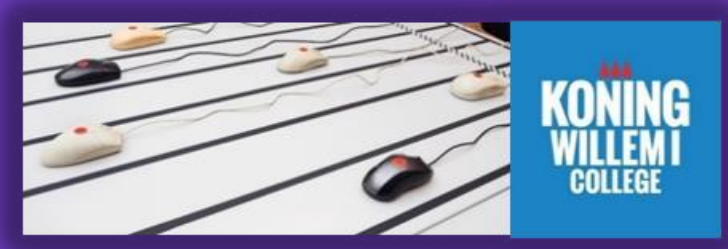
Basis standalone

Realiseren

hoofdstuk

3

Consoleapplicaties



Algemene informatie

Onderwerp	Consoleapplicaties
Leerdoel(en)	1. Het kunnen uitleggen wat een consoleapplicatie is. 2. Het kunnen werken met invoer en uitvoer bij de console. 3. Het kunnen werken met willekeurige getallen. 4. Het kunnen toepassen van de beslissings-structuur if. 5. Het kunnen werken met de lus-structuur while. 6. Het kunnen werken met snippets. 7. Het kunnen inlezen van fysieke bestanden 8. Het kunnen wegschrijven naar fysieke bestanden
Vereiste voorkennis	1. 1. De student heeft kennis van de modules realiseren van thema 1 t/m 4. 2. De student heeft Reader 1 en 2 bestudeert.
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Een consoleapplicatie is een programma dat wordt uitgevoerd in een DOS-box (ook wel Terminal of Commandline genoemd). Dit is een venster, waarin je alleen via tekstuele commandos met het programma kunt werken. Een muis, buttons, etc zijn niet aanwezig.

Maar waarom zou je een applicatie maken dat in een DOS-box wordt uitgevoerd? Tegenwoordig zijn toch alle applicaties grafisch?

Dit heeft twee redenen:

1. Consoleapplicaties komen nog vaak voor op de achtergrond als bijvoorbeeld een service.
2. In deze reader worden consoleapplicaties gebruikt om je beter te leren programmeren. Je wordt dan minder snel afgeleid door alle “toeters en bellen” van de grafische applicaties. Pas na een aantal hoofdstukken ga je werken met Windows Form applicaties.

Inhoud

Wat is een Console-applicatie?

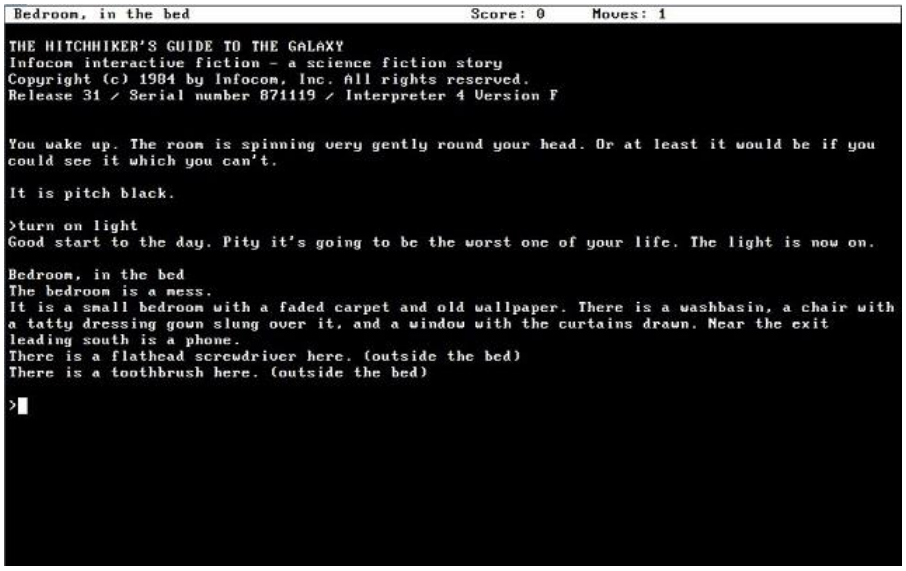
Een consoleapplicatie is een programma dat wordt uitgevoerd in een Opdrachtprompt venster (ook wel *DOS Box*, *terminal* of *commandline* genoemd). Dit is een venster, waarin je alleen via tekstuele commando's met het programma kunt werken. Een muis, buttons, etc zijn niet aanwezig.

Maar waarom zou je dan zo'n applicatie maken? Tegenwoordig zijn toch alle applicaties grafisch? Dit heeft twee redenen:

1. Consoleapplicaties komen nog vaak voor op de achtergrond als bijvoorbeeld een service.
2. In deze reader worden consoleapplicaties gebruikt om je beter te leren programmeren. Je wordt dan minder snel afgeleid door alle “toeters en bellen” van de grafische applicaties.

Op deze link (https://www.myabandonware.com/game/the-hitchhikers-guide-to-the-galaxy-42/play-42) vind je een online versie van het spelletje " The hitchhiker’s guide to the galaxy". In deze applicatie moet je door het oplossen van raadsels verder komen. Dit doe je door de opdrachten die je aan het spel wilt geven in te typen. Zo kun je bijvoorbeeld starten met het commando "Stand up" of "light", waarna het spelletje reageert met een verhaaltje. Dit is een voorbeeld van een console-applicatie die je tegenwoordig eigenlijk niet meer tegenkomt.

In de jaren 80 werden dit soort spellen vaak gespeelt op de PC. Hieronder is het beginscherm weergegeven.



Voorbeeld van een Console applicatie

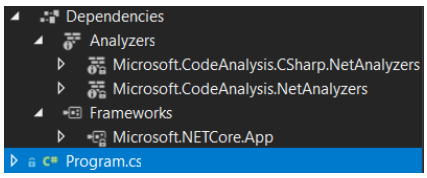
→ Open deze url in een browser: https://www.myabandonware.com/game/the-hitchhikers-guide-to-the-galaxy-42/play-42

→ Speel nu de applicatie (START) en schrijf de code op die je krijgt aan het einde van het spelletje.

→ Ben je het spel beu? Typ dan **Quit**

Je eerste console applicatie

In de vorige reader hebben we een Console Application gemaakt. Wellicht heb je dat project nog open staan, anders kun je even terugkijken in de vorige reader. Zo'n project bestaat uit de volgende bestanden:



Onder *Dependencies* kun je zien van welke libraries je programma standaard gebruik maakt. Daar komen we later nog op terug. Op dit moment is het belangrijk om te weten dat alle programmacode in het bestand Program.cs staat.

Standaard is de inhoud van Program.cs als volgt:

```
using System;

namespace ConsoleApp1
{
    0 references | 0 changes | 0 authors, 0 changes
    class Program
    {
        0 references | 0 changes | 0 authors, 0 changes
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Wat we hier zien:

- Dit bestand laadt via *using* een libraries genaamd System in.
- De *namespace* waarin dit bestand zich bevindt is *ConsoleApp1.*
- Binnen deze *namespace* is de klasse genaamd *Program* gedefinieerd.
- Binnen de klasse *Program* is een methode (ook wel functie in andere talen genoemd) genaamd *Main* gedefinieerd. Deze is *static* en heeft geen returnwaarde (*void*).
- De methode *Main* accepteert één parameter van het datatype *string[] (*dit is een *array)* genaamd *args*.

Bij het opstarten van je applicatie zal de methode *Main* als startpunt van je applicatie dienen. We kunnen dus in deze methode onze code schrijven. Zodra alle code binnen *Main* is uitgevoerd, is de applicatie klaar en zal hij zichzelf weer afsluiten.

We kunnen onze gemaakte applicatie testen. We kunnen dit doen door de programmacode te compilen (omzetten naar voor de computer herkanbare commando's). Dat doe je als volgt:

→ Klik op menuitem **Build** en kies dan voor **Build** *<projectnaam>,*
bijvoorbeeld **Build ConsoleApp1**

Visual studio laat je vervolgens het output scherm zien met iets vergelijkbaars als onderstaande tekst:

```
Build started...
1>------ Build started: Project: ConsoleApp1, Configuration: Debug Any CPU ------
1>ConsoleApp1 -> E:\Applicaties\C-
Sharp\RealiserenThema7\ConsoleApp1\bin\Debug\net5.0\ConsoleApp1.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Het belangrijkste in dit stukje tekst vind je in de laatste regel (`1 succeeded`). Zie het als een algemene programmeerregel, `succeeded` is goed, `failed` is slecht.

Je kan het programma ook uitvoeren, starten. Dit doe je als volgt:

→ Klik op menuitem **Debug** en kies dan voor **Start Debugging** of…

→ Klik op menuitem **Debug** en kies dan voor **Start Without Debugging**

Zie je verschil?

Je Console applicatie iets laten doen

→ Pas je Program.cs bestand aan zodat het eruit zien zoals in de afbeelding hieronder:

```
class Program
{
    // This is where your program starts.
    0 references | 0 changes | 0 authors, 0 changes
    static void Main(string[] args)
    {
        // Prompt user to enter a name.
        Console.WriteLine("Enter your name, please:");
        // Now read the name entered.
        string name = Console.ReadLine();
        // Greet the user with the name that was entered.
        Console.WriteLine("Hello, " + name);
        // Wait for user to acknowledge the results.
        Console.WriteLine("Press Enter to terminate...");
        Console.Read();
    }
}
```

Let hierbij goed op het gebruik van hoofdletters! Dat is in C# ontzettend belangrijk.

→ Klik op menuitem **Build** en kies dan voor **Build** *<projectnaam>,* bijvoorbeeld **Build ConsoleApp1** om de code om te zetten naar ConsoleApp1.exe.

→ Klik op menuitem **Debug** en kies dan voor **Start Without Debugging**

Het zwarte consolevenster opent en vraag om je naam. Nadat je die ingevoerd hebt en op enter hebt gedruk toont het schem *Hello,* gevolgt door de naam die je ingevoerd hebt en laat vervolgens *Press Enter to terminate* zien. Als je op enter drukt sluit het scherm.

Je kan deze applicatie ook buiten Visual studio draaien. Open hiervoor een Command prompt. Dat kan op een van de volgende manieren:

→ Klik op **Windows Startknop** en typ vervolgend **cmd** gevolgd door enter.

→ Klik in Visual Studio op **Tools** - **Command line – Developer Command Prompt**.

→ Typ in de Command prompt het volgende in:

```
CD E:\Applicaties\C-Sharp\RealiserenThema7\ConsoleApp1\bin\Debug\net5.0
consoleapp1
```

Hier is de groene tekst de plek waar jij jouw code hebt opslagen. En is de oranje tekst de naam van jouw applicatie.

Als het goed is laat de applicatie dezelfde output als eerder zien. Je kan ook met een Windows Verkenner naar bovenstaande map navigeren en dubbelklikken op ConsoleApp1.

Het Console Object

In het verleden voor de grafische besturingssystemen was de interactie tussen de gebruikers en de computer gebaseerd op tekst. De combinatie van tekst en toetsenbord is beter bekend als Console.

In C# is alles een object. In onze console applicatie maken we gebruik van het object Console. De namen van classes beginnen ALTIJD met een hoofdletter.

Tekst laten zien in de Console
Als je iets op het scherm wilt laten weergeven in een console applicatie gebruik je het onderstaande stuk code:

```
Console.WriteLine("Enter your name, please:");
```

Je begint met de naam van het object wat je wilt gebruiken **Console**. Binnen dit object gebruik je de methode **WriteLine**. Deze methode zorgt ervoor dat de tekst die tussen de aanhalingstekens staan wordt weergegeven.

Je kunt ook de methode **Write** gebruiken.

```
Console.Write("Hello, ");
Console.Write(name);
```

Je zult zien dat je hetzelfde resultaat als bij de **WriteLine** methode. Het verschil is dat je bij de methode **Write** je steeds verder gaat op dezelfde regel.

Gebruikersinvoer uitlezen en opslaan
Om te zorgen dat de terminal zichtbaar blijft, moet je ervoor zorgen dat je programma wacht op een invoer van de gebruiker. Je gebruikt hiervoor de onderstaande code.

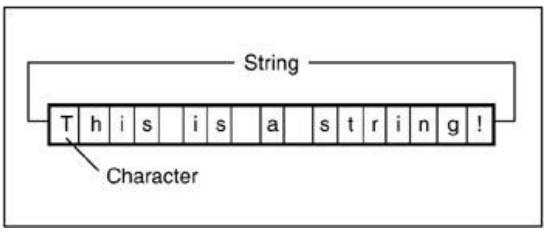
```
Console.ReadLine();
```

Echter kun je ook uitlezen wat de gebruiker invoert in deze console. Dat doe je door via de **ReadLine** methode de invoer op te slaan in een string variabele. Je gebruikt hiervoor onderstaande code.

```
string name = Console.ReadLine();
```

Je hebt hier de invoer van een gebruiker opgeslagen in een variabele van het datatype *string.* Het is belangrijk om te weten, dat invoer van een gebruiker via **ReadLine()** altijd als een *string* in ons programma ontvangen wordt. In deze regel code declareer en initialiseer je een variabele. Je maakt een variabele door het datatype te benoemen met daarachter de naam van de variabele. Dus bijvoorbeeld: `string name`, je initialiseert een variabele door deze gelijk een waarde te geven, bijvoorbeeld zo: `name = Console.ReadLine();`.

Een datatype geeft aan wat voor soort data in de variabele wordt opgeslagen. In reader 4 van thema 1 (Basis statische website) heb je een aantal datatypes van Javascript geleerd. In C# heb je meer verschillende datatypes. Een aantal van deze datatypes zie je in onderstaande tabel:

Datatype	Omschrijving	Voorbeeld waarden
String	▪ Een reeks karakters. ▪ Een string zet je altijd tussen aanhalingstekens. 	"Dit is een string" "Koning Willem I College" "123" "true" "Voorbeeld 5"
Int	▪ Een geheel getal, ook wel **integer** genoemd. ▪ Een integer zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Met strings kun je niet gaan rekenen.	1 6523 -1 -768
Double	▪ Een decimaal getal. ▪ Decimale getallen bevatten een punt in plaats van een komma. ▪ Een double zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Met strings kun je niet gaan rekenen.	123.50 0.78996 -1.23 -9623.6543
Bool of Boolean	▪ Een variabele van het type bool of Boolean kent maar twee mogelijke waarden **true** of **false**. ▪ Een boolean zet je niet tussen aanhalingstekens. Als je dat doet wordt het een string. Je slaat dan de tekst "true" of "false" op. Let op: allemaal kleine letters!	true false
DateTime	▪ Een variabele van het type DateTime bevat een datum EN tijd ▪ Met een variabele van het type DateTime kun je wel 80 verschillende operaties uitvoeren, zoals alleen de maand laten zien, de dag van de week laten zien, dagen, uren, minuten erbij optellentwee datums van elkaar afhalen.	DateTime thisYear = new DateTime(2021, 1, 1); DateTime thisMoment = DateTime.Now; DateTime anHourFromNow = thisMoment.AddHours(1);

De programmeertaal C# is veel strikter dan de taal Javascript. Een variabele is van een bepaald type en dat kan niet veranderen. In Javascript kon je in een variabele eerst een **string** plaatsen en vervolgen een **number**. Dat accepteert C# niet. Hoe kunnen we dan bijvoorbeeld een getal maken van een ongevoerde string? Daarvoor moet we de string omzetten via het door .NET ingebouwde klasse **Convert.** Een voorbeeld van het omzetten naar een *integer* zie je hieronder:

```
// Declareren van variabele (integer)
int amountofCars;
// Gebruikersinvoer vragen via readLine() en direct converteren naar een getal (Int32)
amountofCars = Convert.ToInt32( Console.ReadLine() );
// Tonen van de gebruikersinvoer
Console.WriteLine("Het opgegeven aantal auto's is: " + amountofCars);
```

Via de klasse **Convert** kun je variabelen omzetten naar allerlei andere datatypes (int, double, boolean, DateTime en nog vele meer).

Tekstopbouw in een console applicatie

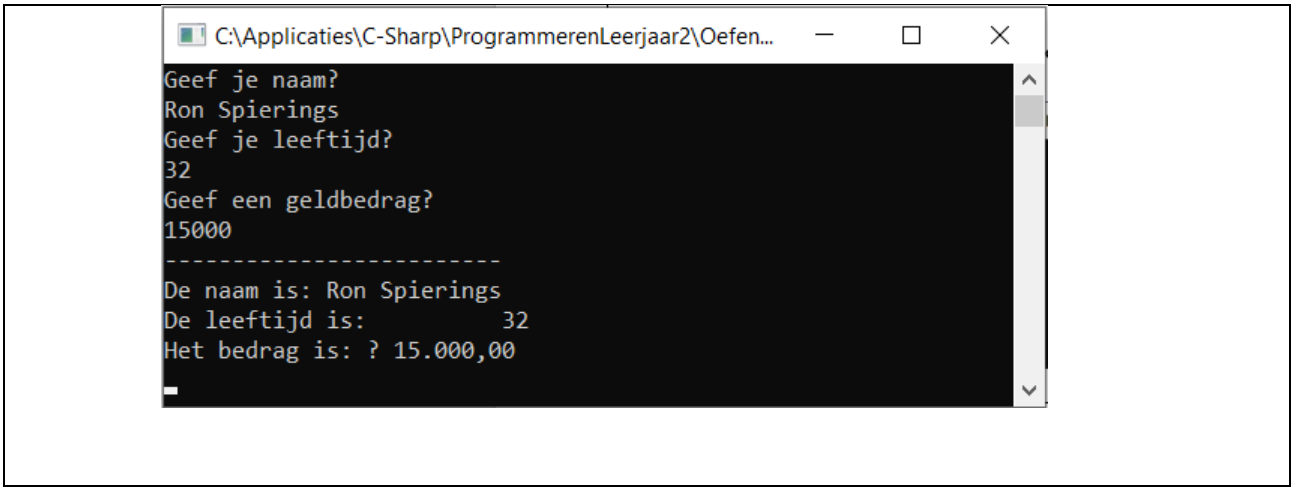
Je hebt in de vorige paragraaf gezien hoe je een tekst kunt laten weergeven op het scherm en omzetten naar een ander datatype via Convert. In deze paragraaf ga je ook wat opmaak toepassen zoals het werken met tabs, enters, enz.

→ Volg onderstaande stappen

- ☑ Maak een nieuw Console project aan in je Instructie map. Noem deze InstructieConsole.
- ☑ Neem onderstaande code over:

```
11      static void Main(string[] args)
12      {
13          // Consoleapplicatie tekstopmaak
14          // Reader 4
15          // Gemaakt door: R. Spierings
16          // Vak: Programmeren
17
18          string name;
19          string age;
20          double amount;
21
22          Console.WriteLine("Geef je naam?");
23          name = Console.ReadLine();
24          Console.WriteLine("Geef je leeftijd?");
25          age = Console.ReadLine();
26          Console.WriteLine("Geef een geldbedrag?");
27          amount = Convert.ToDouble(Console.ReadLine());
28
29          Console.WriteLine("").PadRight(25, '-'));
30          Console.WriteLine("De naam is: " + name);
31          Console.WriteLine("De leeftijd is: \t {0}", age);
32          Console.WriteLine("Het bedrag is: {0,8:c}", amount);
33
34          Console.ReadKey();
35      }
```

- ☑ Start je programma en kijk wat er gebeurt.



Je ziet dat er verschillende codes in staan die ervoor zorgen dat de tekst er iets anders uit komt te zien:

`\n`	Geeft een enter
`\t`	Plaatst een tab
`("").PadRight(25, '-')`	Plaatst een lijn bestaande uit 25 '-'
`{0,8:c}`	Laat het ingevoerde getal zien als een bedrag.

Omdat de voorbeelden zijn gemaakt op een Nederlandstalige versie van Windows 10 en de landinstellingen dus Nederlands zijn, verschijnt het bedrag op de Nederlandse schrijfwijze. Dit kun je aanpassen door de landinstellingen in Windows aan te passen.

Console.Clear()

Het is mogelijk om alle inhoud van de Console leeg te gooien, en dus een leeg scherm te krijgen.

Voer hiervoor onderstaande code uit:

```
// Gooi de inhoud van de Console leeg.
Console.Clear();
```

Kleuren in de Console

Laten we wat gaan "spelen met kleuren". Volg onderstaande stappen:

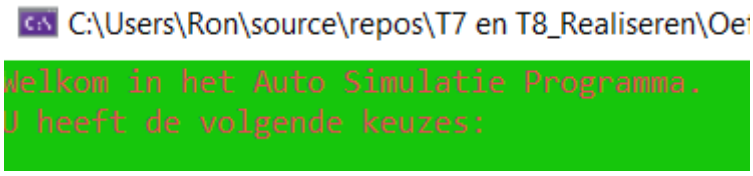
- ☑ Doe de onderstaande stappen na. Doe dit binnen het eerder gemaakte project die bij deze reader hoorde.
- ☑ Voordat je een **WriteLine**() gaat aanroepen, zet de tekstkleur op rood door de property genaamd **ForegroundColor** van het **Console** object aan te passen naar een andere kleur. Neem onderstaand voorbeeld over:

```
Console.ForegroundColor = ConsoleColor.Red;
```

- ☑ Voeg enkele WriteLine()'s toe.
- ☑ Run je programma en bekijk de output.
- ☑ Maak de achtergrondkleur Groen wit, doormiddel van het **BackgroundColor** property op het **Console** object aan te passen naar een andere kleur.

```
Console.BackgroundColor = ConsoleColor.Green;
```

- ☑ Voorbeeld uitvoer:



- ☑ De **ConsoleColor** bevat de volgende kleuren:

```
public enum ConsoleColor
{
    Black = 0,
    DarkBlue = 1,
    DarkGreen = 2,
    DarkCyan = 3,
    DarkRed = 4,
    DarkMagenta = 5,
    DarkYellow = 6,
    Gray = 7,
    DarkGray = 8,
    Blue = 9,
    Green = 10,
    Cyan = 11,
    Red = 12,
    Magenta = 13,
    Yellow = 14,
    White = 15
}
```

Samenvatting

In dit hoofdstuk heb je voor het eerst gewerkt met de ontwikkelomgeving van C#. Je hebt consoleapplicaties gemaakt door gebruik te maken van de klasse Console. Daarnaast heb je gekeken naar de Converter klasse, zodat je nu klaar bent om variabelen om te zetten van het ene datatype, naar het andere (bijvoorbeeld een string naar integer).

Als je het scherm in een consoleapplicatie weer leeg wilt maken, kun je het onderstaande stuk code gebruiken:

```
Console.Clear();
```

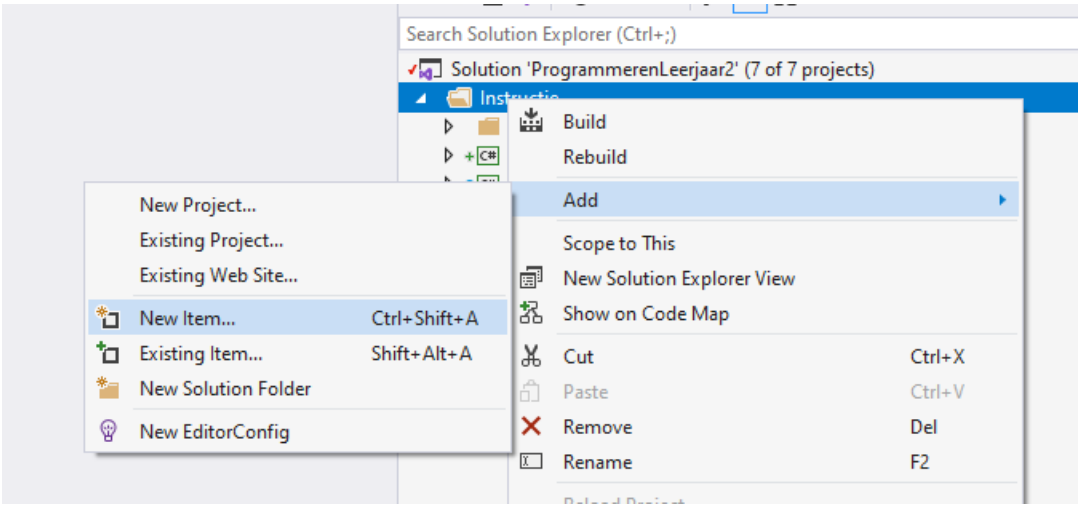
Maak nu **Oefening 3.1**, om te oefenen met de teksten binnen de Console

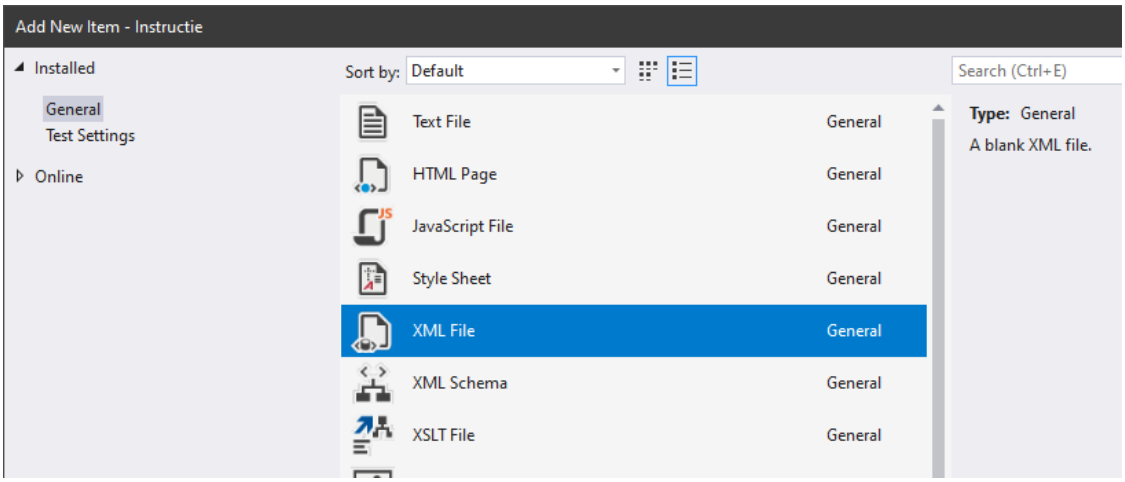
Snippets maken

Een snippet is een code die je op een snelle manier kunt aanroepen. Het is als het ware een snelkoppeling naar veelgebruikte code, zodat je het niet eerdere keer helemaal hoeft te typen.

Omdat je ieder codebestand moet gaan voorzien van een moduleheader, kunnen we van de moduleheader een snippet maken.

- ☑ Open de solution en voeg in de map instructie een XML bestand toe. Klik hiervoor met rechts op de map instructie en kies voor **add** → **New Item.**



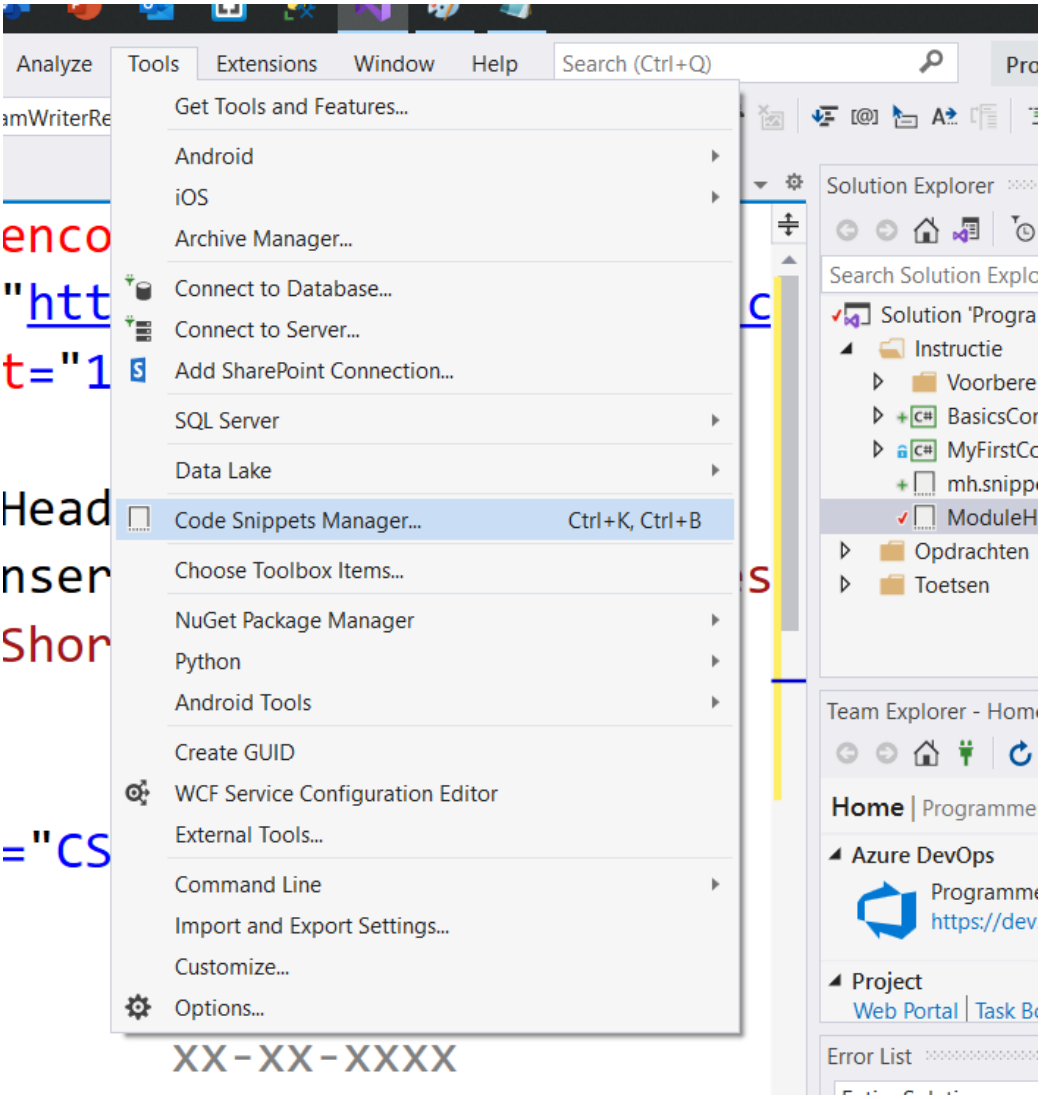


- ☑ Geef het bestand de naam **ModuleHeader.snippet** (dus zonder .xml!).
- ☑ Plak onderstaande code in het bestand.

```
<?xml version="1.0" encoding="utf-8"?>
```

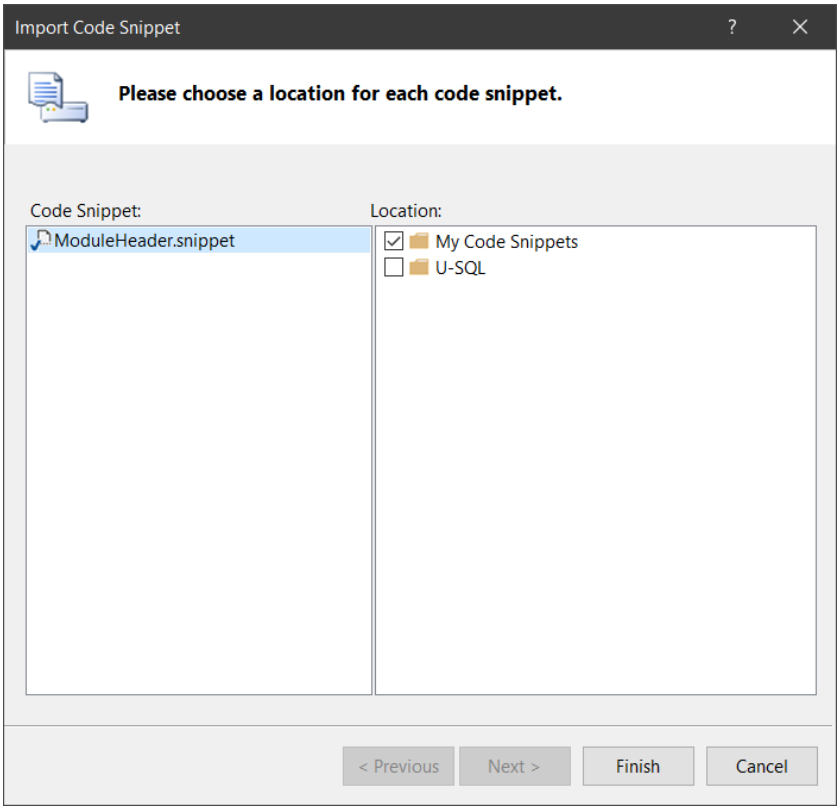
```
<CodeSnippets
xmlns="http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format="1.0.0">
    <Header>
      <Title>Module Header</Title>
      <Description>Inserts a module header</Description>
      <Shortcut>mh</Shortcut>
    </Header>
    <Snippet>
      <Code Language="CSharp">
        <![CDATA[
        //  Author:       Jouw naam
        //  Date:         xx-xx-xxxx
        //  Assignment:   x.x
        //  Description:
        ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

- ☑ Sla het bestand op.
- ☑ Ga in het menu naar Tools → Code Snippets Manager.



- ☑ Kies bij Language voor **Csharp**.
- ☑ Klik onderin het scherm op **Import**.

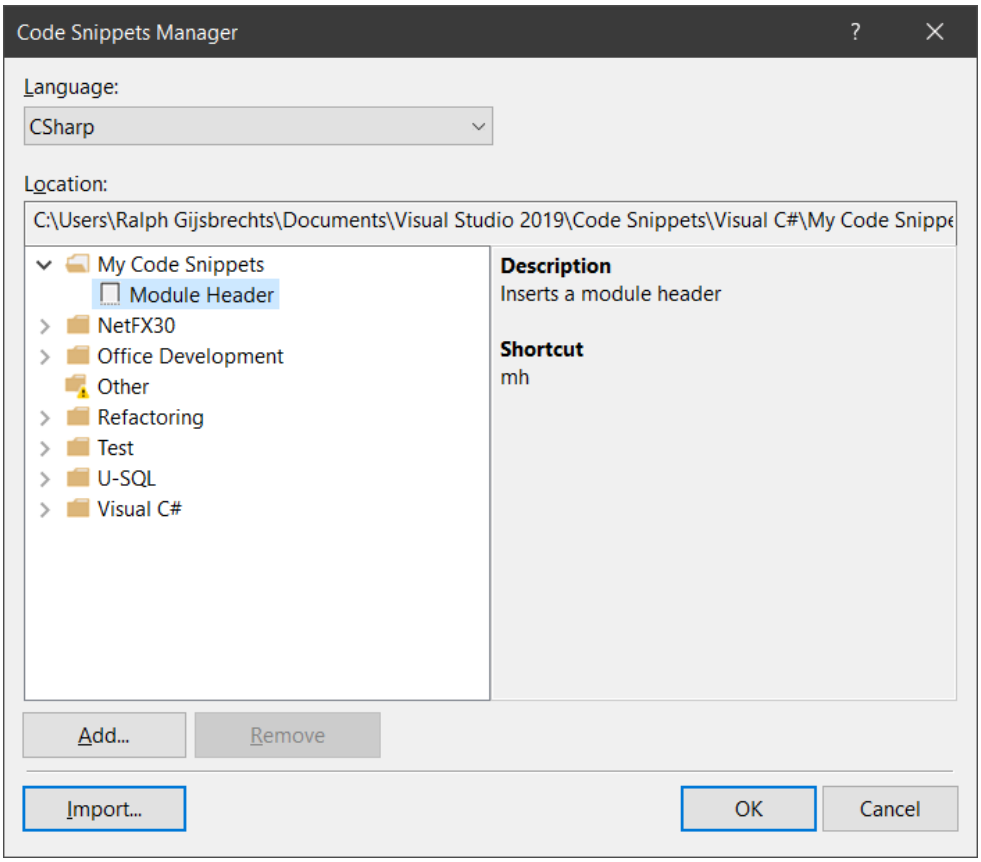
☑ Zoek het zojuist aangemaakte bestand op en klik op **Openen**.



☑ Zorg ervoor dat je als locatie My Code Snippets geselecteerd hebt.

☑ Klik op **Finish**.

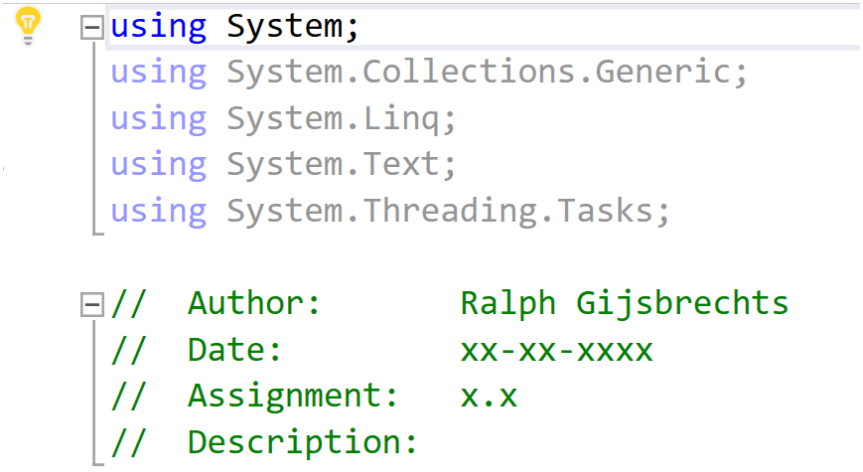
Je kunt nu zien of jouw snippet succesvol is aangemaakt:



Je ziet dat er bij shortcut mh staat. Dit heb je in het XML bestand ook opgegeven in het element shortcut.

Zodra je nu in een C# bestand aan het werk bent kun je **mh** typen gevolgd door 2x tab. De code wordt dan automatisch toegevoegd.

LET OP: plaats de module header onder de usings! Boven de using werkt de snippet niet.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

//  Author:        Ralph Gijsbrechts
//  Date:          xx-xx-xxxx
//  Assignment:    x.x
//  Description:
```

Het inlezen en wegschrijven van bestanden

Op een computer is het gebruikelijk dat je kunt werken met bestanden die te vinden zijn op je harde schijf. Denk bijvoorbeeld aan tekst only bestanden, gemaakt in Kladblok, maar ook aan wat complexere bestanden zoals Word documenten of pdf-documenten. Ieder bestand dat daadwerkelijk op je harde schijf, CD/ DVD speler of USB-stick bevindt, noemen we een fysiek bestand. In deze reader gaan we leren hoe je via een applicatie fysieke bestanden kunt uitlezen, maar ook wegschrijven (aanpassen/ aanmaken). Dit doen we door gebruik te maken van Streaming.

Met het werken in fysieke bestanden in welke programmeertaal dan ook, is altijd een lastige klus. Je moet er altijd rekening mee houden dat er een hoop mis kan gaan (bestand is verwijderd, geen toegang tot het bestand, het bestand wordt tegelijk door een ander programma beschreven, etc). Ook kan een bestand enorm groot zijn (enkele gigabytes) of juist leeg zijn.

Het grootste probleem wat we ondervinden tijdens het werken met fysieke bestanden is de onbekende grootte van een bestanden. Want als je een bestand inlaadt wat 4 gigabyte groot is, zit direct je RAM-geheugen vol en crasht je programma. Daarom dienen we bestanden altijd in te laden via zogeheten **Streamreaders / Writers**. Die zullen een bestand nooit direct volledig inladen, maar alleen de porties van het bestand die jij nodig hebt.

Daarnaast is er een groot verschil tussen het inlezen van een bestand en het wegschrijven van een bestand. Inlezen van bestanden doen we via een **StreamReader**, en het wegschrijven doen we via een **StreamWriter**. Dit zijn 2 klassen die ingebouwd zitten in het .NET Framework en de meest gangbare manier van bestandsafhandeling. In deze reader gaan we enkel werken met simpele tekstbased bestanden (zoals .txt of .html).

Stel je een tekstbestand voor als een stel regels, elk met een wisselend bepaald aantal tekens. Elke regel wordt afgesloten met een **end-of-line markering**, hetzij een newline-feed teken '**\n**', een carriage-return '**\r**' of beide '**\r\n**'. Een newline teken '\n' is de notatie van Windows voor een nieuwe regel, en is de standaard in de computerwereld.

Hoe nam je vroeger je bestanden mee op pad? Via Floppy's! Hoe werkte dit?
https://www.youtube.com/watch?v=EHRc-QMoUE4

Gegevens wegschrijven in een bestand inlezen via StreamWriter

Streaming maakt het mogelijk om in één doorlopende beweging gegevens te lezen of schrijven naar bestanden. Zoals de term al suggereert, "stromen" de gegevens van een fysiek bestand naar het programma en omgekeerd. Belangrijk hierin is om te weten welke stappen jouw programma moet doorlopen om gegevens naar een bestand toe te schrijven:

1. Het bestand openen
2. Gegevens in de juiste volgorde naar het bestand te schrijven (uitvoeren)
3. Het bestand afsluiten als we klaar zijn

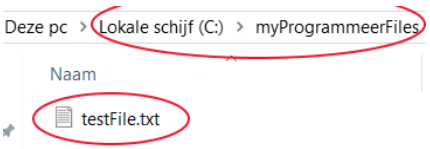
Voor het inlezen en wegschrijven van regels tekst gebruiken we de volgende methoden en klassen:

- *CreateText(),* uit de klasse *File* om een bestand te openen.
- *WriteLine()*,uit de klasse *StreamWiter* om één regel tekst naar een bestand te schrijven.
- *Close()*, uit de klasse *StreamWriter* om een bestand te sluiten.

Deze methodes zijn al voor jou geprogrammeerd in het .NET Framework. Echter moet je nog wel even vertellen dat je deze StreamWriter en StreamReader beschikbaar wilt hebben in jouw project. Dit doe je door een "using System.IO" toe te voegen.

Laten we gaan starten met een voorbeeld hoe je een bestand wegschrijft. Volg daarom onderstaande stappen.

- ☑ Maak op je C:\ schrijf een nieuwe map genaamd "myProgrammeerFiles" aan. Maak binnen deze map een nieuw kladblok bestand genaamd "testFile.txt" aan:



  Dit doen we omdat we standaard geen toegang hebben tot bestanden op de C:\ schijf. Omdat we een nieuwe map aanmaken, hebben we wel toegang!

- ☑ Maak in je instructiemap in je Visual Studio Solution een nieuw project genaamd "StreamWriterExample" aan.

- ☑ Voeg in Program.cs de volgende using toe (bovenaan het bestand). Dit geeft aan dat je alle functionaliteiten van de System.IO library in je deze file wilt kunnen gebruiken.

```
using System.IO;
```

- ☑ Neem onderstaande code over. Bestudeer de code goed:

```
StreamWriter writer = File.CreateText(@"C:\myProgrammeerFiles\testFile.txt");
writer.WriteLine("Testregel 1");
writer.WriteLine("Nog een testregel!");
writer.Close();
```

- ☑ Voer je programma één keer uit. Geen foutmeldingen gezien? Open dan het eerder gemaakte testFile.txt. Als het goed is, zie je de tekst die je via *WriteLine()* hebt opgegeven in het bestand. Zie je dit niet? Dan gaat er iets fout (lees goed eventuele foutmeldingen).

- ☑ Sluit het bestand testFile.txt en start je programmacode een aantal keer opnieuw.

- ☑ Open opnieuw testFile.txt. Hoe ziet het bestand er nu uit?

- ☑ **Vraag:** Waarom worden de oude regels tekst niet bewaard?
  Hint: Lees hier waarom.

- ☑ Verander het filepath dusdanig, dat je in de *CreateText* methode nu het pad "anotherTestFile.txt" staat (zie onderstaande afbeelding). Dit bestand zal dan aangemaakt worden.
  ```
  StreamWriter writer = File.CreateText(@"anotherTestFile.txt");
  ```

- ✓ Voer je programma nogmaals uit, en zoek uit waar je bestand "anotherTestFile.txt" nu heen is geplaatst op je hardeschijf.
  Tip: Bekijk de locatie van je project!

- ✓ Zorg ervoor dat je programma een ongeldige bestandsnaam ontvangt. Wat gebeurt er nu?

Je kunt nu tekst naar een bestand wegschrijven door gebruik te maken van de *StreamWriter* klasse, die zich bevindt in de library System.IO.

Gegevens uit een bestand inlezen via StreamReader

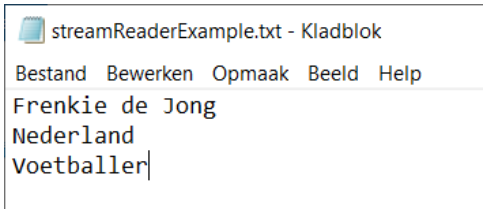
Streaming maakt het mogelijk om in één doorlopende beweging gegevens te lezen of schrijven naar bestanden. Belangrijk hierin is om te weten welke stappen jouw programma moet doorlopen om gegevens uit een bestand uit te lezen:

1. Het bestand openen
2. Gegevens stuk voor stuk lezen en toekennen aan variabelen (invoeren)
3. Het bestand afsluiten als we klaar zijn

Voor het inlezen van bestanden gebruiken we de klasse *StreamReader*. Volg onderstaande stappen om een bestand uit te lezen.

- ☑ Maak in de map C:\myProgrammeerFiles een nieuw bestand aan genaamd "streamReaderExample.txt"

- ☑ Vul dit bestand met de onderstaande gegevens:



- ☑ Maak in je instructiemap in Visual Studio een nieuw project aan genaamd "StreamReaderExample".

- ☑ Voeg aan Program.cs wederom de using System.IO toe om gebruik te kunnen maken van deze library (zie vorige hoofdstuk).

```
using System.IO;
```

- ☑ Voeg onderstaande code toe:

```
// Geef een filepath op
string filepath = @"c:\myProgrammeerFiles\StreamReaderExample.txt";

// Open de file via OpenText
StreamReader reader = File.OpenText(filepath);

// Haal de eerste regel op uit het bestand
string name = reader.ReadLine();

// Haal de 2e regel op uit het bestand
string country = reader.ReadLine();

// Haal de 3e regel op uit het bestand
string profession = reader.ReadLine();

// Sluit het bestand
reader.Close();
```

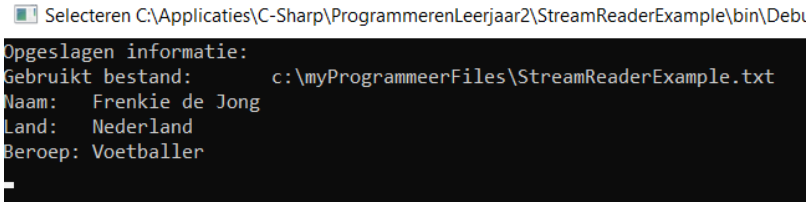
- ☑ Je hebt nu in de 3 regels tekst uit het bestand "StreamReaderExample.txt" opgeslagen in 3 verschillende variabelen opgeslagen.
  De stream is daarnaast afgesloten (via *close()*), maar de variabelen blijven

gewoon bestaan!

- ☑ Plaats onder de vorige code, onderstaande code:

```
Console.WriteLine("Opgeslagen informatie:");
Console.WriteLine("Gebruikt bestand: \t" + filepath);
Console.WriteLine("Naam: \t" + name);
Console.WriteLine("Land: \t" + country);
Console.WriteLine("Beroep:\t" + profession);
Console.ReadLine();
```

- ☑ Je moet dan deze screenshot krijgen:



- ☑ Verander de gegevens uit het bestand "StreamReaderExample.txt" om te controleren of alles werkt zoals gewenst.

- ☑ Bestudeer de code nogmaals. Snap je wat er gebeurt en waarom?

Je kunt nu een bestand waarvan je weet hoeveel regels het heeft inlezen. Maar hoe ga je om met een bestand waarvan je niet weet hoeveel regels het heeft? Dan gebruik je een *while* loop.

- ☑ Via onderstaande code gaan we het bestand doorlopen, totdat we bij het einde komen:

```
string filepath = @"C:\myProgrammeerFiles\testFile.txt";
StreamReader reader = File.OpenText(filepath);
string line = reader.ReadLine();
while(line != null)
{
    Console.WriteLine(line);
    line = reader.ReadLine();
}
reader.Close();
```

- ☑ Probeer nu zelf een .txt-bestand te maken, die je in zijn geheel laat zien via een *while* loop.

Samenvatting

Je hebt geleerd om tekstbestanden te openen en de tekstinhoud regel voor regel uit te lezen. Daarnaast ben je in staat om gegevens naar een bestand toe weg te schrijven, zodat deze voor de langere termijn opgeslagen zijn.

Maak nu **Oefening 3.2**, om te oefenen met bestanden inladen en en wegschrijven