

# Прихващане и хвърляне 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#
6. **Генериране (хвърляне)** на изключения
7. Избор на типа на изключението
8. Препоръки при работа с изключения
9. Създаване на **потребителски изключения**



# 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 породви изключение
}
catch (SomeException)
{
    // Прихващаме хвърленото изключение
}
```



- **catch** блоковете могат да бъдат добавени 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!");
    }
}
```



# Свойства на изключенията

- **Message** дава кратко описание на проблема
- **StackTrace** дава снимка на стека в момента на 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
```

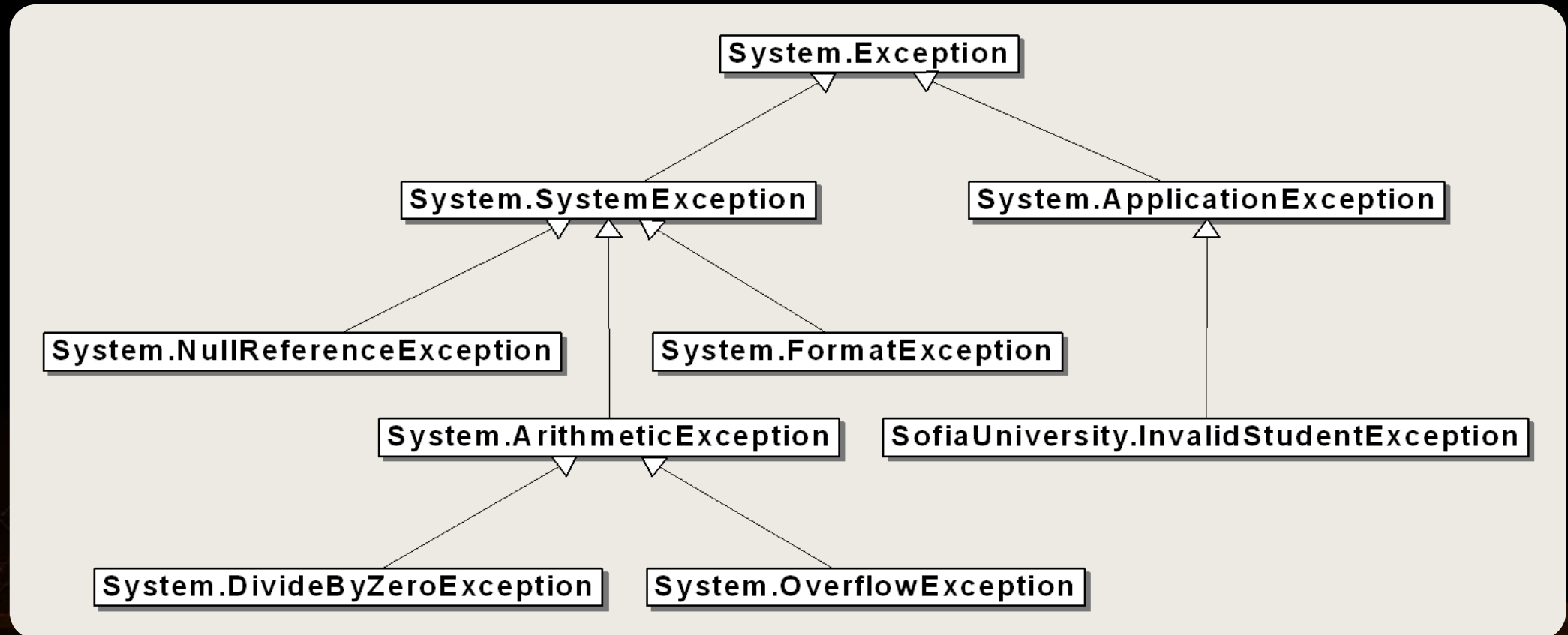
- **InnerException** – изключението, породило текущото (ако има)

# 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);
        }
    }
}
```

# Йерархия на изключенията в .NET

- Изключенията в C# / .NET са класове, организирани в йерархия
- **System.Exception** е базов клас за всички изключения в CLR





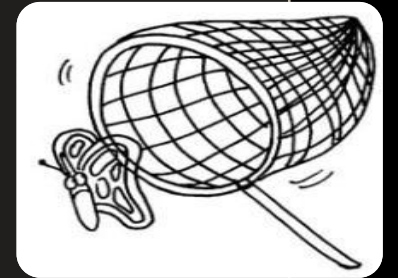
# Типове изключения

- 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 подsigуряваме, 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:

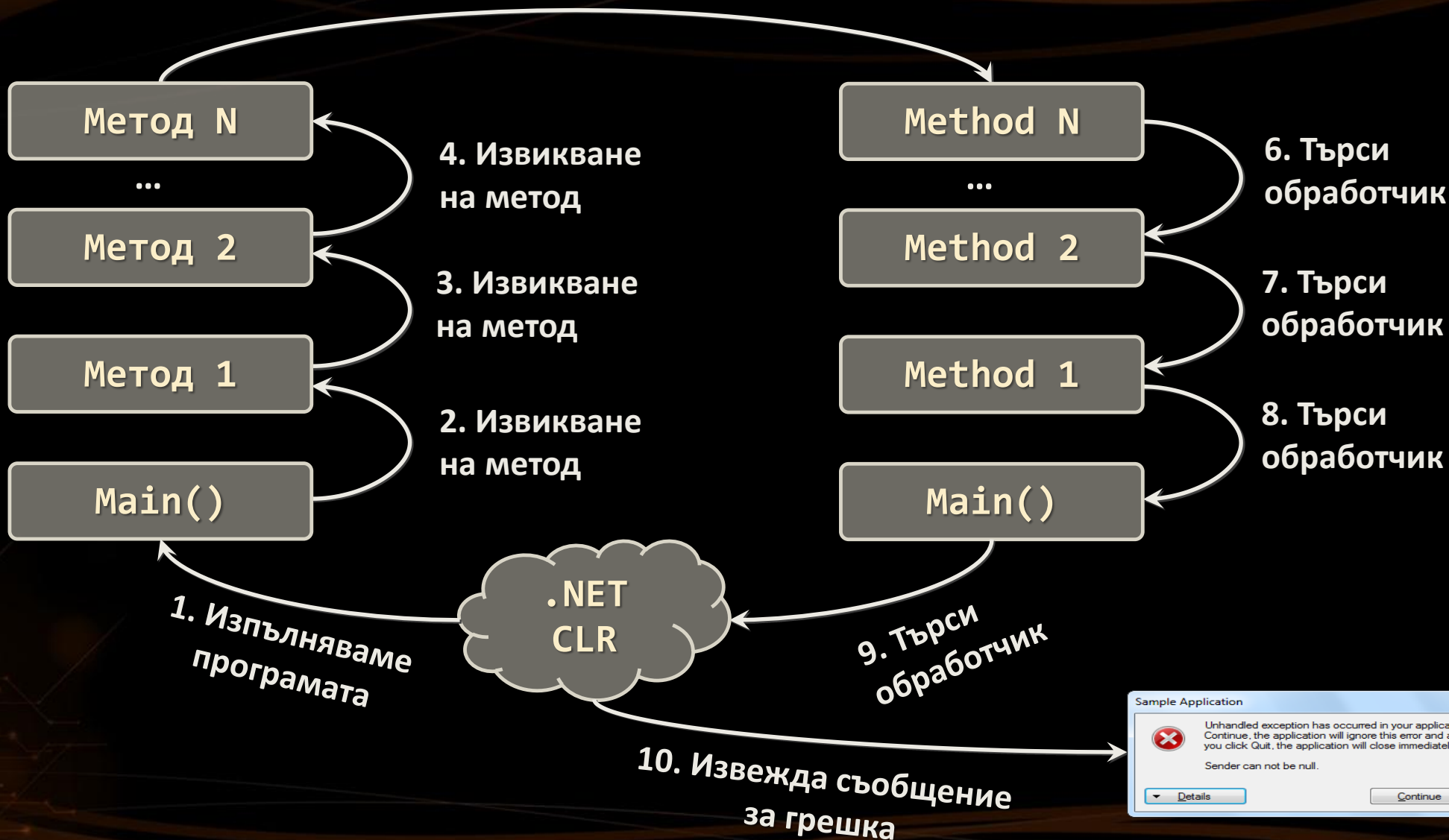
```
StreamReader reader = new StreamReader("somefile.txt");  
using (reader)  
{  
    int lineNumber = 0;  
    string line = reader.ReadLine();  
    while (line != null)  
    {  
        lineNumber++;  
        Console.WriteLine("Line {0}: {1}", lineNumber, line);  
        line = reader.ReadLine();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





# Как работи прихващането на изключенията?

## 5. Хвърляне на изключение



# Хвърляне на изключения

- Изключенията се хвърлят (пораждат) чрез командата **throw**
- Целта е уведомяване на кода, извикал текущия програмен блок, за грешка или друга необичайна ситуация
- Когато се хвърля изключение:
  - Изпълнението на програмата спира
  - Изключението пътува през стека
    - Докато не достигне подходящ **catch** блок, който да го прихване
- Неприхванатите изключения извеждат съобщение за грешка

# Използване на командата throw

- **Хвърляне** на изключение със съобщение за грешка:

```
throw new ArgumentException("Invalid amount!");
```

- Изключението може да приема съобщение и причина:

```
try
{
    ...
}
catch (SQLException sqlEx)
{
    throw new InvalidOperationException("Cannot save invoice.", sqlEx);
}
```

- Бележка: ако и оригиналното изключение не бъде подадено като параметър, ще загубим първоначалната причина за изключението

# Повторно хвърляне на изключение

- Прихванатите изключения може да бъдат хвърлени наново:

```
try
{
    Int32.Parse(str);
}
catch (FormatException fe)
{
    Console.WriteLine("Parse failed!");
    throw fe; // Re-throw the caught exception
}
```

```
catch (FormatException)
{
    throw; // Re-throws the last caught exception
}
```



# Хвърляне на изключения - пример

```
public static double Sqrt(double value)
{
    if (value < 0)
        throw new System.ArgumentOutOfRangeException("value",
            "Sqrt for negative numbers is undefined!");
    return Math.Sqrt(value);
}

static void Main()
{
    try
    {
        Sqrt(-1);
    }
    catch (ArgumentOutOfRangeException ex)
    {
        Console.Error.WriteLine("Error: " + ex.Message);
        throw;
    }
}
```



# Избиране на типа на изключението

- Когато се подаде невалидна стойност в параметър на метод:
  - **ArgumentException, ArgumentNullException, ArgumentOutOfRangeException**
- Когато заявената операция не се поддържа
  - **NotSupportedException**
- Когато методът все още не е реализиран
  - **NotImplementedException**
- Когато няма друг подходящ стандартен клас изключения
  - Създайте ваш собствен клас (наследяващ **Exception**)

# Препоръки при работа с изключения

- **catch** блоковете трябва да започват с изключенията, които са най-ниско в йерархията (т.е. с най-специфичните)
  - И да продължават с по-общите изключения
  - В противен случай ще има грешка при компилация
- Всеки **catch** трябва да обработва само тези изключения, които очаква
  - Ако метод не е компетентен да обработи дадено изключение, той би трябвало да го остави неприхванато
  - Прихващането на всички изключения, независимо от какъв тип са, е популярна лоша практика (анти-шаблон)!

## Препоръки при работа с изключения(2)

- Когато генерирате изключение, винаги подавайте на конструктора достатъчно говорящо **пояснително съобщение**
  - Когато хвърляте изключение, винаги подавайте добро описание на проблема, който го е предизвикал
    - Съобщението на изключението трябва да обяснява какво е породило проблема и как той може да бъде решен
    - Добро: „Размерът трябва да е число в диапазона [1...15]" 
    - Добро: „Невалидно състояние. Извикайте първо Initialize()"
    - Лошо: „Неочакван проблем"
    - Лошо: „Невалиден аргумент"
- 



# Препоръки при работа с изключения(3)

- Изключенията може да намалят производителността на приложението
  - Затова ги хвърляйте само в ситуации, които са наистина необичайни и трябва да бъдат обработени
  - Не хвърляйте изключения при нормалната работа на програма
  - CLR може да хвърли изключения по всяко време, няма как да бъде предвидено това
    - Например **System.OutOfMemoryException**

# Създаване на потребителски изключения

- Потребителските изключения наследяват някой от класовете изключения (най-често **System.Exception**)

```
public class TankException : Exception
{
    public TankException(string msg)
        : base(msg)
    {
    }
}
```

- Те се хвърлят като всяко друго изключение

```
throw new TankException("Not enough fuel to travel");
```

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



## Обобщение (2)

- Изключенията се хвърлят (пораждат) чрез командата **throw**
- Когато се хвърля изключение:
  - Изпълнението на програмата спира
  - Изключението пътува през стека, докато не бъде прихванато от **catch** блок
  - Всеки **catch** трябва да обработва само тези изключения, които очаква
- Прихванато изключение може да бъде хвърлено наново
- Неприхванатите изключения извеждат съобщение за грешка

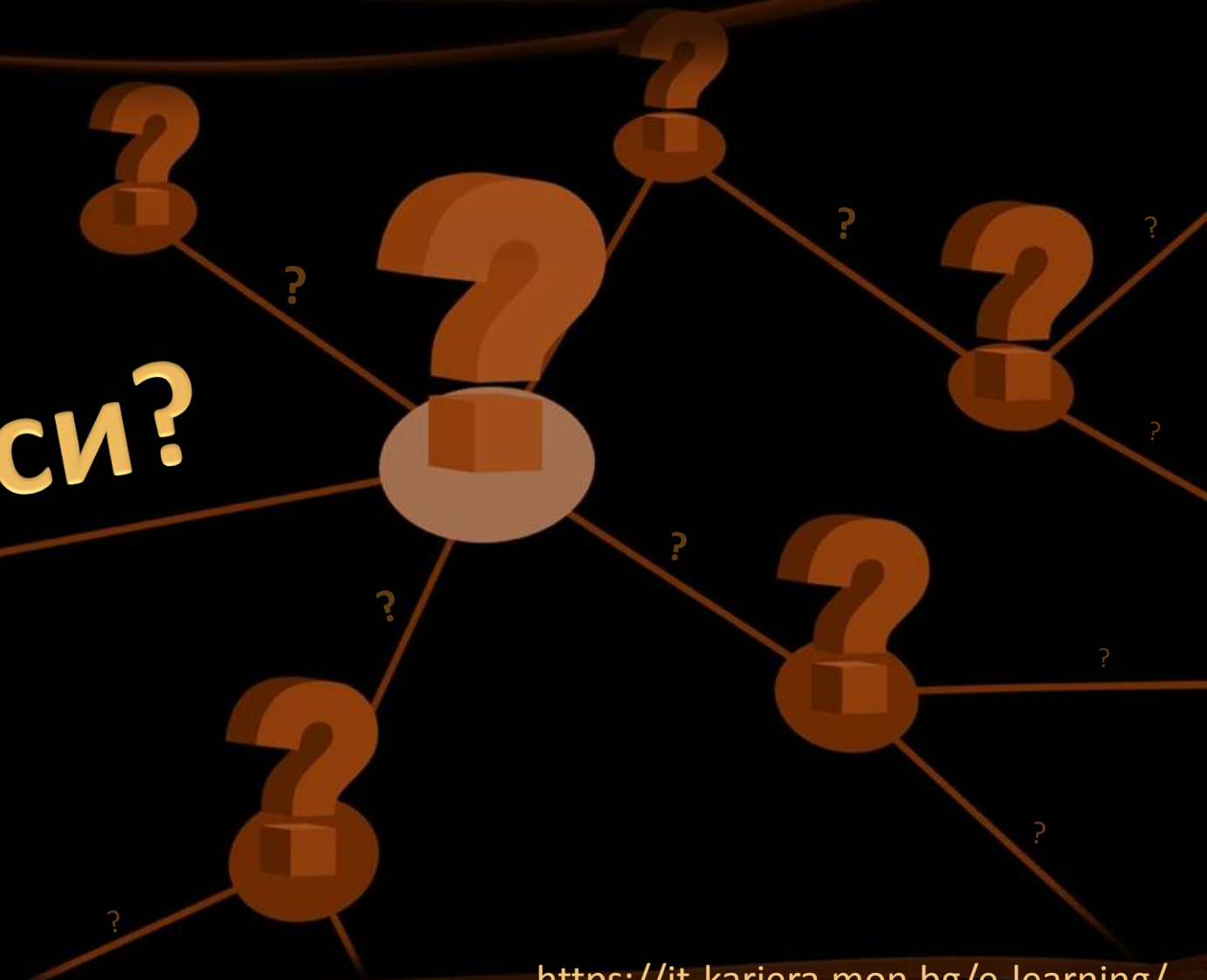




# Прихващане и хвърляне на изключения



Въпроси?





# Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"



- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Основи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA