

Упражнение №1 по ПС

C#

*Среда за разработка Visual Studio.
Разлики между C++ и C#.
Markup languages.*

Целта на това упражнение:

Запознаване с езика C# и средата, на която основно се използва – Visual Studio. Тях ще използваме на всички следващи упражнения. Запознаване с основите принципи на Markup languages, за да можем по-нататък да се запознаем и с езика XAML.

Необходими познания:

От предметите дотук трябва да можете:

- Да използвате езика C++
- Да създавате променливи и функции
- Да използвате операторите if, for, switch, указатели,...
- Да създавате class, struct, interface, enumeration
- Да управлявате изключения
- Да създавате проекти с повече от един файл
- Да създавате конзолни приложения

Задачите в упражнението изграждат:

Университетска информационна система.

В това упражнение: системата за влизане с потребителско име и парола

- клас, който да отговаря за проверка на въведените потребителско име и парола (LoginValidation)

- изброен тип, който да описва различните типове потребители (UserRoles)

- клас, който да отговаря за връзката с наличните в системата потребители (UserData)

- клас, който да описва данните на един потребител (User)

В края на упражнението:

Потребителя ще въвежда потребителско име и парола, системата ще го регистрира и ще изведе какви привилегии има. (За сега регистрацията ще е възможна с произволно име и парола).

Легенда:

Най-важното

- Схематично и накратко

Обяснение. Уточнение.

Пример: или **Задача:** ограденото трябва да изпълните за да е синхронизиран проекта ви

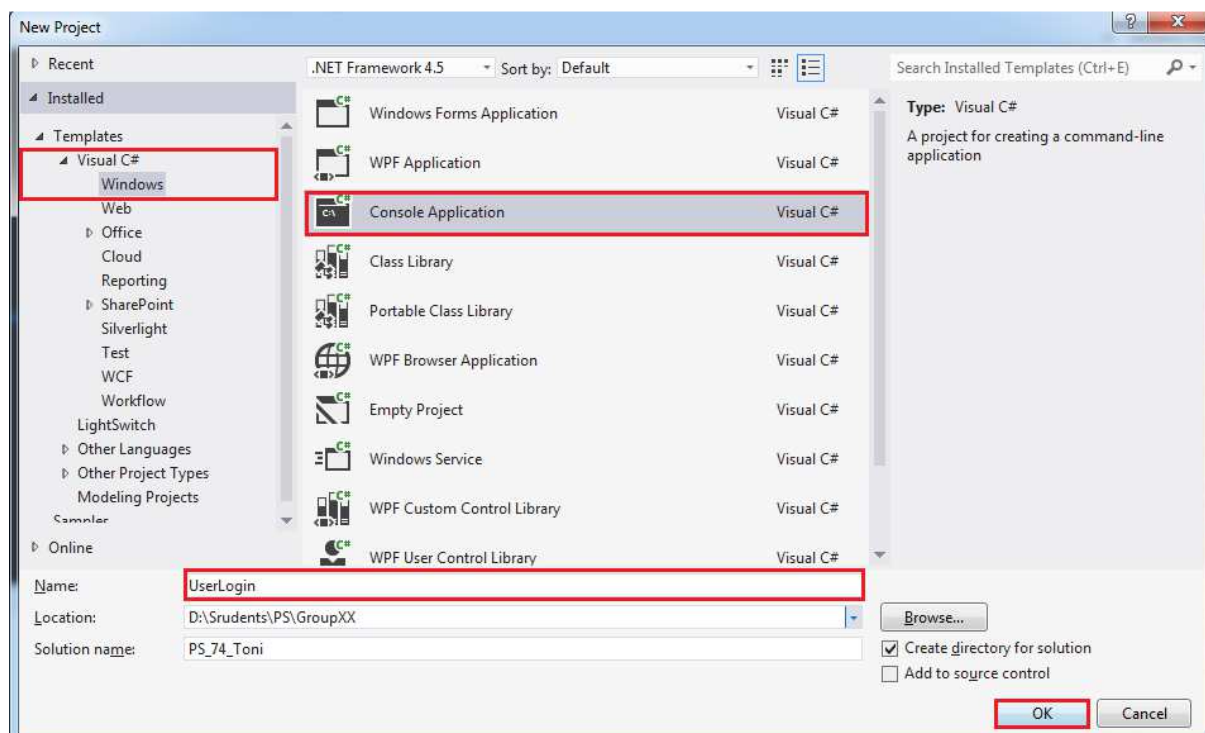
- Подходящо място да стартирате изпълнение за да проверите дали работи вярно.

1. Среда за разработка Visual Studio.

1.1. Създаване на нов проект за конзолно приложение

За да създадете нов проект:

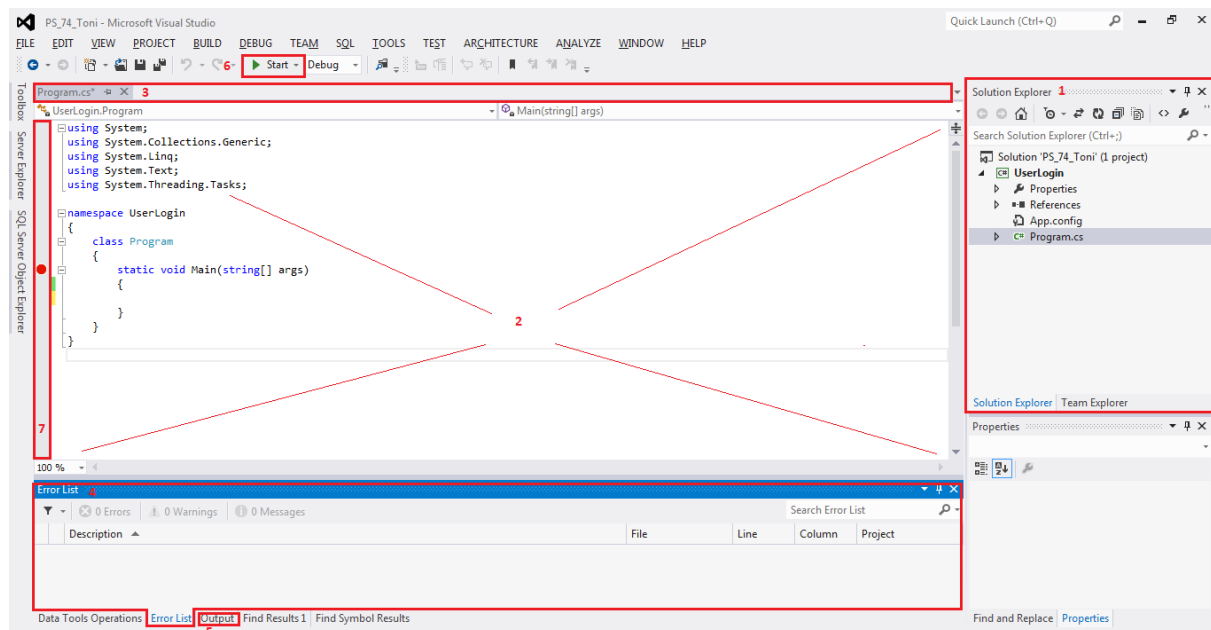
1. Отворете Visual Studio 2012
2. От менюто изберете "File" → "New"
3. В диалоговия прозорец изберете "Visual C#" → "Windows" → "Console Application"
4. Кръстете вашия проект **UserLogin**, но след това задайте друго име за Solution. Нека името представлява PS_<№ на група>_<име>, например PS_74_Toni.



1.2. Изглед на отворен проект

След като е отворен един проект вашето Visual Studio 2012 трябва да има следните елементи:

1. **Solution Explorer** – списък на всички проекти и всички файлове на всеки проект.
(Ако не се вижда: "View" → "Solution Explorer")
2. **Екран за редакция** на отворен файл.
3. Всички отворени файлове.
4. **Error List** – списък на всички установени в кода грешки.
(Ако не се вижда: "View" → "Error List")
5. **Output** – стандартния изход на приложението.
(Ако не се вижда: "View" → "Output")
6. Бутон за **стартране на приложението**
(Ако не се вижда: "View" → "Toolbars" → "Standard")
7. Лента за поставяне на **Breakpoint** за Debug.



Visual Studio помни два режима на средата – Design и Debug. Ако в режим на стартирано приложение някои от тези елементи ви липсват, трябва да си ги отворите от менюто.

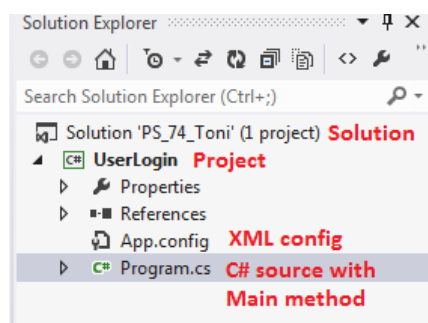
1.3. Структура на Solution и Project

- Solution ≠ Проект
- Проект = Приложение
- Solution = Комплект приложения


(Пример за комплект приложения: основно приложение и множество DLL; сървърно, web и десктоп приложения със свързана функционалност)

Проектът винаги включва

- C# сорс файлове с разширение .cs (Program.cs)
- XML конфигурационен файл (App.config или Web.config)



Задача: Да добавим още един файл:

1. От менюто изберете “Project” → “Add New Item...”
(Или от “Solution Explorer” → “Project” →  → “Add” → “New Item...”).

Не на Solution, не на файл, а на проекта – UserLogin)

2. В диалоговия прозорец изберете “Visual C# Items” → “Code” → “Class”
3. Кръстете новия файл User.cs



В новия файл автоматично ще се създаде празна конструкция на клас с име User. Това ще е класът, който ще описва потребителите на системата.

1.4. Система за подсказки IntelliSense

IntelliSense е общ термин за множество технологии за редакция на код, включващ: code completion, parameter info, quick info, member list

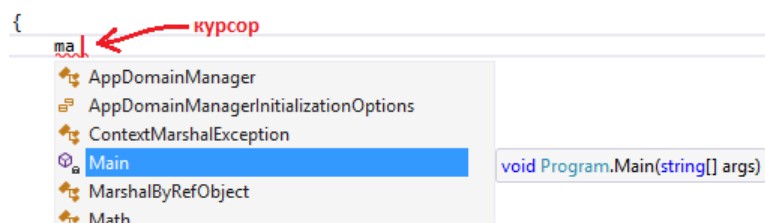
Във Visual Studio технологията IntelliSense е основен способ за писане за код.

IntelliSense се активира:

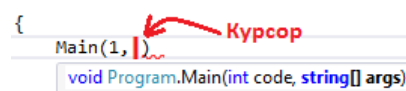
- При изписване на специален символ (най-често точка или скоба)
- При започване на изписване на нов ред
- С натискане на **Ctrl + Space** за подсказка за променливи и типове
- С натискане на **Ctrl + Shift + Space** за подсказка за параметри (само между скобите за параметри на функция)

Пример:

Ctrl + Space



Ctrl + Shift + Space

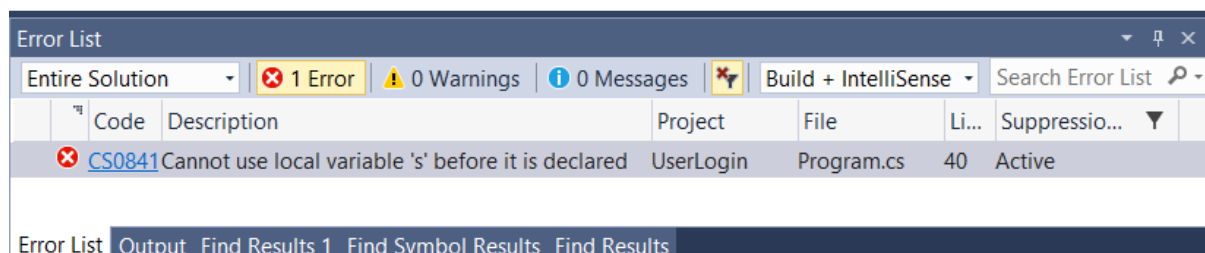
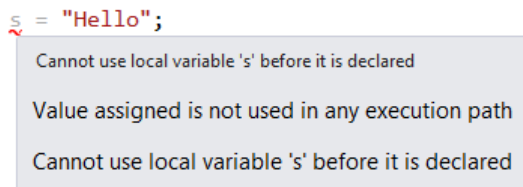


Още за IntelliSense: <https://code.visualstudio.com/docs/editor/intellisense>

1.5. Прозорецът Error List

Ако допуснете грешка при писане на код, която не може да се компилира, се появява ред в Error List.

- Двоен клик върху ред за грешка в Error List ще премести курсора там, където компилатора се сблъсква проблем (т.е. при самата грешка)



Добра практика е да отстранявате грешките щом ги забележите, вместо да изпишете целия код и да ги отстранявате една по една след това.

Освен ако не са грешки от незавършения код, който изписвате точно в момента.

2. Разлики между C++ и C#

2.1. Синтаксис на C#

C# е език от фамилията на C езиците, разработен за най-удобно използване под .NET

Основният синтаксис на C# е сходен с този на C++:

- Обектно ориентиран
- Класове, структури, изброени типове (**class, struct, enum**)
- Основни типове: **int, double, bool**
- Запазени думи: **for, while, if, else,...**
- Синтаксис: къдрави скоби { }, извикване на членове с точка .
- Изключения: **try, catch**

Основните разлики са разгледани надолу.

Задача: Във файла User.cs разширете класа User с публични полета от подходящ тип и име, които да съхраняват:

- Потребителско име

- Парола
- Факултетен номер
- Роля на потребителя

Ролята да е Integer със следната кодировка:

1 – администратор

2 - инспектор

3 – преподавател

4 - студент

2.1.1. Namespace

Namespace се използва за групиране на типове (класове).

```
namespace UserLogin
{
    // Дефиниции на типове
}
```

В C# кодът е организиран по следния начин:

- Всеки файл може да достъпва всички файлове, включени към проекта. Т.е. не се използва #include
- Всеки файл може да съдържа един или повече namespace.
- Един namespace може да е разписан в един или повече файлове.

Пример: В двата файла Program.cs и User.cs се описва един и същ namespace – UserLogin. За това в метода Main типът User е видим.

Program.cs

```
namespace UserLogin
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

User.cs

```
namespace UserLogin
{
    class User
    {
        // Публични полета
    }
}
```

Задача: Да изпробваме новия клас, който добавихме по-горе.

- В метода Main създайте обект от тип User.
- Попълнете полетата му с примерни данни за администратор.

2.1.2. Using

- Using не е задължително да се използва.

- Using скъсява обръщението към типовете в посочения namespace
- Using се използва винаги в началото на файла
- Using клаузата важи за целия файл и само за този файл

В C# достъпа до членове на namespace става с оператор . (точка), вместо ::

Пример: Visual Studio добавя using клаузи за най-често използваните namespace във всеки файл. Например ако премахнем using System от файла User.cs ще трябва да описваме типовете чрез пълно наименование:

Без using System:

```
namespace UserLogin
{
    class User
    {
        public System.String Username;
        public System.String Password;
        public System.String FakNum;
        public System.Int32 Role;
    }
}
```

С using System:

```
using System;

namespace UserLogin
{
    class User
    {
        public String Username;
        public String Password;
        public String FakNum;
        public Int32 Role;
    }
}
```

2.1.3. Стандартен вход/изход

В C# няма потоци cin и cout.

Стандартния вход и изход е капсулиран в класа System.Console

- Ако е деклариран using System достъпа до класа става директно Console.
- Извеждането на съобщения става с Console.WriteLine(*// string to write*)
- Прочитането на въведеното става с Console.ReadLine()

Пример:

```
string s;
s = Console.ReadLine();
Console.WriteLine(s);
```

Задача: В метода Main изведете на екрана данните от обекта с примерните данни за администратор.

- Стартирайте програмата с F5 или зеления триъгълник.

2.1.4. Глобални методи и променливи


В C# глобални методи и променливи не са позволени. Трябва да са членове на клас.

- Всеки файл се състои от namespace-и.
- Всеки namespace може да се състои само от типове: class, struct, interface, enumeration, вложен namespace
- => методи и променливи могат да са единствено част от клас или структура

Пример: Самият метод Main също не е глобален. Той е член на клас:

```
namespace UserLogin
{
    class Program
    {
        static void Main(string[] args)
        {
            ...
        }
    }
}
```

Задача: Ще ни е необходим метод `ValidateUserInput`, който да проверява дали потребителя е попълнил нещо смислено за потребителско име и парола. За целта трябва да му създадем клас с подходящо име:

- Създайте клас с име `LoginValidation`
("Project" → "Add New Item...")
(Или от "Solution Explorer" → "Project" →  → "Add" → "New Item...")
- Добавете в класа публична функция `ValidateUserInput`

Функцията да връща булев резултат. За сега нека е `true` по подразбиране.

Задача: Променете метода `Main` така че:

- да инстанцира обект от тип `LoginValidation`
- да запитва функцията `ValidateUserInput()`
- ако отговора на `ValidateUserInput()` е бил `true`, да извежда на екрана данните от обекта с примерните данни за администратор.

Към момента `ValidateUserInput` винаги ще връща `true`. Ще добавим проверки след малко, но логиката трябва да е: принтираме данни за потребител само ако се е регистрирал, т.е. ако `ValidateUserInput()` е върнал `true`

- Стартирайте програмата с F5 или зеления триъгълник.


2.1.5. Static

В C# статични могат да са само класове и членове на класове.

Тъй като всички типове са част от namespace, а всеки namespace може да е разписан в множество файлове, не може да задаваме статични типове само за текущия файл.

В C# достъпа до статични членове става с оператор `.` (точка), вместо `::`

Задача: Създайте нов статичен клас `UserData`:

- Създайте клас с име `UserData`
("Project" → "Add New Item...")
(Или от "Solution Explorer" → "Project" →  → "Add" → "New Item...")
- Добавете към дефиницията на класа запазената дума `static`.

Този клас ще отговаря за съхраняване и попълване на данните в обекти от класа `User`.

Т.е. той няма да има собствени инстанции, всяка - с различни данни, и за това е статичен.

2.1.6. Директивата `#define`

В C# директивата `#define` не може да се използва за дефиниране на константи.

Подходяща реализация на константи е като:

- Изброен тип
- Статично поле на клас


Задача: Добавете статично публично поле `TestUser` от тип `User` към класа `UserData`.

Задача: Добавете статичен частен метод `ResetTestUserData()`, който да попълва `TestUser`.

- Метод на `static` клас трябва също да е `static`. (както и всички членове на `static` клас)

```
static private void ResetTestUserData()  
{  
    ...  
}
```
- Преместете в новия метод кода от метода `Main`, който създава обект от тип `User` с примерни данни за администратор в новия метод `ResetTestUserData()`, и го преправете да задава данни за `TestUser`
(Т.е. вместо да попълваме някакъв обект в `Main`, да попълним `TestUser` в `ResetTestUserData()`)
(Изкарването на екрана остава в `Main`, само задаването на данните се премества)

Задача: Добавете изброен тип `UserRoles`, който да описва възможните потребители на системата:

1. От менюто изберете "Project" → "Add New Item..."
(Или от "Solution Explorer" → "Project" →  → "Add" → "New Item...")
2. В диалоговия прозорец изберете "Visual C# Items" → "Code" → "Code File"



3. Кръстете новия файл `UserRoles.cs`
4. Декларирайте, че ще пишем в `namespace` `UserLogin`

```
namespace UserLogin  
{  
}
```
5. Вътре в `namespace` блока добавете изброен тип `UserRoles` със следните стойности:
`ANONYMOUS, ADMIN, INSPECTOR, PROFESSOR, STUDENT`

2.1.7. Свойства

Свойство е:

- Член на клас
- Дава гъвкав достъп до private поле
- Използва се като публично поле
- Реализира се чрез функции (get и set)

Т.е. свойството изглежда като поле, но всъщност е функция.

2.1.7.1. Синтаксис на свойства

Синтаксис на свойства в C#:

```
class SomeClass
{
    private double _value;

    public double value
    {
        get {
            // Изчисления с _value
            return // изчислението;
        }
        set {
            // Изчисления с value
            _value = // изчислението ;
        }
    }
}
```

Пример: Нека преправим TestUser да бъде свойство с цел да не може да се променя извън класа и винаги да връща едни и същи данни:

1. В класа UserData добавяме ново частно поле _testUser от тип User. Разбира се в static клас вмяко поле е static.

```
static private User _testUser;
```

2. Добавяме към TestUser синтаксиса за свойство:

```
static public User TestUser
{
    get { }
    set { }
}
```

3. Имплементираме get метода като викаме по-рано написания метод ResetTestUserData() преди да върнем резултата. Така винаги ще връщаме заложеното в ResetTestUserData():

```
get {
    ResetTestUserData();
    return _testUser;
}
```

4. Set метода оставяме празен. Промяна през публичното свойство не предвиждаме.
5. Преправяме метода ResetTestUserData() да попълва частното поле _testUser вместо TestUser.

```
static private void ResetTestUserData()
{
    ... TestUser → _testUser ...
}
```

(→ означава "подменя се с")

Задача: В метода Main променете извеждането на данните да е от обекта `UserData.TestUser`.

- Стартирайте програмата с F5 или зеления триъгълник.

2.1.7.2. Къс синтаксис на свойства

Често има нужда дадено поле да се държи като свойство, без специфична логика за методите `get` и `set`.

За това е късият синтаксис:

```
class SomeClass
{
    public double value
    { get; set; }
}
```

Късият синтаксис на свойство е удобен, когато

- няма специална логика, която да се реализира в `get` и `set` метода, а:
 - само искаме да модифицираме видимостта на `get` или на `set` метода
 - или искаме да е наличен само един, т.е. само `get` или само `set` метод.

Пример: Следните две дефиниции са еквивалентни:

```
class SomeClass
{
    private double _value;

    public double value
    {
        get {return _value;}
        set {_value = value}
    }
}
```

```
class SomeClass
{
    public double value
    { get; set; }
}
```

Задача: Добавете в класа `LoginValidation` статично публично свойство `currentUserRole` от тип `UserRoles`:

- Свойството трябва да може да се чете отвън, но да се задава само в класа. Т.е. `set` метода трябва да е частен.
`private set;`

Задача: Променете Main метода:

- След като изкара данните за тестовия потребител (ако е успешно логнат):
 - изведете на екрана стойността на `LoginValidation.currentUserRole`

За момента ще изкарва ANONYMOUS , защото това е стойността по подразбиране и не е задавана друга. По-нататък ще добавим описание за различните роли и ще задаваме стойности.

- Стартирайте програмата с F5 или зеления триъгълник.

3. Markup languages

Ако имате понятие от езика HTML може да пропуснете този раздел. Тези знания ще са ви напълно достатъчни за да започнете XAML.

За останалите следва кратко въведение в HTML.

3.1. Езикът HTML

HTML = Hypertext Markup Language

Основният маркиращ език за описание и дизайн на уеб страници.

Описанието на документа става чрез специални елементи, наречени **HTML елементи** или маркери, които **се състоят от** етикети или **тагове** (HTML tags) и **ъглови скоби** (като например елемента <html>).

HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница, като заглавия, цитати, раздели, хипертекстови препратки и т.н.

Най-често HTML елементите са групирани по двойки <h1> и </h1>. HTML файлове съдържат текстово съдържание с маркери – инструкции за браузъра за това как да се показва текстът. Например

```
<маркер> Някакъв текст. </край на маркера>
```

Но също и

```
<маркер свойство="стойност" />
```

Пример за HTML файл.

```
<!DOCTYPE html>
<html>
<head>
  <title>My First HTML</title>
  <meta charset="UTF-8">
</head>
<body>

  <h2 title="I'm a header">A Heading with title attribute</h2>
  <p>A paragraph </p>

</body>
</html>
```

Запишете си проекта! USB, DropBox, GoogleDrive, e-mail, SmartPhone, където и да е.

До края на упражненията ще работите върху този проект.

Вкъщи също ще работите върху този проект.

Накрая трябва да го представите завършен.