Прихващане на изключения

Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/

Съдържание

1. Какво са изключенията?
2. Прихващане на изключения
3. Класът `System.Exception`
4. Свойства на изключенията
5. Йерархия на изключенията в C#

Какво са изключенията?

- Изключенията в .NET Framework / Java са класическа реализация на модела на изключенията в ООП
- Предоставят мощен механизъм за централизирано прихващане на грешки и необичайни събития
- Заменят процедурно-ориентирания подход, при който всяка функция връща код за грешка
- Опростяват изграждането и поддръжката на кода
- Позволяват проблематични ситуации да бъдат обработени на множество нива

Прихващане на изключенията

- В C# могат да бъдат прихванати чрез **try-catch-finally** конструкция

```
try
{
    // Вършим някаква работа, която може да породи изключение
}
catch (SomeException)
{
    // Прихващаме хвърленото изключение
}
```

- **catch** блоковете могат да бъдат използвани многократно за обработка на различни типове изключения

Прихващане на изключения – пример

```
static void Main()
{
    string s = Console.ReadLine();
    try
    {
        int.Parse(s);
        Console.WriteLine(
           "You entered a valid Int32 number {0}.", s);
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid integer number!");
    }
    catch (OverflowException)
    {
        Console.WriteLine(
           "The number is too big to fit in Int32!");
    }
}
```

Класът `System.Exception`

- Изключенията в C# / .NET са обекти
- **`System.Exception`** е базов клас за всички изключения в CLR
  - Съдържа информация за причината на грешката / необичайната ситуация
    - **`Message`** – текст, описващ изключението
    - **`StackTrace`** – снимка на стека в момента на хвърлянето на изключението
    - **`InnerException`** – изключението, породило текущото (ако има)
- Подобни са нещата и в Java и PHP

Свойства на изключенията – пример

```
class ExceptionsExample
{
    public static void CauseFormatException()
    {
        string str = "an invalid number";
        int.Parse(str);
    }
    static void Main()
    {
        try
        {
            CauseFormatException();
        }
        catch (FormatException fe)
        {
            Console.Error.WriteLine("Exception: {0}\n{1}",
                fe.Message, fe.StackTrace);
        }
    }
}
```

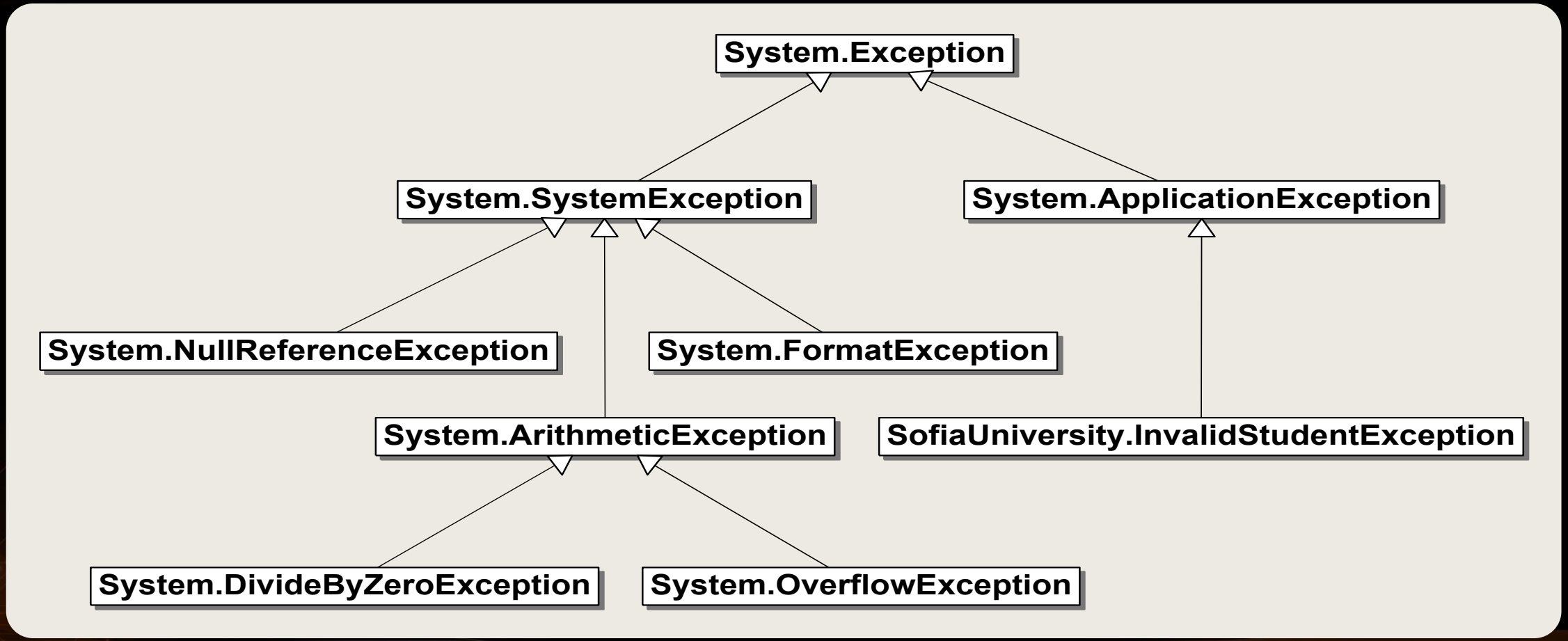
Свойства на изключенията

- Свойството **`Message`** дава кратко описание на проблема
- Свойството **`StackTrace`** е изключително полезно за откриване на причината, породила изключението

```
Exception caught: Input string was not in a correct format.
   at System.Number.ParseInt32(String s, NumberStyles style,
NumberFormatInfo info)
   at System.Int32.Parse(String s)
   at ExceptionsTest.CauseFormatException() in
c:\consoleapplication1\exceptionstest.cs:line 8
   at ExceptionsTest.Main(String[] args) in
c:\consoleapplication1\exceptionstest.cs:line 15
```

Йерархия на изключенията в .NET

- Изключенията в .NET Framework са организирани в йерархия



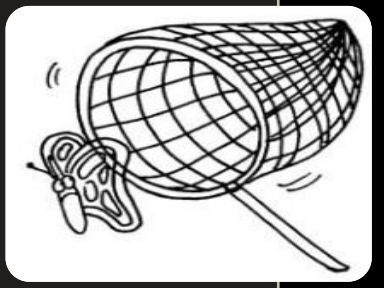
Типове изключения

- Изключенията в .NET са наследници на `System.Exception`
- Системните изключения наследяват `System.SystemException`
  - `System.ArgumentException`
  - `System.FormatException`
  - `System.NullReferenceException`
  - `System.OutOfMemoryException`
  - `System.StackOverflowException`
- Потребителските трябва да наследяват `System.Exception`

Прихващане на изключения

- Когато се прихваща изключение от даден клас, всички негови наследници (наследени изключения) също се прихващат:

```
try
{
    // Do some work that can cause an exception
}
catch (System.ArithmeticException)
{
    // Handle the caught arithmetic exception
}
```



- Прихваща **ArithmeticException** и всички негови наследници **DivideByZeroException** и **OverflowException**

Открийте грешката!

```
static void Main()
{
    string str = Console.ReadLine();
    try
    {
        Int32.Parse(str);
    }
    catch (Exception)
    {
        Console.WriteLine("Cannot parse the number!");
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid integer number!");
    }
    catch (OverflowException)
    {
        Console.WriteLine(
          "The number is too big to fit in Int32!");
    }
}
```

Това трябва да е последно

Никога не се стига дотук

Никога не се стига дотук

Прихващане на всички изключения

- Всички изключения, генерирани в .NET контролиран код наследяват класа **`System.Exception`**
- Неконтролираният код хвърля други изключения
- За прихващане на абсолютно всички изключения използвайте:

```
try
{
    // Do some work that can raise any exception
}
catch
{
    // Handle the caught exception
}
```

Конструкцията Try-finally

- Конструкцията:

```
try
{
    // Do some work that can cause an exception
}
finally
{
    // This block will always execute
}
```

- Подсигурява изпълнението на даден блок във всички случаи
  - Независимо дали ще се генерира изключение в `try` блока
- Използва се за изпълнение на разчистващия код (например освобождаване на заделените в конструкцията ресурси)

Try-finally – пример

```
static void TestTryFinally()
{
    Console.WriteLine("Code executed before try-finally.");
    try
    {
        string str = Console.ReadLine();
        int.Parse(str);
        Console.WriteLine("Parsing was successful.");
        return; // Exit from the current method
    }
    catch (FormatException)
    {
        Console.WriteLine("Parsing failed!");
    }
    finally
    {
        Console.WriteLine("This cleanup code is always executed.");
    }
    Console.WriteLine("This code is after the try-finally block.");
}
```

Командата "Using"

- В програмирането често се ползва "Dispose" шаблона
  - Така се подсигуряваме, че всички ресурси са коректно затворени

```
Resource resource = AllocateResource();
try {
    // Use the resource here …
} finally {
    if (resource != null) resource.Dispose();
}
```

- Същият ефект може да се постигне и чрез "**using**" израза в C#:

```
using (<resource>)
{
   // Use the resource. It will be disposed (closed) at the end
}
```

Четене на текстов файл – пример

- Чете и извежда текстов файл ред по ред:

```
StreamReader reader = new StreamReader("somefile.txt");
using (reader)
{
    int lineNumber = 0;
    string line = reader.ReadLine();
    while (line != null)
    {
        lineNumber++;
        Console.WriteLine("Line {0}: {1}", lineNumber, line);
        line = reader.ReadLine();
    }
}
```

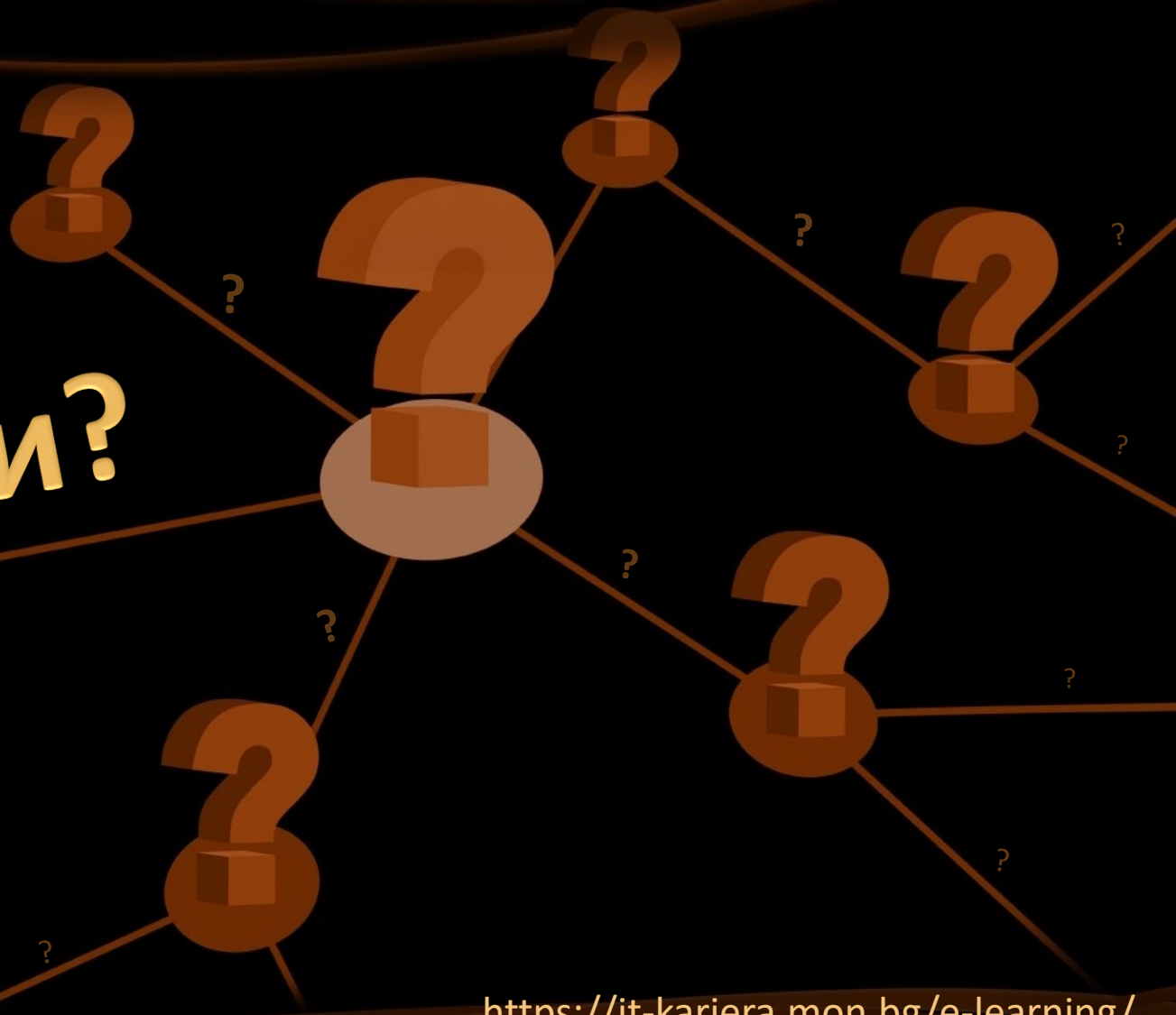
Обобщение

- Изключенията са гъвкав механизъм за обработка на грешки
  - Позволяват грешките да бъдат прихванати на множество нива
  - Всеки прихващач на изключения обработва само грешки от даден тип (и подтиповете му)
    - Другите типове грешки се обработват от други прихващачи по-късно
  - Необработените изключения извеждат съобщения за грешка

- `Try-finally` конструкцията гарантира, че даден блок с код ще се изпълни винаги (дори когато хвърлено изключение)

Прихващане на изключения

Въпроси?



https://it-kariera.mon.bg/e-learning/

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**