Exceptions – Exercises

The exercises are taken from the Bulgarian book: **Fundamentals of Computer Programming with C# - 2013.**

Authors of the book: Svetlin Nakov & Co.

Website: www.introprogramming.info

References: Chapter 12. Exception Handling

1. Exercise 01

- **Description:** Find out all exceptions in the **System.IO.IOException hierarchy.**
- **Instruction:** Search in MSDN for **“IOException”.**

2. Exercise 02

- **Description:** Find out all standard exceptions that are part of the hierarchy holding the class **System.IO.FileNotFoundException**.
- **Instruction:** Search in the MSDN.

3. Exercise 03

- **Description:** Find out all standard exceptions from the **System.ApplicationException** hierarchy.
- **Instruction:** Search in the MSDN.

4. Exercise 04

- **Description:** Explain the concept of exceptions and **exception handling** when they are used and how to catch exceptions.
- **Instruction:** Search on the internet.

5. Exercise 05

- **Description:** Explain when the statement try-finally is used. Explain the relationship between the statements **try-finally and using.**
- **Instruction:** Search on the internet.

6. Exercise 06

- **Description:** Explain the advantages when using exceptions**.**
- **Instruction:** Search on the internet.

7. Exercise 07

- **Description:** Write a program that takes a **positive integer** from the console and prints the square root of this integer. If the input is negative or invalid **print "Invalid Number"** in the console. **In all cases print "GoodBye".**
- **Instruction:** Create **try-catch-finally** statement.

8. Exercise 08

- **Description:** Write a method **ReadNumber(int start, int end)** that reads an integer from the console in the range [start…end]. In case the input integer is not valid, or it is not in the required range throw an appropriate exception. Using this method, write a program that takes 10 integers a1, a2, …, a10 such that 1 < a1 < … < a10 < 100.
- **Instruction:** When an invalid number is used, we can throw an Exception because there is no other exception that can better describe the problem. As an alternative, we can define our own exception class called in a way that better describes the problem, e.g**. InvalidNumberException**.

9. Exercise 09

- **Description:** Write a method that takes as a parameter the name of a text file, reads the file, and returns its content as a string. What should the method do if an exception is thrown?
- **Instruction:** Read the file line by line with **System.IO.StreamReader** class and add the rows in **System.Text**. StringBuilder. Throw all exceptions from the method without catching them. You may cheat and solve the problem in one line of code by using the static method **System.IO.File.ReadAllText().**

10. Exercise 10

- **Description:** Write a method that takes as a parameter the name of a binary file, reads the content of the file, and returns it as an array of bytes. Write a method that writes the file content to another file. Compare both files.
- **Instruction:** Search in Internet to learn more about **binary streams**. After that follow the instructions below for reading a file:
  - **For reading use FileStream** and write the data in a MemoryStream. You have to read the file in parts, for example on portions with 64 KB each, the last one can be smaller.
    Note: Be careful with the method for reading the bytes **FileStream.Read(byte[] buffer, int offset, int count).** This method can read fewer bytes than requested. You have to write as many bytes as you read from the input stream. Create a loop that ends when zero bytes are read.
  - **Use using** to close the streams.
  - Open **FileStream** and start writing the bytes inside the MemoryStream. Use **using** to correctly close the streams.

- **Use a big ZIP archive** to test (for example 300 MB). If the program is not working correctly, it will break the structure of the archive and an error will occur when trying to open it.

- You can cheat by using the system methods System.IO.File. ReadAllBytes() and System.IO.File.WriteAllBytes(byte[]).

11. Exercise 11

- **Description:** Search for information on the Internet and define your own class for exception **FileParseException. The exception has to contain the name of the processed file and the number of the row where the problem occurred.** Add appropriate constructors in the exception. Write a program that reads integers from a text file. If the during reading a row does not contain an integer **throw FileParseException** and pass it to the calling method.
- **Instruction:** Inherit from the **Exception class** and add a constructor to it. **For example, FileParseException(string message, string filename, int line).** Use this exception the same way as using any other exception. The number can be read with the **StreamReader** class.

12. Exercise 12

- **Description:** Write a program that gets from the user the full path to a file (for example C:\Windows\win.ini), reads the content of the file, and prints it to the console. Find in MSDN how to **use the System.IO.File. ReadAllText(…)** method. Make sure all possible exceptions will be caught and a user-friendly message will be printed on the console.
- **Instruction:** Search for all possible exceptions that the method could throw and **for all of them define a catch block** and print a user-friendly message.

13. Exercise 13

- **Description:** Write a program to download a file from the Internet by a given URL, e.g.
  http://introprogramming.info/wp-content/themes/introprograming_en/ images/Intro-Csharp-Book-front-cover-big_en.png.
- **Instruction:** Search for articles on the Internet for "**downloading a file with C#"** or search for information and examples about using the **WebClient class**. Make sure you **catch and process all exceptions** that can be thrown.

Answers

1. Exercise 01

- **Link to the MSDN: https://learn.microsoft.com/en-us/dotnet/api/system.io.ioexception?view=net-8.0**
- **Here are some of them:**
  - **System.IO.DirectoryNotFoundException:** Thrown when trying to access a directory that does not exist.
  - **System.IO.EndOfStreamException:** Thrown when reading past the end of a stream.
  - **System.IO.FileNotFoundException:** Thrown when attempting to access a file that does not exist.
  - **System.IO.FileLoadException:** The exception that is thrown when a managed assembly is found but cannot be loaded.
  - **System.IO.PathTooLongException:** The exception that is thrown when a path or fully qualified file name is longer than the system-defined maximum length.

2. Exercise 02

- **Here are some of them:**
  - **System.Exception:** The base class for all exceptions**.**
  - **System.SystemException:** The base class for all predefined system exceptions.
  - **System.IO.PipeException**: Thrown when an error occurs within a named pipe.**Error! Reference source not found.**

3. Exercise 03

- **Link to the MSDN:** ApplicationException Class (System) | Microsoft Learn
- **Some of the exceptions are:**
  - **Microsoft.JScript.BreakOutOfFinally:** Represents the exception state when code execution breaks out of a finally block.
  - **Microsoft.JScript.ContinueOutOfFinally:** Represents the exception state when code execution continues out of a finally block.

- **Microsoft.JScript.JScriptException:** The exception that is thrown by JScript to notify a common language runtime (CLR) host or program that an error occurred.
- **Microsoft.JScript.NoContextException:** The exception that is thrown when there is no code Context associated with a JScriptException.

4. Exercise 04

- **Exceptions** are notifications that something has interrupted the normal program execution. It is a paradigm in OOP for detecting and handling unexpected situations/events. When an exception arises, the state of the program is saved, the normal flow is interrupted, and the control is passed to an exception handler (if such exists in the current context).
- **Exception handling** is a mechanism for catching and throwing exceptions. It is provided internally by the **CLR.**
- **They are used** when there is an unexpected event, which interrupts the program flow.
- **The Try-catch** construction is used for catching and handling exceptions.
- **Considering the call stack,** once the exception is thrown, the CLR starts looking for a handler and if such does not exist, the CLR catches the exception and prints it on the console or visualizes it on an error message dialog box.

5. Exercise 05

- **The try-finally block** is used for cleaning up some resources. The **finally block** will always execute. There is the release of resources e.g. file streams, database connections, etc.
- **The using statement** releases resources just as the **try-finally block**. In the end, it will free up that resource no matter if an exception is thrown or not. It is a shorter and more simplified version.

6. Exercise 06

- **Some of the advantages** of using exceptions are the separation of code that is responsible only for exception handling, a grouping of different error types, and a more convenient way of error handling compared to procedural programming.

7. Exercise 07

- To solve that problem, we will need to take the user’s input, process it, and do the operation (square root of a number). Here is an example:

The try block:

```
using System;

namespace Ex07_Exeptions_Throwing_And_Handling_Exeptions_Exercises
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Enter a number: ");

            try
            {
                int n = int.Parse(Console.ReadLine());

                if (n < 0)
                {
                    throw new ArgumentException("Invalid number!");
                }

                Console.WriteLine("The squre root of the number {0} is 
{1}", n, Math.Sqrt(n));
            }
```

The catch block:

```
catch (ArgumentException e)
{
      Console.WriteLine(e.Message);
}
catch (FormatException)
{
      Console.WriteLine("Only numbers are allowed!");
}
catch (Exception)
{
      Console.WriteLine("Something went wrong.");
}
```

The finally block:

```
finally
{
	Console.WriteLine("GoodBye");
}
```

8. Exercise 08

- **Solution:** We are creating a new class **InvalidNumberException**. Then this exception is thrown in the method **ReadNumber()** afterwards the **Main()** handles it.

Main method:

```
static void Main()
        {
            int start = 1;
            int end = 100;
            int[] numbers = new int[10];

            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine($"Enter number {i + 1} in the range
[{start}...{end}]:");
                try
                {
                    numbers[i] = ReadNumber(start, end);
                    start = numbers[i];
                }
                catch (InvalidNumberException e)
                {
                    Console.WriteLine(e.Message);
                    i--; // repeat current iteration.
                }
            }
```

The ReadNumber():

```
private static int ReadNumber(int start, int end)
        {
            int number = int.Parse(Console.ReadLine());

            if (number <= start || number >= end)
            {
                throw new InvalidNumberException($"The number is not in
the range [{start}...{end}]");
            }

            return number;
        }
```

The class exception:

```
public class InvalidNumberException : Exception
    {
        public InvalidNumberException(string message) : base(message)
        {
        }
    }
```

9. Exercise 09

➢ In this code, the **ReadFileContent** method reads the entire file with the given filename into a string and returns it. If any exception occurs (such as a **FileNotFoundException** if the file does not exist), the exception is not caught and is instead thrown up to the caller of the method.

The main method:

```
static void Main()
        {
            try
            {
                string content = ReadFileContent("example.txt");
                Console.WriteLine(content);
            }
            catch (FileNotFoundException)
            {
                Console.WriteLine("An error occurred: File does not 
exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine($"An error occurred: 
{e.Message}");
            }
        }
```

The ReadFileContent method:

```
static string ReadFileContent(string filename)
        {
            StringBuilder sb = new StringBuilder();

            using (StreamReader sr = new StreamReader(filename))
            {
                string line;
                while ((line = sr.ReadLine()) != null)
                {
                    sb.AppendLine(line);
                }
            }

            return sb.ToString();

            // return File.ReadAllText(filename); // Alternative
        }
```

10. Exercise 10

- **Explanation:**

  - The **ReadFileContent** method reads a binary file into a byte array using a FileStream. It reads the file in chunks of 64 KB each.
  - The **WriteFileContent** method writes a byte array to a file using a FileStream.
  - The **CompareFiles** method compares the contents of two files by reading them into byte arrays and comparing each byte.

The main method:

```
static void Main()
        {
            string inputFile = "input.zip";
            string outputFile = "output.zip";

            byte[] content = ReadFileContent(inputFile);
            WriteFileContent(outputFile, content);

            Console.WriteLine("Are files identical? " +
CompareFiles(inputFile, outputFile));
        }
```

The ReadFileContent method:

```
static byte[] ReadFileContent(string filename)
        {
            using (FileStream fs = new FileStream(filename,
FileMode.Open, FileAccess.Read))
            {
                byte[] buffer = new byte[64 * 1024]; // 64 KB
                using (MemoryStream ms = new MemoryStream())
                {
                    int bytesRead;
                    while ((bytesRead = fs.Read(buffer, 0,
buffer.Length)) > 0)
                    {
                        ms.Write(buffer, 0, bytesRead);
                    }
                    return ms.ToArray();
                }
            }
        }
```

The WriteFileContent method:

```
static void WriteFileContent(string filename, byte[] content)
        {
            using (FileStream fs = new FileStream(filename,
FileMode.Create, FileAccess.Write))
            {
                fs.Write(content, 0, content.Length);
            }
        }
```

The CompareFiles method:

```
static bool CompareFiles(string file1, string file2)
       {
           byte[] content1 = File.ReadAllBytes(file1);
           byte[] content2 = File.ReadAllBytes(file2);

           if (content1.Length != content2.Length)
           {
               return false;
           }

           for (int i = 0; i < content1.Length; i++)
           {
               if (content1[i] != content2[i])
               {
                   return false;
               }
           }

           return true;
       }
```

11. Exercise 11

- **Explanation:**

  - The **FileParseException** class is a custom exception that includes the filename and line number where the exception occurred.
  - The **ReadNumbers** method reads a text file line by line using a StreamReader. If a line does not contain an integer, it throws a FileParseException.
  - The **Main method** calls ReadNumbers and catches any FileParseException that is thrown, printing an error message to the console.

The FileParseException class:

```
public class FileParseException : Exception
    {
        public string Filename { get; }
        public int LineNumber { get; }

        public FileParseException(string message, string filename, int 
lineNumber)
            : base(message)
        {
            Filename = filename;
            LineNumber = lineNumber;
        }
    }
```

The Main method:

```
static void Main()
        {
            string filename = "numbers.txt"; // replace with your
file

            try
            {
                ReadNumbers(filename);
            }
            catch (FileParseException e)
            {
                Console.WriteLine($"An error occurred in file
{e.Filename} at line {e.LineNumber}: {e.Message}");
            }
       }
```

The ReadNumbers method:

```
static void ReadNumbers(string filename)
        {
            using (StreamReader sr = new StreamReader(filename))
            {
                string line;
                int lineNumber = 1;

                while ((line = sr.ReadLine()) != null)
                {
                    if (!int.TryParse(line, out _))
                    {
                        throw new FileParseException("Non-integer
value found", filename, lineNumber);
                    }

                    lineNumber++;
                }
            }
        }
```

12. Exercise 12

Solution:

```
static void Main()
        {
            try
            {
                Console.Write("Enter the full path to a file: ");
                string path = Console.ReadLine();

                string content = File.ReadAllText(path);
                Console.WriteLine("File content:");
                Console.WriteLine(content);
            }
            catch (ArgumentException)
            {
                Console.WriteLine("The path is a zero-length string, contains only white 
space, or contains one or more invalid characters.");
            }
            catch (PathTooLongException)
            {
                Console.WriteLine("The path exceeds the system-defined maximum length.");
            }
            catch (DirectoryNotFoundException)
            {
                Console.WriteLine("The specified path is invalid (for example, it is on an 
unmapped drive).");
            }
            catch (FileNotFoundException)
            {
                Console.WriteLine("The file specified in path was not found.");
            }
            catch (IOException)
            {
                Console.WriteLine("An I/O error occurred while opening the file.");
            }
            catch (UnauthorizedAccessException)
            {
                Console.WriteLine("Path specified a file that is read-only, or this 
operation is not supported on the current platform, or path specified a directory, or the 
caller does not have the required permission.");
            }
            catch (NotSupportedException)
            {
                Console.WriteLine("Path is in an invalid format.");
            }
            catch (SecurityException)
            {
                Console.WriteLine("The caller does not have the required permission.");
            }
        }
```

Note: Include the necessary libraries: using System.IO, using System.Security.

13. Exercise 13

- **Explanation:**
  - **The Main method** downloads a file from a given URL using WebClient.DownloadFile() and it catches any exceptions that are thrown.
  - If a **WebException** is caught, it checks if the status of the exception is ConnectFailure, which indicates a failure to connect to the network, and prints a user-friendly message.
  - If a **NotSupportedException** is caught, it prints a user-friendly message.
  - If any other type of **Exception** is caught, it prints a user-friendly message.

The main method:

```
static void Main()
        {
            string url = "http://introprogramming.info/wp-content/themes/introprograming_en/images/Intro-Csharp-Book-front-cover-big_en.png"; // replace with your URL
            string filename = "downloaded_file.png";

            try
            {
                using (WebClient client = new WebClient())
                {
                    client.DownloadFile(url, filename);
                }

                Console.WriteLine("File downloaded successfully.");
            }
            catch (WebException e)
            {
                Console.WriteLine($"WebException: {e.Message}");
                if (e.Status == WebExceptionStatus.ConnectFailure)
                {
                    Console.WriteLine("Are you connected to the Internet?");
                }
            }
            catch (NotSupportedException e)
            {
                Console.WriteLine($"NotSupportedException: {e.Message}");
            }
            catch (Exception e)
            {
                Console.WriteLine($"Exception: {e.Message}");
            }
        }
```