# Решения на задачите от глава XXII на книгата "Въведение в програмирането със C#"

Предлагаме ви решения на задачите от [книгата "Въведение в програ­мирането със C#"](http://www.introprogramming.info), заедно с анализ на задачата, описание на използва­ните идеи, алгоритми, подходи за решаване и тестове.

## Авторски колектив

**Веселин Иванов Божинов**

Контакти: ([vesko\_bojinov@abv.bg](mailto:vesko_bojinov@abv.bg))

**Жасмина Тодорова Стоянова**

Контакти: ([jas\_t@abv.bg](mailto:jas_t@abv.bg))

**Калоян Йорданов Которов**

Контакти: ([K.Kotorov@gmail.com](mailto:K.Kotorov@gmail.com))

**Ненчо Великов Ненчев**

Контакти: ([nencho83@yahoo.com](mailto:nencho83@yahoo.com))

# Отговор на Задача 1 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 1. Да се намерят всички стандартни изключения от йерархията на System.IO.IOException. |
| **IOException** е базовият клас за изключения, които се хвърлят, докато се достъпва информация чрез използването на потоци, файлове и директории. Библиотеката на базовия клас включва следните класове, всеки от които е наследник на класа IOException:   * **DirectoryNotFoundException** – изключение от този клас се хвърля, когато част от файл или директория не могат да бъдат намерени * **DriveNotFoundException** - изключение от този клас се хвърля, когато се опитваме да достъпим устройство или дял, които не са налични * **EndOfStreamException** – изключение от този клас се хвърля, когато се опитваме да четем след края на потока * **FileLoadException** - изключение от този клас се хвърля, когато асембли файла е намерен, но не може да бъде зареден. * **FileNotFoundException** - изключение от този клас се хвърля, при неуспешен опит да бъде досъпен файл, който не се намира на диска * **PathTooLongException -** изключение от този клас се хвърля, когато пътя или името на файла е по-дълго от определената от системата максимална дължина * **PipeException -** изключение от този клас се хвърля, при възникване на грешка в рамките на именования канал   Когато е уместно е добре да се използват тези подкласове, вместо базовия клас IOException. |

# Отговор на Задача 2 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 2. Да се намерят всички стандартни изключения от йерархията на System.IO.FileNotFoundException. |
| Изключение от тип **FileNotFoundException** се хвърля, при неуспешен опит да бъде досъпен файл, който не се намира на диска.  Забележка!!!  Класът няма наследници.  http://msdn.microsoft.com/en-us/library/system.io.filenotfoundexception.aspx |

# Отговор на Задача 3 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 3. Да се намерят всички стандартни изключения от йерархията на System.ApplicationException. |
| Изключенията от йерархията на **System.ApplicationException** се хвърлят от потребителската програма(а не от средата за контролирано изпълнение на .NET код - CLR), когато възникне нефатална грешка в приложението. Наследници на класа са следните подкласове:   * [**InvalidFilterCriteriaException**](http://msdn.microsoft.com/en-us/library/system.reflection.invalidfiltercriteriaexception(v=vs.71).aspx)– изключение от този клас се хвърля, при работа с обекти от клас MemberInfo, когато филтриращия критерий не е валиден за типа филтър, който използваме. * [**TargetException**](http://msdn.microsoft.com/en-us/library/system.reflection.targetexception(v=vs.71).aspx) **-** изключение от този клас се хвърля, когато се прави опит да бъде извикан нестатичен метод върху нулев обект, т.е когато целта е невалидна * [**TargetInvocationException**](http://msdn.microsoft.com/en-us/library/system.reflection.targetinvocationexception(v=vs.71).aspx) **-** изключение от този клас се хвърля от методи, извикани чрез рефлекcия. Този клас не може да бъде наследен. * [**TargetParameterCountException**](http://msdn.microsoft.com/en-us/library/system.reflection.targetparametercountexception(v=vs.71).aspx) **-** изключение от този клас се хвърля, когато броя на параметрите за дадено извикване не съвпада с очаквания брой. Този клас не може да бъде наследен.   Когато е уместно е добре да се използват тези подкласове, вместо базовия клас ApplicationException. |

# Отговор на Задача 4 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 4. Обяснете какво представляват изключенията, кога се използват и как се прихващат. |
| По време на изпълнение на програмата могат да възникнат грешки или да се появят необичайни ситуации. Тогава системата или изпълняващата се в момента програма докладва за това чрез хвърляне на изключение, съдържащо информация за възникналата грешка или ситуация. Веднъж хвърлено, изключението се обработва от програмата или от обработчика на изключения по подразбиране(exception handler).  Изключенията трябва да се ползват при изключителни ситуации, а не за очаквани събития, обикновено свързани с функционалността на програмата, тъй като самия процес е бавна операция и може да доведе до видимо намаляване на производителността.  Прихващането на изключения става с помоща на ключовите думи **try**, **catch** и **finally**, за да се проверят действия, които е възможно да не се изпълнят успешно, да се обработят грешки, когато смятаме за необходимо и да се освободят ресурси след това. Изключенията могат да бъдат генерирани от средата за контролирано изпълнение на .NET код, наречена CLR или други външни библиотеки, или от кода на програмата като за целта се използва ключовата дума **throw**. |

# Отговор на Задача 5 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 5. Обяснете ситуациите, при които се използва try-finally конструкцията. Обяснете връзката между try-finally и using конструкциите. |
| Към всеки try блок може да бъде добавен максимум един finally блок.  Конструкцията **try-finally** има вида:  try  {  // Contains the code segment expected to raise the exception.  }  finally  {  // Contains the exception handler and the cleanup code.  }  Блокът **finally** e полезен за освобождаване на един или повече ресурси, заделени в try блока и за изпълнението на код, дори когато се появи изключение в try блока. Обикновено, кодът в блокът finally се изпълнява, когато се излиза от try блока, без значение дали това става в резултат на нормалното изпълнение на кода или на срещането на break,continue,gotoилиreturn. Когато дадено изключение е обработено, тогава винаги се гарантира изпълнението на съответния finally блок, в противен случай обаче това зависи от това дали в блока try средата за изпълнение CLR ще прекрати изпълнението си.  Конструкцията **using** предоставя съкратен запис за освобождаване на някои видове ресурси, без да се интересува от хвърлените изключения или дали има такива. Подобно на finally, using блока гарантира освобождаването на ресурсите, без значение как се излиза от него. Това е вярно дори когато дадено изключение не може да бъде обработено(освен в случаите на StackOverflowException). С него не е нужно да имаме try-finally конструкция за да освободим ресурси, нито да викаме изрично някакви методи за това. |

# Отговор на Задача 6 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 6. Обяснете предимствата на използването на изключения. |
| Използването на изключения се е наложило в практиката поради редица преимущества, най-главните от които са:   * **Отделяне на кода за обработка на грешките** – използването на изключения ни помага да отделим обикновения код в една програма от кода за изключителни ситуации и този за обработка на грешки. Така псевдокода на даден метод е по-добре структуриран, кратък и ясен. * **Групиране на различните видове грешки –** тъй като изключенията са групирани в класове с йерархична структура можем да прихващаме повече от един тип изключение, стига в catch да декларираме тип с наследници, така прихващаме не само декларирания тип, а и цялата йерархия от всички негови наследници. * **Предаване на грешките за обработка в стека на методите – прихващане на нива –** позволява обработката на изключение да се направи в метода, който е най-подходящ. Това става благодарение на възможността, която средата за изпълнение (CLR) предлага като позволява на всеки метод, който е заинтересован да прихване и обработи дадено изключение да го направи и ако не е да се препрати назад в стека на извикване на методите. |

# Отговор на Задача 7 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 7. Изключения при коренуване на цяло число. | |
| **Условие**  Напишете програма, която прочита от конзолата цяло положително число и отпечатва на конзолата корен квадратен от това число. Ако числото е отрицателно или невалидно, да се изпише "Invalid Number" на конзолата. Във всички случаи да се принтира на конзолата "Good Bye". | |
| **Описание на входа**  Входните данни се четат от конзолата. На единствения входен ред се подава цяло положително число за изчисляване на неговият корен квадратен, или произволен стринг за тестване на изключенията. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. На първият ред от изхода бъде написан квадратният корен на входното число, или да бъде изведено съобщението "Invalid Number". На втория ред от изхода трябва да бъде принтирано съобщението "Good Bye". | |
| **Анализ на задачата**  В един try блок се опитване да парснем подаденият вход към uint и да присвоим получената стойност на променлива от тип uint. След това принтираме полученият от коренуването резултат и затваряме try блока. След това с два catch блока прихващаме двата вида изключения които могат да се появят – OverflowException и FormatException. Във всеки от тези блокове принтираме ан конзолата "Invalid Number". Накрая в един блок finally принтираме съобщението "Good Bye" за да сме сигурни че това съобщение се изпише на конзолата – независимо от хода на изпълнение на програмата. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CheckInputData  {  class CheckInputData  {  static void Main(string[] args)  {  uint number = 0;  try  {  number = uint.Parse(Console.ReadLine());  double result = Math.Sqrt(number);  Console.WriteLine(result);  }  catch (OverflowException)  {  Console.WriteLine("Invalid number!");  }  catch (FormatException)  {  Console.WriteLine("Invalid number!");  }  finally  {  Console.WriteLine("Good bye!");  }  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * 0 – отговорът трябва да е 0   Good bye!   * 1 – отговорът трябва да е 1   Good bye!   * Отрицателно число - Invalid number!   Good bye!   * Число с плаваща запетая - Invalid number!   Good bye!   * Без вход - Invalid number!   Good bye!   * Стринг - Invalid number!   Good bye!   * Много голямо число което не се побира във uint - Invalid number!   Good bye! | |
| **Вход** | **Изход** |
| 0 | 0  Good bye! |
| **Вход** | **Изход** |
| 1 | 1  Good bye! |
| **Вход** | **Изход** |
| -1 | Invalid number!  Good bye! |
| **Вход** | **Изход** |
| 3.5 | Invalid number!  Good bye! |
| **Вход** | **Изход** |
| “Enter” | Invalid number!  Good bye! |
| **Вход** | **Изход** |
| word | Invalid number!  Good bye! |
| **Вход** | **Изход** |
| 100000000000000000000000 | Invalid number!  Good bye! |
| **Вход** | **Изход** |
| 9 | 3  Good bye! |
| **Вход** | **Изход** |
| 100 | 10  Good bye! |
| **Вход** | **Изход** |
| 10 | 3,16227766016838  Good bye! |

# Отговор на Задача 8 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 8. Въвеждане на 10 числа в интервала 1-100. | |
| **Условие**  Напишете метод ReadNumber(int start, int end), който въвежда от конзолата число в диапазона [start…end]. В случай на въведено невалидно число или число, което не е в подадения диапазон хвърлете подходящо изключение. Използвайки този метод напишете програма, която въвежда 10 числа a1, a2, …, a10, такива, че 1 < a1 < … < a10 < 100. | |
| **Описание на входа**  Входните данни се четат от конзолата. На десет реда последователно се въвеждат произволни символи. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. След всяко въвеждане на символ или стринг трябва да излиза подходящо съобщение за грешка – ако символът или стрингът не представляват число в диапазона 1-100 или ако текущото въведено число не е по-голямо от предходното въведено число. В противен случай програмата не извежда нищо на конзолата. | |
| **Анализ на задачата**  Създаваме методът ReadNumber(int start, int end), който приема два параметъра. В него декларираме една променлива от тип int. Във един try блок се опитваме да и присвоим стойност прочетена от конзолата и проверяваме дали тази стойност е в интервала между start I end. Ако въведената стойност не е интервала хвърляме ArgumentOutOfRangeException. След това в няколко catch блока прихващаме всички възможни изключения които могат да възникнат. В main метода декларираме един брояч и правим един while цикъл докато броячът не стане равен на 10. В цикъла приемаме върната стойност от ReadNumber(int start, int end)и проверяваме дали е по малка от предходната въведена. Ако е така хвърляме ArgumentOutOfRangeException. Ако всичко е наред увеличаваме брояча с единица. Преди края на while цикъла прихващаме ArgumentOutOfRangeException. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  namespace ReadNumbers  {  class ReadNumbers  {  static void Main(string[] args)  {  int start = 1;  int end = 100;  byte count = 0;  int previousNumber = 0;  while (count < 10)  {  try  {  int nexNumber = ReadNumber(start, end);  if (previousNumber >= nexNumber)  {  throw new ArgumentOutOfRangeException();  }  previousNumber = nexNumber;  count++;  }  catch (ArgumentOutOfRangeException smallNumber)  {  Console.WriteLine(smallNumber.Message);  }  }  }  static int ReadNumber(int start, int end)  {  int number = 0;  try  {  string input = Console.ReadLine();  number = int.Parse(input);  if ((number < start) || (number > end))  {  throw new ArgumentOutOfRangeException();  }  }  catch (OutOfMemoryException outOfMemory)  {  Console.WriteLine(outOfMemory.Message);  }  catch (ArgumentOutOfRangeException outOfRange)  {  Console.WriteLine(outOfRange.Message);  }  catch (ArgumentNullException nullExeption)  {  Console.WriteLine(nullExeption.Message);  }  catch (FormatException formatEx)  {  Console.WriteLine(formatEx.Message);  }  catch (OverflowException overflow)  {  Console.WriteLine(overflow.Message);  }  catch (IOException IO)  {  Console.WriteLine(IO.Message);  }  catch (Exception general)  {  Console.WriteLine(general.Message);  }  return number;  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * 0 – отговорът трябва да е Specified argument was out of the range of valid values.   Specified argument was out of the range of valid values.   * Число с плаваща запетая - Input string was not in a correct format.   Specified argument was out of the range of valid values.   * Без вход - Input string was not in a correct format.   Specified argument was out of the range of valid values.   * Стринг - Input string was not in a correct format.   Specified argument was out of the range of valid values.   * Когато въведеното число е по-малко от предходното - Specified argument was out of the range of valid values. * Много голямо число, което не се побира в int - Value was either too large or too small for an Int32. Specified argument was out of the range of valid values. | |
| **Вход** | **Изход** |
| 0 | Specified argument was out of the range of valid values.  Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| 20 |  |
| **Вход** | **Изход** |
| 10 „10<20“ | Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| 3.5 | Input string was not in a correct format.  Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| “Enter” | Input string was not in a correct format.  Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| word | Input string was not in a correct format.  Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| 100000000000000000000000 | Value was either too large or too small for an Int32.  Specified argument was out of the range of valid values. |
| **Вход** | **Изход** |
| 50 |  |
| **Вход** | **Изход** |
| 60 |  |
| **Вход** | **Изход** |
| 70 |  |

# Отговор на Задача 9 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 9. Съдържание на текстов файл към стринг. | |
| **Условие**  Напишете метод, който приема като параметър име на текстов файл, прочита съдържанието му и го връща като string. Какво е правилно да направи методът с евентуално възникващите изключения? | |
| **Описание на входа**  Входните данни се четат от конзолата. На единствения входен ред се подава пътят до името на файла и неговото име. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. Ако всичко е коректно и позволено – трябва да бъде изведено съдържанието на текстовият файл. В противен случай трябва да се изведе Error Message на съответното изключение което е възникнало. | |
| **Анализ на задачата**  Пишем един метод FileToString който приема като параметър string. Използваме този стринг в един try блок като параметър за File.ReadAllText() и присвояваме получената стойност на променлива от тип string която връщаме като резултат от метода, ако не са възникнали изключения. Ако възникне изключение – методът не трябва да връща никаква стойност, а само да предаде полученото изключение – защото един метод трябва или да свърши това за което е написан или да хвърли изключение.  Във main метода отваряме един try блок и в него използваме нашият FileToString метод. След това в поредица от catch блокове прихващаме възможните изключения които могат да се появят по време на изпълнение на нашият метод. Във всеки catch блок принтираме Error Message на съответното изключение, което сме прихванали. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  using System.Security;  namespace FileContentToString  {  class FileContentToString  {  static void Main(string[] args)  {  string errorMessage = null;  string filePath = Console.ReadLine();  try  {  Console.WriteLine(FileToString(filePath));  }  catch (ArgumentNullException ane)  {  errorMessage = ane.Message;  }  catch (ArgumentException ae)  {  errorMessage = ae.Message;  }  catch (PathTooLongException ptle)  {  errorMessage = ptle.Message;  }  catch (DirectoryNotFoundException dnfe)  {  errorMessage = dnfe.Message;  }  catch (FileNotFoundException fnfe)  {  errorMessage = fnfe.Message;  }  catch (IOException ioe)  {  errorMessage = ioe.Message;  }  catch (UnauthorizedAccessException uae)  {  errorMessage = uae.Message;  }  catch (NotSupportedException nse)  {  errorMessage = nse.Message;  }  catch (SecurityException se)  {  errorMessage = se.Message;  }  catch (Exception e)  {  errorMessage = e.Message;  }  finally  {  if (errorMessage != null)  {  Console.WriteLine(errorMessage);  }  }  }  private static string FileToString(string filePath)  {  string toString = File.ReadAllText(filePath, Encoding.UTF8);  return toString;  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Празен стринг – Empty path name is not legal. * Много дълъг стринг - The specified path, file name, or both are too long. The fully qualified file name must be less than 260 characters, and the directory name must be less than 248 characters. * Нвалиден път. * Път до файл, който не е разрешен за четене. | |
| **Вход** | **Изход** |
| “Enter” | Empty path name is not legal. |
| **Вход** | **Изход** |
| VeryLoooooooooooooooooooooooooooooooooooooooooo  ooooooooooooooooooooooooooooooooooooooooooooooo  ooooooooooooooooooooooooooooooooooooooooooooong | The specified path, file name, or both are too long. The fully qualified file na  me must be less than 260 characters, and the directory name must be less than 24  8 characters. |
| **Вход** | **Изход** |
| C:\WrongFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| D:\CnatReadFile.txt | Access to the path 'D:\CnatReadFile.txt' is denied. |
| **Вход** | **Изход** |
| D:\TextFile.txt | The text in the file... |
| **Вход** | **Изход** |
| C\:EmptyFile.txt |  |
| **Вход** | **Изход** |
| C:\NotExistingFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\TheFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\SomeFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\MissingFile.txt | 1 2 3 4 |

# Отговор на Задача 10 от Глава 12. Обработка на изключения

|  |
| --- |
| Задача 10. Изключения при коренуване на цяло число. |
| **Условие**  Напишете метод, който приема като параметър име на бинарен файл и прочита съдържанието на файла и го връща като масив от байтове. Напишете метод, който записва прочетеното съдържание в друг файл. Сравнете двата файла. |
| **Описание на входа**  Входните данни се четат от конзолата. Входните данни се приемат на два реда. На първият ред се прием пътят до файлът, от който сечете. На вторият ред се приеме пътят до файлът, в който се пише. |
| **Описание на изхода**  Ако не възникнат грешки по време на изпълнение на програмата на конзолата не трябва да се изписва нищо и файла, в който се пише трябва да съдържа байтовете на файла, от който се чете. Ако по време на изпълнение на програмата възникне изключение – на конзолата трябва да се напише адекватно съобщение за прихванатото изключение. |
| **Анализ на задачата**  Прочитаме двата реда на входа и ги записваме във два стринга. Подаваме първият стринг на предварително написан метод, който приема пътят до файла и връща резултатът му като стринг. Полученият стринг подаваме на метод, който приема стринг и връща съдържанието му като масив от байтове. Последният метод използва метода Convert.ToByte, който извлича байтовият код на даден стринг. Използваме предварително написан метод за писане във файл, който приема като аргумент масив байтове и и път до изходен файл, като му подаваме полученият масив то байтове и подаденият ни от конзолата път до изходния файл. В изброените методи използваме try – catch конструкции за прихващане на възможните изключения. |
| **Решение (сорс код)** |
| using System;  using System.IO;  using System.Security;  using System.Text;  namespace CompareFiles  {  class CompareFiles  {  static void Main(string[] args)  {  string inputFile = Console.ReadLine();  string outputFile = Console.ReadLine();  string fileContent = ReadFile(inputFile);  byte[] byteArray = BinaryToByteArray(fileContent);  WriteFile(outputFile, byteArray);  }  static string ReadFile(string fileName)  {  string errorMessage = null;  string binaryStr = null;  try  {  binaryStr = File.ReadAllText(fileName, Encoding.UTF8);  }  catch (ArgumentNullException ane)  {  errorMessage = ane.Message;  }  catch (ArgumentException ae)  {  errorMessage = ae.Message;  }  catch (PathTooLongException ptle)  {  errorMessage = ptle.Message;  }  catch (DirectoryNotFoundException dnfe)  {  errorMessage = dnfe.Message;  }  catch (FileNotFoundException fnfe)  {  errorMessage = fnfe.Message;  }  catch (IOException ioe)  {  errorMessage = ioe.Message;  }  catch (UnauthorizedAccessException uae)  {  errorMessage = uae.Message;  }  catch (NotSupportedException nse)  {  errorMessage = nse.Message;  }  catch (SecurityException se)  {  errorMessage = se.Message;  }  catch (Exception e)  {  errorMessage = e.Message;  }  finally  {  if (errorMessage != null) Console.WriteLine(errorMessage);  }  return binaryStr;  }  static byte[] BinaryToByteArray(string fileContent)  {  int bytesCount = fileContent.Length / 8;  byte[] bytes = new byte[bytesCount];  for (int i = 0; i < bytesCount; i++)  bytes[i] = Convert.ToByte(fileContent.Substring(8 \* i, 8), 2);  return bytes;  }  static void WriteFile(string fileName, byte[] fileContent)  {  string errorMessage = null;  try  {  File.WriteAllBytes(fileName, fileContent);  }  catch (ArgumentNullException ane)  {  errorMessage = ane.Message;  }  catch (ArgumentException ae)  {  errorMessage = ae.Message;  }  catch (PathTooLongException ptle)  {  errorMessage = ptle.Message;  }  catch (DirectoryNotFoundException dnfe)  {  errorMessage = dnfe.Message;  }  catch (FileNotFoundException fnfe)  {  errorMessage = fnfe.Message;  }  catch (IOException ioe)  {  errorMessage = ioe.Message;  }  catch (UnauthorizedAccessException uae)  {  errorMessage = uae.Message;  }  catch (NotSupportedException nse)  {  errorMessage = nse.Message;  }  catch (SecurityException se)  {  errorMessage = se.Message;  }  catch (Exception e)  {  errorMessage = e.Message;  }  finally  {  if (errorMessage != null) Console.WriteLine(errorMessage);  }  }  }  } |
| **Тестове**  Интересните случаи за тестване са следните:   * При нормален ход на програмата не трябва да се изписва нищо на конзолата. В противен случай трябва да се покаже съобщение от прихванато изключение. |

# Отговор на Задача 11 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 11. Дефиниране На собствен клас за изключения. | |
| **Условие**  Потърсете информация в Интернет и дефинирайте собствен клас за изключение FileParseException. Вашето изключение трябва да съ­държа в себе си името на файл, който се обработва и номер на ред, в който е възникнал проблем. Добавете подходящи конструктори за вашето изключение. Напишете програма, която чете от текстов файл числа. Ако при четенето се стигне до ред, който не съдържа число, хвърлете FileParseException и го обработете в извикващия метод. | |
| **Описание на входа**  Входните данни се четат от конзолата. На единствения входен ред се подава пътят до файла и неговото име. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. Ако на някой от редовете във текстовия файл има нещо различно от число трябва да изкарате съобщение за грешка. Съобщението трябва да съдържа името на файла и реда на който е възникнала грешката във следният формат:  Error while parsing file Text.txt on line 6.  Text.txt  6 | |
| **Анализ на задачата**  Създаваме един клаc FileParseExeption, който наследява Exsception и връща във Error Message името на файла и редът на който е възникнало изключението. Създаваме един метод ParseLine който приема като параметри името на файла, редът който парсваме и броят на редовете които файлът съдържа и хвърля изключение от тип FileParseExeption, ако на дадения ред от файла няма число. В main метода на програмата използваме нашият метод ParseLine във един try блок и след това прихващаме изключенията от тип FileParseExeption в един catch блок. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Diagnostics;  using System.IO;  namespace CustomException  {  class CustomException  {  static void Main(string[] args)  {  string filePath = Console.ReadLine();  string fileName = filePath.Substring(filePath.LastIndexOf('\\') + 1);  try  {  string text = File.ReadAllText(filePath, Encoding.UTF8);  string[] fileContent = text.Split('\n');  long length = fileContent.Length;  for (int line = 0; line < length; line++)  {  ParseLine(fileName, fileContent[line].Trim(), line + 1);  }  }  catch (FileParseException fpe)  {  Console.WriteLine(fpe.Message);  Console.WriteLine(fpe.fileName);  Console.WriteLine(fpe.lineNumber);  }  }  static void ParseLine(string fileName, string lineOfFile, long lineNumber)  {  bool containesNumber = true;  long length = lineOfFile.Length;  for (int symbol = 0; symbol < length; symbol++)  {  if ((lineOfFile[symbol] > 47) && (lineOfFile[symbol] < 58))  {  containesNumber = false;  break;  }  }  if (containesNumber)  {  throw new FileParseException(fileName, lineNumber);  }  }  }  class FileParseException : Exception  {  public string fileName { get; private set; }  public long lineNumber { get; private set; }  public FileParseException()  {  }  public FileParseException(string fileName, long lineNumber)  : base(string.Format("Error while parsing file {0} on line {1}.", fileName, lineNumber))  {  this.fileName = fileName;  this.lineNumber = lineNumber;  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Ако въведем път до файл в който има ред в който не се съдържа число. | |
| **Вход** | **Изход** |
| D:\Text.txt | Error while parsing file Text.txt on line 6.  Text.txt  6 |
| **Вход** | **Изход** |
| D:\OnlyNumbers.txt |  |

# Отговор на Задача 12 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 12. Извеждане на съдържанието на текстов файл на конзолата. | |
| **Условие**  Напишете програма, която прочита от потребителя пълен път до даден файл (например C:\Windows\win.ini), прочита съдържанието на файла и го извежда на конзолата. Намерете в MSDN как да използвате метода System.IO.File.ReadAllText(…). Уверете се, че прихващате всички възможни изключения, които могат да възникнат по време на работа на метода и извеждайте на конзолата съобщения за грешка, разбираеми за обикновения потребител. | |
| **Описание на входа**  Входните данни се четат от конзолата. На единствения входен ред се подава пътят до името на файла и неговото име. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. Ако всичко е коректно и позволено – трябва да бъде изведено съдържанието на текстовият файл. В противен случай трябва да се изведе Error Message на съответното изключение което е възникнало. | |
| **Анализ на задачата**  Пишем един метод FileToString който приема като параметър string. Използваме този стринг в един try блок като параметър за File.ReadAllText() и присвояваме получената стойност на променлива от тип string която връщаме като резултат от метода, ако не са възникнали изключения. Ако възникне изключение – методът не трябва да връща никаква стойност, а само да предаде полученото изключение – защото един метод трябва или да свърши това за което е написан или да хвърли изключение.  В main метода отваряме един try блок и в него използваме нашият FileToString метод. След това в поредица от catch блокове прихващаме възможните изключения които могат да се появят по време на изпълнение на нашият метод. Във всеки catch блок принтираме Error Message на съответното изключение, което сме прихванали. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  using System.Security;  namespace FileContentToString  {  class FileContentToString  {  static void Main(string[] args)  {  string errorMessage = null;  string filePath = Console.ReadLine();  try  {  Console.WriteLine(FileToString(filePath));  }  catch (ArgumentNullException ane)  {  errorMessage = ane.Message;  }  catch (ArgumentException ae)  {  errorMessage = ae.Message;  }  catch (PathTooLongException ptle)  {  errorMessage = ptle.Message;  }  catch (DirectoryNotFoundException dnfe)  {  errorMessage = dnfe.Message;  }  catch (FileNotFoundException fnfe)  {  errorMessage = fnfe.Message;  }  catch (IOException ioe)  {  errorMessage = ioe.Message;  }  catch (UnauthorizedAccessException uae)  {  errorMessage = uae.Message;  }  catch (NotSupportedException nse)  {  errorMessage = nse.Message;  }  catch (SecurityException se)  {  errorMessage = se.Message;  }  catch (Exception e)  {  errorMessage = e.Message;  }  finally  {  if (errorMessage != null)  {  Console.WriteLine(errorMessage);  }  }  }  private static string FileToString(string filePath)  {  string toString = File.ReadAllText(filePath, Encoding.UTF8);  return toString;  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Празен стринг – Empty path name is not legal. * Много дълъг стринг - The specified path, file name, or both are too long. The fully qualified file name must be less than 260 characters, and the directory name must be less than 248 characters. * Нвалиден път. * Път до файл, който не е разрешен за четене. | |
| **Вход** | **Изход** |
| “Enter” | Empty path name is not legal. |
| **Вход** | **Изход** |
| VeryLoooooooooooooooooooooooooooooooooooooooooo  ooooooooooooooooooooooooooooooooooooooooooooooo  ooooooooooooooooooooooooooooooooooooooooooooong | The specified path, file name, or both are too long. The fully qualified file na  me must be less than 260 characters, and the directory name must be less than 24  8 characters. |
| **Вход** | **Изход** |
| C:\WrongFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| D:\CnatReadFile.txt | Access to the path 'D:\CnatReadFile.txt' is denied. |
| **Вход** | **Изход** |
| D:\TextFile.txt | The text in the file... |
| **Вход** | **Изход** |
| C\:EmptyFile.txt |  |
| **Вход** | **Изход** |
| C:\NotExistingFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\TheFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\SomeFile.txt | Could not find file 'C:\WrongFile'. |
| **Вход** | **Изход** |
| C:\MissingFile.txt | 1 2 3 4 |

# Отговор на Задача 13 от Глава 12. Обработка на изключения

|  |  |
| --- | --- |
| Задача 13. Изтегляне на файл от интернет по зададен URL адрес. | |
| **Условие**  Напишете програма, която изтегля файл от Интернет по даден URL адрес, примерно (<http://www.devbg.org/img/Logo-BASD.jpg>). | |
| **Описание на входа**  Входните данни се четат от конзолата. На единствения входен ред се подава URL адреса на файла който трябва да се изтегли. | |
| **Описание на изхода**  Изхода трябва да бъде изведен на конзолата. Ако файлът се изтегли без проблеми на конзолата не се изписва нищо. Ако се появи изключение по време на изпълнение на програмата трябва да се изведе Error Message на изключението. | |
| **Анализ на задачата**  Използваме System.Net и неговият клас WebClient. Приемаме URL адреса от конзолата като стринг.  Създаваме инстанция на класа WebClient. В един try блок подаваме на метода DownloadFile() на прочетеният стринг. След това в поредица catch блокове прихващаме възможните изключения. | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Net;  namespace DownloadFileFromInternet  {  class DownloadFileFromInternet  {  static void Main(string[] args)  {  WebClient wc = null;  string errorMessage = null;  string url = Console.ReadLine();  try  {  wc = new WebClient();  // Download a file from i-net to the current directory.  wc.DownloadFile(url, @".\google.png");  // Download a file from i-net to parent directory of parent directory.  wc.DownloadFile(url, @"..\..\google.png");  }  catch (WebException we)  {  errorMessage = we.Message;  }  catch (NotSupportedException nse)  {  errorMessage = nse.Message;  }  catch (Exception e)  {  errorMessage = e.Message;  }  finally  {  if (errorMessage != null) Console.WriteLine(errorMessage);  }  }  }  } | |
| **Тестове**  Интересните случаи за тестване са следните:   * Грешно URL | |
| **Вход** | **Изход** |
| http://www.devbg.org/img/Logo-BASD.jpg |  |
| **Вход** | **Изход** |
| http://www.devbg.org/img/Logo | The remote server returned an error: (404) Not Found. |