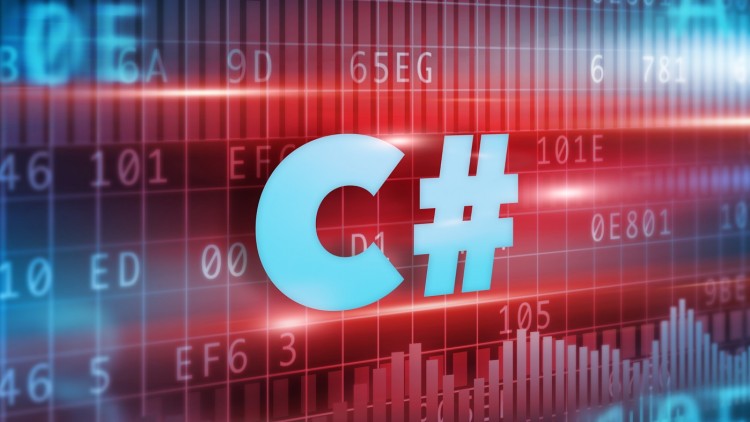
# C# и Microsoft .NET

**Wi-Fi: Knowledge1234**

+359877347912 Георги Панаьотов

# C# и Microsoft .NET

[Home](https://pragmatic.bg) C# и Microsoft .NET



### C# и Microsoft .NET

[Програмиране](https://pragmatic.bg/coursecategory/%d0%bf%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%b8%d1%80%d0%b0%d0%bd%d0%b5/)

16 Лекции

48 часа 0 минути

[650лв. - Записване за курса](https://pragmatic.bg/sign-for-a-course/)

**Обект на курса:**

Ц елта на курса е да запознае аудиторията с Microsoft Visual Studio 2013 и да даде основни познания по езика C#. В рамките на курса се представят добри практики за планиране и реализация на софтуерни системи.

Разглежданите теми покриват основите на езика C#, като се засягат и елементи на визуалното програмиране за Desktop.

**За кого е предназначен**

Курсът е предназначен за хора с нулеви или минимални познания в сферата на софтуерната разработка или работещи софтуерни специалисти, които тепърва се запознават със C# и .NET Framework.  
Въпреки че курса се преподава на български език, е добре курсистите да имат добро ниво на английски. Това в последствие ще им помогне при търсене на работа в сферата.

**Продължителност на курса**

Курса е вместен в 16 лекции, всяка с продължителност 3 часа. Занятията се провеждат два пъти седмично, като общата продължителност на курса е 8 седмици или 2 месеца.  
Занятията ще са смесени (лекции и упражнения) поради практическата насоченост на материала.  
В края на курса ще бъде проведен и тест върху материала на целия курс.

**Практическа част по време на занятия**

За упражненията и практическата част по време на занятията всеки курсист трябва да носи личен лаптоп.

**Цена на курса**

Цената за цялата продължителност на курса е 650 лева. Като предоставяме възможност за разсрочено плащане – 2 вноски по 325 лева в началото на всеки от месеците от курса.

**В кои дни се провежда курса, кога ще започнат нови групи и как може да се запиша?**

Разписанието се публикува в [График](http://pragmatic.bg/schedule), а за записване, моля използвайте [Записване за курс](http://pragmatic.bg/sign-for-a-course/).

## [Обзор на .NET Framework](https://pragmatic.bg/lessons/%d0%be%d0%b1%d0%b7%d0%be%d1%80-%d0%bd%d0%b0-net-framework/)

3 часа 0 минути

Какво е .NET Framework  
Управляван и неуправляван код  
MSIL, CLS, CTS, CLI, CLR и BCL

## [Работа с типове, променливи, оператори и изрази](https://pragmatic.bg/lessons/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d1%82%d0%b8%d0%bf%d0%be%d0%b2%d0%b5-%d0%bf%d1%80%d0%be%d0%bc%d0%b5%d0%bd%d0%bb%d0%b8%d0%b2%d0%b8-%d0%be%d0%bf%d0%b5%d1%80%d0%b0%d1%82%d0%be%d1%80%d0%b8/)

3 часа 0 минути

Типове данни  
Променливи и съхранението им в паметта  
Оператори в C# и приоритет на оператори  
Преобразуване на типове  
Изброени типове

## [Условни конструкции, цикли, масиви, въведение в алгоритмите](https://pragmatic.bg/lessons/%d1%83%d1%81%d0%bb%d0%be%d0%b2%d0%bd%d0%b8-%d0%ba%d0%be%d0%bd%d1%81%d1%82%d1%80%d1%83%d0%ba%d1%86%d0%b8%d0%b8-%d1%86%d0%b8%d0%ba%d0%bb%d0%b8-%d0%bc%d0%b0%d1%81%d0%b8%d0%b2%d0%b8-%d0%b2%d1%8a%d0%b2/)

3 часа 0 минути

if и switch/case  
for, foreach, while и do/while  
Масиви и колекции в .NET  
Методи, итеративни и рекурсивни алгоритми

## [Обектно-ориентирано програмиране, класове, обекти, полета, методи, свойства, конструктори](https://pragmatic.bg/lessons/%d0%be%d0%b1%d0%b5%d0%ba%d1%82%d0%bd%d0%be-%d0%be%d1%80%d0%b8%d0%b5%d0%bd%d1%82%d0%b8%d1%80%d0%b0%d0%bd%d0%be-%d0%bf%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%b8%d1%80%d0%b0%d0%bd%d0%b5-%d0%ba%d0%bb/)

3 часа 0 минути

Пространства от имена и библиотеки от класове в .NET  
Класове и инстанции на класове  
Конструктори  
Полета, методи и свойства на клас  
Статични ресурси на клас  
Етикети за достъп

## [Интерфейси, абстрактни класове, шаблонни класове (Generics)](https://pragmatic.bg/lessons/%d0%b8%d0%bd%d1%82%d0%b5%d1%80%d1%84%d0%b5%d0%b9%d1%81%d0%b8-%d0%b0%d0%b1%d1%81%d1%82%d1%80%d0%b0%d0%ba%d1%82%d0%bd%d0%b8-%d0%ba%d0%bb%d0%b0%d1%81%d0%be%d0%b2%d0%b5-%d1%88%d0%b0%d0%b1%d0%bb%d0%be/)

3 часа 0 минути

Разлика между интерфейс и абстрактен клас  
Шаблонни класове  
Шаблонизиране на интерфейси и абстрактни класове

## [Наследяване](https://pragmatic.bg/lessons/%d0%bd%d0%b0%d1%81%d0%bb%d0%b5%d0%b4%d1%8f%d0%b2%d0%b0%d0%bd%d0%b5/)

3 часа 0 минути

Разлика между наследяване на клас и имплементиране на интерфейс  
Protected полета и методи  
Проектиране на йерархии от класове  
Обръщение към скрити методи на базовия клас

## [Полиморфизъм](https://pragmatic.bg/lessons/%d0%bf%d0%be%d0%bb%d0%b8%d0%bc%d0%be%d1%80%d1%84%d0%b8%d0%b7%d1%8a%d0%bc/)

3 часа 0 минути

Виртуални методи и свойства  
Абстрактни методи и свойства  
Интерфейси и полиморфизъм

## [Управление на изключенията](https://pragmatic.bg/lessons/%d1%83%d0%bf%d1%80%d0%b0%d0%b2%d0%bb%d0%b5%d0%bd%d0%b8%d0%b5-%d0%bd%d0%b0-%d0%b8%d0%b7%d0%ba%d0%bb%d1%8e%d1%87%d0%b5%d0%bd%d0%b8%d1%8f%d1%82%d0%b0/)

3 часа 0 минути

Какво е изключение  
Хвърляне на изключения  
Йерархия от изключения  
Начин на работа на try/catch/finally и using  
Особености на класа Exception

## [Работа с колекции](https://pragmatic.bg/lessons/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d0%ba%d0%be%d0%bb%d0%b5%d0%ba%d1%86%d0%b8%d0%b8/)

3 часа 0 минути

Нетипизирани колекции  
Типизирани колекции  
Хеш таблици  
Речници  
Сортирани колекции

## [Работа с низове, конвертиране на данни от и до низ](https://pragmatic.bg/lessons/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d0%bd%d0%b8%d0%b7%d0%be%d0%b2%d0%b5-%d0%ba%d0%be%d0%bd%d0%b2%d0%b5%d1%80%d1%82%d0%b8%d1%80%d0%b0%d0%bd%d0%b5-%d0%bd%d0%b0-%d0%b4%d0%b0%d0%bd%d0%bd%d0%b8/)

3 часа 0 минути

Класът String и вградена функционалност  
Escaping в символни низове  
Класът StringBuilder  
Преобразуване от низ и до низ  
Класът CultureInfo и формати при преобразуване от и до низ

## [Работа с файлове и потоци](https://pragmatic.bg/lessons/%d1%80%d0%b0%d0%b1%d0%be%d1%82%d0%b0-%d1%81-%d1%84%d0%b0%d0%b9%d0%bb%d0%be%d0%b2%d0%b5-%d0%b8-%d0%bf%d0%be%d1%82%d0%be%d1%86%d0%b8/)

3 часа 0 минути

Какво е поток  
Класът FileStream  
Класовете StreamReader и StreamWriter

## [Въведение в структурите от данни](https://pragmatic.bg/lessons/%d0%b2%d1%8a%d0%b2%d0%b5%d0%b4%d0%b5%d0%bd%d0%b8%d0%b5-%d0%b2-%d1%81%d1%82%d1%80%d1%83%d0%ba%d1%82%d1%83%d1%80%d0%b8%d1%82%d0%b5-%d0%be%d1%82-%d0%b4%d0%b0%d0%bd%d0%bd%d0%b8/)

3 часа 0 минути

Списък, двусвързан списък, стек и опашка  
Дърво и двоично дърво за търсене

## [Визуално програмиране](https://pragmatic.bg/lessons/%d0%b2%d0%b8%d0%b7%d1%83%d0%b0%d0%bb%d0%bd%d0%be-%d0%bf%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%b8%d1%80%d0%b0%d0%bd%d0%b5/)

3 часа 0 минути

Въведение в Microsoft Visual Studio 2012 Express  
Класът MessageBox  
Запознаване с основните визуални контроли  
Прихващане на събития на визуални контроли

## [Нишки и многонишково програмиране](https://pragmatic.bg/lessons/%d0%bd%d0%b8%d1%88%d0%ba%d0%b8-%d0%b8-%d0%bc%d0%bd%d0%be%d0%b3%d0%be%d0%bd%d0%b8%d1%88%d0%ba%d0%be%d0%b2%d0%be-%d0%bf%d1%80%d0%be%d0%b3%d1%80%d0%b0%d0%bc%d0%b8%d1%80%d0%b0%d0%bd%d0%b5/)

3 часа 0 минути

Делегати  
Класът Thread  
Синхронен и асинхронен начин на работа  
Блокиране и синхронизация на нишки

## [Ламбда изрази и LINQ](https://pragmatic.bg/lessons/%d0%bb%d0%b0%d0%bc%d0%b1%d0%b4%d0%b0-%d0%b8%d0%b7%d1%80%d0%b0%d0%b7%d0%b8-%d0%b8-linq/)

3 часа 0 минути

Анонимни методи и ламбда изрази  
Въведение в LINQ

## [Сериализация](https://pragmatic.bg/lessons/%d1%81%d0%b5%d1%80%d0%b8%d0%b0%d0%bb%d0%b8%d0%b7%d0%b0%d1%86%d0%b8%d1%8f/)

3 часа 0 минути

Какво е сериализация на данни  
XML сериализация  
Двоична сериализация

# 01.10.2017 ЛЕКЦИЯ №1: Веведение

* Лектор: Георги Панайотов
* Въведение в програмирането – Езика C# история
* Namespace, class, methods
* Main method
* IDEs: Visual Studio
* Главни VS 2017 Панели (Panеs): Solution Explorer, Properties, Output
* Интелисенс – подсказки за възможните комбинации
* Стейтмънти – командите на всеки ред
* Литерали –
* First program - Hello World
* Run program F5 или Ctrl + F5 (Debug)
* Достъп до сайта – learn.pragmatics.bg

# 03.10.2017 ЛЕКЦИЯ №2

* Да се помисли за домашно – Относно програма, която бихме искали да реализираме
* Plug-in лесно намиране на XPaths
* Смяна на темата на VS 2017: Search -> Themes
* Синхронизация на проектите, които пише в GIT?
* Namespace – обединява
* Синтакс шугар ?!
* Debug-ване /Brake Points

What is bit: 0,1

What is byte: 8 bits

0000 0000

1111 1111

0101 1001

<<------------

0 1 0 1 1 0 0 1

2^8 2^7 2^6 2^5 2^4 2^3 2^2 2^1

2 bit-това система = 4 символа

4 bit-това система = 16 симова

т.е. 2 на степен (^) b bit-a

Kilo byte = 1024 bytes

Mega byte = 1024 Kbytes …

Giga byte = 1024 Mega byte;

What is Data Type?

What is a Variable?

**String name**; низ от символи

**name** = Console.Readline();

**name** = “Ivan”;

**int x** = 5; целочислени числа

**float pi** = 3.14; реални числа

**const float** pi = 3.14; слагането на const не позволява да се променя по нататък стойността

специфично за **C#** е използването на на дейта тайп **VAR** сам опеределя типа на **променливата**

добра практика е да се ползва **var**!

var sample = Calculate(20, 40, 60);

sample = “Joro”;

**парсване** – разчитане на стринг да се конвертира към смислен тип

**кастване** – конвертиране към друг тип

## UNICODE

<https://en.wikipedia.org/wiki/Unicode>

Coding Standards

Синтаксис на използваните ключови думи

Запазени символи

Имената на метходите и класовете започват с главна буква – Main(), Calculate() etc.

Имената на променливите започват с малка буква – name, firstName etc.

В различните езици типовете се различават като големина

## Build-In Data Types

bool 1 bit 0 or 1

byte

signed byte

short

ushort

int

uinit

lomg

ulong

правилото е да се избира тип, който е подходящ за целите ни т.е. да не се позлват големи типове с идеята да се оптимизира паметта

float се представя с приближение заради 0010

положителна и отрицателна безкрайност

NaN = not a number примери: 0 делено на 0 или корен квадратен на отрицателно число

Floating пойнт числата не са точни – не се изполват за смятане на пари

float f = 0.1f; // 32 bit

Console.WriteLine(f); // 0.1 (correct due to rounding)

double d = 0.1f; // 64 bit

Console.WriteLine(d); // 0.100000001490116 (incorrect)

decimal 128 bits – използва се пресмятане на пари

|| = or или

01 = 1

10 = 2

Десетични числа се сбират с +

Докато двойчни се съборат с OR и AND

ASCII char 1 bit

UNICODE char 16 bit - <http://www.unicode.org/charts/>

Character literal

Hexadecimal

Unicode

## Escaping symbols @ “ ”

\’

\”

\\ \ symbol in string literal

\b

\n = new line

\r = carriage return

\t = tab

## Enumeration - изброяване

Нормалните деца броят от 1, програмистите броят от 0

enum cartype

{

Sedan, 100

Wagoon, 50

Coope, 200

Hedgeback 300

}

int x;

cartype car;

## Object Type - тип обект – reference type

Обектите за разлика от обикновенните променливи може да се разширяват

Типове данни – **Value** и **Refence Type**;

Int a = 4;

Int b = a;

a = 7; (b = 4)

## Reference memory

Steak memory

Heap memory

## Type Casting

**Explicit – ние казвам, че разбираме и искаме това преобуразуване**

Double pi = 3.14;

Pi.ToString();

target\_type = (target\_type)original\_type

int x = (int)pi;

**Implicit** – сработва понеже double е по-голямо от float

float pi = 3.14;

Pi.ToString();

target\_type = (target\_type)original\_type

double d = pi;

**Convert**

Int.Covert()

**Parse**

Int.Parse()

Operators in C#

Arithmetic

+, -. \*, %

Logical

&&, ||, !

String

+

Comparison

==, !=, >, < , >=, <=

Assignment

=, +=, -=, etc

&& логиеско И

|  |  |  |
| --- | --- | --- |
| Т | Т | Т |
| Т | F | F |
| F | T | F |
| F | F | F |

|| логиеско ИЛИ

|  |  |  |
| --- | --- | --- |
| Т | Т | Т |
| Т | F | Т |
| F | T | Т |
| F | F | F |

Sum = sum + productPrice;

Sum += productPrice;

Една и съща операция

## Домашна работа – да се разгледа класа Math

Формулите от слайда да се представят с математичните операции от библиотеката.

# 08.10.2018 ЛЕКЦИЯ №3: Strings – Низ от символи

## Debugging Tricks

F10 – Step over

F11 – Step Into

Debug - Using Conditions

Debug – Immediate Window / **C# Interactive** – пишеш C# код и виждаш веднага резултата

С Конзолните приложения също можем да създаваме на POC (доказателства), че дадено решение работи или не.

i = 0

i++ => i = 0, Понеже i++ първо печата резултата, а след това извършва операцията

++I => i = 2, Понеже ++I първо извършва операцията, а след това печата резултата

Snnipets ?!

## ENUMBERATATION

Изброяване enum – дефинира се извън Мейн метода

Винаги задавайте стойности на елементите на енумерацията, с цел да можем на по късен етап да можем да добавяме нови стойности

enum GenderType

{

Unknown = 0, // Default one – стойност по подразбиране

Male = 100,

Female = 200

}

GenderType gender = GenderType.Unknown;

Int age 25;

Bool isAdult = (age >= 18 && gender == GenderType) || (age >= 21 && gender = GenderType.Male)

Console.WriteLine(isAdult);

Console.Readkey();

&& И

|| ИЛИ

## STRUCTURES

Структурите са Value типове

Public struct type\_name

{

Public Type Filed;

\_x;

\_isAdult;

}

public struct Point

{

public float X;

public float Y;

}

public enum Country

{

Bulgaria = 0,

Greece = 1,

Macedonia = 2,

Turkey = 3

}

public struct Address

{

public string AddressLine1;

public string AddressLine2;

public string City;

public Country Country;

}

class Program

{

static void Main(string[] args)

{

Address customerBillingAddress = new Address()

{

AddressLine1 = "Mladost 2",

AddressLine2 = "Drujba",

City = "Sofia",

Country = Country.Bulgaria

};

Console.WriteLine(customerBillingAddress);

Console.ReadKey();

}

}

## STRING

Стойността по подразбиране на Value Type е 0 (нула)

String Не е Value type, а е Reference type

Стойността на подразбиране на String е Null

Разликата между Null и 0, e че 0 е стойност, докато Null e празно.

### String Data Type

**-> Value Type / Not a reference type.**

Represent a set of characters (aka text 😊)

Strings encoding is Unicode (UTF 16)

String are **immutable** – **не променяеми**

String **Automatic Type Conversion**

String msg = “I’m” + 25 + “ years old”

**String.Concat** static method – конкатениране на стрингове

**StringBuilder** – вече не immutable, перформанс ефективна операция

Append, Remove, Insert, Replace

ToString()

### Comparing Strings

== -> сравнява дали съдържанието на стринговете е еднакво

String.Equals();

String.Compare();

### Searching in Strings

Contains method

StartsWith

EndsWith

IndexOf

LastIndexOf

### Changing Strings

Substring()

Insert()

Remove()

Replace()

ToLower()

ToUpper()

Trimming strings

Trim()

TrimEnd()

TrimStart()

### Operations

Remove

Replace

Join

Split

### Formatting

String.Format(…)

Custom String Formatting

String Interpolation

### Globalization

GetCulture()

Най-честия проблем, понеже при пограмиста средата е с разделител точка, докато при клиента е запетайка или според локала на клиента.

10 Invariant Culture

REG EX – Регулар Експрешън

**Coding Styles – различни Кодинг Стандарти за всяка компания**

# 10.10.2018 - ЛЕКЦИЯ №4

Книжка на **Microsoft – Software Development Fundamentals**

**Globalization** – Приложението да се държи адекватно спрямо дръжавата на използване

**Localization** – Превод на ресурсите, предимно езика.

За разлика от **Localization, Globalization** е повече за форматиране на числа и дати – точки, запетайки, преобразуване между различни часови зони.

Базовите типове имат **Parse** и **TryParse**

По-добре да се ползва **TryParse** защото се оптива да го ковертира и връща **Conditional Statement**

**Immutable** – стринга не се променя т.е. след промяна, създава нов стринг, а не променя текущия

Разлика между **Structure** и **Class** не е голяма, главно се изразява в това, че **Structure е value type**, докато **Class е reference type**.

static void Main(string[] args)

{

// Локален вараябъл

}

Изпозване на снипети / snippets <https://en.wikipedia.org/wiki/Snippet_(programming)>

Не е добра практика да се ползва **Copy/Past**

По добре да се изпозлва групово писане **ALT + SHIFT** и селектиране на вертикална колна

**Coding Standards** – добри практики установени в екипа, като да не се ползва копиране на код, стандарт на имената на променливите, писане на коментари и документация.

Object and Disposable Object

В C# няма процедури и фукнции, а има процедури, понеже е **процедурен език**.

**Sonar Cube** – Статичен и Динамичен анализ на кода

**Binary** – bin\debug\Program.exe

Проверява за Memory Leaks, SQL Injections,

Sonar Cube – хвръля греша, ако в методите се използват параметри със стойности по подразбиране

string[] – масив от стрингове

**C# ReSharper** - Помага при писането на код

**PDF Standard Format** - Format specifier

Дефиниция на метода, който сме извикали **CTR + SHFT + SPACE**

**Trim** маха шпации (space) и табулации (tab) // white space

**PAYLOAD**

* XML
* JSON
* YAMEL

## Condition Statements

### If Then Else

NaN – Not An Number

Value types нямат

double totalPrice = 0;

double price = -10;

if (price >= 0)

{

totalPrice += price;

}

else

{

totalPrice = double.NaN;

}

totalPrice = price >= 0 ? totalPrice + price : double.NaN;

### Switch

switch (gender)

{

case Gender.Unknown:

{

}

break;

case Gender.Male:

case Gender.Female:

{

}

break;

default:

break;

}

## Цикли - Loops

### While

### Do / While

### For

### Nested Loops

Нестване – влагане на цикли

For()

{

For()

}

### Foreach

Foreach (type variable\_name in collection)

{

}

## Масиви – Arrays

### Методи – Methods

### Рекурсии

### Да се изпрати каква задача бих искал да направя като проект

PDF – материали от предния път

# 15.10.2018 – ЛЕКЦИЯ №5

## Method

The void return type

The out and ref parameters

public method\_name(arguments)

{

}

## Recursion

Пример за Рекурсия, чрез пресмятане на Факториел

<https://en.wikipedia.org/wiki/Factorial>

public static int Factoriel(int n)

{

if (n == 0)

{

return 1;

}

return (n \* Factoriel(n - 1));

}

Използване на

break – прекъсва цикъла

continue – продължава (продължава напред) със следващото условие, но не излиза от цикъла

## Algorithm

## Data Structure

## Arrays

Какво е масив?

Инициализация?

## Multidimensional Arrays

Two Dimensional

Three Dimensional

2x2, 3x3, 4x4

## Jagged Arrays

Arrays that have arrays in them

Defining a jagged array

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/jagged-arrays>

## Array List

* Add
* Insert
* Remove

Boxing

Unboxing

## Stack

LIFO – Last In First Out

* Push (…)
* Peek (…)
* Pop (…)
* Count

## Queue

FIFO – First In First Out

* Enqueue (…)
* Dequeue (…)
* Count

## Hash table

Hash? – Нетипизирано Dictionary

Keys and values

Elements are Objects

* Add (…)
* Remove (…)
* Dequeue (…)
* Count

Hashtable users = new Hashtable();

users.Add(“smg”, userStruct);

users[“smg”]

## Books

[The Art of Computer Programming, Volumes 1-4 (Box Set) Hardcover – 3. March 2011](https://www.amazon.de/Art-Computer-Programming-Volumes-1-4/dp/0321751043/ref=sr_1_1/262-9394147-2671127?ie=UTF8&qid=1539629087&sr=8-1&keywords=the+art+of+computer+programming)

[Mta Software Development Fundamentals (Microsoft Official Academic Course)](https://www.amazon.de/Software-Development-Fundamentals-Microsoft-Official/dp/047088911X/ref=sr_1_1?s=software&ie=UTF8&qid=1539630055&sr=8-1&keywords=microsoft+software+development+fundamentals)

[Design Patterns C#](https://www.amazon.de/Design-Patterns-Hands-Real-World-Examples/dp/1484236394/ref=sr_1_1?s=books-intl-de&ie=UTF8&qid=1539792498&sr=1-1&keywords=Design+Patterns+C%23)

Services Fundamentals?!

# 17.10.2018 – ЛЕКЦИЯ №6

Draw Board

Quit

Init Board

CheckMate

QCastling

KCastling

Undo

Read Input

Take Figure

Figures

Players

Actions/ Moves

**Github.com/smg-bg/pragmatic**

## [KISS principle](https://en.wikipedia.org/wiki/KISS_principle)

## [Полиморфизъм (информатика)](https://bg.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D1%8A%D0%BC_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0))

# 22.10.2018 – ЛЕКЦИЯ №7

# 24.10.2018 – ЛЕКЦИЯ №8

## ООП – обектно ориентирано програмиране

### Основни принципи

#### Абстрашън

#### Енкапсолейшън

#### Инхеритънс

#### Полиморфизъм

### Класове и Обекти

#### Какво е клас?

#### Какво е обект?

Класовете са референс типове

Променливата сочи към инстанцията

т.е. ако имаме 2 променливи сочещи към една инстанция, която и да променим ще променим обекта в паметта.

### Как да дефинираме клас?

Наименоват се с главна буква. Не се изполва \_, а само редуване на ГолемиМалки букви – CarInsurance

Всичко в C# е клас – enumeration, structure etc. Class = Custom Type

## Access Modifiers

access\_modifier **class** class\_name

{

// Fields and constants

// Properties

// Constructor

// Methods

}

### Type scoped

#### Public – Външни

#### **Internal – Вътрешни – По подразниране**

Абстракцията е много добър начин да се бори комплексити

### Member Scoped

#### Public

#### **Private – По подразниране**

#### Internal

#### Protected

#### Private protected

#### Internal Protected

## Class Anatomy

#### Nested Types - Class in Class in Class

#### Constants and Read only fields

#### Fields

#### Properties

#### Indexers

#### Events

#### Constructors

#### Methods

#### Operators

## Class

### Fields

### Properties

### Constructors

### Methods

Constructors

Overloading – повече методи с едно и същи име, но с различни параметри. Различно е от Overriding.

Default constructor – този, който не прима параметри

Methods

Static

Namespace

Class Library

С Голяма буква се пишат филдовете, които са публик

Методите в C# започват с голяма буква

# 29.10.2018 – ЛЕКЦИЯ №9

Решаване на домашното

$ Интерполейшън

Използвайте само конструнтори, които ви трябват и искате да задължите потребителя да въведе

Създаването на масив от обекти, от даден класов заделя

**SOLID -** [**https://nitinbhojwani-tech-talk.blogspot.com/2018/04/solid-design-principle.html**](https://nitinbhojwani-tech-talk.blogspot.com/2018/04/solid-design-principle.html)

S – Single Responsibility

O – Open-Closed Principle

L – Liskov Substitution Principle

I –

D –

**KISS -** [**https://en.wikipedia.org/wiki/KISS\_principle**](https://en.wikipedia.org/wiki/KISS_principle)

K – Keep

I – It

S – Simple

S – and Stupid

**Design Patterns**

**Добри практики:**

1. Класовете не трябва да са прекалено дълги
2. Всеки клас трябва да е в отделен файл
3. Методи над 150 реда не трябва да има
4. Един метод не трябва да е по-дълъг от един екран

Ползване на **Sonar Cube** – за тестови анализ

Отделните класове трябва да правят само едно единствено нещо

**SOLID - <https://nitinbhojwani-tech-talk.blogspot.com/2018/04/solid-design-principle.html>**

**S**

**SRP: The Single Responsibility Principle**

THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A

CLASS TO CHANGE.

**O**

**OCP: Open-Closed Principle**

SOFTWARE ENTITIES (CLASSES, MODULES, FUNCTIONS, ETC.)

SHOULD BE OPEN FOR EXTENSION, BUT CLOSED FOR

MODIFICATION

**L**

**LSP: Liskov Substitution Principle**

FUNCTIONS THAT USE POINTERS OR REFERENCES TO BASE

CLASSES MUST BE ABLE TO USE OBJECTS OF DERIVED CLASSES

WITHOUT KNOWING IT.

**I**

**ISP: Interface Segregation Principle**

CLIENTS SHOULD NOT BE FORCED TO DEPEND UPON INTERFACES

THAT THEY DO NOT USE.

**D**

**DIP: Dependency Inversion Principle**

A. HIGH LEVEL MODULES SHOULD NOT DEPEND UPON LOW

LEVEL MODULES. BOTH SHOULD DEPEND UPON ABSTRACTIONS.

B. ABSTRACTIONS SHOULD NOT DEPEND UPON DETAILS. DETAILS

SHOULD DEPEND UPON ABSTRACTIONS.

# 31.10.2018 – ЛЕКЦИЯ #10

**Class** - Класа е Шаблон

**Object** - Обекта е Екземпляр направен по Шаблона

**STATIC keyword** – методи, фийлодове и пропъртите, който са общи за всички или **методи, който не зависят от инстанцията**

Примери за **статични методи** и **класове**

* Метод: **Console** e static
* Класа **Math** също е статик

**Non-static** = **Instance** Method

**Static class** не може да се инстанцира

**Singleton** – Използване само на една инстанция

**Конструктура** е Метод със същото име като Класа

**Конструктируте** инициализрат статуса на нашия обект

Колкото **обекти** имаме, толкова пъти ще се извика **инстантния констуркор**

Ако **констрктура е статичен**, тогава той се изивиква само веднъж

**Read Only** може да се променя само през конструктора

**Current Directory** – там където се изпълнява изпълнява изпълнимия файл

Example: Environment.CurrentDirectory

Namespace – определя скоупа на проекта

Името на проекта съвпада с името на неймспейса / същото като package в JAVA

**Class Library** – DLL (Dynamic Link Library)

**Managed DLL** = **Assemblies**, такава библиотека, която може да се използва от .NET

Библиотеките от Windows System папката не могат да се използват в нашите проекти, защото не са писани на .NET

Build Configuration

Build = Compile операция

**BIN\DEBUG** -> **EXE** or **DLL** / **CONGIF** / **PDB** Файлове

**Name Spaces** – Using Namespace

**Reference Manager** – Add Global references or projects

**Abstractions** – Hide all but relevant details from end user

**Encapsulation** – Hide the internal details from end users

**Inheritance** – Reuse (share) logic and state from parent classes

**Polymorphism** – next time

**Inheritance**

**Class Shape** – Inheritance of shape Square, Rectangle, Triangle

Споделяне, чрез наследяване

**Налседяване** - даден клас да наследява базов клас

**Class Diagrams**

## Create UML Class Diagrams from Code

<https://msdn.microsoft.com/en-us/library/ff657806.aspx>

**The base class**

**Access modifiers** ползва се предимно **Protected**

Extend class :

Ultimate Base Class

Object.Equals(car1, car2);

Car1.Equals(car2)

Object

Bool Equals(obj1, obj2) /static/

Bool ReferenceEquals(objA, objB) /static/

Int GetHashCode() – Hash математически метод за ставяняване на данни

Type GetType

ToString

MemberwiseClone /protected/ клонира обекта

Interfaces vs Abstract Classes

Car

ICar

Един клас може да има/наследи само един базов клас, но може да имплементира много интерфейси.

Интерфейсите нямат стейт и конструктури, защото само само рамки

Проект – Домашна Работа

eWallet – Програма за отчет на финансите

или програмка за електронни бележки тип

Дизайн на обектна ирархия

# 05.11.2018 – ЛЕКЦИЯ №11 - Полиморфизъм

## Abstract classes

Public class Human:

{

Public void Speak(string msg);

}

Human h = new Human();

h.BreastEating();

h.Speak(“Wow”);

**Когато не искаме потребителите да вдигат инстанция на даден клас, тогава го правим Абстрактен.**

Когато класа е абстрактен, не можем да го инстанцираме, можем само да го наследяваме.

Mamal m = new Mamal();

Access\_modifier abstract class class\_name {…}

Public abstract void DoWork()

{

}

Не можем да имаме абстрактни фийлдове и константи, понеже те приемат стойности.

## Interfaces

**В интерфейсите не може да имаме конструктори**

Public interface IMortable

{

}

Интерфейсите се използват, когато искаме да имплементираме общо поведение

**Разликата между абстрактен клас и интерфейс**

1. Абстрактни класове: Има само единично наследяване и стейт (фийлд)

2. Интерфейси – Множествено наследяване и бехейвиър (методи)

\*Пропъртита могат да бъдат и стейт и бехейвиър

## Inheritance

Test : BaseTest();

## Polymorphism

Полимофизъм – достъпване на стейта на даден клас през неговия родител

abstract class Mamal

{

public **virtual** bool Eat(Food f)

{

If(f == null)

return false;

}

}

class Human : Mamal

{

public **override/new** bool Eat {}

base.Eat();

}

Human h = new Human();

Mamal m = new Human();

m.Eat(); Ще извика Eat() от Human

m.Speak(); Ще върне грешка, понеже няма декалрация на Speak в класа Human()

## Virtual Methods

Overridden methods

New methods

Abstract vs Virtual

## Type Casting

Type casting and polymorphism

Имплицитно

Float f = 7f;

Double d = f;

Екплицитно – със съгласие

Double d = 7

Float f = (float)(d);

Figure f = new Pawn();

IMortable m = new ;

new Die();

Интерфейсите не се наследяват, те се имплементират

Може да се наследява само един клас

И да се имплементират множество интерфейси

Base\_Class – базовия клас е наследник на клас родител

Base\_Class var\_name = new Super\_Class();

((Super\_Class)var\_name).Super\_Method();

В C# всичко е клас т.е. на всичко родителя му Object

Тези, които са value type, наследяват също в основата си Object.

Полиморфизма се използва подобно на кондъпънъл стейтмънт т.е. вместо да се изпозлват IF или CASE

Всички наследнитеци са длъжни да имат същия конструктур с или без параметри като родителя / базовия клас.

Protected – само наследниците имат достъп до него

С abstract защитаваме достъпа до базовия клас

В интерфейсите не можем да имем стейт / field

### Early and Late Binding

<https://www.geeksforgeeks.org/early-binding-late-binding-c/>

## Class Diagram

Install Class Designer

Add new Class Diagram in project

Избират се всички класове и се драгрват в Class Diagram

# 07.11.2018 – ЛЕКЦИЯ №12 - Ексепшъни

## What is an exceptions?

**Errors** – Compile Time Errors

**Exceptions** – Run Time Errors

## Why handle Exceptions

Не прихващата грешка, която не знаете как да обработите.

Освен ако не ползваме Wrapping Exceptions т.е. ще съдържа грешката и ще я обедини с информация от главната програма.

## Advanced exception handling

Exceptions and Exception Handling (C# Programming Guide)

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/>

## Exceptions List

Exception също е клас

Наследници на този клас са:

Преди релийз проверка в кода за:

//TODO: [MAK] Task …

//NotImplementedException

## Handling Exceptions

**Try** section

**Catch** section – Multiple catch sections

**Throw** keyword – Re-throwing exceptions

**Finally,** section

Не трябва:

Да се хваща родителя на всички Exceptions

Да има празен catch

Real life examples

1. Do I throw expectations or do I return false

Този метод НЕ очаква да фейлне. По-бърз метод

Int int.Parse (String str)

{

Throw new SystemException.ParseExeption();

}

Този метод очаква да фейлне. По-бавен метод

Bool int.TryParse(String str, out int Result )

{

}

1. Base Class Library = .NET Standard

## Exception Details

Exception Message

Stack Trace

Inner Exception

Building custom exceptions

## Example

VS Setting: Project Properties – Threat warnings as errors

Unhandled Exception - System Format Exception

Възникле ли ексепшън, изпълнението на програма спира

Изпълението на Try изисква поне един Catch или Finally

Java Super Class

C# Derived Class

Object -> Exception -> System Exception -> Derived -> System Format Exception

На крайния потребител не трябва да се показва exceptions, поради security причини. Добрата практика е да му се покзват говорещи съобщения.

Exception -> System Exception -> System Format Exception

Exception -> System Exception -> Argument Exception -> Argument Null Exception

Ексепшъните се редят да се хващат от най-конкретния до най-глобания т.е. отзад на напред

Само един експешън може да е активен и може да бъде хвърлен в конкретен случай

Т.е. всеки път хвърля по 1 ексепшън, защото стигнели се до първия троу, то тогава програмата спри и връща метода нагоре – стак трейс

CLR – Common Language Runtime кетчва system exceptions

Finally – освобожда заетия обект

Finally блока винаги минава без значение дали сме минали през catch блок

# 12.11.2018 – ЛЕКЦИЯ №13 Exceptions

## Exceptions and Examples

* Handling
* Throwing
* Wrapping
* Debugging
* Custom Exceptions

## Debugging Exceptions

Debug Conditions

Панел Exception Settings

Break When Throws – Format Exceptions

## Custom Exceptions

Създаваме си нов клас

Пример – Car Garage

Коли – 3 вида, според двигателя

* Бензинови
* Дизелови
* Електрически

Колите **имат** **двигатели** т.е. по общото, което имат се използава клас, по които да опише сама себе си.

1 файл = 1 клас

Клас CarGarage

TakeNewCar

RepairCar

ReturnCar

Клас Car

ChangeOil

ChangeTires

В ООП вместо енумерации и структури се ползват класове за да се опишат свойствата на обекта.

Един клас не може да бъде протектед, а само публик и интернал.

Абстрактен Клас Engine – за да не може да се интанцира. Ще инстанцираме наследниците. Видовете дригатели

Наследници

Клас ElectricEngine

Клас InternalCombistionEngine

Наследници

Клас PetrolEngine

Клас DieselEngine

Unsafe Casting

Var specific = (Type)Engine

Safe Casting

Var specific = Engine as Type

Кастването е много удобно, но може да доведе до грешки, за това се ползва Generics.

Инхеритънс (Inheritance) – наследяване изцяло поведението и стейта от даден клас

Композиция (Associations)– обедияняване на поведението и стейта от други класове

# 12.11.2018 – ЛЕКЦИЯ №13 Generics

Параметризиране на класовете, без значение дали подаваме кола или друг обект.

## Generics Declaration

Methods

return\_type MethodName<T> (T items) [where T: class | struct | type\_name | etc]

{

// Do something with item

}

T – означение за Generics

I - означение за Interface

Tuple и Caple

Classes and Interfaces

Interface | class TypeName<T> [where T: class | struct | type\_name | etc]

{

T \_item

}

## .NET Framework Generic Types

System.Collections.Generic namespace

* List<T>
* Dictionary<TKey, TValue>
* Stack<T>
* Queue<T>
* Etc.

System.Linq namespace

* Where<T>(…)
* Max<T>

Add Integers (int a, int b)

Add Bytes (byte a, byte b)

Add Double (double a, double b)

Add<T> (T a, T b)

Какъвто параметър му дадеш такъв тип ще се избере автоматично

T = Type или параметризиране на аргументите на методите.

# 12.11.2018 – ЛЕКЦИЯ №13 Extension Methods

Using Extension Methods

.NET Framework

System.Linq namespace

В .NET се използват много extensions methods т.е. колкото повече using extension\_methods ползваме, толкова повече възможности имаме.

С екстеншън методи се разширява стандартната фукционалност

# 14.11.2018 – ЛЕКЦИЯ №14 Delegates and Lambda expressions

## Last Time

Generics – Позвоялва ни да параметризираме типовете в класовете и интерфейсите.

* Methods
* Classes and Interfaces

Extension Methods – Позволяват ни да разширяваме функционалността даден клас или интерфейс без да променяме тяханта дефиниция

## Delegates

Делегата е тип, който представя референция към методите, чрез даден параметър или return type.

Declaring Delegates

access\_modifier delegate return\_type Delegate\_name(int a);

.NET Delegates

Func<>

Action<>

**Полза: Можем да го подаваме като параметър на метод (call back method)**

## Lambda Expressions

Анонимни функции и методи, който ни позволяват да ползваме делегати без да ги декларираме експлицитено.

Можеда да пишем локални функции, които да ги подаваме като агрументи.

Func <int, int, int> doWork = (a, b) => a \* b;

.NET Methods that work with Lambdas

* Where<>()
* FindAll<>()
* First<> or FirstOrDefault<>()
* Select<>()
* etc.

## Synchronous and asynchronous programming

Стандарното програмите, което до сега правим е синхронно т..е изпълняване програмата и всеки метод ичаква изпълнението на другите

А програмитенето, при което паралелно тече изпълнението на няколко процеса, все едно имаме няколко мейна в програмата, тогава имаме асинхронно програмитане и се ползват нишки за всеки процес.

<https://docs.microsoft.com/en-us/dotnet/csharp/async>

## Remote / File or Console Log

Log записване на информация как работи нашата програма

Ретърн типа и параметрите на Делегата трябва да съвпадат с тези на метода, които се опитваме да присвойм

Делегатите са референс типове, ако не асайнати ще хвърлят Null Reference Exception.

## Command Line arguments

- еднобуквен агрумент

-- еднодумов агрумент

## Generic Delegates

// normal Delegate

public delegate int SumDelegate(int a, int b)

// generic Delegate

public delegate TResult SumDelegate<TResult, TArg>(TArg a, TArg b)

or

public delegate T SumDelagate<T>(T a, T b)

delegate се вика без име т.е. анонимно извикване

Стандартни предифинирани generic делегати Func<> и Action<>

**Func<>** връща стойност, но не изпълнява действие

**Action<>** изпълнява действие, но не връща стойност

Where<> връша List<>