

# Упражнение №2 по ПС

## C#

Среда за разработка Visual Studio.

Разлики между C++ и C#.

### Целта на това упражнение

*Запознаване с езика C#, прихващане на грешки, делегати и други.*

### Задачите в упражнението изграждат:

*Малка студентска информационна система*

### В това упражнение:

*Конзолно приложение, което надгражда Упражнение №1.*

- Създаване и работа с делегати
- Прихващане на грешки
- Логъри и как да ги използваме.

### В края на упражнението:

*Ще записваме в конзолата събития чрез използването на логъри, ще прихващаме грешки хвърлени от приложението и ще умеем да споделяме код писан в друг код на нашият Solution.*

### За домашно:

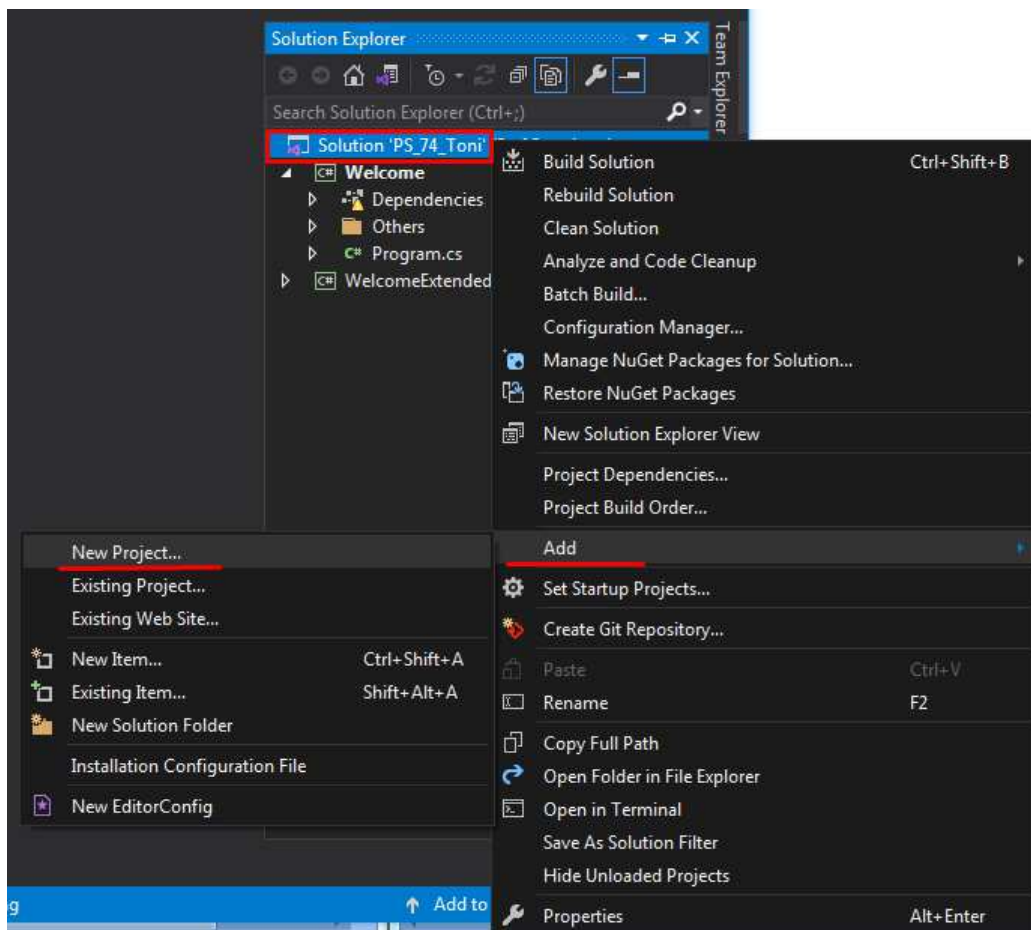
Да се промени кода така, че:

1. Да се промени кода на HashLogger така, че да може да принтираме всички записани съобщения.
2. Да се промени кода на HashLogger така, че да принтираме събитие по дадено eventId.
3. Да се промени кода на HashLogger така, че да можем да изтриваме събитие по дадена eventId.
4. Да се добави Logger, който да записва във файл.

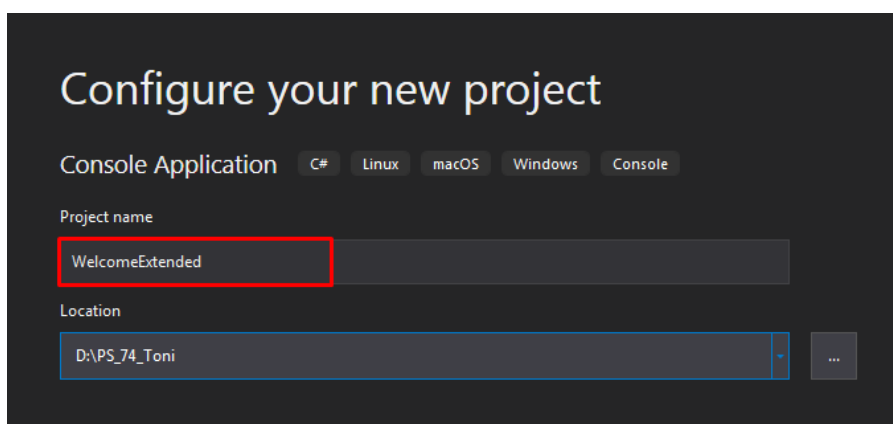
**Важни знания от упражнението: Exceptions, Delegates, Shared Projects, NuGet Packages, Loggers, ConcurrentDictionary, Interfaces**

## Създаване на проект

1. Отворете **Visual Studio 2022** (или по-ново)
2. Заредете **Solution**-а създаден в предишното упражнение.
3. Създаваме нов проект, това става чрез **десен клик** в **Solution Explorer** на **Solution**-а



4. Изберете **Add → New Project**
5. Изберете конзолно приложение като в предното упражнение ("**C#**" → "**Windows**" → "**Console**" → "**Console App**")
6. Въведете име на проекта **WelcomeExtended**.

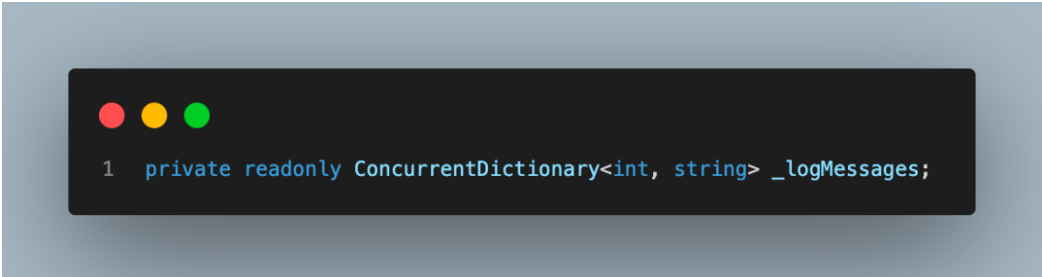


## Създаване на папки

1. От **Solution Explorer** избираме проекта **WelcomeExtended**
2. Кликаме с десен бутон върху **WelcomeExtended**, след което избираме **Add → New Folder**
3. Създаваме следните папки: **Helpers, Loggers, Others**

## Създаване на клас HashLogger

1. Кликнете с десен бутон мишката на папката **Loggers**
2. От контекстното меню изберете **Add → New Item...**
3. От новоотвореният прозорец изберете **Class**
4. В полето **Name**, въведете за име **HashLogger.cs**
5. В новосъздаденият клас добавяме частно **readonly** поле **\_logMessages** от тип **ConcurrentDictionary<int, string>**



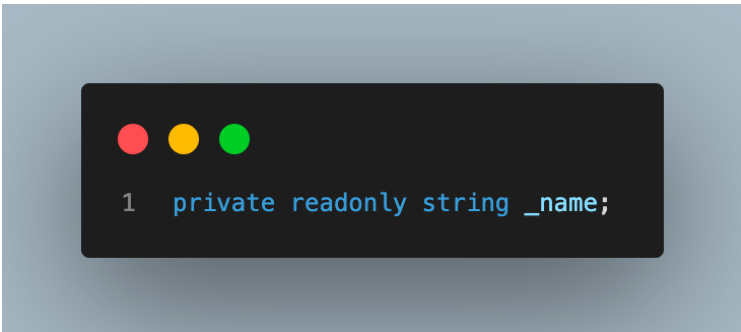
```
1 private readonly ConcurrentDictionary<int, string> _logMessages;
```

Пояснение:

- **readonly** се използва за деклариране на поле, чиято стойност може да бъде задавана/променяне еднократно, по време на декларацията или в конструктора на класа.

- **ConcurrentDictionary** е структура от данни, която позволява паралелна работа със стойности, съхранявани в речник. Създадена е като аналог на обикновените речници в C#, но е по-сигурна при работа от множество нишки, като позволява едновременна работа от много места. **<int, string>** са типовете които **ConcurrentDictionary** използва за ключа и стойността които ще се запаметяват в речника.

6. Създаваме поле с име **\_name** и тип **string**



```
1 private readonly string _name;
```

7. Създаваме конструктор, който
  - а. приема: един входящ параметър **string name**,

- b. а в тялото на конструктора:  
присвояваме **name** на полето **\_name**,  
присвояваме нова инстанция на **ConcurrentDictionary** на полето **\_logMessages**.

## Използване на NuGet Packages

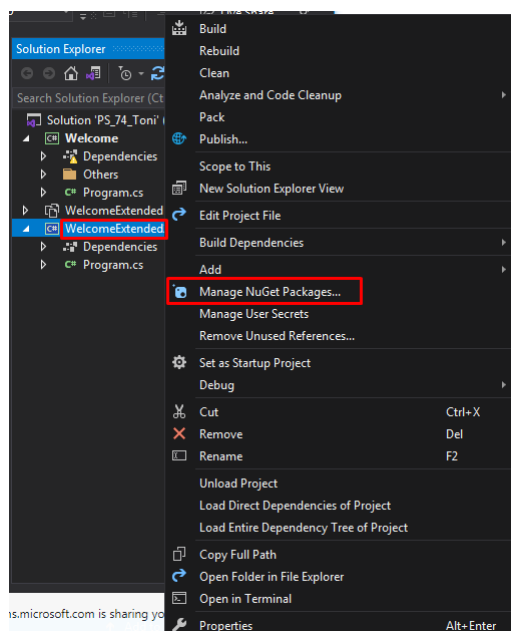
1. Класът **HashLogger**, трябва да имплементира интерфейса **ILogger** в C#, това става чрез използването на **ILogger**.  
след класа.
2. Интерфейсът **ILogger** се намира в **Microsoft.Extensions.Logging** :

```
using Microsoft.Extensions.Logging;
```

Библиотеката **Microsoft.Extensions.Logging** се намира извън вашия проект (не е включена по подразбиране.

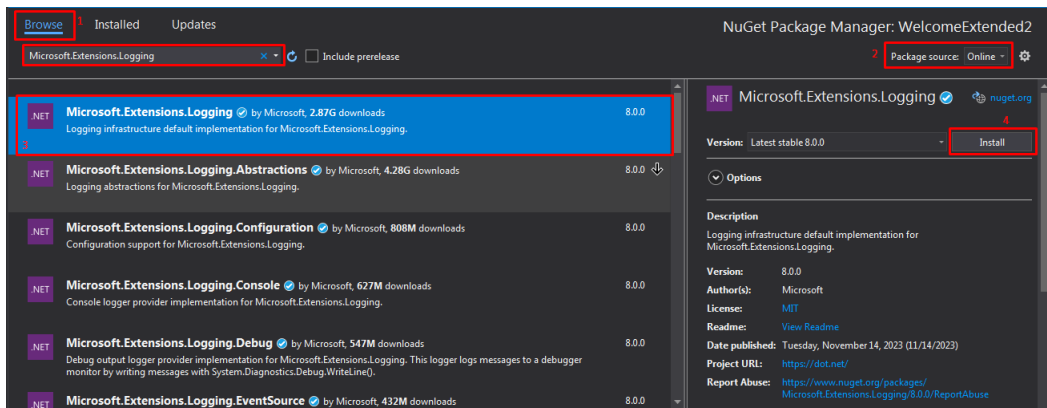
За да я включа им трябва да добавим **външен пакет** към проекта. Това става чрез системата **NuGet Package Manager**:

3. Кликнете с десен бутон мишката на **проекта WelcomeExtended** и от контекстното меню изберете **Manage NuGet Packages...**



4. В отвореният се **NuGet Package Manager** потърсете онлайн библиотеката и за нея изберете **Install**

(Изображението е от Visual Studio 2019):



5. За да бъде валидна имплементацията нашият клас трябва да има имплементация на следните методи

```
0 references
public IDisposable BeginScope<TState>(TState state)
{
    throw new NotImplementedException();
}

0 references
public bool IsEnabled(LogLevel logLevel)
{
    throw new NotImplementedException();
}

0 references
public void Log<TState>(LogLevel logLevel, EventId eventId,
    TState state, Exception? exception,
    Func<TState, Exception?, string> formatter)
{
    throw new NotImplementedException();
}
```

6. Добавяме следният код във всеки от трите метода

```
1 public IDisposable BeginScope<TState>(TState state)
2 {
3     // This logger does not support scopes.
4     return null;
5 }
```



```
1 public bool IsEnabled(LogLevel logLevel)
2 {
3     // This logger is always enabled.
4     return true;
5 }
```



```
1 public void Log<TState>(LogLevel logLevel, EventId eventId, TState state, Exception exception, Func<TState, Exception, string> formatter)
2 {
3     var message = formatter(state, exception);
4     switch (logLevel)
5     {
6         case LogLevel.Critical:
7             Console.ForegroundColor = ConsoleColor.Red;
8             break;
9         case LogLevel.Error:
10            Console.ForegroundColor = ConsoleColor.DarkRed;
11            break;
12        case LogLevel.Warning:
13            Console.ForegroundColor = ConsoleColor.Yellow;
14            break;
15        default:
16            Console.ForegroundColor = ConsoleColor.White;
17            break;
18    }
19    Console.WriteLine("- LOGGER -");
20    var messageToBeLogged = new StringBuilder();
21    messageToBeLogged.Append($"[{logLevel}]");
22    messageToBeLogged.AppendFormat(" [{0}]", _name);
23    Console.WriteLine(messageToBeLogged);
24    Console.WriteLine($" {formatter(state, exception)}");
25    Console.WriteLine("- LOGGER -");
26    Console.ResetColor();
27    _logMessages[eventId.Id] = message;
28 }
```

В този метод, единственото което правим е да принтираме грешката и да я запамятаваме в `_logMessages`, `switch`-а се използва за да оцветим грешката в цвят спрямо типа ѝ.

## Създаване на Клас LoggerProvider

1. В папката **Loggers** създайте още един клас – **LoggerProvider**, като имплементирате интерфейса **ILoggerProvider**, като в метода **CreateLogger** (който се изисква от интерфейса) добавяте **return new HashLogger(categoryName);**

*Този файл ще се използва за създаването на нова инстанция на Logger, това е код с илюстративна цел, но не спазва добри практики.*

## Създаване на клас LoggerHelper

1. В папката **Helpers** създайте още един клас – **LoggerHelper**, трябва да направим класа **static**
2. В класа добавяме статичен метод **GetLogger** който връща **ILogger**, като входящ параметър приема **string categoryName**
3. Добавяме следният код

```
1 public static ILogger GetLogger(string categoryName)
2 {
3     var loggerFactory = new LoggerFactory();
4     loggerFactory.AddProvider(new LoggerProvider());
5
6     return loggerFactory.CreateLogger(categoryName);
7
8 }
```



## Създаване на клас Delegates

1. В папката **Others** създайте още един клас – **Delegates**
2. В класа добавете следното поле

```
1 public static readonly ILogger logger = LoggerHelper.GetLogger("Hello");
```

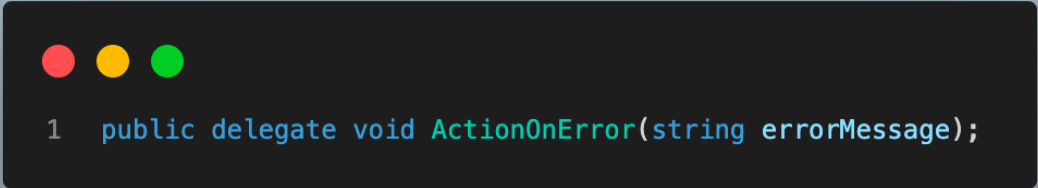
3. Добавяме два статични метода, както следва:

```
1 public static void Log(string error)
2 {
3     logger.LogError(error);
4 }
5
6 public static void Log2(string error)
7 {
8     Console.WriteLine("- DELEGATES -");
9     Console.WriteLine($"{error}");
10    Console.WriteLine("- DELEGATES -");
11 }
```

Първият метод принтира грешката посредством Logger-а, докато вторият принтира директно в конзолата.

## Създаване на делегатен тип ActionOnError

1. В папката **Others** създайте **delegate** ActionOnError.  
(Десен бутон > Add... > New Item... > Code File)  
Той се дефинира като метод, като няма тяло, само връщан тип и входящи параметри, които трябва да отговарят на тези в горните два метода:

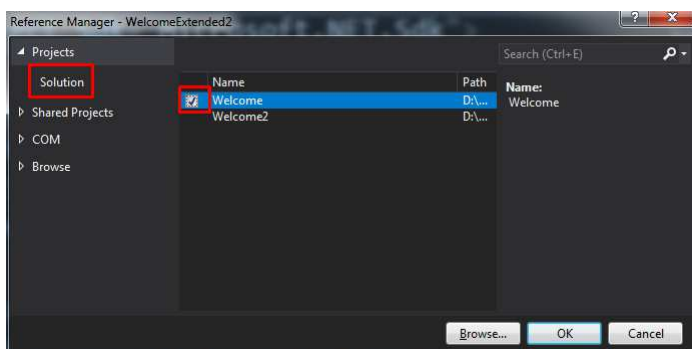


```
1 public delegate void ActionOnError(string errorMessage);
```

*Delegates в C# представляват тип данни, който може да съхранява референции към методи с определен тип параметри и тип връщане. Те се използват за предаване на функционалност от един метод към друг метод, без да е необходимо да се знае точно кой метод ще бъде извикан.*

## Връзка между проектите

1. Кликнете с десен бутон мишката на проекта **WelcomeExtended**
2. От контекстното меню изберете **Add → Project Reference...**
3. От новоотвореният прозорец от вашия Solution маркирайте проекта **Welcome** и изберете OK.



Така проекта **Welcome** става преизползван (dependency) от проекта **WelcomeExtended**.

4. От проекта **Welcome** ще са видими за преизползване от други проекти само онези класове, които са обявени за **public**. Например:

```
public class User
{
    ...
}
```

This method has 0 reference(s). (Alt+Z)

*Трябва добре да помислите кои класове ще са публични.*

5. Също за да достъпвате класа ви е необходим подходяща **using** клауза. Например:

```
using Welcome.Model;

namespace WelcomeExtended
{
    2 references
    class Program
    {
        2 references
        static void Main(string[] args)
        {
            User u = new User(66, "John Smith");
        }
    }
}
```

## Навързване на кода

1. Във файла Program.cs, в метода Main добавете следният код:  
(Методът **Log** по-долу не е достъпен директно. Къде се намираше?)

```
1  try
2  {
3      // Example 2
4      var user = new User
5      {
6          Name = "John Smith",
7          Password = "password123",
8          Role = UserRolesEnum.Student
9      };
10
11     var viewModel = new UserViewModel(user);
12
13     var view = new UserView(viewModel);
14
15     view.Display();
16
17     // Throw error here
18     view.DisplayError();
19 }
20 catch (Exception e)
21 {
22     var log = new ActionOnError(Log);
23     log(e.Message);
24 }
25 finally
26 {
27     Console.WriteLine("Executed in any case!");
28 }
```

Както може да забележите изпълняваният код се намира в try блока на try-catch-a, там се изпълнява кода, който очакваме да върне грешка. В catch-a, прихващаме грешката, като входящият параметър идва от самият CLR (Runtime-a на .NET). Finally в тази конструкция се изпълнява винаги, независимо от това дали приложението е хвърлило грешка или не.

2. Създайте метод **DisplayError()** в класа **UserView** в проекта **Welcome**, метода трябва да връща грешка. Това става посредством `throw new Exception("ТЕКСТ НА ГРЕШКАТА");`