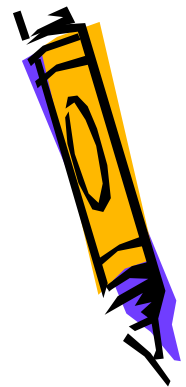




Лекция 11. Исключения в .NET

1. Основна концепция при обработка на изключения

- Изключение се нарича НЕОЧАКВАНО СЪБИТИЕ, което възниква по време на изпълнение на програмата, заради което програмата не може да бъде изпълнена успешно.
- Изключения могат да настъпят при:
 - А) грешки в кода, например:
 - Деление на 0 - `System.DividedByZeroException`
 - Излизане от границите на масив - `System.IndexOutOfRangeException`
 - Не може да се намери файл - `System.IO.FileNotFoundException`
 - Б) при изчерпване на ресурс на операционната система, например изчерпва се наличната работна памет (`System.OutOfMemoryException`), при препълване на стека (`System.StackOverflowException`), неочаквано поведение в .NET средата (примерно невъзможност за верификация на даден код) и в много други ситуации.



- Изключение (exception) в .NET представлява събитие, което уведомява програмиста, че е възникнало непредвидено обстоятелство (НЕОЧАКВАНО СЪБИТИЕ): методът, в който е възникнало непредвиденото обстоятелство (грешката) хвърля изключение - това е специален обект, съдържащ информация за вида на грешката, мястото в програмата, където е възникнала и състоянието на програмата (stack trace) в момента на възникване на грешката.
- Така, ако по време на изпълнение на програмата някой от извикваните от метод А методи, например, метод В неочаквано хвърли изключение, то как реагира CLR?
- Когато възникне изключение, изпълнението на програмата спира. CLR запазва състоянието на стека и го претърсва: търси блока код (handler), който да обработи възникналото изключение. Ако не го намери в границите на текущия метод В, CLR търси в извикващия метод А. Ако и в него не го намери, то търси метода по-надолу и т. н. Ако никой от извикващите методи не прихване изключението, то изключението се прихваща от CLR, който извежда на потребителя информация за възникналия проблем.



Моделът на обработка на
изключенията ЕН (Exception
Handling) в .Net се базира на 2
концепции:

1. Всяко хвърлено изключение е
обект от клас `System. Exception` или
неговите наследници
2. Организиране на защитавани
блокове: `try - catch`



2. Прихващане, обработка и генериране (хвърляне) на изключения

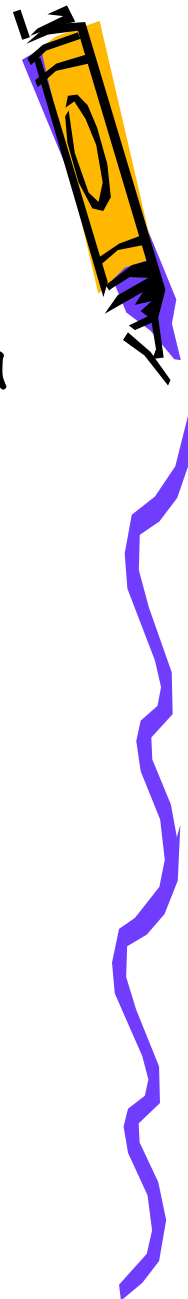
И така, как един метод реагира на изключение?

1. Прихваща и обработва изключението (дефинира обработчици, т.е handler-и на изключение).
2. Генерира (предизвиква, хвърля - throw) ново изключение от същия или различен тип, което да се обработи от друг handler (в друг метод).
3. Пропуска изключението, тоест не реагира, с идеята, че изключението **може би** ще се обработи от handler в някой от извикващите методи.



2.1. Прихващане и обработка на изключения - Общ синтаксис

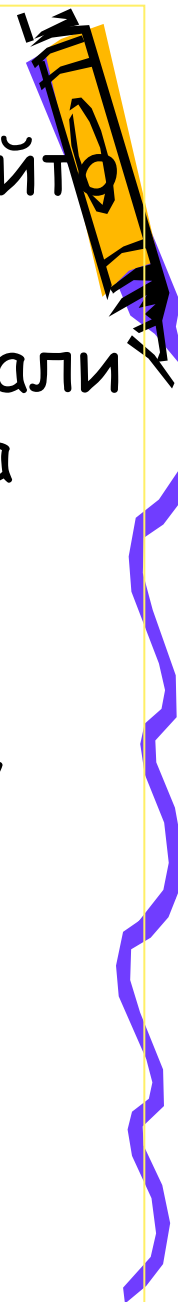
```
try
{ // Критичен код, който може да предизвика
  изключение
}
catch (<изключение 1>)
{ // Обработка се изключение 1 }
catch (<изключение 2>)
{ // Обработка се изключение 2 } ...
finally
{ //довършителна работа }
```



- Особености:
- В **try** блока се пише критичния код (код, който може да предизвика изключение).

Резонен е въпроса: Как може да се прецени дали един код е критичен? Единствено на база на документацията може да се прецени кой метод, какви изключения генерира.

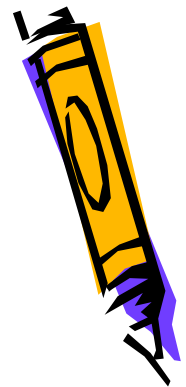
- След него има 0 или повече **catch** блокове, това са обработчици (**handler** - и) на изключения, като всеки **catch** прихваща и обработва определен тип изключение.



Синтаксис на catch блок:

- `catch (<изключение 2>){... }`
- `catch (<изключение 2>
<идентификационни променливи>
{... }`
- `catch {... }`

В първите 2 случая, в скоби след catch се указва типа (ExceptionType) и понякога името (ExceptionType objectName) на прихващаното изключение (нар. филтър за прихващане на изключения).



- Ако филтърът бъде пропуснат, се прихващат всички изключения, независимо от типа им. Такъв `catch` се нарича общ, трябва да бъде последен и трябва да бъде само един.
- `catch` изразът може да присъства няколко пъти в `try`, съответно за различни типове изключения, които трябва да бъдат прихванати:
`catch (FormatException e) {...}`
`catch (OverflowException) {...}`
`catch {Всяко изключение се обработва тук }`



- Редът на catch филтрите е важен, защото catch филтрите се обхождат отгоре надолу:

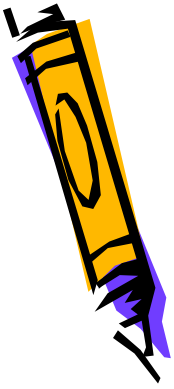
- Те трябва да са подредени така, че да започват от изключенията най-ниско в йерархията и да продължават с по-общите.
- Така ще бъдат обработени първо по-специфичните изключения и след това по-общите.
- В противен случай кодът за по-специфичните никога няма да се изпълни.



- Докато управляваният .NET код може да предизвика само изключения, наследници на `System.Exception`, то неуправляваният код може да предизвика и други изключения. Общият `catch` се използва за прихващане на всякакви други изключения в C#.
- `Finally` блокът се изпълнява винаги, както при нормално изпълнение на кода в `try`, така и при възникване на изключение. Затова в този блок се освобождават системни ресурси, например затварят се файлове и конекции с БД.



- В примера по-долу има недостижим код, защото първи е `catch` блока, който ще обработи всички типове изключения
(`catch (System.Exception)...`).
- По-специфичните блокове след него няма да се изпълнят никога:



```
static void Main()
```

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

```
try {Int32.Parse(s);}
```

```
catch (System.Exception)
```

```
// Трябва да е най-накрая
```

```
{Console.WriteLine("Can not parse the number!"); }
```

```
catch (FormatException)
```

```
// Този код е недостижим
```

```
{Console.WriteLine("Invalid integer number!"); }
```

```
catch (OverflowException)
```

```
// Този код е недостижим
```

```
{Console.WriteLine("The number is too big!");}
```

```
}
```



```
static void Main()
```

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

```
try {Int32.Parse(s);}
```

```
catch (System.Exception)
```

```
// Трябва да е най-накрая
```

```
{Console.WriteLine("Can not parse the number!"); }
```

```
catch (FormatException)
```

```
// Този код е недостижим
```

```
{Console.WriteLine("Invalid integer number!"); }
```

```
catch (OverflowException)
```

```
// Този код е недостижим
```

```
{Console.WriteLine("The number is too big!");}
```

```
}
```



- Всички възможни изключения ли трябва да обработваме?

Препоръката е да се залага на обработката на специални изключения, а не на - общи.

- **Пример:** Въвежда се число от конзола и се присвоява на променлива от тип `int`.



using System;
class Program

{ static void Main()

{ try { int a = Int32.Parse(Console.ReadLine()); }

catch (FormatException e)

{ Console.WriteLine(e.Message); }

catch (OverflowException)

{ Console.WriteLine("Number too big to fit
in Int32!"); }

finally { Console.WriteLine("Inside Finally"); }

Console.WriteLine("Outside Finally");

}

}



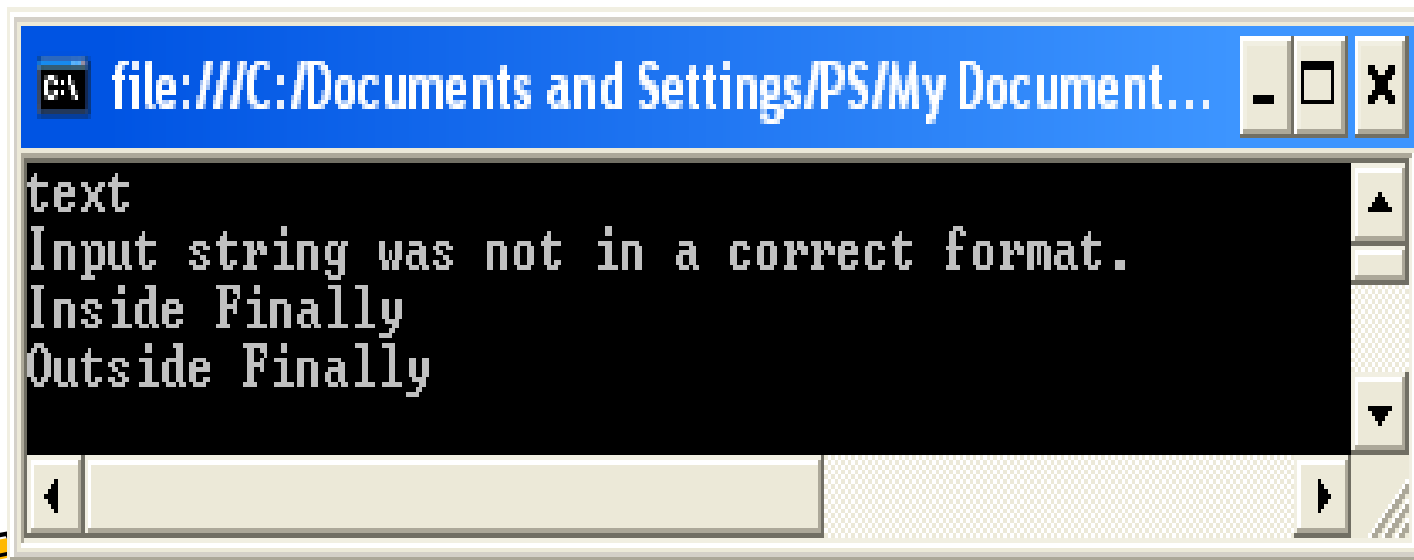
Сценарий 1: Въвеждаме число.



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "file:///D:/Violeta_new/C...". The command prompt shows the output of a script: "234", "Inside Finally", and "Outside Finally". The window has a scroll bar at the bottom.

```
file:///D:/Violeta_new/C...  
234  
Inside Finally  
Outside Finally
```

Сценарий 2: Въвеждаме текст - обработка се изключението `FormatException`



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "file:///C:/Documents and Settings/PS/My Document...". The command prompt shows the output of a script: "text", "Input string was not in a correct format.", "Inside Finally", and "Outside Finally". The window has a scroll bar at the bottom.

```
file:///C:/Documents and Settings/PS/My Document...  
text  
Input string was not in a correct format.  
Inside Finally  
Outside Finally
```



- Сценарий 3: Въвеждаме голямо цяло число и изключение `OverflowException`:



A screenshot of a Windows command prompt window. The title bar is blue and contains the text "file:///D:/Violeta_new/C#proekti1/Cons...". The command prompt has a black background with white text. The output shows a large number "1256721567154", followed by the error message "Number too big to fit in Int32!", and then the text "Inside Finally" and "Outside Finally".

```
1256721567154
Number too big to fit in Int32!
Inside Finally
Outside Finally
```

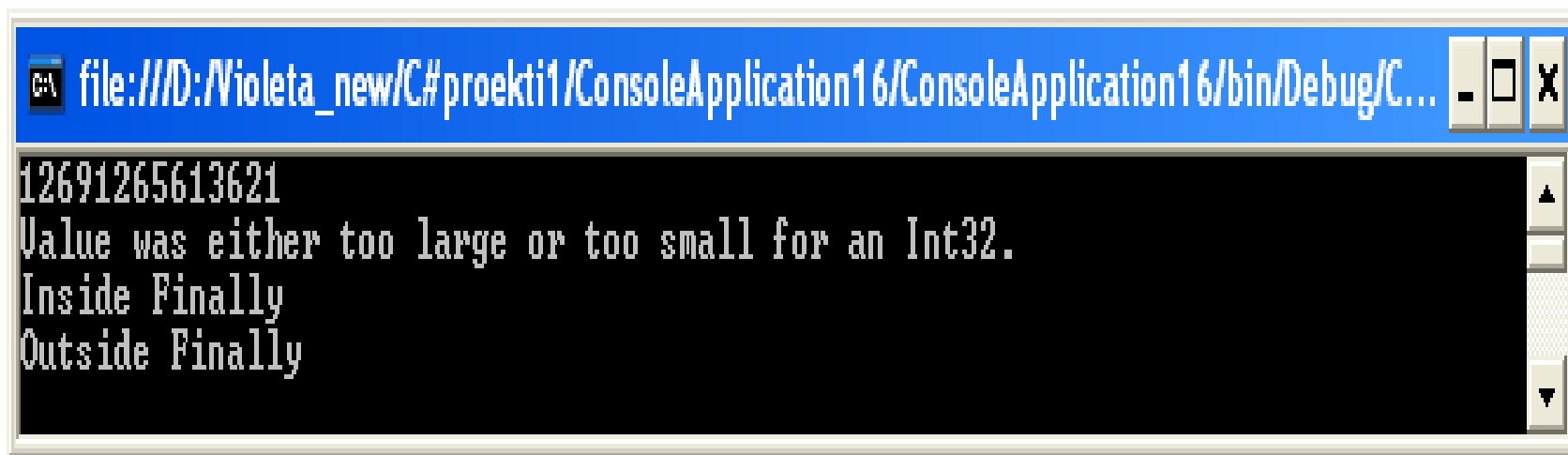
- Всеки `catch` блок е подобен на метод който приема точно един аргумент от определен тип изключение. Този аргумент може да бъде зададен само с типа на изключението, както е в по-горния пример, а може да се зададе и променлива от съответния тип:



catch (OverflowException ex)

```
{ Console.WriteLine(ex.Message); }
```

- Тук посредством от променливата `ex`, която е инстанция на класа **System.OverflowException**, можем да извлечем допълнителна информация за възникналото изключение.
- Сценарий 4 - вариант 2: Голямо цяло число:



The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "file:///D:/Violeta_new/C#proekti1/ConsoleApplication16/ConsoleApplication16/bin/Debug/C...". The command prompt area has a black background with white text. The text displayed is: "12691265613621", "Value was either too large or too small for an Int32.", "Inside Finally", and "Outside Finally".

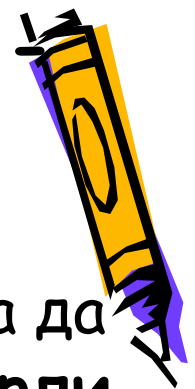
```
12691265613621
Value was either too large or too small for an Int32.
Inside Finally
Outside Finally
```

2.2. Генериране (предизвикване, хвърляне **throw**) ново изключение

- Разгледахме как в един метод се прихващат и обработват изключения, предизвикани отвън. Нека да разгледаме как методът може да **предизвика (хвърли - throw)** изключение, което да се обработи от друг метод (например, от извикващият метод).
- За да се хвърли изключение, с което да се уведоми извикващият код за даден проблем, в **C#** се използва оператора **throw**, на който се подава инстанция на класа на изключението (обект от някой наследник на класа **System.Exception**) и текстово описание на възникналия проблем. Ето един пример, в който се хвърля изключение **ArgumentException**:

throw new

System.ArgumentException("Invalid argument!");



- Обикновено преди да бъде хвърлено изключение, то се създава чрез извикване на конструктора на класа, на който то принадлежи. Почти всички изключения дефинират следните два конструктора:

Exception(string message);

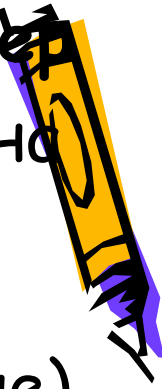
Exception(string message, Exception InnerException);

- Първият конструктор приема текстово съобщение, което описва възникналия проблем, а вторият приема и изключение = причинител на възникналия проблем.
- При хвърляне на изключение от извикания метод, CLR прекратява изпълнението на програмата и обхожда стека (отгоре-надолу) до достигане на **catch** блок, обработващ съответното изключение.



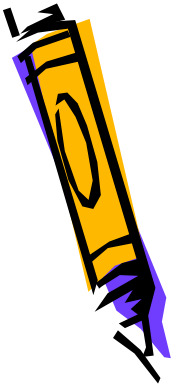
Хвърляне и прихващане на изключения - пример

- Следва един пример, демонстриращ хвърляне на изключение в даден метод (в случая `S(double aValue)`) и обработката на изключението от извикващия метод. Нека в метод `S(double aValue)` се изчислява корен квадратен на `aValue`.
Подаването на отрицателно число към метода `S(double aValue)` е една възможна ситуация при която методът `S()` хвърля изключение (`ArgumentException`). Извикващият метод `Main()` е този, който обработва изключението, след като бъде информиран за проблема от извикания метод `S()`. В `Main()` метода изключението се прихваща и се отпечатва грешка.



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

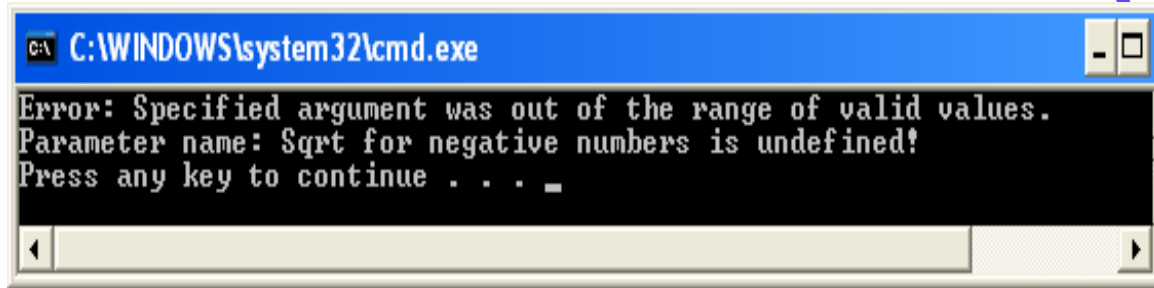
```
static void Main()
{ try {S(-1);}
  catch (ArgumentOutOfRangeException ex)
    {Console.Error.WriteLine("Error: " +
    ex.Message); }
}
```



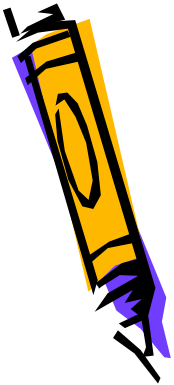
```
public static double S(double aValue)
{ if (aValue < 0)
  { throw new
    System.ArgumentOutOfRangeException(
      "Sqrt for negative numbers is undefined!");
  } return Math.Sqrt(aValue);
}
```

```
static void Main()
{ try { S(-1); }
```

```
catch (ArgumentOutOfRangeException ex)
{ Console.Error.WriteLine("Error: " +
  ex.Message); }
}
```

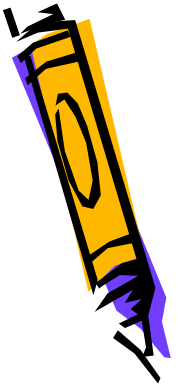


A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays an error message: "Error: Specified argument was out of the range of valid values. Parameter name: Sqrt for negative numbers is undefined! Press any key to continue . . . _". The error message is displayed in white text on a black background.



2.3. Хвърляне на прихванатото изключение

- В `catch` блокове прихванатите изключения могат да се хвърлят отново.
- Пример за такова поведение е следния: В метода `Calculate(...)` прихванатото аритметично изключение се обработва като се отпечатва на конзолата "**Calculation failed!**" и след това се хвърля отново (чрез изрази `throw;`). В резултат същото изключение се прихваща и от `try-catch` блока в `Main()` метода.



```
public static int Calculate(int a, int b)
{ try {return a/b; }
  catch (DivideByZeroException)
  { Console.WriteLine("Calculation failed!");
    throw;
```

//само в catch блок има такава форма!!!

```
  }
} static void Main()
{
  try { Calculate(1, 0); }
  catch (Exception ex)
  { Console.WriteLine(ex); }
}
```



Още един пример:

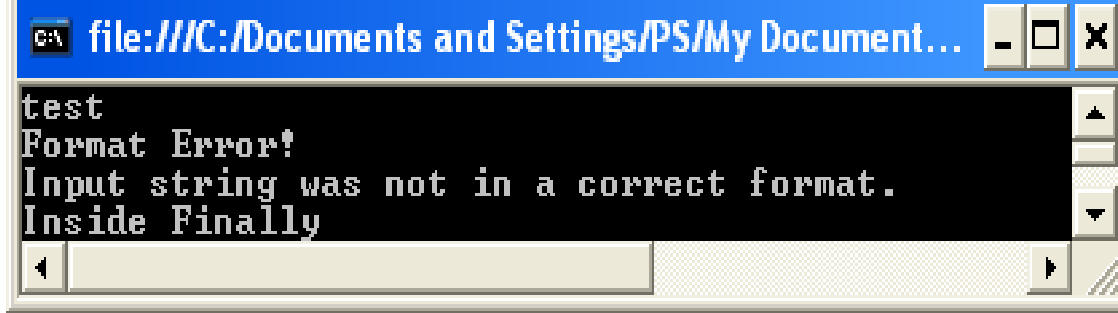
using System;

class Program

```
{    public static int C(string s)
    {    try    {    int a = Int32.Parse(s); return a;    }
        catch (FormatException)
        {    Console.WriteLine("Format Error!");
            throw; //само в catch блок има такава форма!!!
        }
    }
```

static void Main()

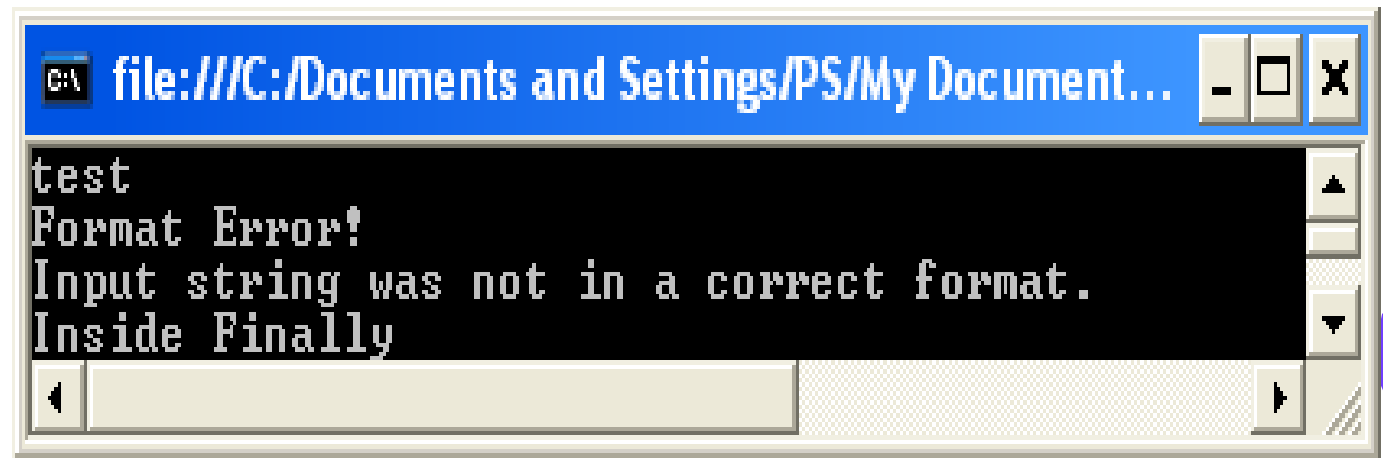
```
{ try { string s = Console.ReadLine(); C(s); }
    catch (OverflowException)
    { Console.WriteLine("Number too big to fit in Int32!"); }
    catch (FormatException ex)
    { Console.WriteLine(ex.Message); }
    finally
    { Console.WriteLine("Inside Finally"); }
    Console.ReadLine(); }
}
```



```
test
Format Error!
Input string was not in a correct format.
Inside Finally
```



- В метода `C(...)` прихванатото `FormatException` се обработва като се отпечатва на конзолата `Format Error!` и след това се хвърля отново (чрез израза `throw;`). В резултат, същото изключение се прихваща и от `try-catch` блока в `Main()` метода.



```
file:///C:/Documents and Settings/PS/My Document...
test
Format Error!
Input string was not in a correct format.
Inside Finally
```

The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "file:///C:/Documents and Settings/PS/My Document...". The command prompt has a black background with white text. The text displayed is: "test", "Format Error!", "Input string was not in a correct format.", and "Inside Finally". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. A vertical scrollbar is visible on the right side of the text area.



3. Свойства на изключенията

- Изключенията в .NET Framework са обекти. Класът **System.Exception** е базов клас за всички изключения в CLR. Той дефинира свойства, общи за всички .NET изключения, които съдържат информация за настъпилата грешка или необичайна ситуация.

Ето и някои често използвани свойства:

- **public virtual string Message { get; }** - текстово описание на грешката.



- **public virtual string StackTrace { get; }** - текстова визуализация на състоянието на стека в момента на възникване на изключението. Дава информация за метода, файла и реда във файла, в който е възникнало изключението. Имената на файловете и редовете са налични само при компилиране в Debug режим.
- **public virtual string InnerException { get; }** - извлича изключението, което е причина за възникване на текущото изключение (ако има такова). Така, ако X се хвърля като пряк резултат от изключение Y, InnerException на X съдържа референцията на Y.



using System; //Един Пример

class ExceptionsTest

{public static void CauseFormatException()
{ string s = "an invalid number";Int32.Parse(s);}

static void Main(string[] args)

{ try {CauseFormatException(); }

catch (FormatException fe)

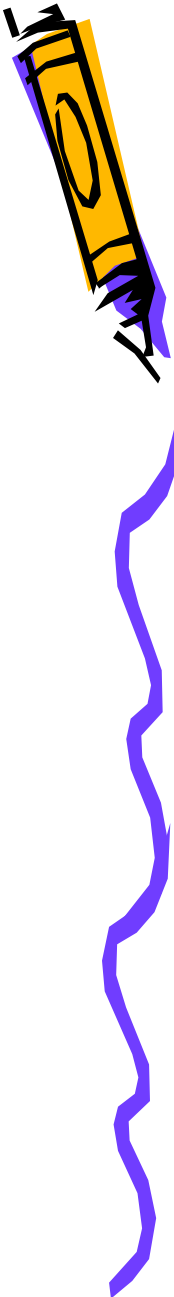
{Console.Error.WriteLine(

"Exception caught: {0}\n{1}",

fe.Message, fe.StackTrace);}

}

}

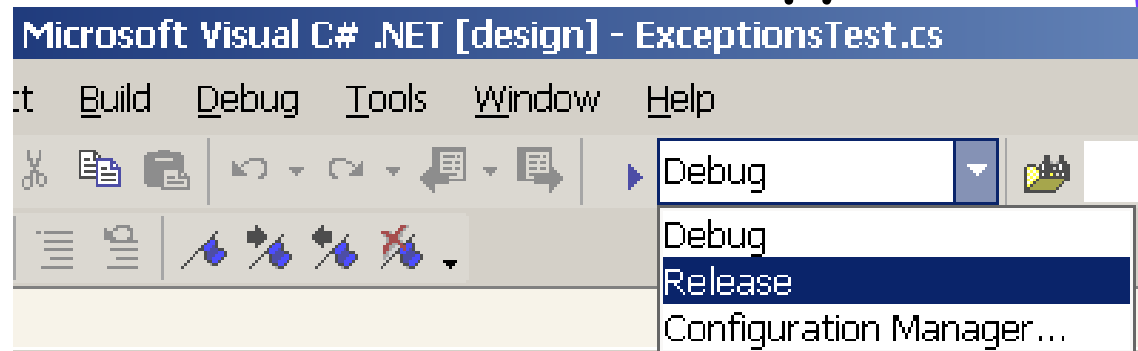


- Свойството **StackTrace** е изключително полезно при идентифициране на причината за изключението. Резултатът от примера е информация за прихванатото в **Main()** метода изключение, отпечатана върху стандартния изход за грешки:

- 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



- Имената на файловете и номерата на редовете са достъпни само ако сме компилирали с дебъг информация. Ако компилираме по-горния пример в Release режим, ще получим много по-бедна информация от свойството StackTrace:
- Exception caught: Input string was not in a correct format.
- at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
- at ExceptionsTest.Main(String[] args)
- Превключването между Debug и Release режими във Visual Studio.NET става от лентата с инструменти за компилация:



4. Йерархия на изключенията

- Почти всички изключения в .NET Framework наследяват класа **System.Exception** (някои наследяват директно **System.Object**). **System.Exception** клас има важни наследника, които директно го наследяват - **ApplicationException** и **SystemException** и които, от своя страна, се наследяват от други класове. Това се вижда от следната диаграма:

System.Exception

→ **System.SystemException Class**

- **System.ArithmeticException**
 - **System.DivideByZeroException**
 - **System.OverflowException**
- **System.NullReferenceException**
- **System.FormatException**

→ **System.ApplicationException Class**

- **StudentException**
 - **StudentNotFoundException**
 - **CreateStudentException**
- **TeacherException**
 - **TeacherNotFoundException**
 - **CreateTeacherException**



- Системните изключения, които се използват от стандартните библиотеки на .NET и вътрешно от CLR наследяват класа **System.SystemException**.

Ето някои от тях:

- **System.ArithmeticException** - грешка при изпълнението на аритметична операция, например деление на 0 (**System.DivideByZeroException**), препълване на целочислен тип (**System.OverflowException**) и др.
- **System.FormatException** - хвърля се, при грешен формат на аргумент на метод.
- **System.NullReferenceException** - опит за достъп до обект, който има стойност null.
- **System.ArgumentException** - невалиден аргумент при извикване на метод.
- **System.OutOfMemoryException** - паметта е свършила.



- `System.StackOverflowException` - препълване на стека. Обикновено възниква при настъпване на безкрайна рекурсия.
- `System.IndexOutOfRangeException` - опит за излизане от границите на масив.
- `System.OverflowException` - хвърля се, при препълване, при аритметичен, кастващ или преобразуващ оператор.



- Препоръчително е, изключенията дефинирани от потребителя да наследяват класа **System.ApplicationException**. В по-големите приложения, тези изключения се разделят логически в категории и за всяка категория се дефинира по един базов клас (**StudentException**, **CourseException**, **TeacherException...**), който се наследява от по-конкретни изключения.
- Възможно е потребителски-дефинирано изключение да наследи директно **System.Exception**, а не **System.ApplicationException**, но това не се счита за добра практика. Идеята е, че когато потребителските програми предизвикват изключения само от **System.ApplicationException** или негови наследници, това би дало възможност по-лесно да се разбере дали проблемът е на ниво потребителски код или е свързан със системна грешка. Въпросът обаче е дискуссионен: някои експерти твърдят, че наследяването на **ApplicationException** усложнява излишно йерархията, докато други смятат, че е по-важно да се разграничават системните от потребителските изключения. Използването на собствените класове за изключения става по същия начин, както и системните изключения.

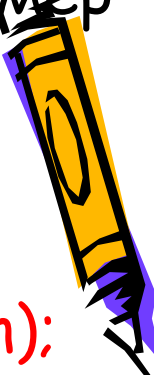


- Когато се дефинира собствено изключение (например `MyException`), се препоръчва да се дефинират и следните два конструктора:

`MyException(string message);`

`MyException(string message, Exception InnerException);`

- Препоръчва се също, имената на изключенията да завършват с "**Exception**", както в примерите (`MyException`, `StudentException`, `CourseException`, `TeacherException`).



5. Препоръчвани практики

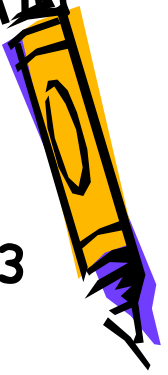
- Изключенията са утвърден механизъм за обработка на грешки, но неправилното им или прекомерното им използване, може да доведе до отрицателен ефект по отношение на коректната работа и производителността на приложението. Препоръчителни са следните практики при работата с изключения:
 - **catch** блоковете да се подреждат така, че да започват от прихващане на изключенията на най-ниско ниво в йерархията, в посока – към по-високите. Така ще бъдат обработени първо по-специфичните изключения и след това по общите изключения. В противен случай, кодът за по-специфичните изключения никога няма да се изпълни.



- Всеки **catch** блок трябва да прихваща само изключенията, които очаква. Не се счита за добър стил прихващането на всички изключения, тоест препоръчително е да се избягват конструкциите **catch (Exception) {...}** или общ **catch**, тъй като различните видове изключения изискват специфични действия за справяне с възникналата проблемна ситуация.
- Добър програмистки стил е при дефиниране на собствени изключения да се наследява **System.ApplicationException**, а не директно **System.Exception**. По този начин може да се направи разграничение на това дали изключението е от .NET Framework или е от приложението.
- Добър програмистки стил е имената на класовете на всички изключения да завършват на **Exception**. Това прави кода по-разбираем и по-лесен за поддръжка.



- Добър програмистки стил е при създаване на инстанция на изключение да се подава в конструктора подходящо съобщение. Това съобщение ще бъде достъпно по-късно чрез свойството **Message** на изключението и ще помогне на програмиста, който използва дадения клас, по-лесно да идентифицира проблема.



- Прекомерното използване на изключенията се отразява на производителността. Това е така защото всяко хвърлено изключение инстанцира клас (а това отнема време), инициализира членовете му (това също отнема време), извършва търсене в стека за подходящ **catch** блок (и това отнема време) и накрая след като инстанцията стане неизползваема, тя се унищожава от **garbage collector** (това също отнема време). Затова, изключенията трябва да бъдат хвърляни само при ситуации, които наистина са изключителни и трябва да се обработят. Препоръчва се, когато е възможно да се прави проверка за дадено действие, вместо да се използват изключения.



- Добра практика е прихващането на непредсказуеми по време изключения, като например `System.OutOfMemoryException`, да се стане на най-високо ниво в програмата - в **Main()** метода, с цел да се прекрати по подходящ начин изпълнението на програмата.

