

C# Essentials

Foutafhandeling en debugging

Lector: Tom Quareme

Fouten

- **Verschillende soorten fouten**
 - **Syntaxisfouten:**
 - Fouten tegen de taalregels (grammatica) van programmeertaal.
 - Zie je aan rode gekleurde golfjes tijdens programmeren.
 - **Logische fouten/bugs:**
 - Gebruik van foute logica in je programma. vb/ foute indexwaarde
 - Kan je nagaan door te debuggen met breakpoints.
 - **Runtime-fouten:**
 - Crashen van je programma terwijl het runt.
 - Opvangen via gestructureerde foutafhandeling (try-catch).

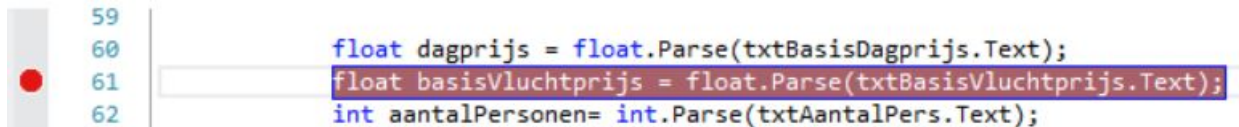
Debugging: fouten opsporen

- **Breakpoints:**

- Onderbrekingspunten aanduiden op een bepaalde regel in je code.
- Om code te pauzeren op die regel.
- Kijken welke waarden de variabelen hebben op dat moment.

- **Breakpoints instellen:**

- Klik in Visual Studio op de grijze blak vóór het regelnummer.
- OF: klik op een regel code, ga naar Debug > Toggle Breakpoint of druk F9.



- Start programma of druk F11.
- Controleer waarden van variabelen door met de muis erover te hoveren.

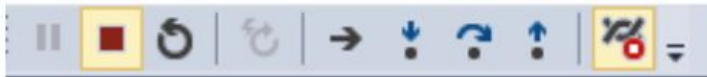
float basisVluchtprijs = f]

basisVluchtprijs 0.0 ⇐

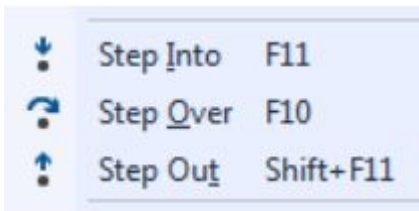
Debugging: fouten opsporen

- **Programma doorlopen na breakpoint in te stellen:**

- Bovenaan Visual Studio is een balkje verschenen:









- Betekenis:



- **Step Into (F11):** spring in de method waar je nu zit.
- **Step Over (F10):** ga 1 regel verder in code
- **Step Out (Shift+F11):** spring uit de method waar je nu zit.
- **Run To Cursor (Ctrl+F10):** stuk code uitvoeren tot waar cursor is








Debugging: fouten opsporen

- **Watch venster (wanneer je aan het debuggen bent ga naar: Debug > Windows > Watch > Watch 1)**
 - Is het meest gebruikte debug venster!
 - Typ eerst variabele namen in om deze te gaan inspecteren.
 - Waardes en eigenschappen van lokale variabelen bekijken.
 - Rood geeft de laatste wijziging aan.

Watch 1		
Name	Value	Type
 vluchtprijs	640.0	float
 dagprijs	60.0	float
 aantalDagen	7	int
 verblijfsprijs	1176.0	float
 aantalPersonen	4	int
 teBetalen	0.0	float

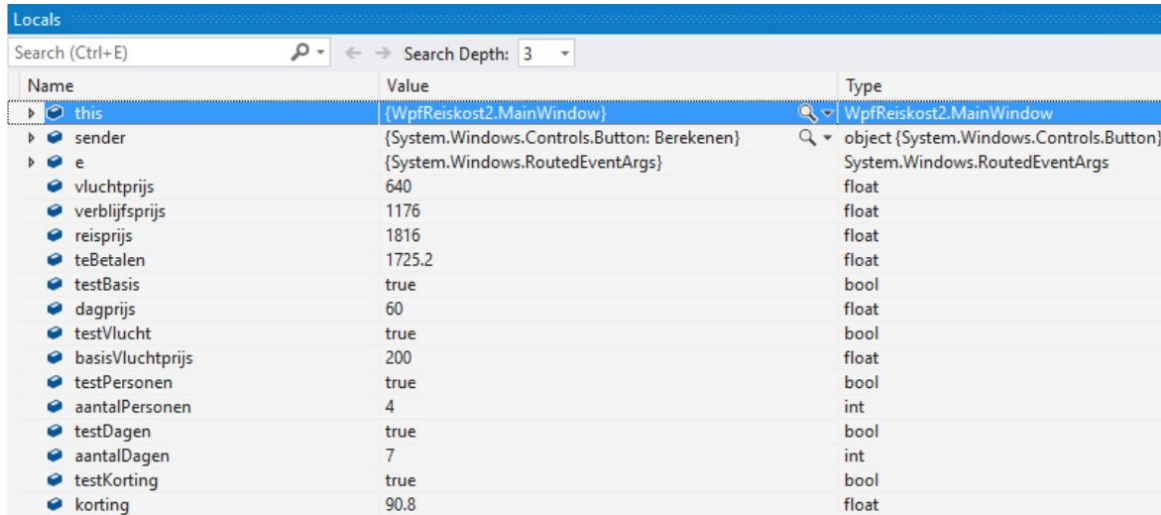
Debugging: fouten opsporen

- **Autos venster (Debug > Windows > Autos)**
 - Toont de waarden en eigenschappen van het huidig en vorige statement.
 - Rood geeft de laatste wijziging aan in de huidige method.

Autos		
Name	Value	Type
 korting	90.8	float
 reisprijs	1816.0	float
 teBetalen	0.0	float
  this	{reiskost2.frmReiskost2, Text: }	reiskost2.frmReiskost2
  txtKortingspercentage	{System.Windows.Forms.TextBox, Text: 5}	System.Windows.Forms.TextBox

Debugging: fouten opsporen

- **Locals venster (Debug > Windows > Locals)**
 - Toont alle lokale variabelen binnen de huidige method tijdens uitvoering van programma.



The screenshot shows the 'Locals' window in Visual Studio. At the top, there is a search bar with the text 'Search (Ctrl+E)' and a magnifying glass icon, and a 'Search Depth' dropdown set to '3'. Below this is a table with three columns: 'Name', 'Value', and 'Type'. The table lists various local variables, with 'this' selected at the top. The variables include 'sender' (a button), 'e' (event args), and several numerical and boolean variables like 'vluchtprijs', 'verblijfsprijs', 'reis prijs', 'teBetalen', 'testBasis', 'dagprijs', 'testVlucht', 'basisVluchtprijs', 'testPersonen', 'aantalPersonen', 'testDagen', 'aantalDagen', 'testKorting', and 'korting'.

Name	Value	Type
this	{WpfReiskost2.MainWindow}	WpfReiskost2.MainWindow
sender	{System.Windows.Controls.Button: Berekenen}	object {System.Windows.Controls.Button}
e	{System.Windows.RoutedEventArgs}	System.Windows.RoutedEventArgs
vluchtprijs	640	float
verblijfsprijs	1176	float
reis prijs	1816	float
teBetalen	1725.2	float
testBasis	true	bool
dagprijs	60	float
testVlucht	true	bool
basisVluchtprijs	200	float
testPersonen	true	bool
aantalPersonen	4	int
testDagen	true	bool
aantalDagen	7	int
testKorting	true	bool
korting	90.8	float

Debugging: fouten opsporen

- **Immediate Window (Debug > Windows > Immediate Window)**
 - Waarden van variabelen of expressies tijdens het runnen aanpassen.

Immediate Window

? reiskost

error CS0103: The name 'reiskost' does not exist in the current context

? reisprijs

1816

reisprijs = 1500

1500

? reisprijs

1500

Exceptions: fouten afhandelen

- **Runtime-fouten:**
 - Crashen van je programma terwijl het runt.
 - Software of hardware probleem.
 - Kunnen gegevens doen verloren gaan in bestanden
 - Kunnen fouten in bestanden veroorzaken ⇒ beschadigd bestand
- **Runtime-fouten afhandelen:**
 - Fouten opvangen via gestructureerde foutafhandeling (**try-catch**).
 - Runtime-fouten behandel je als exceptions.

Exceptions: fouten afhandelen

- **Algemene vorm van foutafhandeling:**

```
try
{
    'Beschermd code'
    Statements die een runtimefout veroorzaken
}
catch
{
    'ErrorHandler of exceptionhandler'
    Statements die uitgevoerd worden bij een runtimefout
}
finally
{
    Optionele statements die altijd uitgevoerd worden
}
```

Exceptions: fouten afhandelen

- try ... catch ... finally voorbeeld

```
try
{
    int[] mijnNummers = {1, 2, 3};
    Console.WriteLine(mijnNummers[10]); // runtime fout: out of bounds
}
catch (Exception ex)
{
    // fout afhandelen en programma niet laten crashen
    Console.WriteLine("Dit is een fout!"); // een eigen melding
    Console.WriteLine(ex.Message); // toon de echte foutmelding die het systeem geeft
}
finally
{
    Console.WriteLine("Deze code wordt ALTIJD uitgevoerd na try indien geen fout of na catch indien wel een fout");
}
```

Exceptions: fouten afhandelen

- **try ... catch ... finally**
 - **Try blok:**
 - Hierin komt je gewone code waarin mogelijks een fout zit.
 - Je probeert (try) die code uit.
 - Als er een fout optreedt belanden we in een catch blok.
 - **Catch blok:**
 - Hierin komt code die optreedt wanneer een fout voorkomt.
 - **ER KUNNEN MEERDERE CATCH BLOKKEN ZIJN!**
 - **Finally blok:**
 - Deze code wordt altijd uitgevoerd.
 - Meestal voor opruimcode (bestanden sluiten, databaseconnectie sluiten).

Exceptions: fouten afhandelen

- **Meerdere catch blokken**

```
int deeltal, deler, quotient;
try
{
    deeltal = Convert.ToInt32("10");
    deler = Convert.ToInt32("0");
    quotient = deeltal / deler;
    string quotientText = quotient.ToString();
}
catch (DivideByZeroException) // deling door 0
{
    Console.WriteLine("Delen door 0 mag niet!");
}
catch (FormatException) // een getal is k of 1.6 bijvoorbeeld => een conversiefout
{
    Console.WriteLine("Je moet 2 gehele getallen ingeven!");
}
catch (Exception ex) // andere fout
{
    Console.WriteLine(ex.Message); // druk foutmelding af gegenereerd door het systeem
}
```

Exceptions: fouten afhandelen

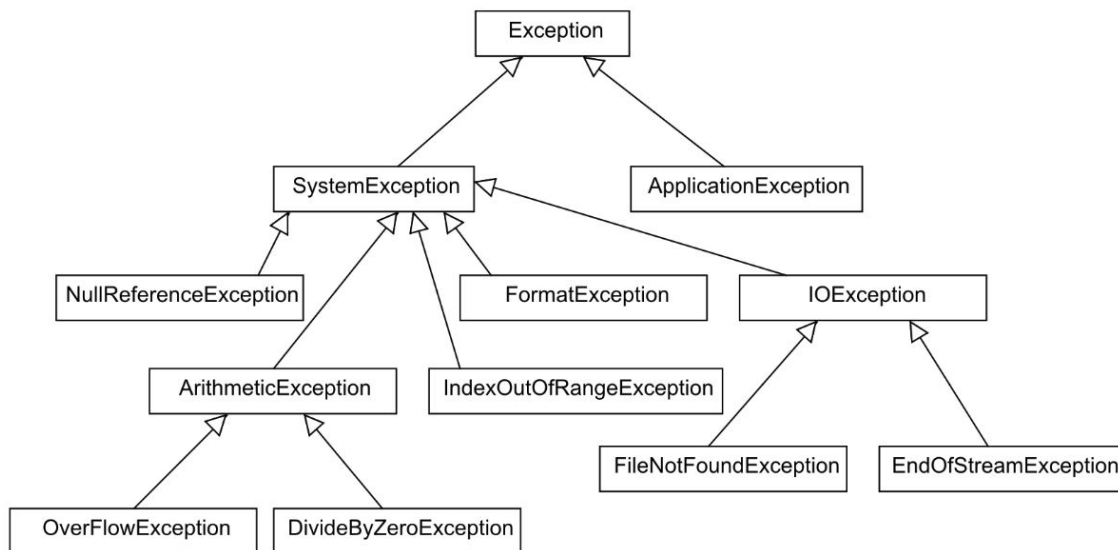
- **Meerdere catch blokken**

- Catch blokken één voor één testen.
- Catch blok met overeenkomstig type exception wordt uitgevoerd!
- Als een exception niet gelijk is aan 1 van deze types, wordt een catch-blok zonder type exception doorlopen.

```
catch  
{  
    // ...  
}
```

Class hierarchie van Exceptions

- Algemene exceptions staan bovenaan hierarchie. Specifiekere beneden.
- Zorg dat je eerst catch-blokken met specifieke fout zet en daarna pas algemenere.
 - Anders wordt de fout als een algemene exception opgevangen!!!



Exceptions: fouten afhandelen

- **Geneste try catch blokken (om nog een fout binnenin op te vangen)**

```
int[] numbers = {8, 17, 24, 5, 25}; // array numbers heeft meer elementen dan array delers heeft
int[] delers = {2, 0, 0, 5}; // array delers heeft minder elementen dan array numbers heeft
try
{
    for (int i = 0; i < numbers.Length; i++)
    {
        try
        {
            Console.WriteLine($"Nummer: {numbers[i]}, Deler: {delers[i]}, Quotient: {numbers[i]/delers[i]}");
        }
        catch (DivideByZeroException) // delen door 0 opvangen en nadien nog verder lopen
        {
            Console.WriteLine("Binnenste Try Catch blok.");
        }
    }
}
catch (IndexOutOfRangeException ex) // index van een array is kleiner dan 0 of te groot (hier te groot)
{
    Console.WriteLine("Buitenste Try Catch blok.");
    Console.WriteLine(ex.Message);
}
```


Exceptions gooien (throw new)

VERSCHILLENDE VORMEN

```
throw new FormatException();  
throw new FormatException(string);
```

- **Zelf exceptions gooien** (Zelf een runtime-fout genereren)

```
private static int WoordNaarNummer(string woord)  
{  
    int resultaat = 0;  
    if (woord.Equals("tien"))  
        resultaat = 10;  
    else if (woord.Equals("honderd"))  
        resultaat = 100;  
    else  
        throw new FormatException("Verkeerde invoer: " + woord); // eigen melding meegeven  
    return resultaat;  
}
```

- **Zelf weer opvangen met try catch:**

```
try  
{  
    Console.WriteLine(Convert.ToString(WoordNaarNummer("hXnderd")));  
}  
catch (FormatException ex)  
{  
    Console.WriteLine(ex.Message);  
}
```

Defensief programmeren: exceptions voorkomen

- **Defensief programmeren:**

- Beter exceptions voorkomen dan dat we ze opvangen.
- Gebruik een if-test om te valideren of er een fout kan komen of niet.
 - Gebruik if-test als er een grote kans is op fout.
- Maar: Gebruik try catch voor zeldzame fouten (< 25% kans).

```
int getal = 10;
int deler = 0;

if (deler == 0)
{
    Console.WriteLine("Delen door 0 mag niet!");
}
else
{
    Console.WriteLine($"Resultaat: {getal/deler}.");
}
```