

Разработка приложений на платформе .NET 4.7.2

Шаталов Юрий

Знакомство

▶ О себе

- Шаталов Юрий
- Закончил отделение второго высшего на ВМК
- Разработчик в крупной российской компании
- Microsoft Certified Professional Developer (MCPD)
- Курс читается 9 год.
 - Изменяется от года к году
 - Рассказ о современном положении дел
- e-mail
- Давайте общаться на ты и по именам

Знакомство

► О Вас

- Представьтесь
- Каково ваше отношение к IT?
 - работаете в IT, собираетесь, интересуетесь
- Что знаете / слышали о .NET, C# / C++ / C / Java ?
- Что ожидаете от курса ?

Программа курса

1 семестр

- ▶ Основы программирования на языке C# 7.3 в среде .NET Framework 4.7.2

2 семестр

- ▶ Разработка Windows приложений на платформе Microsoft .NET.
 - Windows Presentation Foundation (WPF)
- ▶ Доступ к данным и манипуляция данными
 - ADO.NET, LINQ, ADO.NET Entity Framework
- ▶ Разработка распределенных приложений
 - Windows Communication Foundation (WCF).
- ▶ Разработка Web-приложений на платформе .NET
 - ASP.NET MVC

Отчетность

- ▶ В конце каждого семестра
 - Экзамен
 - Зачет
- ▶ Для зачета необходимо сделать ВСЕ домашние работы
- ▶ Зачет и Экзамен независимые* и необязательные**
- ▶ Бонус
 - При сдаче любого сертификационного экзамена Microsoft на пути к MCSD по разработке ПО – зачет и экзамен за семестр автоматом
 - При получении статуса MCSD App Builder (Microsoft Certified Solutions Developer) – автоматом зачеты и экзамены за год

Литература

Основы C# и .NET

- ▶ Э. Троелсен, Ф. Джепикс. Язык программирования C# 7 и платформы .NET и .NET Core (2018)
- ▶ Д. Албахари, Б. Албахари. C# 7.0. Справочник. Полное описание языка (2018)
- ▶ Джеффри Рихтер. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# (2013/2017)
- ▶ Старайтесь читать литературу по .NET в оригинале (на английском языке)
- ▶ Help <https://docs.microsoft.com/ru-ru/dotnet/>
- ▶ Исходники .NET <http://referencesource.microsoft.com/>

Разное

► Софт

- Visual Studio 2017. Можно бесплатно получить лицензионную версию в комнате 705.
- Resharper (<http://www.jetbrains.com/resharper/>) – помогает быстрее и качественней разрабатывать

► Сайт

- <http://msudotnet.ru/>

► Почта

- E-mail

Разновидности .NET

▶ .NET Framework

- Наиболее полная версия .NET
- Кроссплатформенная для Windows
- Входит в дистрибутив и обновления Windows
- Актуальная версия .NET Framework 4.7.2

▶ .NET Core

- Платформа разработки с открытым кодом
- Поддерживается Майкрософт и сообществом .NET на сайте GitHub
- Кроссплатформенная: Windows, MacOS и Linux
- Активно развивается
- Актуальная версия .NET Core 2.1

▶ .NET Standard

- Поддерживается всеми платформами .NET
- Открытый стандарт
- Актуальная версия .NET Standard 2.0


▶ Xamarin

- Позволяет единым образом создавать приложения для Android, iOS, Mac

▶ Mono

- Урезанная версия NET для Unix и Mac
- Open source
- Не Майкрософт

Сегодня

- ▶ Предыстория
 - ▶ Понятие платформы .NET
 - ▶ Первая программа на C#
 - ▶ Отличия от C++
 - ▶ Основные типы
- 

Предыстория

- ▶ Язык C. Использование Win API
 - Процедурный стиль
 - Не объектный
 - Сложный API
- ▶ Язык C++. Использование различных обертток над Win API
 - Сложная работа с указателями
 - Ручное управление памятью
 - Переносимость, оптимизация
- ▶ Visual Basic 6
 - Нет классического наследования
 - Трудности с многопоточностью
- ▶ Java
 - Подразумевает использования только языка Java
- ▶ COM
 - Нет наследования
 - Не гарантии совместимости типов
 - Необходимость регистрации в реестре
 - Невозможность использования разных версий одного и того же модуля

Что дает платформа .NET


- ▶ Кроссплатформенность
 - Разные версии Windows
 - Частично клоны Unix <http://www.mono-project.com>
 - Mac OS <http://www.mono-project.com>
 - .NET Core (Core CLR). Open Source <https://github.com/dotnet/coreclr>
 - Xamarin
- ▶ Поддержка нескольких языков программирования
 - C#, VB.NET, JScript.NET, Managed C++ (C++/CLI), F#
 - https://en.wikipedia.org/wiki/List_of_CLI_languages
- ▶ Общая среда выполнения для различных языков программирования
 - Прозрачное межязыковое взаимодействие
- ▶ Библиотека базовых классов
 - Web, Mobile, Desktop, Services, Data Access, Cloud, ...
- ▶ Упрощенная работа с памятью
- ▶ Простое развертывание (не нужна регистрация в реестре)
- ▶ Безопасность
- ▶ Взаимодействие со старым кодом (native, COM и т.д.)

Быстрое написание сложных программ

Версии .NET Framework

Версия .NET	Год выхода	По умолчанию в версиях Windows	Нововведения	Visual Studio	Версия C#
.NET Framework 1.0	2002			Visual Studio .NET	C# 1
.NET Framework 1.1	2003	Windows Server 2003		Visual Studio .NET 2003	C# 1.2
.NET Framework 2.0	2005		Обобщенные типы, анонимные методы, x64 и IA-64.	Visual Studio 2005	C# 2
.NET Framework 3.0	2006	Windows Vista, Windows Server 2008	WPF, WCF, WF	Visual Studio 2005 + расширения	C# 3
.NET Framework 3.5	2007	Windows 7, Windows Server 2008 R2	LINQ, ADO.NET Entity Framework, ASP.NET AJAX	Visual Studio 2008	
.NET Framework 4.0	2010		Dynamic, PLINQ, F#, Windows Azure Apps	Visual Studio 2010	C# 4
.NET Framework 4.5, 4.5.1, 4.5.2	2012, 2013, 2014	Windows 8 (8.1), Windows Server 2012 (R2)	Windows Store Apps (Windows 8, 8.1)	Visual Studio 2012, 2013	C# 5
.NET Framework 4.6, 4.6.1, 4.6.2	2015, 2016	Windows 10, Windows Server 2016	RyuJIT, UWP, .NET Native, поддержка экранов с высоким DPI, расширение криптографии	Visual Studio 2015	C# 6
.NET Framework 4.7, 4.7.1	2017	Windows 10 RS2–RS3	High DPI for WinForms, TLS 1.2 support	Visual Studio 2017	C# 7
.NET Framework 4.7.2	04.18	Windows 10 RS4 (1803)	DI ASP.NET, минорные изменения	Visual Studio 2017 (Update 15.7)	C# 7.3 (09.18)

Сегодня

- ▶ Предыстория
 - ▶ **Понятие платформы .NET**
 - ▶ Первая программа на C#
 - ▶ Отличия от C++
 - ▶ Основные типы
- 

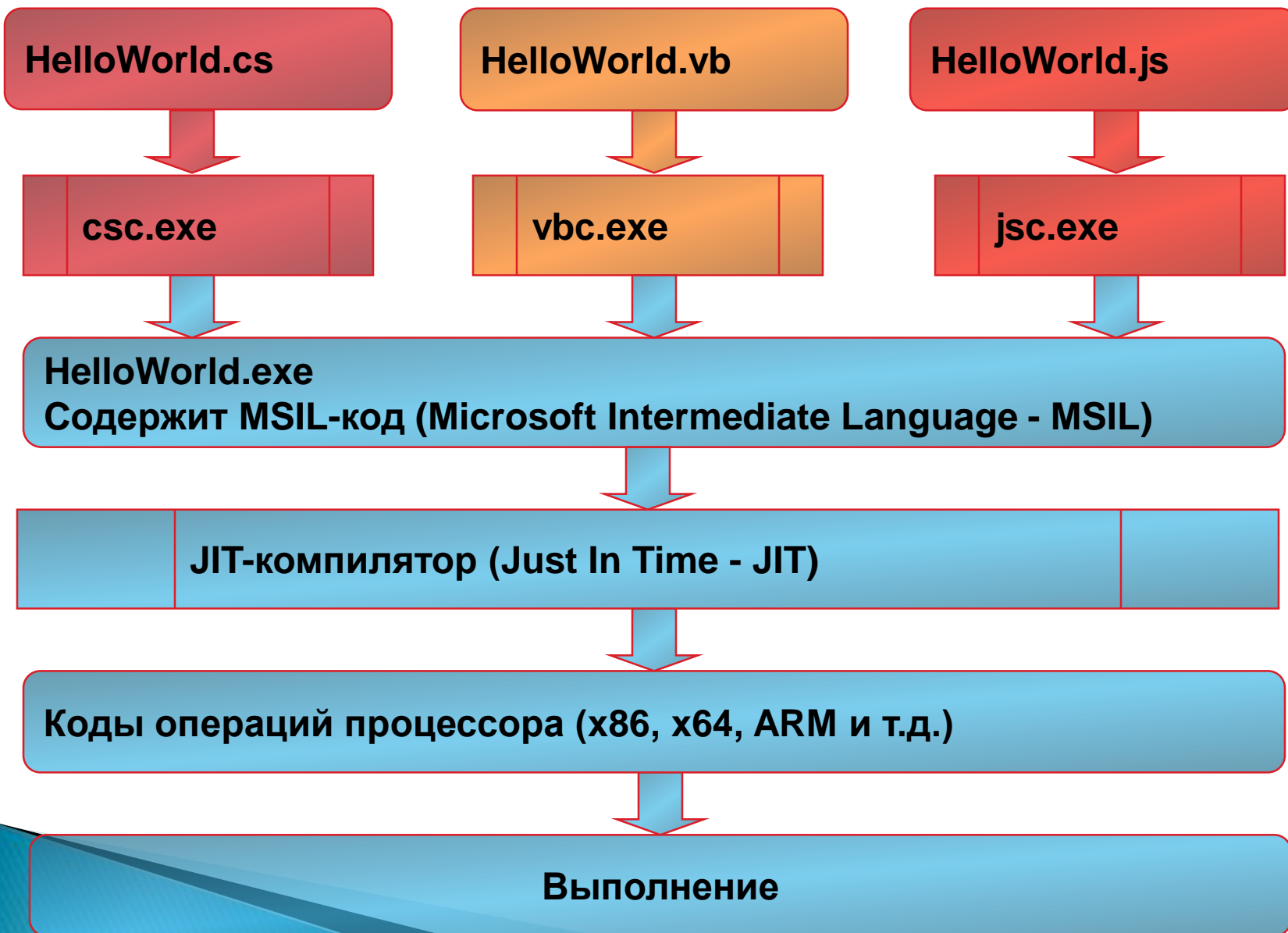
Платформа .NET

- ▶ **Общезыковая среда выполнения**
Common Language Runtime – CLR
 - MSIL JIT-компилятор
 - Сборщик мусора
- ▶ Единая система типов
Common Type Specification – CTS
- ▶ Общезыковая спецификация
Common Language Specification – CLS
- ▶ Библиотека базовых классов
Base Class Library – BCL
 - Global Assembly Cache – GAC

Общезыковая среда выполнения

- ▶ **Common Language Runtime – CLR**
- ▶ Виртуальная исполняющая среда
- ▶ Запускается при старте вашего приложения
- ▶ Отвечает за:
 - Загрузку сборок
 - Just In Time компиляцию
 - Управление памятью
 - Управление безопасностью

MSIL-компиляция



JIT компиляция и CLR

- ▶ Программный код компилируется в промежуточный код (Intermediate Language – IL, MSIL, CIL)
- ▶ CLR – не интерпретатор. Компиляция происходит 1 раз. Повторно не компилируется, а используется уже откомпилированный код

— • Более медленный старт и работа приложения

- +
- Экономия памяти
 - Код на IL обычно занимает меньше места
 - Компилируется только тот код, который выполняется
 - JIT компилятор получает высоко оптимизированный код (заточенный под конкретную аппаратную модель)
 - CLR отслеживает частоту вызова и может производить оптимизацию налету

Общезыковая среда выполнения

- ▶ Common Language Runtime – CLR
- ▶ Отвечает за:
 - Загрузку сборок
 - Just In Time компиляцию
 - **Управление памятью**
 - Управление безопасностью

Управление памятью

▶ Автоматическая сборка мусора

```
// Утечка памяти в C
char *f(int a)
{
    char *p = (char)malloc(...);
    ...
    return p;
}
...
void g(){ f(1); }
```

Утечка
памяти

функция f():

char *p → "text"

функция g():

→ × → "text"

Сборщик мусора (Garbage Collector - GC) отслеживает ссылки на объекты. Он обнаружит, что на область памяти p больше нет ссылок и освободит эту область.

▶ CLR может перенести часто используемые объекты для оптимизации доступа к страницам памяти

Общезыковая среда выполнения

- ▶ Common Language Runtime – CLR
- ▶ Отвечает за:
 - Загрузку сборок
 - Just In Time компиляцию
 - Управление памятью
 - Управление безопасностью

Сборки

- ▶ Независимая единица кода
- ▶ Файл с расширением dll или exe *
- ▶ Состоит из:
 - **Манифеста**
 - Содержит информацию о текущей версии сборки, культуре, перечень ссылок на все внешние сборки, необходимые для работы сборки
 - **Метаданных типов**
 - Описание всех типов внутри сборки, их публичных членов
 - **Промежуточного кода (IL)**
 - **Ресурсов**
- ▶ Благодаря самоописанию, значительно упрощает повторное использование (не нужна сложная COM инфраструктура)
- ▶ Сборка может содержать цифровую подпись

Утилиты ILDASM, [dotPeek](#), [.NET Reflector](#)

Структура .NET сборок


Демонстрация



Общезыковая среда выполнения

- ▶ CLR (Common Language Runtime)
 - Загрузка сборок
 - Just In Time компиляция
 - Управление памятью
 - Управление безопасностью

Платформа .NET

- ▶ **Общезыковая среда выполнения**
 - ▶ **Единая система типов**
Common Type Specification – CTS
 - ▶ **Общезыковая спецификация**
Common Language Specification – CLS
 - ▶ **Библиотека базовых классов**
Base Class Library – BCL
 - **Global Assembly Cache – GAC**
- 

Межязыковое взаимодействие

Демонстрация



Единая система типов (CTS)

- ▶ Типы одинаковые на всех языках
- ▶ Поскольку в силу особенности языков не все языки могут поддерживать все типы (CTS) выделено подмножество типов. Это подмножество типов описано в Общеязыковой спецификации (CLS). Все типы в CLS обязаны поддерживаться всеми .NET языками.



Типы описанные в CLS могут использоваться для межязыкового взаимодействия

Могут использоваться, но не в публичных интерфейсах (если конечно нужно межязыковое взаимодействие)

Платформа .NET

- ▶ Общезыковая среда выполнения
- ▶ Единая система типов
- ▶ Общезыковая спецификация
- ▶ Библиотека базовых классов
 - Base Class Library – BCL
 - Global Assembly Cache – GAC


Библиотека базовых классов (BCL)

- ▶ Расположена в **Global Assembly Cache – GAC**
- ▶ `c:\Windows\Microsoft.NET\assembly`
- ▶ `c:\Windows\assembly;`
- ▶ Может использоваться всем программами
- ▶ Позволяет сохранять и использовать разные версии одной и той же сборки
- ▶ `mscorelib.dll` – основная сборка.
Используется во всех программах.
Содержит пространство имен `System`.

Платформа .NET

- ▶ **Общезыковая среда выполнения**
Common Language Runtime – CLR
 - MSIL JIT-компилятор
 - Сборщик мусора
- ▶ **Единая система типов**
Common Type Specification – CTS
- ▶ **Общезыковая спецификация**
Common Language Specification – CLS
- ▶ **Библиотека базовых классов**
Base Class Library – BCL
 - **Global Assembly Cache – GAC**

Сегодня

- ▶ Предыстория
 - ▶ Понятие платформы .NET
 - ▶ **Первая программа на C#**
 - ▶ Отличия от C++
 - ▶ Основные типы
- 


Hello, World!

```
using System;

namespace Hello
{
    class HelloWorld
    {
        /// <summary> Entry point </summary>
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, C# World!");

        } // end of Main()

    } // end of HelloWorld
} // namespace
```



Первая программа на C#


Демонстрация



Замечания

- ▶ Пространство имен
 - объединяет группу семантически связанных между собой типов
 - Позволяет отделять типы с одинаковыми названиями
- ▶ Варианты метода Main
 - `static void Main() {...}`
 - `static int Main() {... return 0; }`
 - `static void Main(string[] args) {...}`
 - `static int Main(string[] args) {... return 0; }`
- ▶ **using** позволяет сократить полное название типа (System.Console).
Как бы объединяет пространства имен* с текущим (*или тип в C# 6)
- ▶ .NET использует Unicode.
 - Название типов можно заводить и на русском языке (но не рекомендуется)
- ▶ Языки для .NET чувствительны к регистру
 - Main() и main() разные методы
- ▶ Вывод на консоль: `System.Console.WriteLine("текст")`
- ▶ Чтение данных с консоли: `string s = System.Console.ReadLine()`

Сегодня

- ▶ Предыстория
 - ▶ Понятие платформы .NET
 - ▶ Первая программа на C#
 - ▶ **Отличия от C++**
 - ▶ Основные типы
- 

Отличия от С

- ▶ Не нужны заголовочные файлы и вообще заголовки. Сборки сами себя описывают
- ▶ Указателей **«нет»**. Но есть ссылки
- ▶ Нормальный строковый тип **string**
- ▶ Логический тип **bool**
- ▶ Нет глобальных полей и функций
 - Любое поле/метод – член класса
- ▶ Нет «провала» в **switch()** {}
- ▶ Имеет атрибуты, что позволяет использовать аспектноориенированное программирование
- ▶ Контекст вычислений
 - **checked** / **unchecked**

Логический тип bool

while(условие-продолжения)
 оператор

do
 оператор
while(условие-продолжения)

Условие должно иметь
логический тип!

```
int i = 10;  
while (i--) // ошибка!  
    Console.Write(i);
```

if (условие)
 оператор1
else
 оператор2

if (условие)
 оператор

Условие – только тип bool

Оператор switch

```
switch (day) {  
    case 1:  
        Console.Write("Понедельник");  
        break;  
    case 2:  
    case 3:  
        Console.Write("Вторник или среда");  
        break;  
    default:  
        Console.Write("Другой день недели");  
        break;  
}
```

- ✓ Каждая альтернатива должна завершаться break, return, throw, continue
- ✓ switch() работает и со строками, и с другими типами при использовании pattern matching (C# 7)

Сегодня

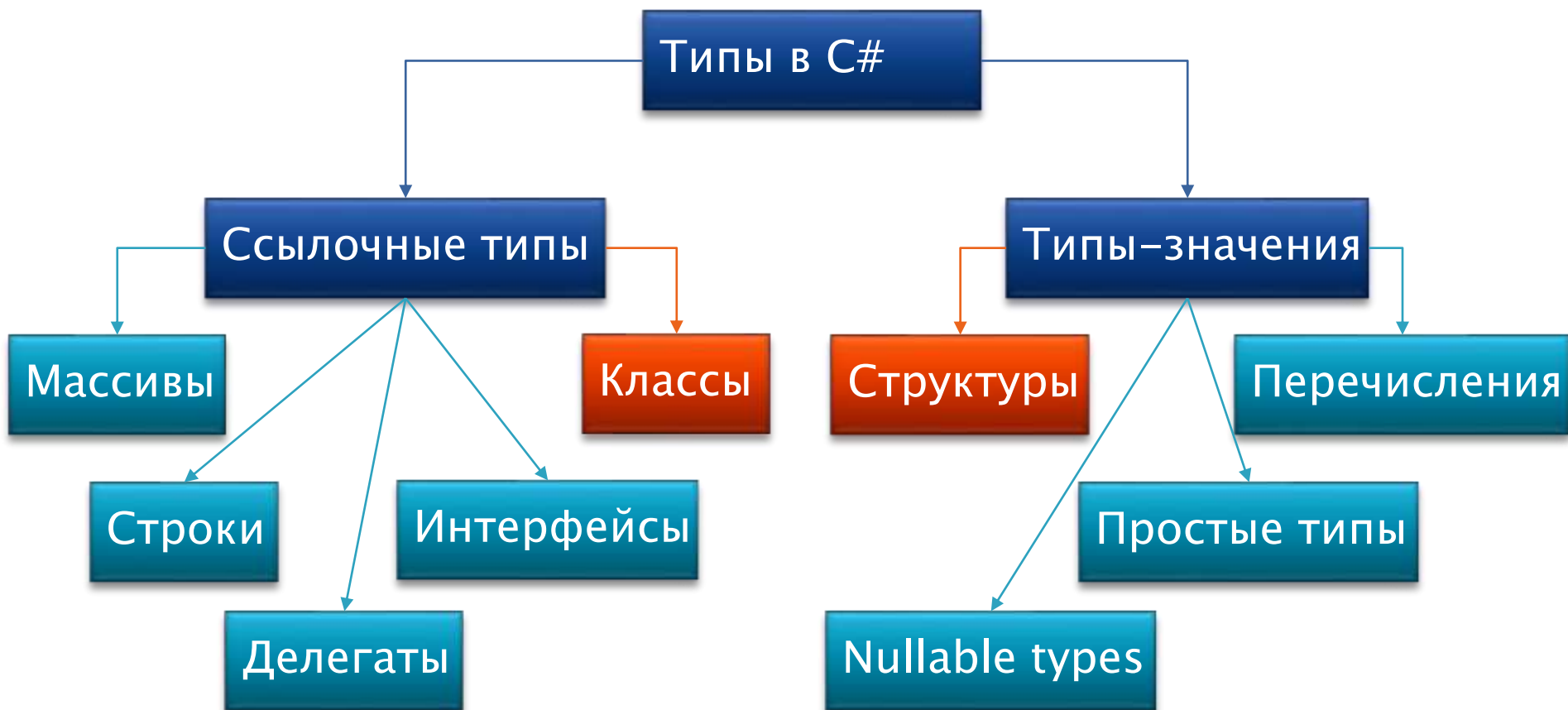
- ▶ Предыстория
- ▶ Понятие платформы .NET
- ▶ Первая программа на C#
- ▶ Отличия от C++
- ▶ **Основные типы**

Строгая типизация в C#

- ▶ Каждая переменная и экземпляр объекта в системе относится к четко определенному типу !!!
- ▶ Все типы происходят от одного корневого предка – типа **object** *

* – точнее приводятся к типу *object*

Типы в C#



* - условная схема, поскольку все ссылочные типы (кроме интерфейсов) – классы, все типы значения - структуры

Ссылки и значения


Признак	Ссылки	Значения
Переменная	Ссылка на объект	Сам объект
Память	Куча	Стек или куча
Присваивание	Копирование ссылки	Копирование объекта
Значение по умолчанию	null	0, 0.0, '\0', false

Ссылки и значения

Демонстрация



Самые важные типы

- ▶ **int** – 32-битное целое (System.Int32)
 - ▶ **bool** – логический тип (System.Boolean). Значения только **true** и **false**
 - ▶ **float, double** – вещественные типы (System.Single и System.Double)
 - ▶ **char** – символьный тип Unicode
 - ▶ **string** – строка текста (Unicode)
 - ▶ **DateTime** – дата и время
- 

Простые целые типы

Тип (C#)	Полное название типа	Диапазон	Описание	Размер (бит)
sbyte	System.Sbyte	-128 до 127	Знаковое целое	8
byte	System.Byte	0 до 255	Без знаковое целое	8
short	System.Int16	-32 768 до 32 767	Знаковое целое	16
ushort	System.UInt16	0 до 65 535	Без знаковое целое	16
int	System.Int32	-2 147 483 648 до 2 147 483 647	Знаковое целое	32
uint	System.UInt32	0 до 4 294 967 295	Без знаковое целое	32
long	System.Int64	$-9 * 10^{19}$ до $9 * 10^{19}$	Знаковое целое	64
ulong	System.UInt64	0 до $18 * 10^{19}$	Без знаковое целое	64
char	System.Char	U+0000 до U+ffff	Символ в Unicode	16

* Все типы – типы значения

Вещественные типы

Тип (C#)	Полное название типа	Диапазон	Точность	Размер (бит)
float	System.Single	$\pm 1.5 * 10^{-45}$ до $\pm 3.4 * 10^{38}$	7 знаков	32
double	System.Double	$\pm 5.0 * 10^{-324}$ до $\pm 1.7 * 10^{-308}$	15–16 знаков	64
decimal **	System.Decimal	$(-7.9 \times 10^{28}$ до $7.9 \times 10^{28}) / (10^0$ до $2^8)$	28–29 знаков	128

** Не имеет аппаратной поддержки
Всегда проверяет диапазон

* Все типы – типы значения

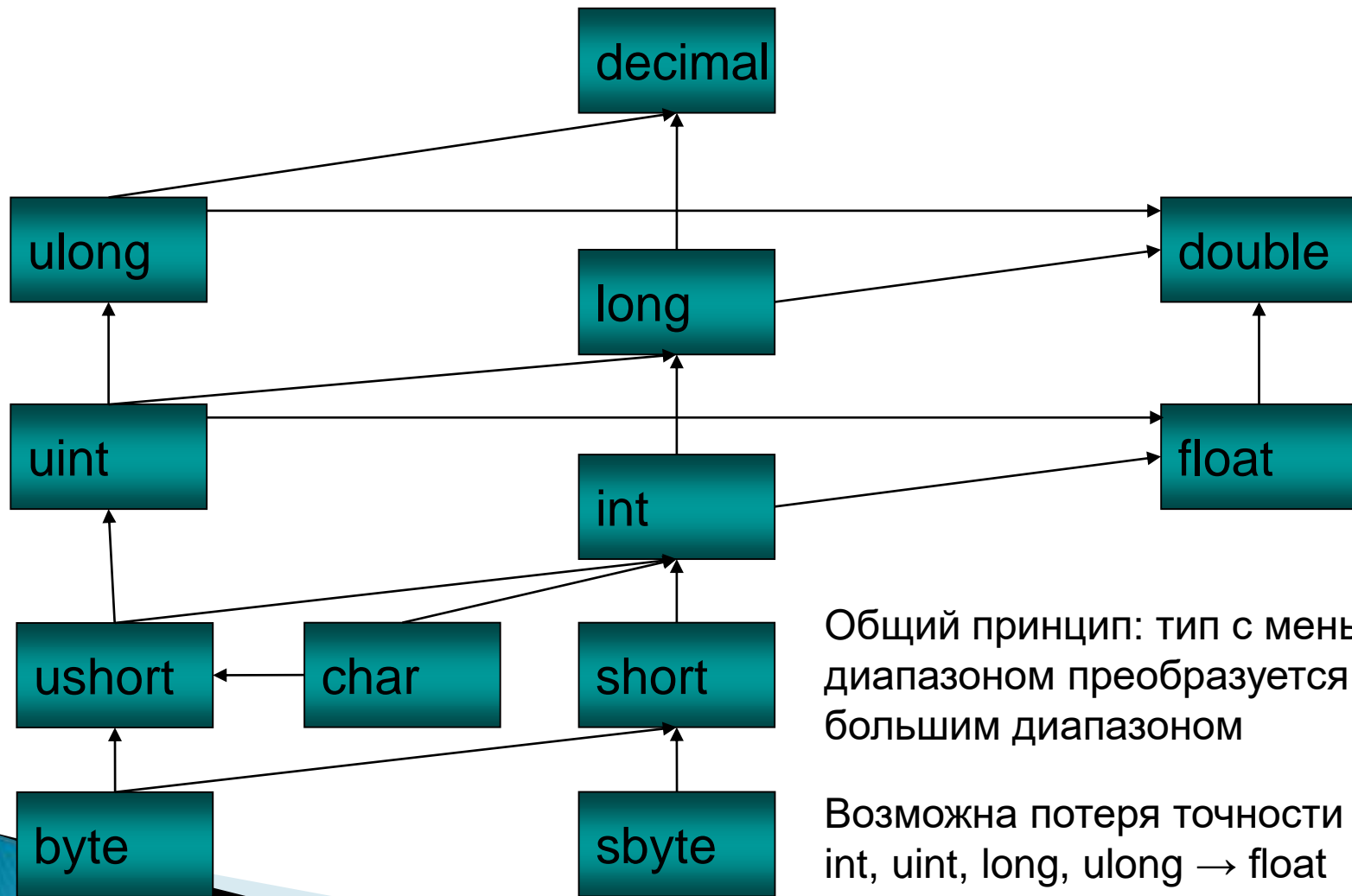
Важные типы

- ▶ **bool** – логический тип
 - System.Boolean
 - Значения только **true** и **false**
 - Тип значение
- ▶ **string** – строка текста (Unicode)
 - System.String
 - неограниченной длины
 - Ссылочный тип
- ▶ **DateTime** – дата и время
 - Структура (тип-значение)
 - От 1 января 1 года до 31 декабря 9999 года
 - Точность 100 нс
 - Работает с временными зонами

Типы данных по умолчанию

- ▶ Если значение целое и оно помещается в `int` – то подразумевается `int`
 - 5 – тип `int` Пример: `int i = 45;`
- ▶ Если значение вещественное – то подразумевается `double`
 - 5.6 – тип `double` Пример: `double d = 12.277;`
- ▶ Для обозначения конкретных типов служат “суффиксы”
 - 5l – `long` Пример: `long l = 5l;`
 - 5.4f – `float` Пример: `float f = 5f;`
 - 4m – `decimal` Пример: `decimal d = 0m;`
- ▶ Шестнадцатеричное число `0x`ЧИСЛО
 - `0x0099` Пример: `int i = 0x1234FFFF;`
- ▶ Восьмеричное число `0`ЧИСЛО
 - `06789` Пример: `int i = 05777;`
- ▶ Двоичное число `0b`ЧИСЛО (C# 7). `_` – разделитель разрядов (C# 7)
 - `0b1010`, `0b1101_10101` Пример: `int i = 0b0111_0000_1111_0101;`
- ▶ Выражения. Тип выражения определяется в порядке приоритета:
 - Если в выражении присутствует `decimal`, то результат операции – `decimal`
 - Если присутствует вещественное число, то результат операции – `double`
 - `ulong`, если присутствует тип `ulong`
 - `long`, если присутствует тип `long`
 - Результат операции с целыми числами – `int`

Неявное приведение типов



Общий принцип: тип с меньшим диапазоном преобразуется в тип с большим диапазоном

Возможна потеря точности при:
int, uint, long, ulong → float
long, ulong → double.

Явное приведение типов

- ▶ Синтаксис:

- *(type) expression*

- ▶ Пример:

```
double d = 5.5;
```

```
int i = (int)d;
```

- ▶ Применяется при преобразованиях типов с возможной потерей точности
- ▶ При «зашкаливании» результат определяется контекстом

Контекст проверки вычисления

- ▶ 2 контекста
 - **checked** – проверяет на переполнение
 - **unchecked** – не проверяет
- ▶ Устанавливаются
 - Глобально (опции проекта)
 - Локально (блоки **checked** и **unchecked**)
 - Не распространяется на вызовы функций
- ▶ По умолчанию проверка выключена.
 - Однако, если значение выражения может быть вычислено в процессе компиляции, то употребляется контекст **checked**
 - `byte h = (byte) (255 + 100); // вызовет ошибку в процессе компиляции`

Checked и unchecked

Демонстрация

Перечисление

- ▶ Служит для кодирования возможных значений или магических чисел

- ▶ `enum MyEnum {`

- `Monday,`
 - `Thursday,`
 - `...`

- ▶ `}`

- ▶ `enum OneMoreEnum {`

- `Monday = 2,`
 - `Thursday,`
 - `Среда = 4,`
 - `...}`

```
enum Имя [:базовый целочисленный тип]
{
    Имя1 [=значение1]
    [, ... ИмяN [=значениеN]]
}
```

- ▶ Объявление и использование:

- `OneMoreEnum my = OneMoreEnum. Среда ;`

- ▶ По умолчанию “наследуются” от `int`, но могут “наследоваться” от другого целочисленного типа

- ▶ Если не указано значение, то для первого по умолчанию – 0, для каждого последующего – предыдущее + 1

- ▶ Возможно приведение типов: `int l = (int)my; int j = (int)OneMoreEnum. Среда;`

Массивы

- ▶ Содержат элементы только одного типа
- ▶ Длина
 - Задаётся при выделении
 - Изменить потом нельзя
- ▶ Бывают
 - Одномерные
 - Многомерные
- ▶ Допускается вложенность
- ▶ Нумерация элементов всегда с 0
 - В C# – всегда с 0, но .NET позволяет задавать массивы и с не нулевой нижней границей индекса

Одномерные массивы

► Объявление:

- `type_name[] var_name [= init_expr];`

► Создание массива (обязательное задание длины массива)

- `int[] arr1; // объявление переменной`
- `arr1 = new int[5]; // создание массива из 5 элементов`
- `byte[] digits = new byte[10];`

► Объявление и инициализация:

- `int[] arr1 = new int[5]{10, 20, 30, 40, 50};`
- `int[] arr1 = {1, 2, 3, 4, 5}; // длина вычисляется автоматически`

► Индексация:

- `int j = arr1[2];`
- `arr1[23] = 345;`
- Компилятор добавляет квазистатическую проверку выхода из диапазона
- Нумерация начинается с 0 (в C#)

► Получить длину массива

- `int len = arr1.Length;`

Многомерные массивы

► Прямоугольные многомерные массивы

- `type_name[,...,:] var_name [= init_expr]`
- **Объявление и создание массива**
 - `int[,] matrix = new int[4,7];`
 - `int[, ,] matrix3d = new int[4, 2, 3];`
- **Инициализация массива при создании**
 - `string[,] array2d = new string[3, 2] { { "one", "two" }, { "three", "four" }, { "five", "six" } };`
- **Можно создать массив без указания размера. Размер вычисляется по правой части выражения**
 - `int[,] matrix2d = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`
 - `int[,] matrix2x2;`
 - `array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };`
- **Обращение к элементам массива**
 - `int element = matrix[3,5];`
 - `matrix[2,1] = 34;`
- **Получение длины массива**
 - `int dimension1 = matrix.GetLength(1); // размер первого измерения`
 - `int dimension = matrix.Length; // Общее количество элементов в массиве`

Многомерные массивы

► Рваные многомерные массивы (массивы массивов)

- `type_name[][]...[] var_name [= init_expr]`
- **Объявление и создание массива**
- `int[][] matrix = new int[3][];`
- `matrix[0] = new int[3];`
- `matrix[1] = new int[7];`
- `matrix[2] = new int[] { 11, 22 }; // сразу заполнение измерения. Размер измерения вычисляется автоматически`
- **Инициализация всего массива при создании. Размеры вычисляются автоматически**
 - `int[][] jaggedArray =`
 - `{`
 - `new int[] {1,3,5,7,9},`
 - `new int[] {0,2,4,6},`
 - `new int[] {11,22}`
 - `};`
- **Обращение к элементам массива**
 - `int element = matrix[3][5];`
 - `matrix[2][1] = 34;`
- **Получение длины массива**
 - `int dimension = matrix.Length; // количество элементов (массивов) в массиве`

Массивы

Демонстрация



Сегодня

- ▶ Предыстория
 - ▶ Понятие платформы .NET
 - ▶ Первая программа на C#
 - ▶ Отличия от C++
 - ▶ Основные типы
- 