Блок 1. Базовый C#

1.1 — Основы C#

План занятия

- Работа с консолью
- Типы данных
- Ресурсы и литература

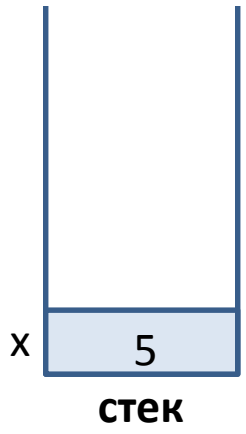
Объекты и типы данных

- Тип данных — специальная структура данных, объединяющая набор переменных (полей), функций (методов) и прочих членов в единую запись, являющуюся шаблоном для объектов.
- Объект — строго типизированный контейнер, отражающий состояние и функциональные возможности конкретной реальной или виртуальной сущности;

Размещение объектов в памяти

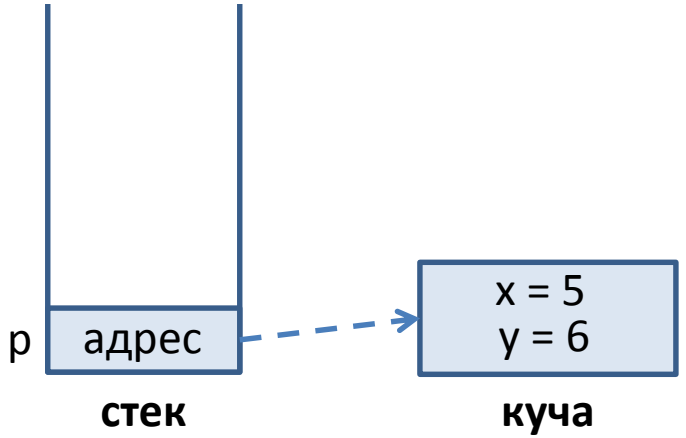
Экземпляр структуры

```
int x = 5;
```



Экземпляр класса

```
MyPoint p = new MyPoint();
p.x = 5; p.y = 6;
```



http://en.wikipedia.org/wiki/Stack_(data_structure)
http://ru.wikipedia.org/wiki/Стек_вызовов
http://en.wikipedia.org/wiki/Call_stack
http://en.wikipedia.org/wiki/Dynamic_memory_allocation

Виды типов данных

- Все типы данных в .NET делятся на две основные группы:
  - **структуры** (value type, значимые типы):
    - Объекты могут храниться непосредственно в стеке;
    - Время жизни объекта определяется текущей областью видимости;
  - **классы** (reference type, ссылочные типы):
    - Объекты хранятся в управляемой куче (динамическая память);
    - Ссылка может вести на несуществующий объект по нулевому адресу — **null**;
    - Объект может быть уничтожен только *сборщиком мусора* (при отсутствии живых ссылок).

Встроенные типы C#

Тип C#	Тип .NET	Описание типа данных
byte	System.Byte	Целочисленный, 8 бит, беззнаковый
sbyte	System.SByte	Целочисленный, 8 бит, знаковый
short	System.Int16	Целочисленный, 16 бит, знаковый
ushort	System.UInt16	Целочисленный, 16 бит, беззнаковый
int	System.Int32	Целочисленный, 32 бита, знаковый
uint	System.UInt32	Целочисленный, 32 бита, беззнаковый
long	System.Int64	Целочисленный, 64 бита, знаковый
ulong	System.UInt64	Целочисленный, 64 бита, беззнаковый
char	System.Char	Аналог ushort, хранит код символа в UTF16
float	System.Single	Вещественный с плавающей точкой, 32 бита
double	System.Double	Вещественный с плавающей точкой, 64 бита
decimal	System.Decimal	Вещественный с увеличенной мантиссой, 128 бит
bool	System.Boolean	Логический: true или false
string	System.String	Строка
object	System.Object	Объект

структуры

классы

Неизменяемые поля

- Константы:
  - Поддерживаются только встроенные типы C# (за исключением object);
  - Значение определяется на этапе компиляции и не может быть изменено;
  - Являются членами самого типа данных, а не его объектов (статические члены типа).

```
const int X = 5;
```

- Поля только для чтения:
  - Нет ограничений по типу данных;
  - Значение определяется при создании объекта;
  - Могут являться членами как типа данных, так и его объектов;
  - Если типом объекта является класс, то неизменяемой является только ссылка на объект; поля объекта изменить можно.

```
readonly int X = LoadXFromFile();
```

Пример перечисления

- Объявление:

```
enum DaysOfWeek
{
    Sunday,      // 0 - по умолчанию нумерация начинается с нуля
    Monday,      // 1 - каждый последующий элемент на единицу больше
    Tuesday,     // 2
    Wednesday,   // 3
    Thursday,    // 4
    Friday = 7,  // 7 - значение может быть присвоено явно
    Saturday,    // 8 - в этом случае нумерация продолжается
}
```

- Использование:

```
DayOfWeek day = DayOfWeek.Monday;
Console.WriteLine(day);
Console.WriteLine((int)day);
```

```
Monday
1
```

Перечисления

- Пользовательская структура, содержащая набор именованных констант указанного основного типа (по умолчанию int);
- Поддерживаются все встроенные целочисленные типы, кроме char;
- Объект перечисления может содержать любое значение основного типа, даже если для него отсутствует явно заданная константа;
- По умолчанию объект перечисления равен нулю.

Объект перечисления как набор битовых флагов

```
[Flags]
enum Actions : byte
{
    None = 0,                                   // 00000000
    Write = 1,                                  // 00000001
    Read = 2,                                   // 00000010
    ReadWrite = Actions.Read | Actions.Write,   // 00000011
    Append = 4,                                 // 00000100
}

Actions a = Actions.Append | Actions.Write;     // 00000101
if (a.HasFlag(Actions.Write))
{
    // ...
}
```

```
  00000101
& 00000001
----------
  00000001
```

```
  00000100
& 00000001
----------
  00000000
```

http://www.dotnetperls.com/enum-flags
http://www.dreamincode.net/forums/topic/15494-c%23-flags/

Расширенная работа с перечислениями

- К объектам перечисления применимы операции алгебры логики (&, |, ^, ~);
- При инициализации констант степенями двойки объект перечисления может выступать в роли набора битовых флагов;

Собственные структуры пользователя

- Объявление:

```
struct MyPoint
{
    public int x;
    public int y;
}
```

- Инициализация:

```
MyPoint p;
p.x = 1;
p.y = 2;
```

объект инициализирован

или

```
MyPoint p = new MyPoint();
p.x = 5;
p.y = 6;
```

объект инициализирован

Собственные классы пользователя

- Объявление:

```
class MyPoint
{
    public int x;
    public int y;
}
```

- Создание объекта:

```
MyPoint p = new MyPoint();
p.x = 5;
p.y = 6;
```

объект инициализирован

Различия между структурами и классами

Структура

```
struct SPoint
{
    public int x;
    public int y;
}

SPoint p = new SPoint();
p.x = 5; p.y = 6;
SPoint p1 = p;
p.x = 10;
```

Класс

```
class CPoint
{
    public int x;
    public int y;
}

CPoint p = new CPoint();
p.x = 5; p.y = 6;
CPoint p1 = p;
p.x = 10;
```

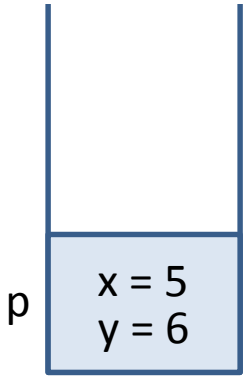
Чему равен p1.x?

Оператор присвоения значения

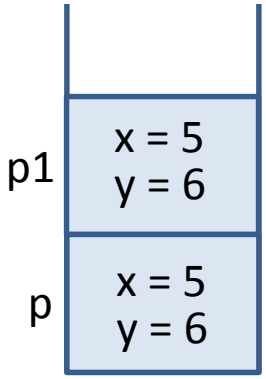
- Оператор присвоения значения (=) копирует непосредственное значение (в стеке) правой переменной в левую переменную;
- Объект структуры хранится в стеке, следовательно оператор присвоения копирует его значение;
- Объект класса хранит в стеке только *ссылку*, следовательно оператор присвоения копирует *ссылку* на **тот же** объект в куче.

Различия между структурами и классами

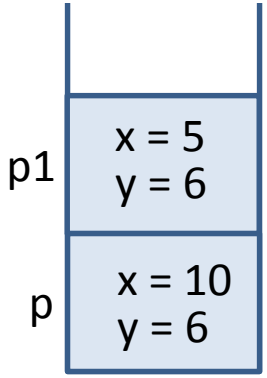
Структура




```
SPoint p = new SPoint();
p.x = 5; p.y = 6;
```



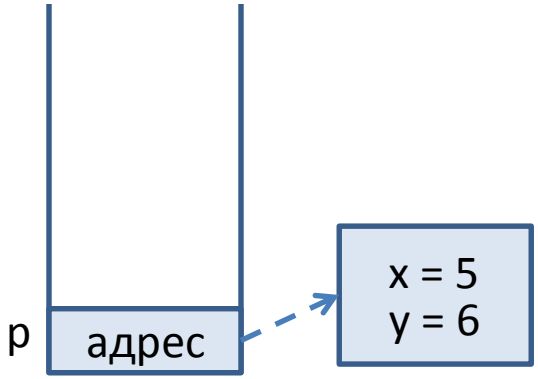

```
SPoint p1 = p;
```



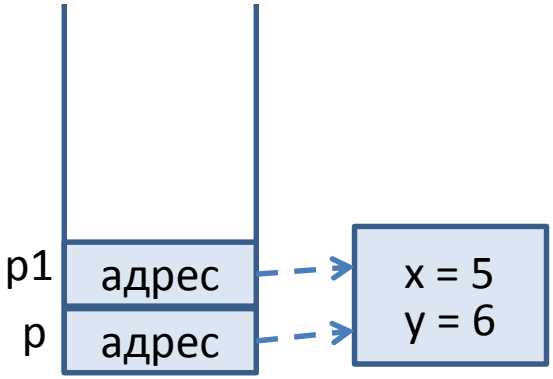

```
p.x = 10;
```

Различия между структурами и классами

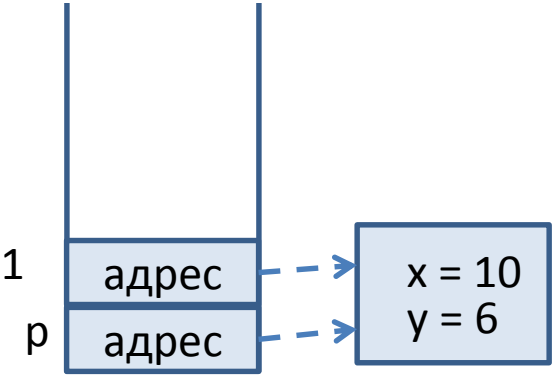
Класс




```
CPoint p = new CPoint();
p.x = 5; p.y = 6;
```




```
CPoint p1 = p;
```




```
p.x = 10;
```

Различия между структурами и классами

Возможность	Структура	Класс
Наличие потомков		
Виртуальные и абстрактные члены		
Явное задание предка		
Хранение пустого указателя (null)		
Переопределение конструктора по умолчанию		
Размещение в стеке		
Инициализация полей при инициализации объекта		
Обязательная инициализация всех полей в явно заданном конструкторе		

Структура Nullable<T>

Объявление:

```
Nullable<тип> имя = значение;
тип? имя = значение;
```

Пример:

```
Nullable<int> n = 20;
int? n = 20;
```

Получение значения:

```
int n1 = n.Value;
int n1 = (int)n;
```

Определение наличия значения:

```
if (n.HasValue) { }
if (n != null) { }
```

Получение значения с защитой от null:

```
int n1 = (n.HasValue) ? n.Value : 0;
int n1 = n ?? 0;
int n1 = n.GetValueOrDefault(0);
```

Структура Nullable<T>

- Позволяет хранить в себе значение выбранной основной структуры + специальное значение null;
- Основные свойства:
  - Value — значение основного типа. Если объект содержит null, при взятии Value срабатывает исключение;
  - HasValue — наличие значения. Возвращает false, если объект содержит null.

Автоматическая типизация

- Если объявляемая переменная *сразу* получает значение, указание её типа можно заменить ключевым словом **var**;
- Переменная получит тип присваиваемого ей объекта;
- Если тип результата определить невозможно, использование **var** недопустимо.

```
var i = 5;
```
struct System.Int32
Represents a 32-bit signed integer.

```
var s = "Hello";
```
class System.String
Represents text as a series of Unicode characters.

```
var a = new[] { 0, 1, 2 };
```
Int32[]

```
for (var x = 0; x < 10; x++)
{ }
```
struct System.Int32
Represents a 32-bit signed integer.

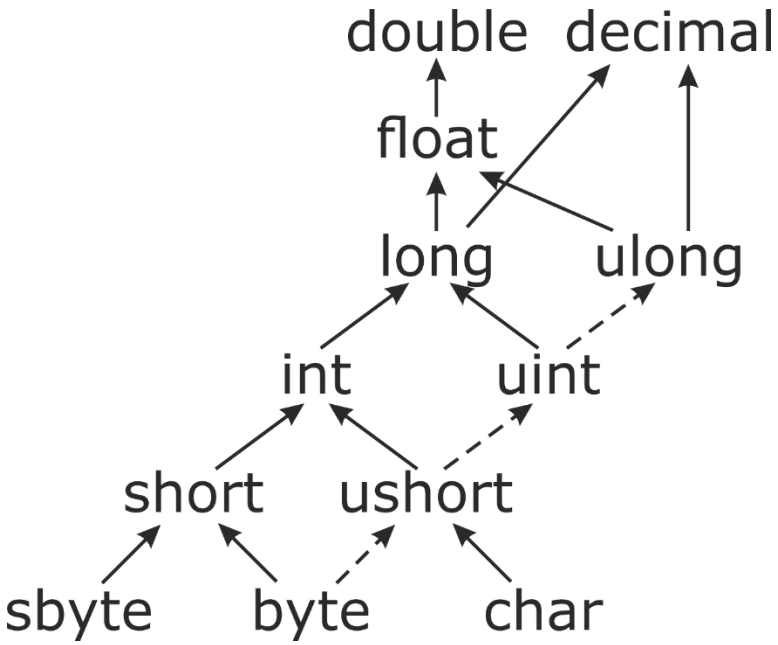
```
foreach (var x in a)
{ }
```
struct System.Int32
Represents a 32-bit signed integer.

http://msdn.microsoft.com/ru-ru/library/bb384061(v=vs.90).aspx

Динамическая типизация

- Динамический объект содержит конструкции, контролирующие обращение к его членам;
- Фактически в различные моменты времени объект может обладать произвольным набором членов;
- Синтаксически обращение к произвольным членам динамического объекта возможно при использовании специального типа dynamic.

Неявное приведение встроенных типов C#



предпочтительный путь

допустимый путь

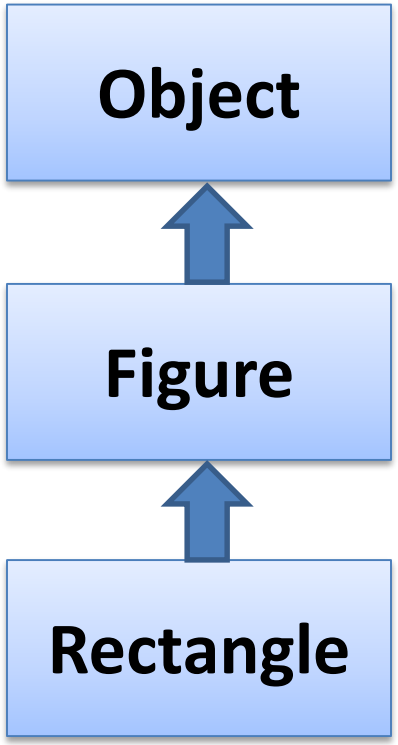
Явное приведение

```
int x = 2, y = 5;
double z = ((double)x) / y;
```

Приведение объектов классов

Неявное приведение

```
Figure f = new Rectangle();
object o = f;
```



Явное приведение

```
// Существует два способа, отличающиеся
// реакцией на ошибку приведения типов:

Figure f1 = (Figure)o;
// возникает InvalidCastException

Rectangle r = f as Rectangle;
// присваивается значение null
```

Проверка ссылочного типа типа

(тип) значение

```
if (f is Rectangle)
{
    r = (Rectangle)f;
    // использование r
}
```

значение as тип

```
r = f as Rectangle;
if (r != null)
{
    // использование r
}
```

Упаковка и распаковка

Упаковка / Boxing

```
int x = 5;
object obj = x;
Type t = obj.GetType();
if (obj.Equals(x)){ }
```

Распаковка / Unboxing

```
int y = (int)obj;
```

Действия при упаковке/распаковке

Упаковка

1. Выделяется память в управляемой куче
2. Поля значимого типа копируются в память
3. К объекту добавляется указатель на тип и SyncBlockIndex.
4. Возвращается адрес объекта

Распаковка

1. Возвращается указатель на упакованное значение.

Примечание: обычно распаковка происходит вместе с копированием

Операции

- Арифметические

  `+ - / * %`

- Сравнения

  `> < >= <= == !=`

- Логические

  `&& || ! & |`

- Битовые

  `& | ~ ^`

- Присваивания

  `= += -= /= *= %= ++ -- &= |= ...`

Вывод данных

Значение переменной

```
Console.WriteLine(x);
```

Ручное объединение строк (конкатенация)

```
Console.WriteLine("x = " + x + ", y = " + y);
```

Форматирование

```
Console.WriteLine("x = {0}, y = {1}", x, y);
```

Работа с консолью

- Консоль доступна только в проектах типа **Console Application**;
- Работа с консолью осуществляется с помощью класса System.Console;
- Основные методы для работы:
  - ReadKey — считывает одну клавишу, нажатую пользователем;
  - ReadLine — считывает строку, введённую пользователем;
  - Write — выводит на экран значение переменной или результат выражения;
  - WriteLine — выводит на экран значение переменной или результат выражения + символ новой строки.

Управляющие символы

Вид	Наименование
\a	Звуковой сигнал
\b	Возврат на шаг назад
\f	Перевод страницы
\n	**Перевод строки**
\r	Возврат каретки
\t	**Горизонтальная табуляция**
\v	Вертикальная табуляция
****	**Обратная косая черта**
\'	**Апостроф**
\"	**Кавычки**

Пример настройки форматирования даты

```
DateTime dt = new DateTime(2008, 3, 9, 16, 5, 7, 123);
Console.WriteLine("{0:\\y-\t y yy yyy yyyy}", dt);
Console.WriteLine("{0:\\M-\t M MM MMM MMMM}", dt);
Console.WriteLine("{0:\\d-\t d dd ddd dddd}", dt);
Console.WriteLine("{0:\\h\\H-\t h hh H HH}", dt);
Console.WriteLine("{0:\\m-\t m mm}", dt);
Console.WriteLine("{0:\\s-\t s ss}", dt);
Console.WriteLine("{0:\\f-\t f ff fff ffff}", dt);
Console.WriteLine("{0:\\F-\t F FF FFF FFFF}", dt);
Console.WriteLine("{0:\\z-\t z zz zzz}", dt);
```

```
y-      8 08 2008 2008
M-      3 03 мар Март
d-      9 09 Вс воскресенье
hH-     4 04 16 16
m-      5 05
s-      7 07
f-      1 12 123 1230
F-      1 12 123 123
z-      +4 +04 +04:00
```

Настройки форматирования даты

Описатель формата	Описание
d	День
f	Доли секунд
F	Доли секунд (без нулей)
h	Часы в 12-часовом формате
H	Часы в 24-часовом формате
m	Минуты
M	Месяц
s	Секунды
y	Год
z	Смещение времени

Пример ввода данных

```
Console.Write("N = ");

string str = Console.ReadLine();
int n = int.Parse(str);

Console.WriteLine("Your input: {0}", n);
Console.ReadLine();
```

Ввод данных

- Ввод данных с клавиатуры возможен только в строковом формате;
- Для приведения строки к конкретному типу, как правило, используются методы Parse и TryParse результирующего типа, а также класс System.Convert;
- При ошибке преобразования типа может произойти исключительная ситуация, однако предусматривать её не обязательно.

Блок try ... catch ... finally

- Предназначен для перехвата исключительных ситуаций;
- Синтаксис:

```
try
{
        действия, способные привести к исключительной
}
catch [(тип [переменная])]
{
        обработка исключения
}
finally
{
        действия, выполняемые в любом случае
}
```

Блоки catch и finally являются необязательными и могут быть опущены, но не одновременно.

Обработка неверного ввода перехватом исключения

```
string str = Console.ReadLine();

try
{
    int n = int.Parse(str);
}
catch (FormatException)
{
    Console.WriteLine("Wrong input!");
    return;
}
```

Ресурсы для начинающего разработчика .NET

- Visual Studio Community: http://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx
- MSDN: http://msdn.microsoft.com/ru-ru/library
- StackOverflow: http://stackoverflow.com/
- Хабрахабр: http://habrahabr.ru/
- Google: https://google.ru/

Литература для начинающего разработчика .NET

- Джеффри Рихтер — CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#
- Эндрю Троелсен — Язык программирования C# 5.0 и платформа .NET 4.5
- Андерс Хейлсберг, Мэдс Торгерсен, Скотт Вилтамут, Питер Голд — Язык программирования C#
- Павел Агуров — C#. Сборник рецептов

- Роберт Мартин, Мика Мартин — Принципы, паттерны и методики гибкой разработки на языке C#
- Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес — Приёмы объектно-ориентированного проектирования. Паттерны проектирования

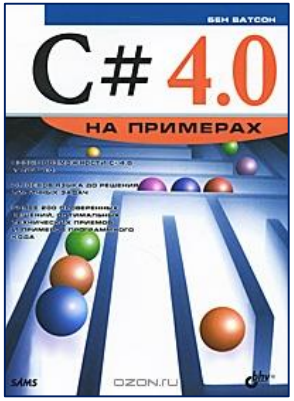
Литература

Джеффри Рихтер

CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#

Павел Агуров

C#. Сборник рецептов (+CD-ROM)



Бен Ватсон

C# 4.0 на примерах

Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер

C# 4.0 и платформа .NET 4 для профессионалов (+ CD-ROM)

Спасибо за внимание!

Контактная информация:
Дмитрий Вересkун
Инструктор
EPAM Systems, Inc.
Адрес: Саратов, Рахова, 181
Email: Dmitry_Vereskun@epam.com
http://www.epam.com