным с исполнительной средой, и средства языка напрямую отображаются в средства исполнительной среды.