Полезни неща от курса по ООП с C# .NET

Седмица 1:

Shortcuts:

Rename var/func/class/whatever -> CTRL + R CTRL + R

Copy Line -> CTRL + D

Mark whole line HOME SHIFT + END

Mark whanted area - hold ALT mouse mark

Call Hierrarhy - CTRL + K CTRL + T

Regions - CTR + K, S

Snippets:

cw ctor prop propfull

Как се прави сниппет:

XML file -> .snipped ->

<?xml version="1.0" encoding="utf-8"?>

<CodeSnippets

xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">

<CodeSnippet Format="1.0.0">

<Header>

<Title>

<!--TITLE-->

</Title>

<Shortcut>

<!--SHORCUT-->

</Shortcut>

</Header>

<Snippet>

<Code Language="CSharp">

<![CDATA[ <!--CODE--> $end$]]>

</Code>

<Declarations>

<Literal>

<ID><!--PARAM--></ID>

<Default>true</Default>

<ToolTip><!--INFO--></ToolTip>

</Literal>

</Declarations>

</Snippet>

</CodeSnippet>

</CodeSnippets>

Стил:

Класовете и методи започват с главна буква. Променливи малка (CamelCase)

Всяка променлива на нов ред, по-нагледно, ако трябват коментари.

За WPF приложения:

XAML:

Border

StackPanel

Grid - ColumnDefinitions, ColumnDefinition Width=\*, RowDefinitions, RowDefinition Heigth=\*, Grid.Column, Grid.Row

TextBlock

TextBox

ComboBox - ComboBoxItem

CheckBox

GroupBox

Функционалност на бутоните:

Click=”event\_name” - имаме име на евент, когато натиснем бутона

private void event\_name\_Click(object sender, RoutedEventArgs e) - функция която се изпълнява когато се натисне бутона

(string)((Button) sender).Content //switch

самите елементи могат да имат имена, които във файла, където пишем код на C#, можем да достъпваме, променяме и извличаме данни от/в тях и т.н.

Полезни неща:

TryParse string->int ex. int a = int32.TryParse(str)

Седмица 2:

Полезно:

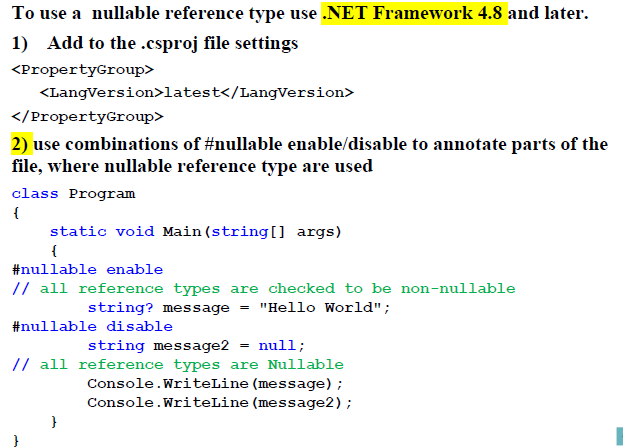
Трябва да пишем пред класовете public.

Нови обекти на клас създаваме <клас-име> обект = new <клас-име>() <- конструктор

Полета и пропъртита - propful tab tab -> field името е с малка започваща, пропартито е същото име с главна започваща. Пропъртито има гет и сет методи, които са със синтаксис

get{...return field\_name} set{...field\_name = value} (value e стойността която се подава при извикване на сет метод). В пропъртито можем да слагаме private пред set / get -> readonly writeonly.

Nullable variables <тип>? име. Това позволява на стойността да бъде null.

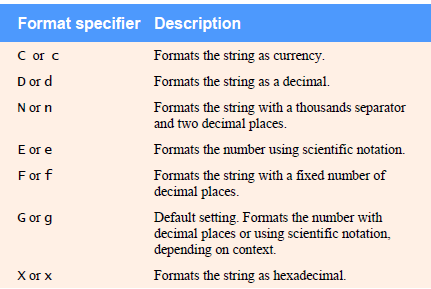


Tuples - когато искаме да пакетираме няколко неща в едно нещо. Например искаме функцията ни да върне наредена твойка инд стринг дабъл -> ползване този тип (int, string double), като пред всяко можем да слагаме имена и след това когато ползваме променливата, която съдържа тези данни да извикваме желаните данни.

“=>” този символ казва, че ако имаме една фунцкия Ф1() директно върни нещо. Като например:

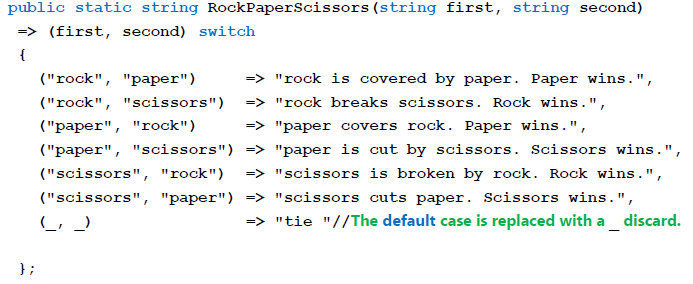
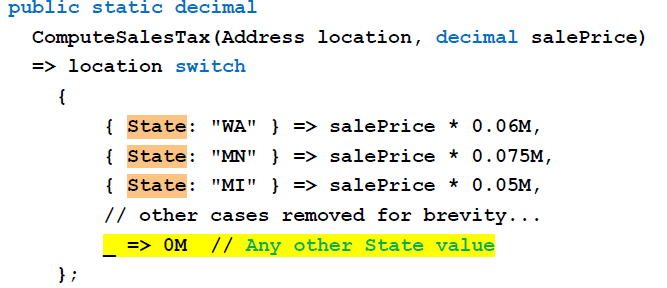
(int dummy, int max, int min) MinMax(int i)

=> (0, int.MinValue, int.MaxValue);



Pattern matching с switch и when. В условията на суйтча можем да кажем <тип> име when (булево условие).

PropertyPattern matching синтаксис:



“is” примерно проверява дали инпута е инт if (input is int count) -> count става стойността на input.

Class Diagram -> Project -> New Item -> Class Diagram

Toolbox добавяме каквото ни е нужно.

Седмица 3

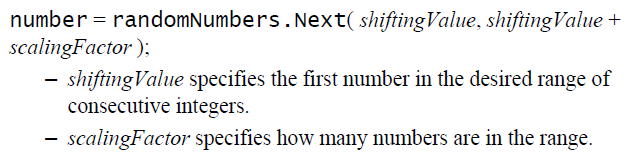
Полезно:

Правим малки смислени функции, които да правят едно нещо, преизползваме код, предодвратяваме повтарящ се код.

Статични методи/променливи. Име на класа . <static\_method/ member>

Random функция.

Random myRandom = new Random() -> myRandom.Next(1, 7) //[1,7)



Множество от константи - enum <name>{EL1, EL2 ...}

private enumMyEnum: *typeName*{*Constant1*,*Constant2*,*...*}

Можем да overload-ваме методи - едно и също име на метод, но различни параметри.

Подаване на аргумент на функция по референция. Можем да използваме думите ref(когато очакваме подадения арг да е инициализиран предварително) и out(когато очакваме, че функцията ще му даде стойност). И когато подаваме аргумент задължително пишем следното func(ref/out varName).

След c#7 можем да извикваме фънк(out int variable) и тази променлива е валидна в скоупа в който е ползвана, ползва се в TryParse

Ако класа, от който искаме да създадем инстанция няма конструктор който да работи с конкретни параметри можем да направим следното

construcotr\_name(<member\_name>: <val>...)

Структура от данни са колекции от дании които са свързани помежду си

Масиви: Lenght, създаваме като <type>[] c = new <type>[number], при създаването на масив всички елементи приемат дефолтни стойности, числовите 0, булевите лъжа, референтните NULL, static Array.Resize(ref myArr, newSize)), ArrayInitializer

foreach (type id in arr){statement}

var когато ползваме тази дума задължително трябва да кажем = някаква стойност иначе ще доведе до грешка, удобно е да се ползва във foreach

Подаване на масив като аргумент на функция става само чрез неговото име без скоби, а в метода трябва да бъде обявено, че параметъра е масив (т.е. със скоби). Подаване без реф прави копие на референцията към която сочи масива, но може чрез нея да се променят елементите

двуметни масиви - int[ , ] myArray = {{1, 2},{3 ,4}} (всеки “ред” има еднакъв брой елементи), но ако си искаме наш си брой правим int[][] myCoolArray{new int[] {2,3 ,3}, new int[] {2,1,2}...}

int[ , ] b; b= new int[3,5], int[][] c; c = new int[2][]; c[0] = new int[5].....

GetLength...или два вложени foreach-а един по редове и другия колони

params (на последно място) пример foo(params double[] numbers)

Седмица 4

Полезно:

Масиви - даваме име на масива в множествено число

\* int size = int.Parse(Console.ReadLine())

масив с размер 0 не е нулл

var names = new[]{........} var e без []

Масивите имат функции и пропъртита от клас Array .NET Framework

-Length

Не можем да променяме данните на масив в foreach трябва for

При копиране на масиви, масивите са референтни типове така, че когато направим int[] a1.. int[] a2 = a1 имаме 2 референции към едно и също нещо. Ако искаме да копираме данните правим int[] anew = new int[aold.Length] и завъртаме един for. Може и с aold.CopyTo(anew,0)

Collections <3:

ArrayList - Remove(само пренарежда елементите),Add(ако се наложи се преоразмерява), Insert(ако се наложи се преоразмерява), [] (но не е препоръчителен заради лоша производителност, ползва се за хетерогенни колекции) вместо това->

List - ползва се за хомогенни колекции

Queue(FIFO) - at the back Enqueue, Dequeue (с един цикъл може да я изпразним queue.Count > 0 Dequeu), Dequeue-то връща това, което декювва

Стек(FILO) - push, pop

HashTable - асоциативен списък има ключ/стойност, реализира се с 2 обекта от тип масив, един за ключовете другия за стойноститте които мапваме, не можем да имаме дублиращи се ключове, ContainsKey, (key value)

