CHAPTER 1

Text File Handling

FHICT

Om te kunnen schrijven naar een Text File of het lezen van een Text File moet je bovenaan in de C#-File het volgende toevoegen:

```
using System.IO;
```

1.1 Voorbeeldcode voor het lezen uit een TextFile

Om een `Text File` te kunnen lezen heb je een zogenaamde `Stream` nodig op de File.

> **Note** Het pad waar de File wordt weggeschreven is relatief ten opzichte van de executable, dus zoek (by default) in je *bin debug*. Met '..' ga je een directory hoger: '../bloemkool.txt'

Maak een StreamReader aan op de stream:

```
StreamReader reader = new StreamReader(fileStream);
```

en vervolgens kan één regel ingelezen worden met

```
regel = reader.ReadLine();
```

Na de laatste regel retourneert *ReadLine()* een `null`. Door een loop te maken kan dus gelezen worden tot het einde. Tot slot moeten de *reader* en de *stream* ge*Closed* worden.

Er omheen zetten we een `try...catch...finally`. Als bij uitvoering van de code in het `try`-blok er een `Exception` optreedt (kan zijn dat de File niet bestaat, of er geen read-rechten zijn) springt de programma-uitvoering meteen

door naar het `catch`-block: hier kan gekeken worden wat er fout is. Er kan een melding aan de gebruiker gegeven worden wat er mis is, of er wordt iets in een log opgeslagen. Tot slot het `finally` blok: dit wordt ALTIJD uitgevoerd: zowel als er een Exception is als wanneer het `try` blok succesvol is uitgevoerd. Door de *Close* in de `finally` te zetten weet je zeker dat na afloop de file gesloten is.

Dat ziet er (met overdadig commentaar) alsvolgt uit:

```
String line;
StreamReader reader = null;
try
{
  reader = new StreamReader("C:\\bloemkool.txt");
  // read the first line
  line = reader.ReadLine();

  //until you get to the end: read line by line.
  while (line != null)
  {
    // 'Verwerk' moet je zelf programmeren:
    Verwerk(line);
    // read next line:
    line = reader.ReadLine();
  }
}
catch(Exception exc)
{
  // Handel het probleem af.
  HandleException(exc);
}
finally
{
  // and close the reader.
  reader.Close();
}
```

1.2 Schrijven van een Text File

Het schrijven gaat vergelijkbaar met het lezen:

```
StreamWriter writer = null;
try
{
  writer = new StreamWriter("C:\\spruitjes.txt");
  // Schrijf wat je wilt naar de File:
  // let op het verschil tussen Write en WriteLine:
  writer.WriteLine("Hello File.");
  writer.Write("Nog ");
```

```
  writer.WriteLine("meer text.");
}
catch(Exception exc)
{
  // Handel het probleem af.
  HandleException(exc);
}
finally
{
  writer.Close();
}
```

Het aanmaken van een StreamWriter kan ook met andere parameters, bijvoorbeeld:

```
new StreamWriter("C:\\spruitjes.txt", true, Encoding.ASCII);
```

- De encoding geeft aan hoe de karakters in de File worden geëncodeerd.
- De *true* geeft in het voorbeeld hierboven aan of er ge*append* moet worden of dat de File overschreven moet worden.

1.3 Terminologie

Dit kun je verder nog uitzoeken als je het nodig hebt:

- Wat is de betekenis van de volgende woorden?
  - File, Bestand
  - Directory, Pad, Folder, Map
  - UNC
  - URL
- Hoe ziet een pad er uit in Windows Verkenner, Command Prompt, Linux, Mac OSX?

1.4 File handling klassen

Het .NET framework biedt een aantal klassen* om met bestanden te kunnen werken:

- File
- Directory
- DirectoryInfo
- Path

1.5 Iets over Exception Handling

`FileNotFoundException`, `NullPointerException`, enzovoort. Al die fouten die jij ziet als je een programma test ziet de gebruiker ook. Maar als de gebruiker ze ziet dan crasht zijn programma! Als software engineer dien je mogelijke exceptionele situaties te voorkomen. C# heeft hiervoor de keywords `try`, `catch` en `finally`.

Regelmatig wil je de structuur van je code niet helemaal aanpassen aan exceptionele situaties die je als een *uitzondering* wilt beschouwen. Een voorbeeld is wanneer je programma een `connectie` met een `database` nodig heeft, of `requests` naar internet stuurt: je wil dan niet elke keer checken of de connectie naar database of internet nog wel werkt, maar **als** de connectie verloren is gegaan dan moet je programma daar wel mee om kunnen gaan.

In C# doe je dat door om je code een `try-catch-clause` te zetten, zoals je hierboven gezien hebt.

Andere bronnen over Exception Handling in C#

Exceptions at CSharp-station[1]

Exceptions at MSDN[2]

MSDN[3]

[1] http://www.csharp-station.com/Tutorial/CSharp/Lesson15
[2] https://msdn.microsoft.com/en-us/library/ms173162.aspx
[3] https://msdn.microsoft.com/en-us/library/2kzb96fk.aspx