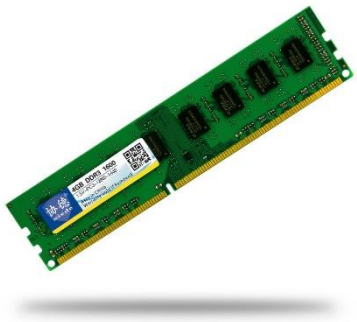
C# 2 – Les 1: Tekstbestanden en errors opvangen met try/catch

Tot nu toe hebben we de gegevens van ons programma bijgehouden in het werkgeheugen van de computer. Bij het beëindigen van het programma verdwijnen al de gegevens aangezien het werkgeheugen bestaat uit RAM-geheugen (= “vluchtig” geheugen).



Je kan de gegevens ook **bewaren op je harde schijf** zodat de gegevens behouden blijven als we het programma afsluiten. We gaan in dit document gebruik maken van **tekstbestanden** om onze gevens op te slaan. (Een andere optie is om de gegevens op te slaan in een database-systeem, dat doen we in deze les niet.)

Om alles goed te laten werken moeten we zowel kunnen **lezen** als **schrijven** in bestanden. Gelukkig gaat dit met C# heel gemakkelijk. We gaan eerst uit een bestand lezen.

Het is belangrijk dat je het bestand, waaruit je wil lezen, in de juiste map op je computer plaatst. Als het programma het bestand niet vindt crasht het programma met een foutmelding.

Een bestand inlezen

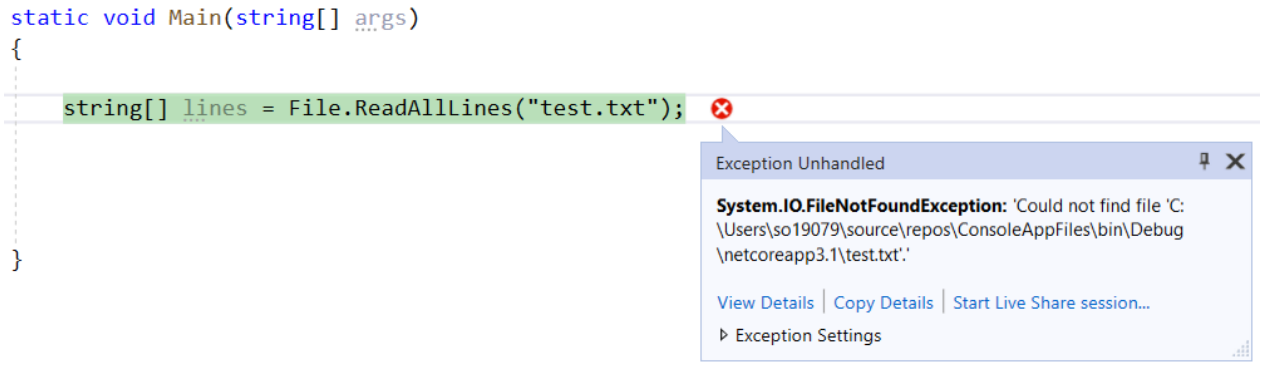
Een tekstbestand bestaat normaalgezien uit meerdere regels (elke regel wordt afgesloten door het speciale teken *newline*, ook wel een *enter* of *return* genoemd). Een tekstbestand wordt bewaard met de extensie txt. Je kan met onderstaande code iedere regel uit een bestand gaan inlezen. De regels worden in een Array van strings geplaatst.

```
string[] lines = File.ReadAllLines("test.txt");
```

File.ReadAllLines zit in System.IO, Visual Studio zal vragen om een using regel toe te voegen:

```
using System.IO;
```

Als je de File.ReadAllLines in je Main methode zet en het programma probeert uit te voeren zonder dat het bestand test.txt bestaat zal het programma crashen en ons ook zeggen waar het programma het bestand verwacht had.



In dit geval moet het bestand in de "… bin\Debug\netcoreapp3.1" map geplaatst worden, dat is de plaats waar het programma wordt uitgevoerd.

Tekstbestand aanmaken

Maak een tekstbestand met een paar regels in en bewaar het als test.txt op de juiste locatie. Op Windows kan je hiervoor het programma Notepad (= Kladblok) gebruiken.

Op Mac kan je Teksteditor gebruiken, maar zorg wel dat je de tekst omzet in "platte tekst" ('Opmaak' => 'Converteer naar platte tekst…') zodat je kan bewaren naar een .txt-bestand.

Als alternatief kan je een website zoals https://www.editpad.org/ gebruiken om snel een tekstbestand te maken.

Als je het bestand op de juiste plaats hebt gezet zou de foutmelding verdwenen moeten zijn. De regels van het bestand zijn nu in de Array *lines* geplaatst.

```
static void Main(string[] args)
{
    string[] lines = File.ReadAllLines("test.txt");

    foreach (var l in lines)
    {
        Console.WriteLine(l);
    }
}
```

try/catch

Zoals je hierboven hebt gemerkt kan het programma crashen als het tekstbestand niet aanwezig is of een foute naam heeft. In plaats van het programma te laten crashen kunnen we het programma normaal laten eindigen met een vriendelijke boodschap voor de gebruiker.

We gebruiken hiervoor een try- en catchblok. Als de ReadAllLines-methode het bestand niet vindt vangen we de fout op: het catch-blok wordt uitgevoerd en we tonen de foutmelding aan onze gebruiker en stoppen in dit geval het programma met *return*.

```
static void Main(string[] args)
{
    string[] lines = {};

    try
    {
        lines = File.ReadAllLines("test.txt");
    }
    catch (FileNotFoundException e)
    {
        // Bestand is niet gevonden:
        // we tonen de fout-boodschap en stoppen het programma
        Console.WriteLine(e.Message);
        return; // deze regel stopt de Main methode (dus het programma stopt)
    }

    // Als het try-blok gelukt is gaat het programma hier verder
    foreach (var l in lines)
    {
        Console.WriteLine(l);
    }
}
```

Schrijven naar een bestand

Met File.WriteAllLines kunnen we schrijven naar een tekstbestand. Omdat we lijnen willen gaan toevoegen gebruiken we een List in plaats van een Array. De Array die File.ReadAllLines teruggeeft zetten we daarom om naar een List met de ToList methode (je hebt hier een extra using-regel voor nodig):

```
static void Main(string[] args)
{
    List<string> lines = new List<string> {};

    try
    {
        lines = File.ReadAllLines("test.txt").ToList();
    }
```

```
    catch (FileNotFoundException e)
    {
        Console.WriteLine(e.Message);
        return; // deze regel stopt de Main methode
    }

    Console.WriteLine("Regels in het bestand: ");

    foreach (var l in lines)
    {
        Console.WriteLine(l);
    }

    Console.Write("Geef een nieuwe regel: ");
    string input = Console.ReadLine();

    lines.Add(input);

    File.WriteAllLines("test.txt", lines);

    Console.WriteLine("Nieuwe regels in het bestand: ");

    foreach (var l in lines)
    {
        Console.WriteLine(l);
    }
}
```

Probeer het programma eens twee keer achtereen uit, de nieuw toegevoegde regels blijven bewaard.

Oefening: TODO-list

Maak een programma dat een tekstbestand inleest en de regels toont op het scherm in een genummerde todo-lijst.

Zorg dat de gebruiker de mogelijkheid heeft om

- Een regel toe te voegen
- Een regel te veranderen
- Een regel te verwijderen

(tot “q” wordt ingegeven)

Zorg dat de regels in het bestand worden bewaard.

EXTRA: Voorzie een mogelijkheid om items in de lijst aan te duiden als afgehandeld. Al de afgehandelde items toon je in het groen, al de nog af te handelen items in het rood.

TIP: Hiervoor heb je een extra stukje informatie nodig dat je mee opslaat per regel. Je kan dit achteraan elke regel toevoegen en elke regel met .Split(…) opslitsen.