Лекция 1.

Введение.

Эпиграф

Используя платформу .NET (Framework/Core) мы понимаем, что работаем на благо компании Microsoft.

С каждой написанной строчкой C# кода, растет мощь операционной системы Windows.

C# - это только язык. Ничего больше. 107 ключевых слов и собственный взгляд на ООП.

Windows - источник денег корпорации Microsoft.

.NET Framework - основная причина популярности Windows среди пользователей и программистов.

.NET Core - главная надежда корпорации Microsoft по порабощению пользователей Linux.

Мы (программисты) пишем код на C# на радость пользователей Windows (а скоро и на радость пользователей Linux).

За это пользователи Windows платят деньги нам и корпорации Microsoft.

Чем проще программисту написать код - тем богаче Microsoft.

Чем проще пользователю использовать код - тем богаче Microsoft.

Обе эти задачи решает платформа .NET.

Программсит просто учит язык C#, алгоритмы и структуры данных,

а платформа .NET гарантирует, что код успешно выполнится на компьютере пользователя.

В чем сложность?

В реальном мире существуют недочеты (препятсвия к сказочному обогащению Microsoft):

1) зоопарк hardware конфигураций компьютеров

2) сложность разработки (освобождение ресурсов, необходимость сразу писать производительный код)

3) зоопарк языков программирования со своим взглядом на типы данных и парадигмы программирования

4) злые хакеры, которые крадут данные

5) высокие требования пользователей к быстродействию и отзывчивости

Часть 1. Унификация зоопарка.

Раньше, что бы программу можно было запускать на разных компьютерах, её исходных код компилировали разными компиляторами.

Один компилятор = одна конфигурация физической машины (условно).

Такой подход плохо масштабируется. А что если вышла еще одна конфигурация физической машины?

Microsoft придумала решение проблемы - Common Language Runtime и Intermediate Language (IL).

Не надо компилировать исходный код под все возможные конфигурации физических машин.

Компилируйте только в один промежуточный код (IL) и отдавайте его пользователю.

Пусть пользователь запустит ваш IL-код на своей машине (с установленной платформой .NET),

и платформа сама преобразует ваш IL код в машинный код, понятный конкретно этой конфигурации физической машины.

Это преобразование на стороне пользователя осуществляется Jit компилятором (Just-in-time).

Таким образом проблема номер 1 была решена.

Часть 2. Мольба программиста.

Раньше программисты писали платформозависимый код на языках C/С++.

Жизнь их была тяжела и неказиста. Приходилось самим выделять и освобождать оперативную память, очень аккуратно использовать системные ресурсы, писать производительный код.

Все это было чревато ошибками. А пользователи не хотят платить Microsoft за систему, которая не доставляет им радость.

Microsoft придумало решение проблемы - Garbage collector, IDisposable и оптимизацию кода на этапе Jit компиляции.

Больше не нужно думать об освобождении оперативной памяти.

Использовать системные ресурсы - проще некуда, реализуй IDisposable и живи спокойно.

Производительность кода - 90% возможных оптимизаций за тебя сделают, и ты об этом даже не узнаешь.

Производительность так же улучшается из-за наличия стека вызовов и возможности хранить ValueType переменные прямо в нем.

Таким образом проблема номер 2 была решена.

Часть 3. Под одну гребенку.

Раньше программисты писали код на множестве разных языков. Потому что языки программирования в разной степени подходят под различные задачи.

Но вот появилась платформа .NET, как убедить программистов работать во имя корпорации Microsoft?

Microsoft придумала решение проблемы - Common Type System.

Пишите программы на любимом языке, а мы сами сопоставим ваши привычные типы данных на наши типы данных.

Вы получите все плюшки платформы .NET.

Вы даже сможете скомпилировать ваш старый исходный код под новую платформу.

Все, что сделала Microsoft, - написала новый компилятор под платформу .NET для каждого языка программирования (условно).

И тогда стало возможным использование библиотек, написанных на разных языках, внутри одного solution'а!

Таким образом проблема номер 3 была решена.

Часть 4. Встроенная защита.

Раньше жизнь была проще. Того, кто мог настроить принтер, считали программистом.

Того, кто понимал как работает компьютер, считали хакером. Никто не крал деньги со счетов.

А потом кто-то первый украл. И все поняли, что человек может влиять на программы и надо от этого защищаться.

Потом чья-то кривая программа сломалась на той же машине, на которой работала система запуска ядерных ракет.

И все поняли, что программы могут влиять друг на друга, и надо от этого защищаться.

Microsoft придумала решение проблемы - Application Domain.

Теперь вы можете быть уверены, что ваш код выполняется абсолютно изолированно.

Ни вы не можете влиять на другие программы, ни другие программы не могут влиять на ваш код.

Появились дополнительные проверки самого кода на предмет того, что и как он делает.

Таким образом проблема номер 4 была решена.

Часть 5. Процесс, который смог.

Раньше компьютеры были не очень мощные и никому в голову не приходило требовать от программ мгновенной реакции.

Если интерфейс программы не реагирует, значит работает. Хорошо, что работает.

Потом пользователи захотели большего. И всё зависло.

Microsoft придумала решение проблемы - многопоточность и асинхронное программировнаие.

Теперь главный поток процесса может создавать сколько угодно новых потоков и дилегировать им выполнение долгих задач.

Асинхронность позволяет наиболее оптимально использовать имеющиеся потоки и тоже позволяет не блокировать главный поток на долгих операциях к ресурсам.

Таким образом проблема номер 5 была решена.