

C# Essentials Foutopsporing

Koen Bloemen

Wim Bervoets

Kristof Palmaers



**DE HOGESCHOOL
MET HET NETWERK**

Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be





- Foutopsporing
- Foutafhandeling
- Exceptions
- Debugging

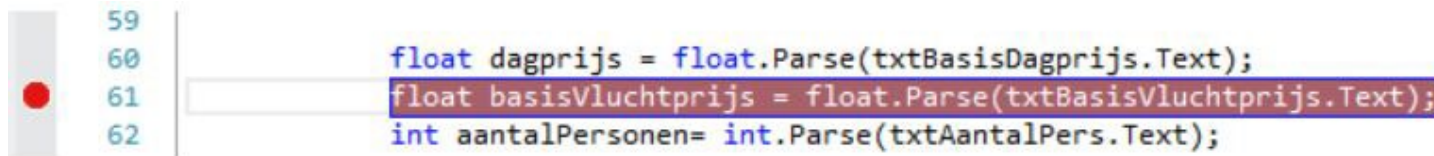
Foutopsporing

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






Debugging: code doorlopen

Programma doorlopen na breakpoint in te stellen

- Bovenaan Visual Studio is een balkje verschenen







- 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

	Step <u>I</u> nto	F11
	Step <u>O</u> ver	F10
	Step <u>O</u> ut	Shift+F11








Debugging: waardes controleren

- **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: waardes controleren

- **Autos** venster (**Debug > Windows > Autos**)
- Toont de waardes 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

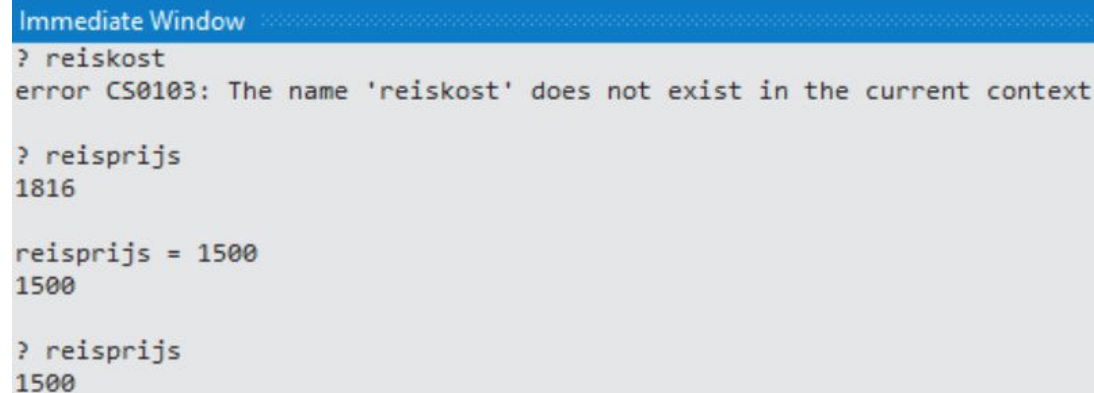
Debugging: waarden controleren

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

Locals		
Search (Ctrl+E)  Search Depth: 3		
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
reispreis	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: waarden controleren

- **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

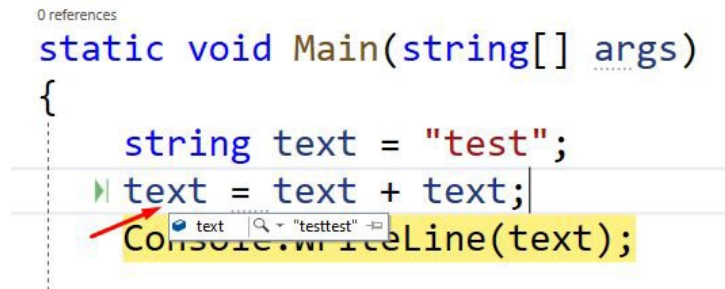
reisprijs = 1500
1500

? reisprijs
1500
```

Debugging: waardes controleren

- Hover effect om waarde te tonen

```
0 references  
static void Main(string[] args)  
{  
    string text = "test";  
    ▶ text = text + text;  
    Console.WriteLine(text);  
}
```

A screenshot of a code editor showing a C# program. The code is: `static void Main(string[] args) { string text = "test"; text = text + text; Console.WriteLine(text); }`. A red arrow points to the variable `text` in the assignment `text = text + text;`. A hover tooltip is displayed, showing the variable name `text` and its current value `"testtest"`. The line `Console.WriteLine(text);` is highlighted in yellow.

Debugging: 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 \Rightarrow beschadigd bestand
- Runtime-fouten afhandelen
 - Fouten opvangen via gestructureerde foutafhandeling (try-catch)
 - Runtime-fouten behandel je als exceptions

Debugging: 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
}
```

Debugging: fouten afhandelen

- Voorbeeld van foutafhandeling

```
try
{
    string nr = "vijf";
    int intrnr = int.Parse(nr); // runtime fout: Input string was not in a correct format.
}
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");
}
```

Debugging: 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)

Debugging: fouten afhandelen

- Voorbeeld meerdere catch blokken

```
int deeltal, deler, quotient;
try
{
    deeltal = Convert.ToInt32(tbGetal1.Text);
    deler = Convert.ToInt32(tbGetal2.Text);
    quotient = deeltal / deler;
    string quotientText = quotient.ToString();
}
catch (DivideByZeroException) // deling door 0
{
    LblInfo.Content = "Delen door 0 mag niet!";
}
catch (FormatException) // een getal is k of 1.6 bijvoorbeeld => een conversiefout
{
    LblInfo.Content = "Je moet 2 gehele getallen ingeven!";
}
catch (Exception ex) // andere fout
{
    LblInfo.Content = ex.Message; // druk foutmelding af gegenereerd door het systeem
}
```

Debugging: fouten afhandelen

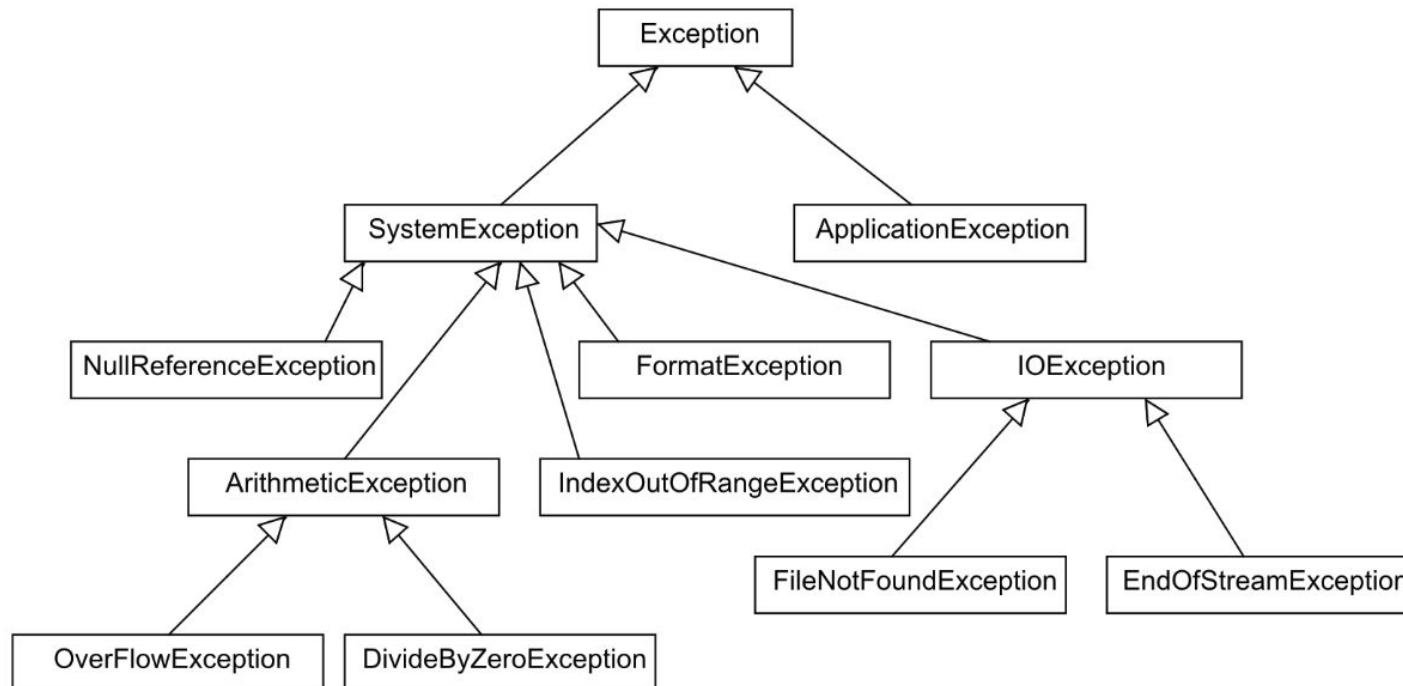
Meerdere catch blokken

- 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 // andere fout
{
    //.....
}
```

Debugging: fouten afhandelen

- Algemeneren exceptions staan bovenaan hiërarchie, specifiekere beneden
- Eerst catch-blokken met specifieke fout en daarna pas algemenere
- Anders wordt de fout als een algemene exception opgevangen!!!



Debugging: fouten afhandelen

- Geneste try catch blokken

```
int deeltal, deler, quotient;
try
{
    deeltal = Convert.ToInt32(tbGetal1.Text);
    deler = Convert.ToInt32(tbGetal2.Text);
    for (int i = 3; i >= 0; i--)
    {
        try
        {
            deler /= i;
        }
        catch (DivideByZeroException)
        {
            LblInfo.Content = "Delen door 0 mag niet! Binnenste afhandeling.";
            deler = Convert.ToInt32(tbGetal2.Text);
        }
    }
    quotient = deeltal / deler;
    string quotientText = quotient.ToString();
}
catch (DivideByZeroException) // deling door 0
{
    LblInfo.Content = "Delen door 0 mag niet!";
}
catch (FormatException) // een getal is k of 1.6 bijvoorbeeld => een conversiefout
{
    LblInfo.Content = "Je moet 2 gehele getallen ingeven!";
}
catch (Exception ex) // andere fout
{
    LblInfo.Content = ex.Message; // druk foutmelding af gegenereerd door het systeem
}
```

Debugging: exceptions thrown

- 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);
}
```

Debugging: exceptions throwen

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}.");
}
```