

# Исключения и обработка ошибок

Юрий Литвинов  
yurii.litvinov@gmail.com

05.03.2021г

## TestCaseData (NUnit)

```
public class StackTest
```

```
{
```

```
    [TestCaseSource(nameof(Stacks))]
```

```
    public void StackShouldNotEmptyAfterPush(IStack stack)
```

```
{
```

```
        stack.Push(1);
```

```
        Assert.IsFalse(stack.IsEmpty());
```

```
}
```

```
private static IEnumerable<TestCaseData> Stacks
```

```
    => new TestCaseData[]
```

```
{
```

```
    new TestCaseData(new ArrayStack()),
```

```
    new TestCaseData(new ListStack()),
```

```
};
```

```
}
```

# Бросание исключений

```
if (t == null)
{
    throw new NullPointerException();
}

throw new NullPointerException("Hello!");
```

# Обработка исключений

```
try {  
    // Код, который может бросать исключения  
} catch (Type1 id1) {  
    // Обработка исключений типа Type1  
} catch (Type2 id2) {  
    // Обработка исключений типа Type2  
} catch {  
    // Обработка всех оставшихся исключений  
} finally {  
    // Код, выполняющийся в любом случае  
}
```

# Что делается

- ▶ На куче создаётся объект-исключение
- ▶ Исполнение метода прерывается
- ▶ Если в вызывающем методе есть обработчик этого исключения, вызывается он, иначе исполнение вызывающего тоже прерывается
- ▶ И т.д., пока не найдём обработчик или прервём Main

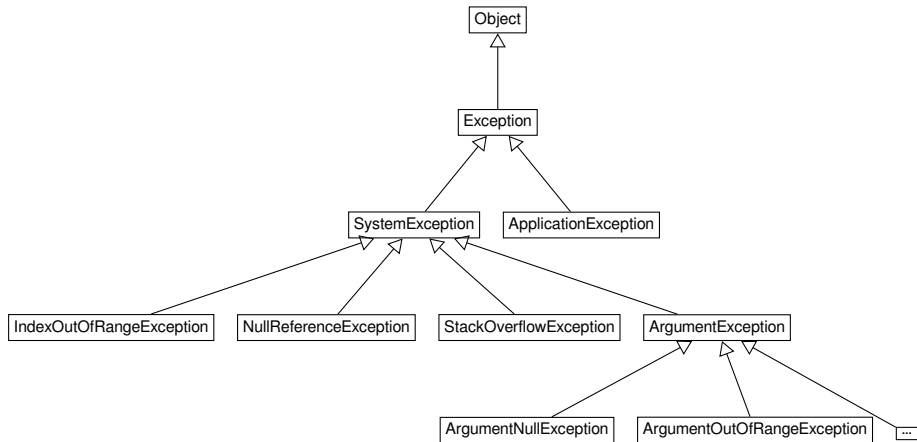
## Пример

```
private void ReadData(String pathName)
{
    FileStream fs = null;
    try {
        fs = new FileStream(pathName, FileMode.Open);
        // Делать что-то полезное
    }
    catch (IOException) {
        // Код восстановления после ошибки
    }
    finally {
        // Закрывать файл надо в любом случае
        if (fs != null)
        {
            fs.Close();
        }
    }
}
```

## Фильтры исключений

```
public static string MakeRequest()
{
    try
    {
        var request = WebRequest.Create("http://se.math.spbu.ru");
        var response = request.GetResponse();
        var stream = response.GetResponseStream();
        var reader = new StreamReader(stream);
        var data = reader.ReadToEnd();
        return data;
    }
    catch (WebException e)
        when (e.Status == WebExceptionStatus.ConnectFailure)
    {
        return "Connection failed";
    }
}
```

# Иерархия классов-исключений





# Свойства класса Exception

- ▶ Data
- ▶ HelpLink
- ▶ InnerException
- ▶ Message
- ▶ Source
- ▶ StackTrace
- ▶ HResult

## Пример (плохой)

```
try
{
    throw new Exception("Something is very wrong");
}
catch (Exception e)
{
    Console.WriteLine("Caught Exception");
    Console.WriteLine("e.Message: " + e.Message);
    Console.WriteLine("e.ToString(): " + e.ToString());
    Console.WriteLine("e.StackTrace:\n" + e.StackTrace);
}
```

# Что выведется

Caught Exception

e.Message: Something **is** very wrong

e.ToString(): System.Exception: Something **is** very wrong

в CSharpConsoleApplication.Program.Main(String[] args) в

c:\Projects\CSharpConsoleApplication\CSharpConsoleApplication\Program.cs: строка 15

e.StackTrace:

в CSharpConsoleApplication.Program.Main(String[] args) в

c:\Projects\CSharpConsoleApplication\CSharpConsoleApplication\Program.cs: строка 15

Для продолжения нажмите любую клавишу . . .

## Перебрасывание исключений

```
try
{
    throw new Exception("Something is wrong");
}
catch (Exception e)
{
    Console.WriteLine("Caught Exception");
    throw; // "throw e;" тоже работает, но сбросит stack trace
}
```

Или

```
throw new Exception("Outer exception", e);
```

# Свои классы-исключения

```
public class MyException : Exception
{
    public MyException()
    {
    }

    public MyException(string message)
        : base(message)
    {
    }
}
```

# Идеологически правильное объявление исключения

[Serializable]

```
public class MyException : Exception
{
    public MyException() { }
    public MyException(string message) : base(message) { }
    public MyException(string message, Exception inner)
        : base(message, inner) { }
    protected MyException(
        System.Runtime.Serialization.SerializationInfo info,
        System.Runtime.Serialization.StreamingContext context)
        : base(info, context) { }
}
```

## “Интересные” классы-исключения

- ▶ Corrupted state exceptions (CSE) — не ловятся catch-ем
  - ▶ StackOverflowException
  - ▶ AccessViolationException
  - ▶ System.Runtime.InteropServices.SEHException
- ▶ FileLoadException, BadImageFormatException, InvalidProgramException, ...
- ▶ ThreadAbortException
- ▶ TypeInitializationException
- ▶ TargetInvocationException
- ▶ OutOfMemoryException
- ▶ Можно кидать null. При этом рантайм сам кидает NullReferenceException.

# Environment.FailFast

```
try
{
    // Делать что-то полезное
}
catch (OutOfMemoryException e)
{
    Console.WriteLine("Закончилась память :(");
    Environment.FailFast(
        String.Format($"Out of Memory: {e.Message}"));
}
```



# Исключения и тесты

```
[TestMethod]
[ExpectedException(typeof(DivideByZeroException))]
public void DivideTest()
{
    var target = new DivisionClass();
    int numerator = 4;
    int denominator = 0;
    int actual = target.Divide(numerator, denominator);
}
```

или (более правильно)

```
[TestMethod]
public void DivideTest()
{
    var target = new DivisionClass();
    int numerator = 4;
    int denominator = 0;
    Assert.Throws<DivideByZeroException>(target.Divide(numerator, denominator));
}
```

# Исключения, best practices

- ▶ Не бросать исключения без нужды
  - ▶ В нормальном режиме работы исключения бросаться не должны вообще
- ▶ Свои исключения наследовать от `System.Exception`
- ▶ Документировать все свои исключения, бросаемые методом
- ▶ Не ловить `Exception`, `SystemException`
  - ▶ Исключения, указывающие на ошибку в коде (например, `NullReferenceException`) уж точно не ловить
- ▶ По возможности кидать библиотечные исключения или их наследников:
  - ▶ `InvalidOperationException`
  - ▶ `ArgumentException`
- ▶ Имя класса должно заканчиваться на “Exception”
- ▶ Код должен быть безопасным с точки зрения исключений
  - ▶ Не забывать про `finally`
  - ▶ Или `using`, `lock`, `foreach`, которые тоже генерят `finally`

## using, IDisposable

```
public static class Program {  
    public static void Main() {  
        // Create the bytes to write to the temporary file.  
        var bytesToWrite = new Byte[] { 1, 2, 3, 4, 5 };  
        // Create the temporary file.  
        using (var fs = new FileStream("Temp.dat", FileMode.Create)) {  
            // Write the bytes to the temporary file.  
            fs.Write(bytesToWrite, 0, bytesToWrite.Length);  
        }  
        // Delete the temporary file.  
        File.Delete("Temp.dat");  
    }  
}
```

# Особенности современного C#

## Кортежи и деконструкция

```
static (int prev, int cur) Fibonacci(int n)
{
    var (prevPrev, prev) = n <= 2 ? (0, 1) : Fibonacci(n - 1);
    return (prev, prevPrev + prev);
}
```

```
private static void Main(string[] args)
{
    var (_, result) = Fibonacci(7);
    Console.WriteLine(result);
}
```

## Именованные поля кортежей

```
int count = 5;  
string label = "Colors used in the map";  
var pair = (count: count, label: label);
```

```
int count = 5;  
string label = "Colors used in the map";  
var pair = (count, label); // element names are "count" and "label"
```

# Инициализация свойств

```
public class Person
{
    public int Age { get; set; } = 0;
    public string Name { get; set; } = "Anonymous";
}
```

# Null propagation

```
var first = person?.FirstName;
```

```
int? age = person?.Age;
```

```
if (age.HasValue)
```

```
{
```

```
    int realAge = age.Value;
```

```
}
```

```
var otherFirst = person?.FirstName ?? "Unspecified";
```

# Локальные функции

```
public int Fibonacci(int x)
{
    if (x < 0)
        throw new ArgumentException("Less negativity please!", nameof(x));
    return Fib(x).current;

    (int current, int previous) Fib(int i)
    {
        if (i == 0) return (1, 0);
        var (p, pp) = Fib(i - 1);
        return (p + pp, p);
    }
}
```



# Индексеры

```
class SampleCollection {  
    private int[] arr = new int[100];  
    public int this[int i]  
    {  
        get  
        {  
            return arr[i];  
        }  
        set  
        {  
            arr[i] = value;  
        }  
    }  
}
```

# Индексная инициализация

```
private Dictionary<int, string> webErrors = new Dictionary<int, string>
{
    [404] = "Page not Found",
    [302] = "Page moved, but left a forwarding address.",
    [500] = "The web server can't come out to play today."
};
```

## Именованные и необязательные аргументы

```
PrintOrderDetails(productName: "Red Mug", 31, "Gift Shop");
```

```
public void ExampleMethod(int required,  
    string optionalStr = "default string",  
    int optionalInt = 10)
```

```
anExample.ExampleMethod(3, optionalInt: 4);
```

# params

```
public static void UseParams(params int[] list)
{
    for (int i = 0; i < list.Length; i++)
    {
        Console.Write(list[i] + " ");
    }
    Console.WriteLine();
}

...
UseParams(1, 2, 3, 4);
```

## switch expression (C# 8)

<https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8>

```
public static RGBColor FromRainbow(Rainbow colorBand) =>
    colorBand switch
    {
        Rainbow.Red    => new RGBColor(0xFF, 0x00, 0x00),
        Rainbow.Orange => new RGBColor(0xFF, 0x7F, 0x00),
        Rainbow.Yellow => new RGBColor(0xFF, 0xFF, 0x00),
        Rainbow.Blue   => new RGBColor(0x00, 0x00, 0xFF),
        Rainbow.Indigo => new RGBColor(0x4B, 0x00, 0x82),
        Rainbow.Violet => new RGBColor(0x94, 0x00, 0xD3),
        _ => throw new ArgumentException(
            message: "invalid enum value",
            paramName: nameof(colorBand)),
    };

```

## switch по свойствам (C# 8)

```
public static decimal ComputeSalesTax(Address location,  
    decimal salePrice) =>  
    location switch  
{  
    { State: "WA" } => salePrice * 0.06M,  
    { State: "MN" } => salePrice * 0.75M,  
    { State: "MI" } => salePrice * 0.05M,  
    // ...  
    _ => 0M  
};
```

## switch по кортежам (C# 8)

```
public static string RockPaperScissors(string first, string second)
=> (first, second) switch
{
    ("rock", "paper") => "rock is covered by paper. Paper wins.",
    ("rock", "scissors") => "rock breaks scissors. Rock wins.",
    ("paper", "rock") => "paper covers rock. Paper wins.",
    ("paper", "scissors") => "paper is cut by scissors. Scissors wins.",
    ("scissors", "rock") => "scissors is broken by rock. Rock wins.",
    ("scissors", "paper") => "scissors cuts paper. Scissors wins.",
    (_, _) => "tie"
};
```

# Deconstruct (C# 8)

```
public class Point
{
    public int X { get; }
    public int Y { get; }

    public Point(int x, int y) => (X, Y) = (x, y);

    public void Deconstruct(out int x, out int y) =>
        (x, y) = (X, Y);
}
```



## Позиционные шаблоны (C# 8)

```
static string Quadrant(Point p) => p switch
{
    (0, 0) => "origin",
    (var x, var y) when x > 0 && y > 0 => "Quadrant 1",
    (var x, var y) when x < 0 && y > 0 => "Quadrant 2",
    (var x, var y) when x < 0 && y < 0 => "Quadrant 3",
    (var x, var y) when x > 0 && y < 0 => "Quadrant 4",
    (var x, var y) => "on a border",
    _ => "unknown"
};
```

## using var (C# 8)

```
static void WriteLinesToFile(IEnumerable<string> lines)
{
    using var file = new System.IO.StreamWriter("WriteLines2.txt");
    foreach (string line in lines)
    {
        // If the line doesn't contain the word 'Second',
        // write the line to the file.
        if (!line.Contains("Second"))
        {
            file.WriteLine(line);
        }
    }
    // file is disposed here
}
```

## Nullable reference-типы (C# 8)

- ▶ **string?** name; — переменная может быть **null**
- ▶ **string** name; — переменная не может быть **null**
  - ▶ Попытки присвоить **null** или что-то, что может быть **null**, вызовут предупреждения компилятора
- ▶ name!.Length — null-forgiving operator
- ▶ Как включить:

```
<PropertyGroup>  
  <Nullable>enable</Nullable>  
</PropertyGroup>
```

в проектнй файл

- ▶ Управляется директивами `#nullable disable` и `#nullable enable`
- ▶ Подробности:  
<https://docs.microsoft.com/en-us/dotnet/csharp/nullable-references>
- ▶ Для нового кода обязательно к применению

## Продвинутое сопоставление шаблонов (C# 9)

```
public static bool IsLetterOrSeparator(char c) =>  
    c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z') or '.' or ',';
```

```
if (e is not null)  
{  
    // ...  
}
```

## Продвинутый вывод типов (C# 9)

```
private List<WeatherObservation> observations = new();
```

```
public WeatherForecast ForecastFor(  
    DateTime forecastDate, WeatherForecastOptions options)
```

```
...
```

```
var forecast = station.ForecastFor(DateTime.Now.AddDays(2), new();
```

```
WeatherStation station = new() { Location = "Seattle, WA" };
```

## Record-ы (C# 9)

```
public record Person
{
    public string LastName { get; }
    public string FirstName { get; }

    public Person(string first, string last)
        => (FirstName, LastName) = (first, last);
}
```