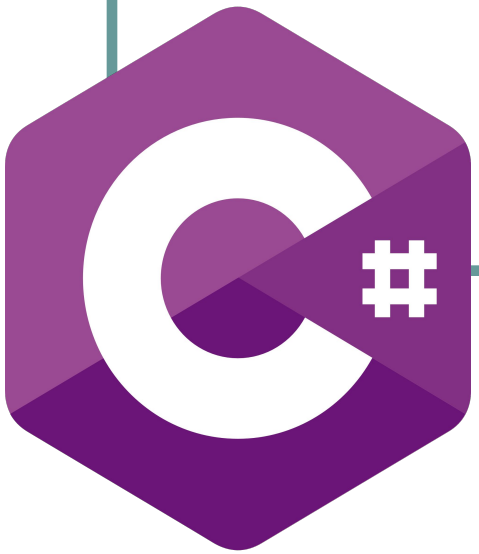
.Net Introduction



Программа курса “Основы .Net”

- Введение в платформу .Net – понятия CLR, IL, BCL, рефлектор
- Основы языка C# - переменные, условия, циклы, массивы, строки
- Работа с классами – конструкторы, this, частичные типы, ref и out
- Обработка исключений – throw, try, catch, finally, checked и unchecked
- Перегрузка операторов операций
- Индексаторы и свойства
- Наследование и полиморфизм, интерфейсы
- Структуры и перечисления, делегаты и события
- Обобщения (generics), сбор мусора
- Взаимодействие с файловой системой
- Сериализация и десериализация
- Работа с унаследованным кодом
- Экзамен (устное собеседование с заведующей кафедры)

Что будет дальше

- ***Windows Forms*** — технология создания клиентских приложений Windows;
- ***Windows Presentation Foundation (WPF)*** — технология создания клиентских приложений с гораздо большими возможностями и имеющими более насыщенный дизайн чем в Windows Forms;
- ***ADO.NET*** — технология, позволяющая работать с базами данных из приложений;
- **SP** — системное программирование, в ходе этой дисциплины изучается работа с потоками и все что с этим связано;
- **NP** — сетевое программирование — изучаются способы передачи информации по сети с использованием различных протоколов;
- ***Windows Communication Foundation (WCF)*** — сервис-ориентированная технология разработки приложений (создание и применение сервисов);
- ***ASP.NET*** — технология создания web-приложений.

План этой презентации

- Причины возникновения платформы Microsoft .NET
- Базовые понятия – CLR, CTS, CLS, CIL, CLI, BCL (FCL), сборка, манифест, метаданные
- Уровни архитектуры платформы .Net
- Преимущества и недостатки платформы .Net
- Список .Net-языков
- Схема исполнения приложения
- Понятие промежуточного языка
- Рефлекторы и обфускаторы, практика
- Тест-резюме
- Ссылки и полезные материалы

Немного истории. Язык С

В начале 1970-х Дэнис Ритчи разрабатывает язык С. Как и все популярные языки программирования, этот язык произошёл из кризиса программного обеспечения, реализовав новаторский подход своего времени — «**структурное программирование**».

Как правило, потребность в разработке новых языков заключается в необходимости новых средств масштабирования программных решений и самого программного кода.

До языка «С» программирование в основном было императивным, что вызывало сложности при увеличении размеров программных проектов. Однако, язык С, хоть и решал некоторые проблемы, связанные с масштабированием кода (привносил такие элементы как макроопределения, структуры и тд.), но всё ещё имел серьёзный недостаток — невозможность справляться с большими проектами.

Немного истории. Язык C++

Следующим этапом развития семейства стал язык C++, разработанный Бьярном Страуструпом в 1979-ом году, который реализовывал парадигму объектно-ориентированного подхода в программировании.

Язык C имел большой успех, в связи с тем, что в нём сочетались гибкость, мощность и удобство. Поэтому новый язык C++ стал развитием языка C. Можно сказать, что C++ — это объектно-ориентированный C, а **причиной его возникновения стала мода на ООП**.

Тесное родство C++ и C обеспечило популярность новому языку программирования, поскольку C-программисту не нужно было изучать новый язык программирования, достаточно было освоить новые объектно-ориентированные возможности и без того удачного языка.

Новые требования

Но время предъявило новые требования в области разработки ПО. Они заключались в том, что у конечного потребителя возникала необходимость в **межплатформенной переносимости** программного обеспечения, упрощения передачи проектов по сетям коммуникации, а также в уменьшении времени, которое затрачивается на разработку.

Для решения этой проблемы необходима была новая технология, которая смогла бы работать одинаково эффективно на всех платформах (Windows, Unix, Linux, Mac OS), и обеспечивала бы отсутствие конфликтов при переносе приложения с одной ОС на другую. Вместе с этим необходим был новый язык программирования, который, с одной стороны, был бы языком реализации данной технологии, а с другой — обеспечивал бы гибкость и скорость разработки проектов.

Появление Java

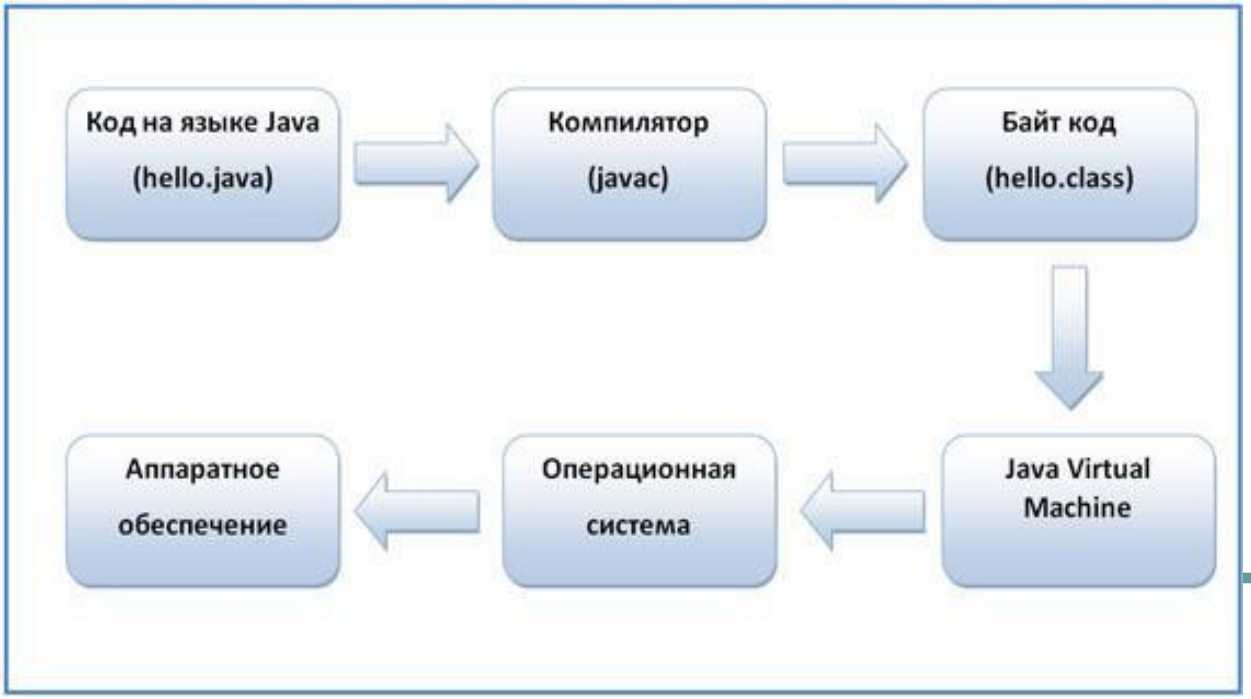
В 1991 году компания Sun Microsystems предложила решение этой проблемы на базе своего нового языка «Oak», который впоследствии стал называться «Java». Авторство этого языка приписывают Джеймсу Гослингу. На базе языка Java была создана среда исполнения «Java Runtime».

Межплатформенная переносимость обеспечивалась за счёт существования интегрированной среды исполнения Java, которая могла исполнять приложения, созданные на языке Java на любых платформах, на которых она установлена. Ограничение заключалось лишь в одном — существовании такой среды исполнения для всех существующих ОС. И тогда компания Sun Microsystems (которая сейчас уже принадлежит компании Oracle) взяла и разработала варианты такой среды исполнения практически для всех популярных операционных систем.

Создатели языка Java

Java-архитектура

- Программист создаёт файл с расширением JAVA и помещает в него исходный код программы
- С помощью компилятора на основании исходного кода создаётся файл с расширением CLASS, в него компилятор помещает байтовый код программы - команды для виртуальной машины
- При запуске программы виртуальной машиной загружается и выполняется файл с байт-кодом



Инструкции байт-кода

Инструкции можно разделить на несколько групп:

- загрузка и сохранение (например, ALOAD_0, ISTORE)
- арифметические и логические операции (например, IADD, FCMPL)
- преобразование типов (например, I2B, D2I)
- создание и преобразование объекта (например, NEW, PUTFIELD)
- управление стеком (например, DUP, POP)
- операторы перехода (например, GOTO, IFEQ)
- вызовы методов и возврат (например, INVOKESTATIC, IRETURN)

Microsoft strikes back ;)

Однако, язык Java решал далеко не все проблемы (например, проблему межъязыкового взаимодействия). Приложения, созданные на Java, исполняются достаточно медленно, что не позволяет использовать их на малопроизводительных платформах.

К тому же, Java на момент начала 2000-х (да и сейчас, в общем-то, тоже =)) не содержала современных языковых средств и эффективных механизмов, которые были так нужны программистам.

Поэтому, в 2002 году компания Microsoft явила свой мощный ответ на набравшую к тому времени популярность платформу Java. И сие решение получило название

.NET Framework.

Что такое .Net Framework?

.NET Framework — программная платформа, выпущенная компанией **Microsoft** в 2002 году. Основой платформы является общеязыковая среда исполнения **Common Language Runtime** (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

Список .Net языков

https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_.NET-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2

ПЕРЕЙТИ ПО ССЫЛКЕ!!!

Языки программирования .NET — компьютерные языки программирования, используемые для создания библиотек и программ, удовлетворяющих требованиям Common Language Infrastructure. За исключением некоторых серьёзных оговорок, большинство CLI-языков целиком компилируются в Common Intermediate Language (CIL), промежуточный язык, который может быть оттранслирован непосредственно в машинный код при помощи виртуальной машины Common Language Runtime (CLR), являющуюся частью Microsoft .NET Framework, Mono и Portable.NET.

Что такое .Net?

Платформа **Microsoft .NET** — это технология, которая поддерживает создание и выполнение приложений самых разных типов (консоль, рабочий стол, веб, мобильные, микросервисы, облако, ML, разработка игр, Internet of Things).

Главной идеей разработки .NET было стремление сделать кроссплатформенную виртуальную машину для выполнения одного и того же кода в различных ОС. И пускай далеко не сразу, но в один прекрасный день - 27 июня 2016 года - у Microsoft это-таки получилось https://ru.wikipedia.org/wiki/.NET

Ранние независимые проекты

.NET Framework (теперь уже как часть .NET) является патентованной технологией корпорации Microsoft и официально рассчитана на работу лишь под операционными системами семейства Microsoft Windows, но существуют и другие независимые проекты (Mono, .Net Core, Portable .Net, Xamarin.iOS, Xamarin.Android), позволяющие запускать программы .NET на других операционных системах.

Начиная с версии .NET 5 (20 ноября 2020 года), платформа стала называться просто **.NET** (уже без использования «Core» или “Framework” в названии), **что символизировало объединение .NET Core, Xamarin/Mono и .NET Framework.**

Ранние независимые проекты

What is .NET?

Languages: C#, Visual Basic, F#

Platforms:

- **.NET Core:** (runs anywhere!) Windows, Linux, and macOS
- **.NET Framework:** websites, services, and apps on Windows
- **Xamarin/Mono**: a .NET for mobile

https://docs.microsoft.com/ru-ru/dotnet/?WT.mc_id=dotnet-35129-website

Цели разработки платформы

Одной из целей разработки новой платформы было объединение всех наиболее удачных наработок в рамках единой платформы и их унификация. Кроме того, ставилась задача следования всем актуальным тенденциям в области программирования на тот момент. Так, например, новая платформа должна была напрямую поддерживать объектно-ориентированность, безопасность типов, сборку мусора и структурную обработку исключений.

Понятие платформы

Платформа .NET — это среда, которую видит код в процессе исполнения. Это означает, что .NET занимается исполнением кода: запускает его, даёт ему соответствующие права, выделяет память для хранения данных, помогает с освобождением памяти и ресурсов, которые больше не требуются, и тд. Помимо этого .NET предоставляет обширную библиотеку классов, так называемую библиотеку базовых классов .NET, для выполнения большого числа задач Windows. В этом плане .NET можно рассматривать с двух сторон: как управляющую выполнением кода и предоставляющую коду различные службы.

Преимущества

Платформа .NET основана на единой объектно-ориентированной модели; все сервисы, предоставляемые программисту платформой, оформлены в виде единой иерархии классов. Платформа предоставляет автоматическое управление ресурсами. Это решает многие распространенные проблемы, такие как утечки памяти, повторное освобождение ресурса и тп. Всё полностью на ООП, даже элементарные типы данных!

Преимущества

- Си-подобный синтаксис, на 80% те же ключевые слова, что и в C++
- Тонны «синтаксического сахара»
- Огромное количество готовых классов на все случаи жизни
- Интеграция с неуправляемыми языками (управляемый код, managed code - любой код, который разработан для исполнения в .NET. Код, который работает под управлением Windows, называется неуправляемым)

Преимущества

Код, сгенерированный для .NET, может быть проверен на безопасность. Это гарантирует, что приложение не может навредить пользователю или нарушить функционирование операционной системы (так называемая "модель песочницы"). Таким образом, приложения для .NET могут быть сертифицированы на безопасность.

Преимущества

Обработка ошибок в .NET всегда производится через механизм исключительных ситуаций. Это решает неоднозначность ситуации, когда некоторые ошибки обозначаются с помощью кодов ошибки платформы Win32, некоторые возвращают HRESULT и тп.

Главное преимущество!

Вероятно, самым большим достижением .NET остаётся **межъязыковое взаимодействие** (language interoperability). Впервые в истории программирования появляется единая модель, позволяющая на равных пользоваться различными языками для создания приложений. Так как MSIL не зависит от исходного языка программирования или от целевой платформы, в рамках .NET становится возможным развивать новые программы на базе старых программ.

Межъязыковое взаимодействие

Например, служба, написанная на C++ для Microsoft .NET, может обратиться к методу класса из библиотеки, написанной на Delphi.NET. На C# можно написать класс, наследованный от класса, написанного на Visual Basic .NET.

Недостатки .Net

- **Несущественное замедление выполнения программ**. Это и неудивительно, так как между исходным языком и машинным кодом вводится дополнительный уровень, MSIL (промежуточный код).
- **Дистрибутивы для работы приложений**. Для работы приложений .Net должны быть установлены специальные дистрибутивы. Каждую новую версию дистрибутивов .Net нужно устанавливать отдельно. Если зайти в менеджер приложений, можно удивиться сколько их установлено.
- **Больше кода - больше ответственности**. Каждая технология постоянно улучшается, зависимости в них меняются. Иногда возникает ситуация, где нужно использовать старую версию какой-нибудь библиотеки. Из-за зависимости версий приходиться изменять версии других библиотек и часто обнаруживается ситуация, что кое-что работает некорректно.

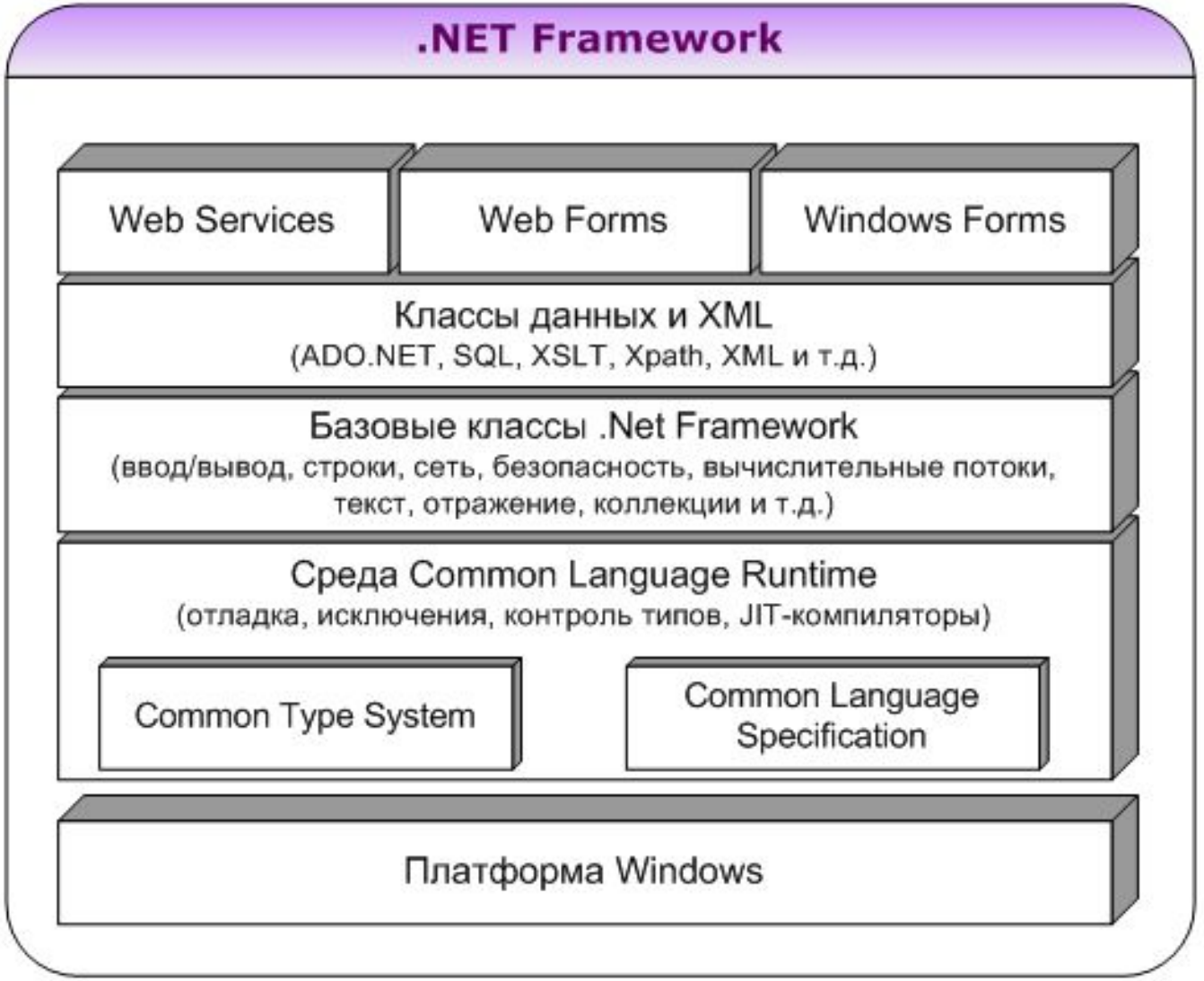
MSIL

MSIL - это язык, на котором должен быть написан код, загружаемый и запускаемый средой исполнения .NET. При компиляции управляемого кода компилятор генерирует код на промежуточном языке, а CLR выполняет заключительную стадию компиляции непосредственно перед исполнением кода. Язык IL разработан таким образом, чтобы обеспечить быструю компиляцию в машинный код, но в то же время он поддерживает все особенности .NET.

Есть ещё один недостаток

Кстати, ещё одна проблема .NET заключается в том, что при её создании основной упор был сделан на C-подобные языки (например, конструкторы с именем, равным имени класса). Это ограничивает возможности интеграции некоторых языков с более богатыми возможностями, особенно с принципиально отличающимися языками, такими как функциональные языки (ML, Haskell, Scheme) или устаревшие языки (Кобол, PL/I).

Архитектура .Net Framework



Архитектура платформы

Платформа .NET Framework является надстройкой над операционной системой, в качестве которой может выступать любая версия Windows. На сегодняшний день платформа .NET Framework включает в себя:

- пять официальных языков: C#, VisualBasic.NET, Managed C++, F# и JScript .NET
- объектно-ориентированную среду CLR (Common Language Runtime), совместно используемую этими языками для создания приложений разных типов
- ряд связанных между собой библиотек классов под общим именем FCL (Framework Class Library, aka BCL)

Архитектура платформы

Самым важным компонентом платформы .NET Framework является **CLR** (Common Language Runtime), предоставляющая среду, в которой выполняются программы. Главная её роль заключается в том, чтобы обнаруживать и загружать типы .NET и производить управление ими в соответствии с полученными командами. CLR включает в себя виртуальную машину, во многих отношениях аналогичную виртуальной машине Java. Среда исполнения активизирует объекты, производит проверку безопасности, размещает объекты в памяти, выполняет их, а также запускает сбор мусора.

Сбор мусора

Под сбором мусора понимается освобождение памяти, занятой объектами, которые стали бесполезными и не используются в дальнейшей работе приложения. В ряде языков программирования (например, C/C++) память освобождает сам программист, в явной форме отдавая команды как на создание, так и на удаление объекта. В этом есть своя логика — "я тебя породил, я тебя и убью". Однако в CLR задача сбора мусора (и другие вопросы, связанные с использованием памяти) решается в нужное время и в нужном месте исполнительной средой, ответственной за выполнение вычислений.

Framework Class Library

Над уровнем CLR находится набор базовых классов платформы, над ним расположены слой классов данных и XML, а также слой классов для создания Web-служб (Web Services), Web- и Windows-приложений (Web Forms и Windows Forms). Собранные воедино, эти классы известны под общим именем **FCL** (Framework Class Library). Это одна из самых больших библиотек классов в истории программирования. Она открывает доступ к системным функциям, включая и те, что прежде были доступны только через API Windows, а также к прикладным функциям для Web-разработки (ASP.NET), доступа к данным (ADO.NET), обеспечения безопасности и удаленного управления. Имея в своём составе **более 4000 классов**, библиотека FCL способствует быстрой разработке настольных, клиент-серверных и других приложений и Web-служб.

Архитектура платформы

Набор базовых классов платформы — нижний уровень FCL — не только прячет обычные низкоуровневые операции, такие как файловый ввод/вывод, обработка графики и взаимодействие с оборудованием компьютера, но и обеспечивает поддержку большого количества служб, используемых в современных приложениях (управление безопасностью, поддержка сетевой связи, управление вычислительными потоками, работа с рефлексией и коллекциями и тд.).

Классы для работы с SQL и XML

Над этим уровнем находится уровень классов, которые расширяют базовые классы с целью обеспечения управления данными и XML. Классы данных позволяют реализовать управление информацией, хранящейся в серверных базах данных. В число этих классов входят классы для работы с SQL и XML.

Архитектура платформы

Microsoft .NET поддерживает не только языковую независимость, но и языковую интеграцию. Это означает, что разработчик может наследоваться от классов, обрабатывать исключения и использовать преимущества полиморфизма при одновременной работе с несколькими языками. Платформа .NET Framework предоставляет такую возможность с помощью спецификации **CTS** (**Common Type System** — общая система типов), которая полностью описывает все типы данных, поддерживаемые средой выполнения, определяет, как одни типы данных могут взаимодействовать с другими и как они будут представлены в формате метаданных .NET.

Архитектура платформы

Важно понимать, что не во всех языках программирования .NET обязательно должны поддерживаться все типы данных, которые определены в CTS. Спецификация CLS (Common Language Specification — общая языковая спецификация) устанавливает основные правила, определяющие законы, которым должны следовать все языки: ключевые слова, типы, примитивные типы, перегрузки методов и тп. Спецификация CLS определяет минимальные требования, предъявляемые к языку платформы .NET. Компиляторы, удовлетворяющие этой спецификации, создают объекты, способные взаимодействовать друг с другом. Любой язык, соответствующий требованиям CLS, может использовать все возможности библиотеки FCL. CLS позволяет и разработчикам, и поставщикам, и производителям программного обеспечения не выходить за пределы общего набора правил для языков, компиляторов и типов данных.

CLR

Common Language Runtime (англ. CLR — общеязыковая исполняющая среда) — исполняющая среда для байт-кода CIL (MSIL), в который компилируются программы, написанные на .NET-совместимых языках программирования (C#, Managed C++, Visual Basic .NET, F# и прочие). CLR является одним из основных компонентов пакета Microsoft .NET Framework.

В отличие от переносимых виртуальных машин Java, абстрагирующихся от нижележащих операционных систем, CLR позиционируется как не "виртуализированная" платформа, тесно связанная с операционной системой Microsoft Windows (в том числе для целей сохранения инвестиций Microsoft в операционную систему).

CLR

CLR,— это то, что собственно управляет исполнением нашего кода. Можно рассматривать её как системный код, который обеспечивает загрузку нашей программы, её выполнение и предоставление всех необходимых служб. CLR физически имеет вид библиотеки ***mscoree.dll*** (Common Object Runtime Execution Engine).

CTS

Common Type System (стандартная система типов) — часть .NET Framework, формальная спецификация, определяющая, как какой-либо тип (класс, интерфейс, структура, встроенный тип данных) должен быть определён для его правильного выполнения средой .NET. Кроме того, данный стандарт определяет, как типы и специальные значения типов представлены в компьютерной памяти. Целью разработки CTS было обеспечение возможности программам, написанным на различных языках программирования, легко обмениваться информацией. Как это принято в языках программирования, тип может быть описан как определение набора допустимых значений (например, «все целые от 0 до 10») и допустимых операций над этими значениями (например, сложение и вычитание).

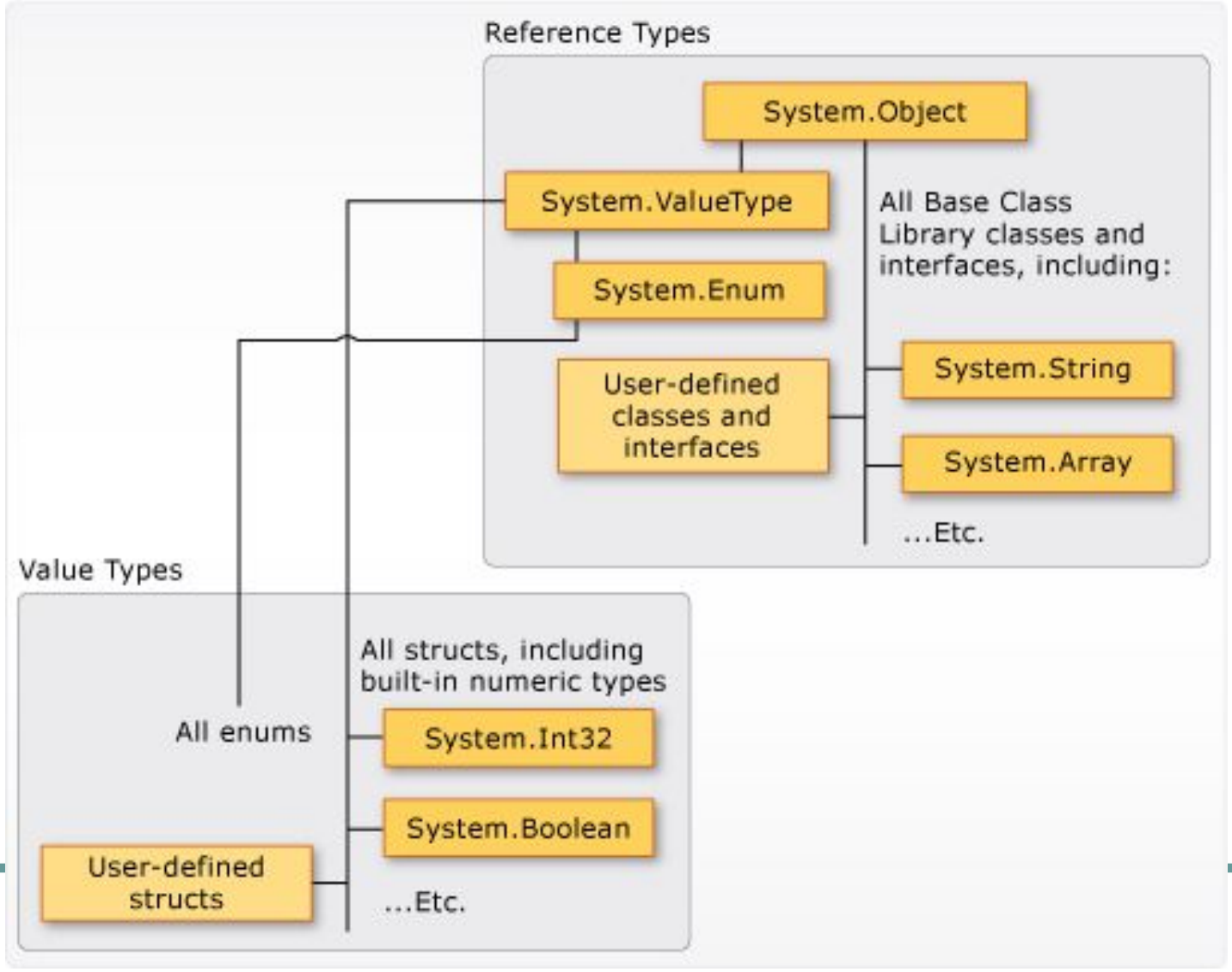
Функции CTS

- Формирует фреймворк, способствующий межъязыковой интеграции, безопасности типов, а также высокой производительности исполнения кода
- Обеспечивает объектно-ориентированную модель, поддерживающую полную реализацию множества языков программирования
- Определяет правила, которым должны следовать языки, что в том числе позволяет гарантировать, что объекты, написанные на разных языках могут друг с другом взаимодействовать
- CTS определяет правила, управляющие наследованием типов, виртуальными методами и продолжительностью существования объектов

Типы, поддерживаемые CTS

- Переменные **значимых типов** непосредственно содержат данные, а экземпляры значимых типов располагаются или в стеке или непосредственно в теле других объектов. Значимые типы могут быть встроенными (реализуются средой исполнения), пользовательскими или перечислениями. Значение значимого типа может быть преобразовано в значение ссылочного типа путем применения к нему процедуры упаковки (боксинга).
- Переменные **ссылочных типов** хранят лишь ссылку на адрес в памяти, по которому хранится значение (объект). Экземпляры ссылочных типов обычно располагаются в куче.

Два вида типов



CLS

Платформа .NET Framework является независимой от языка. Это означает, что код можно разрабатывать на одном из многих языков, ориентированных на .NET Framework, например C#, C++/CLI, Eiffel, F#, IronPython, IronRuby, PowerBuilder, Visual Basic, Visual COBOL и тд. Чтобы использовать типы и члены библиотек классов, разработанных для платформы .NET Framework, не требуется знать их исходный язык. Если вы разрабатываете компоненты, они будут доступны всем приложениям .NET вне зависимости от используемого языка.

Чтобы обеспечить полное взаимодействие между объектами вне зависимости от их языка, объекты должны предоставлять вызывающим объектам функции, общие для всех языков. Этот общий набор компонентов определяется общеязыковой спецификацией (**Common Language Specification**) — рядом правил, который применяется к создаваемым сборкам.

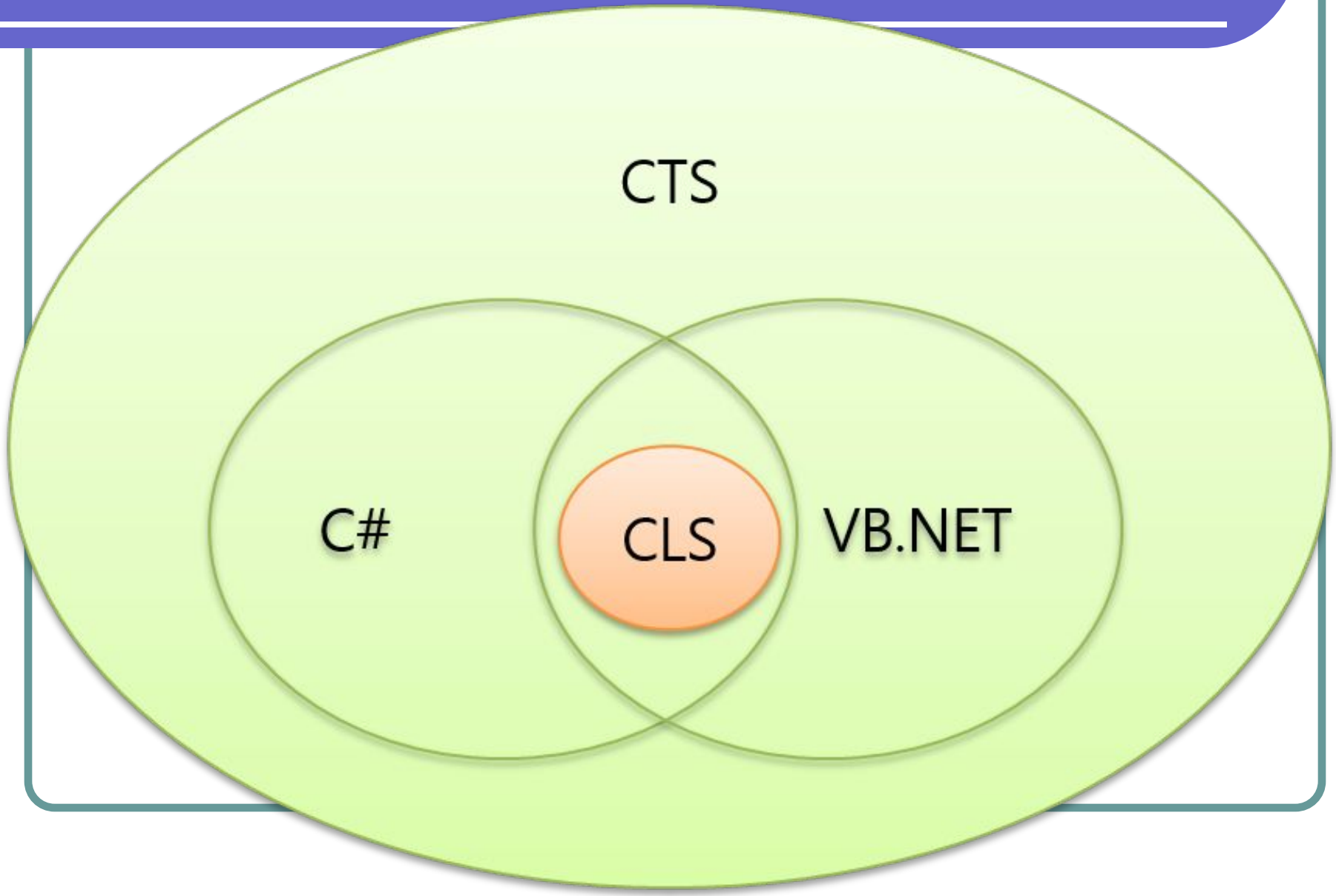
CLS

Не все языки .NET могут работать с некоторыми встроенными типами данных CTS. Поэтому очень важно было определить такой набор типов (и программных конструкций), с которым гарантированно смогут работать любые языки программирования в среде .NET. Этот набор и называется CLS.

CLS

CLS – это минимальный набор стандартов, который гарантирует, что доступ к коду может быть осуществлён из любого языка. Все компиляторы, предназначенные для .NET, должны поддерживать CLS. CLS формирует подмножество функций, доступных в .NET и IL, и полезна для кода, использующего особенности вне компетенции CLS. Если в сборке присутствуют какие-либо не CLS особенности, они могут быть недоступны в некоторых языках.

CLS – подмножество CTS!



BCL (FCL)

Base Class Library, сокращённо BCL — стандартная библиотека классов платформы .NET. Программы, написанные на любом из языков, поддерживающих платформу .NET, могут пользоваться классами и методами BCL — создавать объекты классов, вызывать их методы, наследовать необходимые классы BCL и тд.

Не все языки, поддерживающие платформу .NET, предоставляют или обязаны предоставлять одинаково полный доступ ко всем классам и всем возможностям BCL — это зависит от особенностей реализации конкретного компилятора и языка.

BCL

В отличие от многих других библиотек классов, вроде MFC, ATL/WTL или SmartWin, библиотека BCL не является некой «надстройкой» над функциями операционной системы или над каким-либо API – она является органической частью самой платформы .NET Framework, её «родным» API. Её можно рассматривать как API виртуальной машины .NET.

BCL обновляется с каждой версией .NET Framework.

BCL

Вероятно, одним из самых больших достоинств управляемого кода, помимо упрощения процесса написания кода, является возможность использования библиотеки базовых классов .NET.

Базовые классы .NET представляют собой большую коллекцию классов управляемого кода. Они были созданы Microsoft и позволяют выполнять практически любые задачи, которые ранее были доступны благодаря Windows API.

BCL

Замечательной особенностью базовых классов .NET является то, что они просты в использовании и самодокументированы. Например, для запуска потока необходимо вызвать метод Start() класса Thread. Для открытия файла нужно вызвать метод Open() класса File. Для того чтобы сделать неактивным TextBox, необходимо присвоить значение false свойству Enabled объекта TextBox. Идея самодокументированных классов знакома разработчикам Visual Basic и Java, чьи библиотеки так же просты в применении.

BCL

Возможно, это будет большим облегчением для программистов на C++, которые вынуждены иметь дело с такими функциями API, как GetDIBits (), RegisterWndClassEx () и IsEqualIID (), а также с целой плеядой функций для обработки дескрипторов Windows. С другой стороны, разработчики на C++ всегда могут получить доступ к Windows API, в то время как разработчики на Visual Basic и Java ограничены в применении функциональности Windows на низком уровне. Новым в базовых классах .NET является то, что они сочетают в себе лёгкость использования библиотек Visual Basic и Java с полным доступом ко всем функциям API.

Сферы BCL

- Поддержка Windows GUI, элементов управления и тп.
- Формы Web (ASP.NET)
- Доступ к данным (ADO.NET)
- Доступ к каталогам
- Доступ к файловой системе и реестру
- Работа с сетью и просмотр Web
- Атрибуты .NET и рефлексия
- Доступ к некоторым объектам операционной системы Windows, переменным окружения и тп.
- Доступ к исходному коду и компиляторам различных языков
- Совместимость с COM https://ru.wikipedia.org/wiki/Component_Object_Model
- Графика (GDI+)

Mscorelib.dll

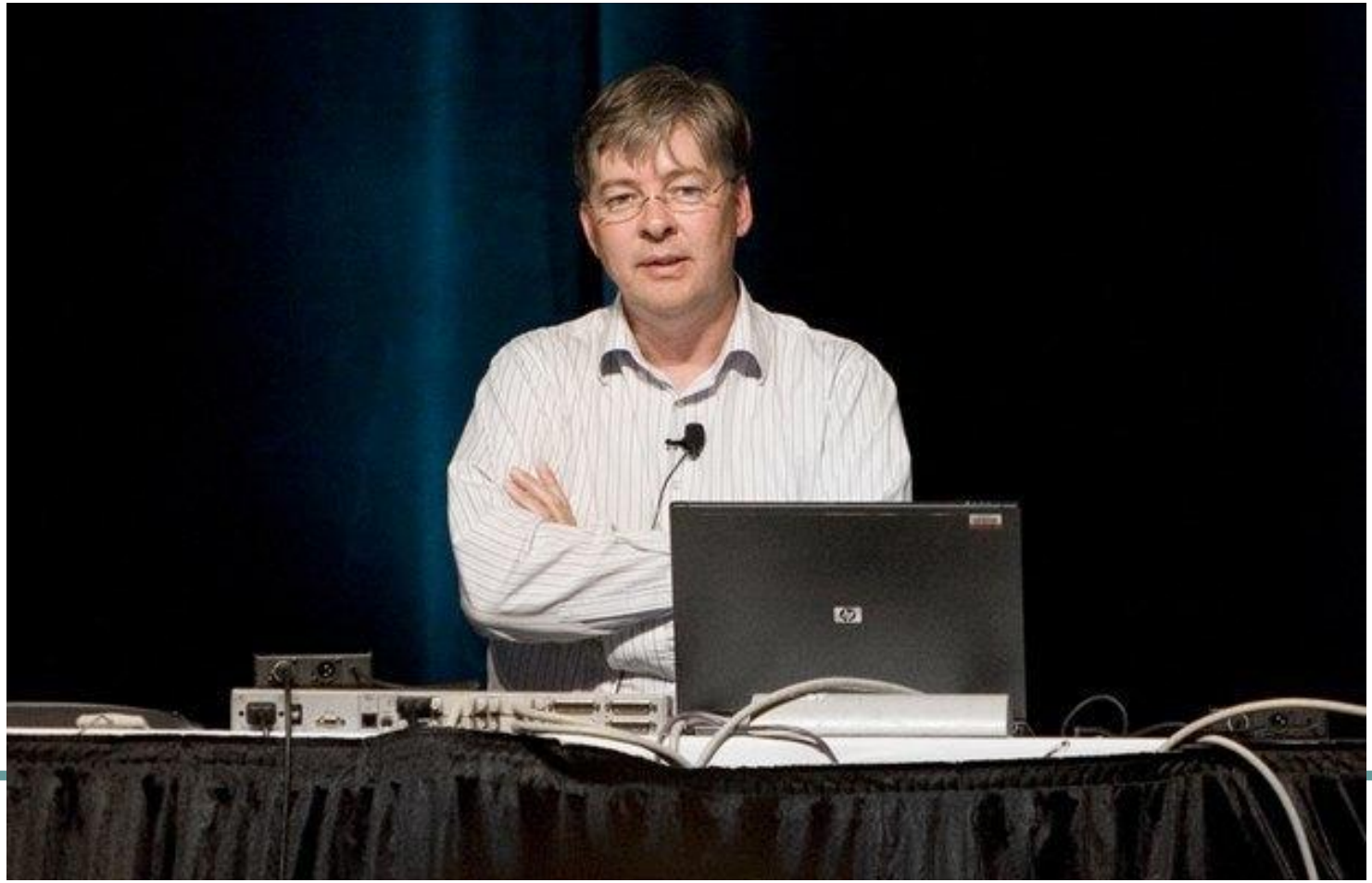
Вся библиотека базовых классов поделена на ряд отдельных сборок, главной среди которых является сборка *mscorelib.dll.* В этой сборке содержится большое количество базовых типов, охватывающих широкий спектр типичных задач программирования.

Возможности языка C#

- автоматическое управление памятью
- расширенные возможности обработки исключительных ситуаций
- богатый набор средств фильтрации ввода-вывода
- набор стандартных коллекций: arraylist, lists, sets, maps и т.д.
- наличие простых средств создания сетевых приложений
- наличие классов, позволяющих выполнять HTTP-запросы и обрабатывать ответы
- встроенные в язык средства создания многопоточных приложений
- унифицированный доступ к базам данных
- поддержка обобщений и многое другое

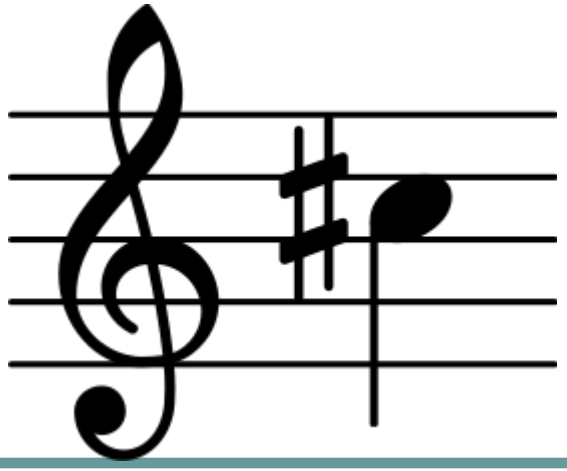
Создатель языка C#

Андерс Хейлсберг, датский инженер-программист, также создатель Turbo Pascal, Delphi и TypeScript.



Название языка

Название «Си шарп» (от англ. sharp — диез) происходит от музыкальной нотации, где знак диез означает повышение соответствующего ноте звука на полутон, что аналогично названию языка C++, где «++» обозначает инкремент переменной. Название также является игрой с цепочкой C → C++ → C++++(C#), так как символ «#» можно составить из 4 знаков «+».



Версии .NET и языка C#

Версия	Дата	.NET	Visual Studio
C# 1.0	Январь 2002	.NET Framework 1.0	Visual Studio .NET (2002)
C# 1.1 C# 1.2	Апрель 2003	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	Ноябрь 2005	.NET Framework 2.0 .NET Framework 3.0	Visual Studio 2005
C# 3.0	Ноябрь 2007	.NET Framework 2.0 (кроме LINQ) .NET Framework 3.0 (кроме LINQ) .NET Framework 3.5	Visual Studio 2008
C# 4.0	Апрель 2010	.NET Framework 4.0	Visual Studio 2010
C# 5.0	Август 2012	.NET Framework 4.5	Visual Studio 2012
C# 6.0	Июль 2015	.NET Framework 4.6 .NET Core 1.0 .NET Core 1.1	Visual Studio 2015
C# 7.0	Март 2017	.NET Framework 4.6.2 .NET Framework 4.7	Visual Studio 2017 15.0
C# 7.1	Август 2017	.NET Core 2.0	Visual Studio 2017 15.3
C# 7.2	Ноябрь 2017		Visual Studio 2017 15.5
C# 7.3	Май 2018	.NET Core 2.1 .NET Core 2.2 .NET Framework 4.8	Visual Studio 2017 15.7
C# 8.0	Сентябрь 2019	.NET Core 3.0 .NET Core 3.1 .NET Framework 4.8	Visual Studio 2019 16.3
C# 9.0	Сентябрь 2020	.NET 5.0	Visual Studio 2019 16.8
C# 10.0	Июль 2021	.NET 6.0	Visual Studio 2022 17.0

Новые возможности версий

Общая информация по версиям

C# 2.0	C# 3.0	C# 4.0	C# 5.0	C# 6.0
• Обобщения • Смешанные типы • Анонимные методы • Итераторы • Нуль-типы	• Неявно типизируемые локальные переменные • Инициализаторы объектов и коллекций • Автоматическая реализация свойств • Анонимные типы • Методы расширения • Запросы • Лямбда-выражения • Деревья выражений	• Динамическое связывание • Именованные и дополнительные аргументы • Обобщенная ковариантность и контрвариантность	• Асинхронные методы • Сведения о вызывающем объекте	• Компилятор как сервис • Импорт членов статических типов в пространство имён • Фильтры исключений • Await в блоках catch/finally • Инициализаторы автосвойств • Значения по умолчанию для get-свойств • Операторы с условием NULL • Интерполяция строк • Оператор nameof • Инициализатор словаря

Типы данных языка C#

Тип данных C#	Тип данных .NET	Количество байт	Описание
bool	System.Boolean	1	Логический тип
sbyte	System.SByte	1	Однобайтовое знаковое целое (от -128 до 127)
byte	System.Byte	1	Беззнаковое однобайтовое целое (от 0 до 255)
short	System.Int16	2	Короткое целое со знаком
ushort	System.UInt16	2	Короткое беззнаковое целое (от 0 до 65 535)
int	System.Int32	4	Целое
uint	System.UInt32	4	Беззнаковое целое
long	System.Int64	8	Длинное целое со знаком
ulong	System.UInt64	8	Беззнаковое длинное целое
char	System.Char	2	Символьный тип
float	System.Single	4	Вещественный тип данных с одинарной точностью с плавающей точкой
double	System.Double	8	Вещественный тип данных с двойной точностью с плавающей точкой
decimal	System.Decimal	16	Вещественный тип данных с фиксированной точкой
object	System.Object		Базовый тип для всех остальных
string	System.String		Тип данных для представления Юникодовой строки

CLI

CLI (**Common Language Infrastructure**) — спецификация общеязыковой инфраструктуры. Наиболее известными реализациями этого стандарта являются Microsoft .NET Framework, Mono, DotGNU Portable.NET. Спецификация CLI определяет, в частности, архитектуру исполнительной системы .NET — CLR и сервисы, предоставляемые CLR выполняемым программам, классы, предоставляемые библиотекой BCL, синтаксис и мнемонику общего промежуточного языка (IL).

Схема исполнения приложения

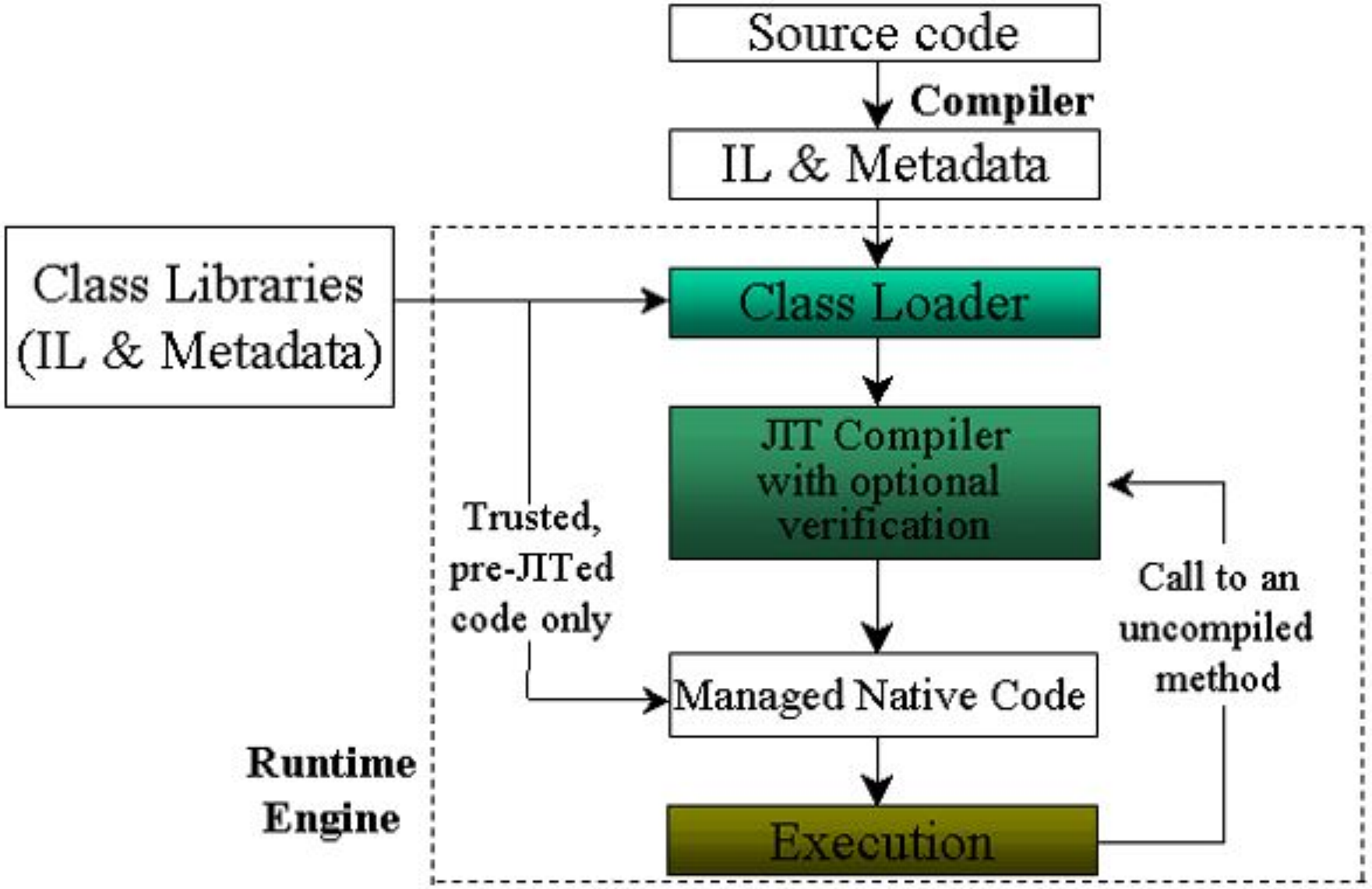


Схема исполнения приложения

Во время выполнения программы среда исполнения .NET загружает первую сборку, ту, что содержит точку входа основной программы. Среда использует хэш для проверки целостности сборки и метаданные для того, чтобы просмотреть определенные типы и убедиться, что среда сможет выполнить сборку. Правильно разработанные коммерческие приложения должны явно указывать, какие привилегии .NET им могут потребоваться (например, понадобится ли приложению доступ к файловой системе, реестру и тд.). В этом случае CLR обратится к политике безопасности системы и к учётной записи, под которой выполняется программа, и проверит, может ли она предоставить необходимые привилегии. Если код не запрашивает права явно, они будут предоставлены ему по первому требованию.

Схема исполнения приложения

CLR создает процесс, в котором будет исполняться код, и отмечает область приложения, в которой размещается главный поток приложения. В некоторых случаях программа может потребовать поместить её в уже имеющийся процесс запущенного ранее кода, тогда CLR создаст для неё только новую область приложения.

CLR берет первую часть кода, которая требуется для исполнения, и компилирует её с промежуточного языка на язык ассемблера, после чего выполняет её из соответствующего потока программы. Каждый раз, когда в процессе исполнения встречается новый метод, не исполнявшийся ранее, он компилируется в исполняемый код. Процесс компиляции происходит только один раз. Как только метод откомпилирован, его адрес заменяется адресом компилированного кода. Таким образом, производительность не ухудшается, так как компилируются только те участки кода, которые действительно используются. Этот процесс носит название компиляции just-in-time. JIT-компилятор может в зависимости от параметров компиляции, указанных в сборке, оптимизировать код в процессе компиляции, например, путём подстановки некоторых методов (inline) вместо их вызовов.

JIT

Компиляция **Just-In-Time** (JIT) - это процесс выполнения заключительной стадии компиляции с промежуточного языка в машинный код. Название определяется тем, что части кода компилируются по мере необходимости.

Схема исполнения приложения

Во время выполнения кода CLR следит за использованием памяти. На основе этих наблюдений она в определенные моменты будет останавливать программу на короткий промежуток времени (речь о миллисекундах) и запускать сбор мусора, который проверит переменные программы и выяснит, какие из областей памяти активно используются программой, для того, чтобы освободить неиспользуемые участки.

CIL

Common Intermediate Language (сокращённо CIL) — «высокоуровневый ассемблер» виртуальной машины .NET. Промежуточный язык, разработанный фирмой Microsoft для платформы .NET Framework. JIT-компилятор CIL является частью CLR (Common Language Runtime) — общей среды выполнения программ, написанных на языках .NET. Ранее язык назывался «Microsoft Intermediate Language».

CIL

Все компиляторы, поддерживающие платформу .NET, должны транслировать код с языков высокого уровня платформы .NET на язык CIL. В частности, код на языке CIL генерируют все компиляторы .NET фирмы Microsoft, входящие в среду разработки Microsoft Visual Studio (C#, Managed C++, Visual Basic .NET, Visual J# .NET).

CIL

По синтаксису и мнемонике язык CIL напоминает язык ассемблера. Его можно рассматривать как ассемблер виртуальной машины .NET. В то же время язык CIL содержит некоторые достаточно высокоуровневые конструкции, повышающие его уровень по сравнению с ассемблером для любой реально существующей машины, и писать код непосредственно на CIL легче, чем на ассемблере для реальных машин. Поэтому CIL можно рассматривать как своеобразный «высокоуровневый ассемблер».

Hello, world! на CIL

```
.assembly Hello {}
.method public static void Main() cil managed
{
    .entrypoint
    .maxstack 1
    ldstr "Hello, world!"
    call void
[mscorlib]System.Console::WriteLine(string)
    ret
}
```

Код C#

```
static void Main ( string [] args )
{
   for ( int i = 2; i < 1000; i++ )
   {
      for ( int j = 2; j < i; j++ )
      {
         if ( i % j == 0 )
            goto outer;
      }
      Console.WriteLine( i );
   }
outer:
}
```

А теперь то же самое на CIL ☺

```
.method private hidebysig static void  Main( string [] args ) cil managed
{
 .entrypoint
 .maxstack  2
 .locals init ( [0] int32 i,
         [1] int32 j )
 IL_0000:  ldc.i4.2
         stloc.0
         br.s       IL_001f
 IL_0004:  ldc.i4.2
         stloc.1
         br.s       IL_0011
 IL_0008:  ldloc.0
         ldloc.1
         rem
         brfalse.s  IL_0000
         ldloc.1
         ldc.i4.1
         add
         stloc.1
 IL_0011:  ldloc.1
         ldloc.0
         blt.s      IL_0008
         ldloc.0
         call       void [mscorlib]System.Console::WriteLine(int32)
         ldloc.0
         ldc.i4.1
         add
         stloc.0
 IL_001f:  ldloc.0
         ldc.i4     0x3e8
         blt.s      IL_0004
         ret
}
```

CIL

Промежуточный язык и байт-код Java в своей основе имеют одну и ту же идею: это языки низкого уровня с простым синтаксисом (основанным на числовых кодах, а не на тексте), который может быть быстро оттранслирован в родной машинный код. Целью байт-кода Java является обеспечение платформенной независимости.

Целью промежуточного языка является не просто платформенная независимость, а ещё и языковая независимость в объектно-ориентированной среде. Идея заключается в том, что должна существовать возможность компиляции кода с любого языка, и компилированный код должен быть совместим с кодом, откомпилированным с других языков. Эта совместимость достигается в .NET, так как с её помощью можно писать код, который компилируется в промежуточный язык, на C++, VB.NET или C#. Из-за требований языковой независимости и совместимости промежуточный язык гораздо сложнее байт-кода Java.

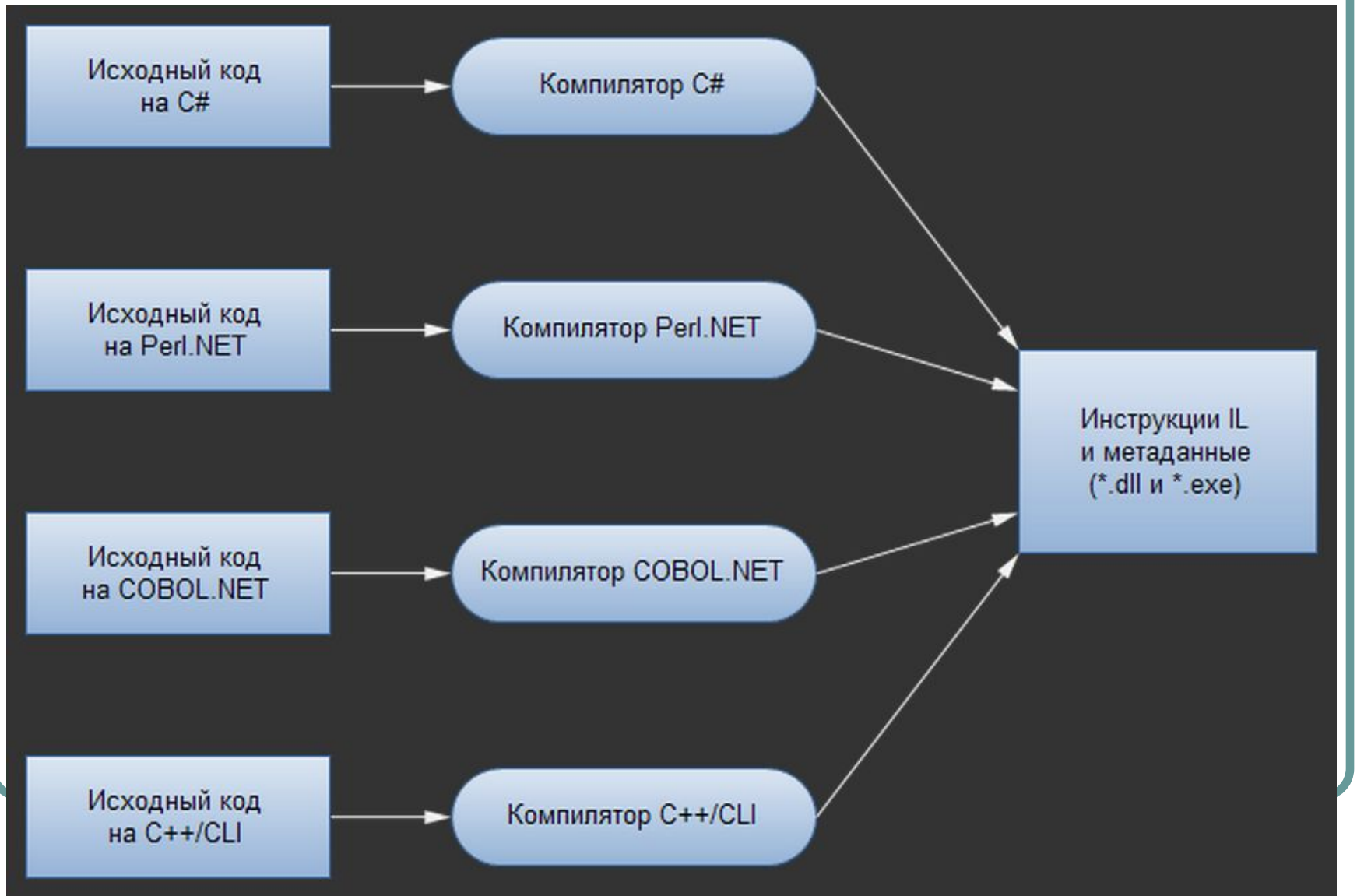
Виртуальный процессор

MSIL-код содержит инструкции, независимые ни от языка программирования, ни от ОС, ни от процессора. Важно понимать, что **программа на языке MSIL – это по-прежнему ООП-код.** Среда CLR исполняет **управляемый ООП-код, оперируя объектами, и** в этом заключается идея виртуального процессора. В то время как среда исполнения неуправляемого кода выполняет низкоуровневые инструкции **процедурного кода.** Это главное отличие среды исполнения CLR от среды исполнения неуправляемого кода!

Понятие сборки

Сборка — это логическая единица, которая содержит компилированный код, предназначенный для .NET. С этой точки зрения сборка аналогична DLL и исполняемому файлу. Сборка полностью описывает себя и является логической, а не физической единицей, так как может располагаться более чем в одном файле. Если сборка хранится в нескольких файлах, то существует один главный файл, содержащий точку входа и информацию о других файлах сборки.

Понятие сборки



Метаданные

Важной характеристикой сборок является то, что они содержат **метаданные**, описывающие типы и методы, определенные в соответствующем коде. Сборка содержит также метаданные, которые описывают саму сборку. Метаданные сборки, хранящиеся в области, известной как манифест, позволяют проверить версию сборки, её целостность и ряд других сведений.

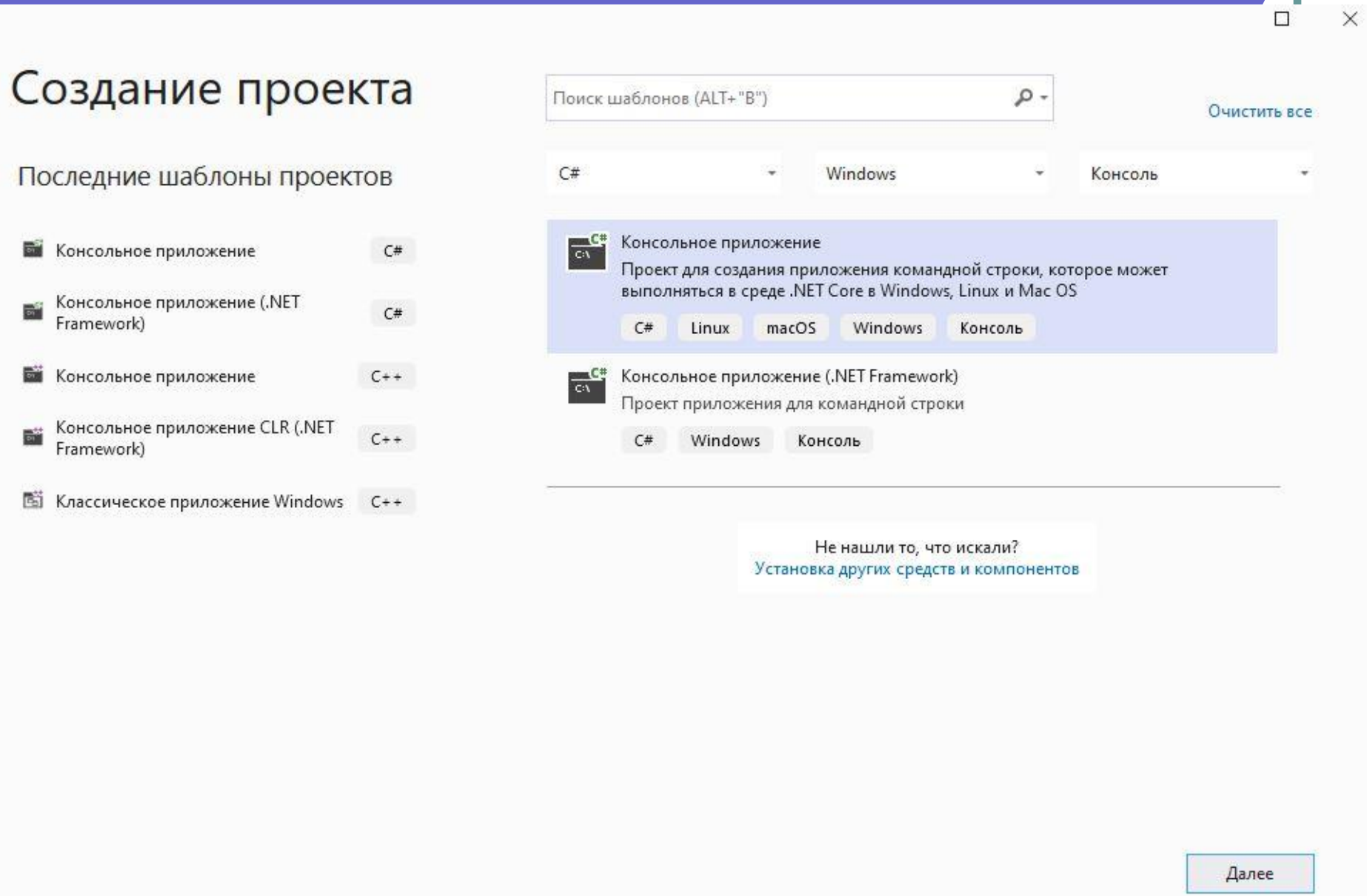
Метаданные

Сборка содержит метаданные о программе, а это означает, что программы и сборки, которые вызывают код этой сборки, не должны обращаться к реестру или иному источнику данных для выяснения того, как использовать сборку. При применении сборок отсутствует риск нарушения синхронизации, так как все метаданные хранятся вместе с исполняемыми инструкциями программы. Хотя сборки могут храниться в нескольких файлах, это всё равно не создает проблемы рассинхронизации данных. Дело в том, что файл, содержащий точку входа основной программы, имеет также информацию и хэш, сформированный на основе содержимого других файлов. Если один из файлов будет удалён, заменён или каким-либо образом модифицирован, это будет сразу же обнаружено, и сборка не станет загружаться.

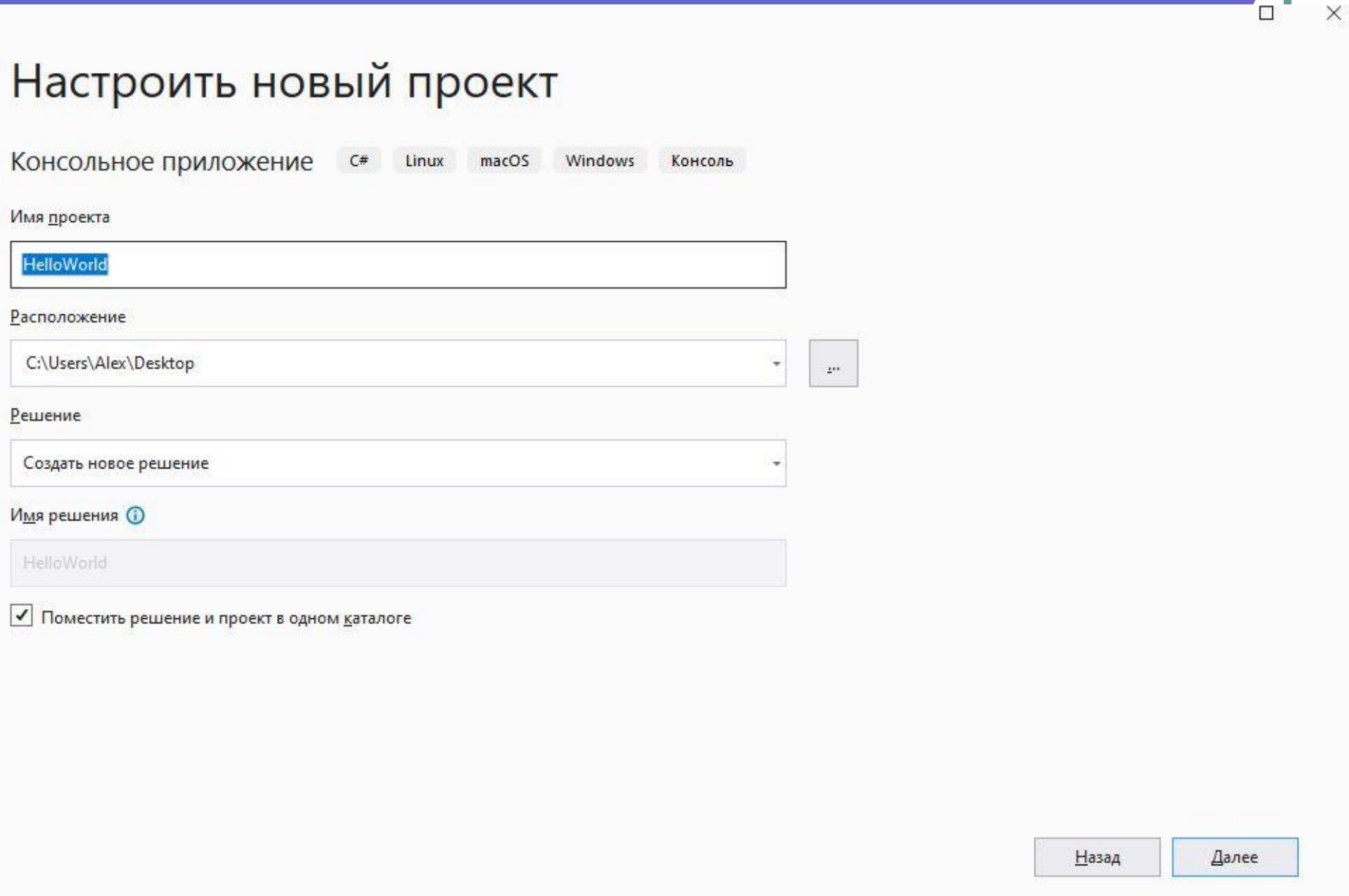
Манифест

Манифест – это сведения о самой сборке (номер версии сборки, языковые настройки, список других внешних сборок, необходимых для нормальной работы программы и тд).

Создание проекта



Создание проекта



Вот такой теперь шаблон :)

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

Что там в коде на самом деле

```
using System;

namespace ConsoleApplication1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

Рефлексия

То, что сборки полностью себя описывают, открывает теоретическую возможность программного доступа к метаданным сборки. В .NET существует несколько базовых классов, которые разработаны с этой целью. Рефлексия – это процесс, во время которого программа может отслеживать и модифицировать собственную структуру и поведение во время выполнения. Парадигма программирования, положенная в основу отражения, называется рефлексивным программированием. Это один из видов метапрограммирования.

Рефлексия

В большинстве современных компьютерных архитектур программные инструкции (код) хранятся как данные. Разница между кодом и данными в том, что выполняя код, компьютеры обрабатывают данные. То есть инструкции выполняются, а данные обрабатываются так, как предписано этими инструкциями. Однако программы, написанные с помощью некоторых языков, способны обрабатывать собственные инструкции как данные и выполнять, таким образом, рефлексивные модификации. Такие самомодифицирующиеся программы в основном создаются с помощью высокоуровневых языков программирования, использующих виртуальные машины.

Рефлексия

Рефлексия может использоваться для наблюдения и изменения программы во время выполнения. Рефлексивный компонент программы может наблюдать за выполнением определённого участка кода и изменять себя для достижения желаемой цели. Модификация выполняется во время выполнения программы путём динамического изменения кода.

Рефлексия

Программы, написанные на языках программирования, поддерживающих рефлексию, наделены дополнительными возможностями, реализация которых на языках низкого уровня затруднительна. Вот некоторые из них:

- поиск и модификация конструкций исходного кода (блоков, классов, методов, протоколов и тп.) как объектов класса во время выполнения
- изменение имён классов и функций во время выполнения
- анализ и выполнение строк кода, поступающих извне
- создание интерпретатора байт-кода нового языка

Практическое применение

- Загрузчики классов виртуальных машин
- Чтение комментариев (метаданных) на этапе выполнения (атрибуты, аннотации)
- Получение подробной информации о типе
- Сериализация и десериализация

.Net Reflector

.NET Reflector — платная утилита для Microsoft .NET, комбинирующая браузер классов, статический анализатор и декомпилятор. MSDN Magazine назвал её одной из десяти «Must-Have» утилит для разработчиков.

Программа используется для навигации, поиска и анализа содержимого .NET-компонентов, а также сборок и переводить двоичные данные в форму, пригодную для чтения человеком. Reflector позволяет производить декомпиляцию .NET-сборок на языки C#, Visual Basic .NET и MSIL. Reflector также включает дерево вызовов, которое может использоваться для навигации вглубь IL-методов с целью определения, какие методы они вызывают. Программа отображает метаданные, ресурсы и XML-документацию. .NET Reflector может быть использован .NET-разработчиками для понимания внутренней работы библиотек кода, для наглядного отображения различий между двумя версиями сборки, и того, как различные части .NET-приложения взаимодействуют друг с другом.

Рефлекторы

.NET Reflector может использоваться для нахождения мест, имеющих проблемы с производительностью и поиска багов. Он также может быть использован для поиска зависимостей сборки. Программа может быть использована для эффективной конвертации кода между C# и VB.NET.

Избавление .NET программы от регистрации на примере:

http://habrahabr.ru/post/111330/

Взлом программ для чайников:

http://habrahabr.ru/post/109117/

Скачать рефлекторы:

- http://www.red-gate.com/products/dotnet-development/reflector/
- http://www.telerik.com/products/decompiler.aspx
- http://confluence.jetbrains.com/display/NETPEEK/dotPeek+Early+Access+Program
- http://ilspy.net/

dotPeek decompiler

Free .Net decompiler:

https://www.jetbrains.com/decompiler/

Практика

- Установка .Net Reflector
- Изучение исходного кода игры Smile

ILdasm

Дизассемблер IL — сопутствующее средство Ассемблера IL (Ilasm.exe). Ildasm.exe принимает входной исполняемый файл, содержащий код на промежуточном языке (IL), и создает соответствующий текстовый файл в качестве входных данных для Ilasm.exe. Это средство автоматически устанавливается с Visual Studio.

- https://msdn.microsoft.com/en-us/library/aa309387(v=vs.71).aspx
- http://www.codespread.com/how-to-view-intermediate-code-in-c-using-ildasm-exe.html#.Vfe8MtLtmkq
- https://msdn.microsoft.com/ru-ru/library/f7dy01k1(v=vs.110).aspx

Скорее всего, он здесь:

C:\Program Files\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools

Обфускация

Обфускация (от лат. obfuscare — затенять, затемнять; и англ. obfuscate — делать неочевидным, запутанным, сбивать с толку) или запутывание кода — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

Обфускаторы

Запутывание кода может осуществляться на уровне алгоритма, исходного текста и/или ассемблерного текста. Для создания запутанного ассемблерного текста могут использоваться специализированные компиляторы, использующие неочевидные или недокументированные возможности среды исполнения программы. Существуют также специальные программы, производящие обфускацию, называемые обфускаторами (англ. obfuscator).

Цели обфускации

- Затруднение декомпиляции/отладки и изучения программ с целью обнаружения функциональности
- Затруднение декомпиляции проприетарных программ с целью предотвращения обратной разработки или обхода систем проверки лицензий
- Оптимизация программы с целью уменьшения размера работающего кода и (если используется некомпилируемый язык) ускорения работы
- Демонстрация неочевидных возможностей языка и квалификации программиста (если производится вручную, а не инструментальными средствами)

Пример обфускации

Исходный вариант кода:

```
int COUNT = 100;
float TAX_RATE = 0.2;
for (int i=0; i<COUNT; i++)
{
    tax[i] = orig_price[i] * TAX_RATE;
    price[i] = orig_price[i] + tax[i];
}
```

Код после обфускации:

```
for(int a=0;a<100;a++){b[a]=c[a]*0.2;d[a]=c[a]+b[a];}
```

Обфускация

Как правило, обфускация на уровне машинного кода увеличивает время выполнения программы. Поэтому она применяется в критичных к безопасности, но не критичных к скорости местах программы.

Простейший способ обфускации машинного кода — вставка в него бесполезных конструкций.

Обфускация

В отличие от обычных языков, таких как C++ и Паскаль, компилирующих в машинный код, язык Java и языки платформы .NET компилируют исходный код в промежуточный код (байт-код), который содержит достаточно информации для адекватного восстановления исходного кода. По этой причине, для этих языков рекомендуется применять обфускацию промежуточного кода.

Обфускация

Декомпиляция программ Java и .NET достаточно проста. Поэтому, обфускатор оказывает неоценимую помощь тем, кто хочет скрыть свой код от посторонних глаз. Зачастую после обфускации декомпилированный код повторно не компилируется.

Про обфускацию программ:

http://habrahabr.ru/post/255871/

Как написать свой обфускатор:

http://eax.me/good-obfuscator/

Обфускаторы

Обзор обфускаторов для .NET:

http://habrahabr.ru/post/97062/

Принципы работы:

http://habrahabr.ru/post/74463/

Скачать обфускатор:

http://www.eziriz.com/downloads.htm

#	Вопрос	Ответы
1	Укажите основные причины возникновения платформы Microsoft .NET :	

#	Ответ	Верный
1	Использование современных языков программирования	
2	Необходимость межплатформенной переносимости:.	✔
3	Межъязыковая интеграция.	✔
4	Упрощение процесса развёртывания программ и контроля версий.	✔
5	Возможность использовать указатели.	

#	Вопрос	Ответы
2	Укажите компоненты .NET Framework:	

#	Ответ	Верный
1	1. Common Language Specification (CLS).	✔
2	NET Framework Class Library (.NET FCL).	✔
3	Common Language Runtime (CLR).	✔
4	MSIL	✔
5	Just In Time компилятор (JIT-компилятор).	✔
6	Base Class Library	✔

		1	1. Управление кодом (загрузка и выполнение). ✔
		2	2. Компиляция кода C# в машинный код.
3	Укажите задачи, выполняемые Common Language Runtime (CLR):	3	3.Управление памятью. ✔
		4	4. Отладка промежуточного кода.MISL
		5	5. Проверка безопасности кода. ✔
		6	6. Преобразование промежуточного языка в машинный код. ✔

		#	Ответ	Верный
		1	1. Устраняют необходимость в заголовочных и библиотечных файлах при компиляции.	✔
		2	Обеспечение динамической подсказки (IntelliSence) при разработке программы.	✔
		3	Описывают все типы данных, размещенные в сборке.	✔
4	Укажите назначение метаданных	4	Обеспечивают поддержку сериализации.	✔
		5	Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов	✔
		6	Обеспечивают быструю загрузку кода	
		7	Обеспечивают межязыковое взаимодействие	

#	Ответ	Верный
1	1. манифест	✔
2	2. MSIL	✔
3	3. OBJ	
4	4. EXE	
5	5. метаданные	✔
6	6. реестр	

5 Укажите состав сборки на языке программирования C#:

#	Ответ	Верный
1	1. В процессе выполнения программы компилируются только те ее части, которые требуется выполнить в данный момент	✔
2	2. При ПОВТОРНЫХ вызовах метода JIT компилятор осуществляет повторную быструю перекомпиляцию	
3	3. Откомпилированные инструкции JIT сохраняет в специальные файлы.	
4	4. Повторное выполнение откомпилированных инструкций выполняется быстрее.	✔
5	5. JIT компилятор учитывает особенности архитектуры CPU компьютера.	✔

6 Укажите особенности JIT компилятора:

7

Укажите все корректные соответствия между типами

#	Ответ	Верный
1	int -System.Int32	✔
2	long – System.Int64	✔
3	char – System.char	
4	float – System.Single	✔
5	bool– System.Bool	
6	string-System.String	✔
7		

8

Укажите основные библиотеки платформы .NET

#	Ответ	Верный
1	1. mscoree.dll	✔
2	2. frameworkNET.dll	
3	3.mscorlib.dll	✔
4	4. user32.dll	
5	5. mssharp.dll	

		#	Ответ	Верный
9	Укажите неверные диапазоны типов C#(.NET)	1	1. bool - 1 или 0	✔
		2	2. byte - 0 до 256	✔
		3	3. string - ограничен объемом системной памяти	
		4	4. System.Boolean - true или false	
		5	5. sbyte - от -128 до 127	

		#	Ответ	Верный
10	Укажите верных создателей языков программирования:	1	1. C# - Андерс Хейлсберг	✔
		2	2. C++ Дэннис Ритчи	
		3	3. Си Бьярни Страуструп	
		4	4. Java - Джеймс Гослинг, Патрик Ноутон	✔
		5	5. Objective C - С. Джобс	

Уроки

Текстовые уроки:

- https://metanit.com/sharp/
- https://professorweb.ru/my/csharp/charp_theory/level1/index.php

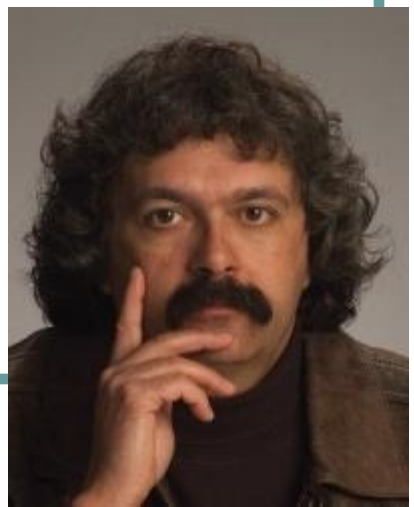
Видео-уроки:

Уроки Сергея Байдачного!

Литература по C#

- **Джефри Рихтер**
- Герберт Шилдт

Книги уже в беседе




FINALLY