



C# Essentials

Foutafhandeling

Sander De Puydt

**DE HOGESCHOOL
MET HET NETWERK**

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Inhoudstafel

- Debugging
- Exceptions
- throw
- Defensief programmeren



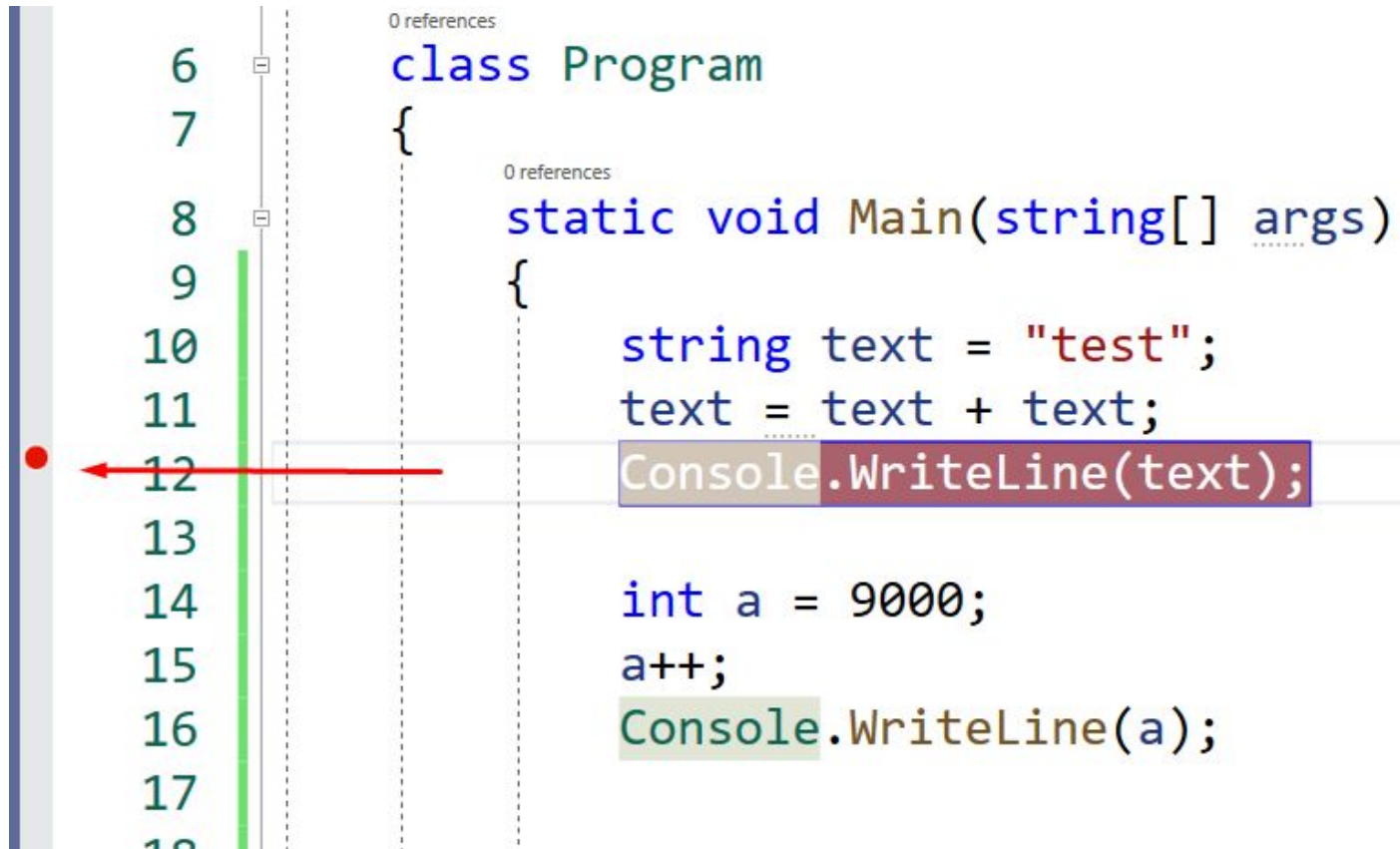
Debugging

- Bij het uitvoeren van een programma kunnen er verschillende fouten optreden.
- Wanneer deze niet behandeld worden, dan crasht het programma.
- Met de onderbrekingsmodus kan je inspecteren waar het fout loopt.



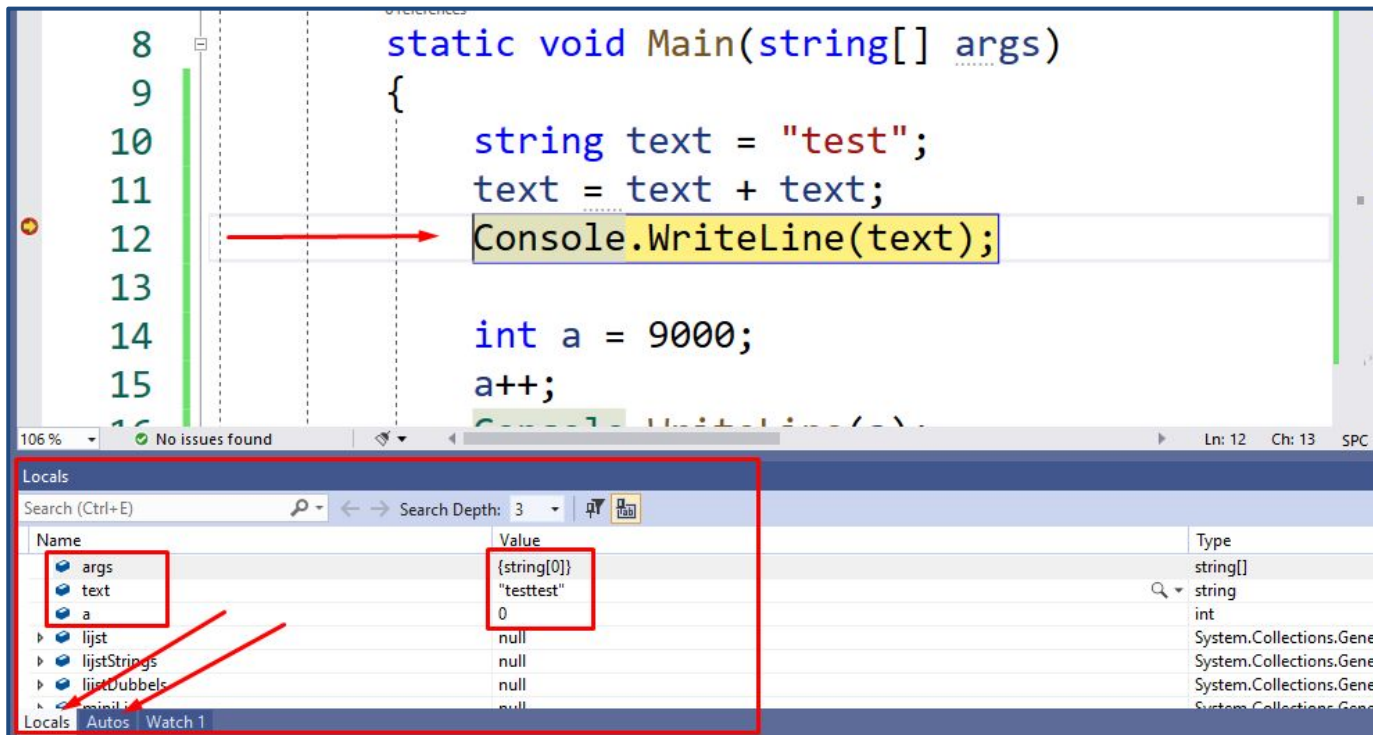
Debugging

- Debug punt ontbreekt de uitvoering.



Debugging

- Je kan de waarden evalueren in het programma op het moment van de onderbreking.



Debugging

Locals: weergeeft alle lokale variabelen.

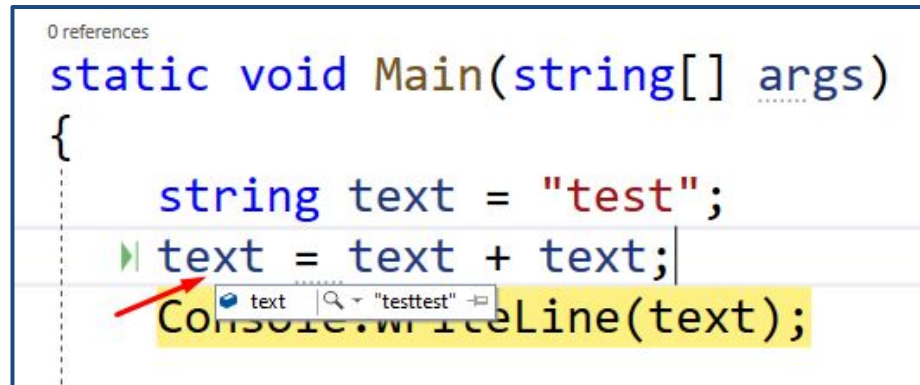
Autos: weergeeft de variabelen van het huidige en vorige statement.

Watch: weergeeft een lijst van alle opgegeven variabelen.



Debugging

- Met een hover effect kan je de waarde inspecteren.



A screenshot of a code editor window. The code is in C# and shows a `Main` method. The variable `text` is assigned the value `"test"`. The next line is `text = text + text;`, and the following line is `Console.WriteLine(text);`. A red arrow points to the variable `text` in the assignment statement. A tooltip is displayed over the `text` variable, showing the value `"testtest"`. The tooltip also includes a search icon and the text `0 references`.

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

- Met de Debug toolbar kan je stap voor stap de code doorlopen.



Step Into	F11
Step Over	F10
Step Out	Shift+F11

Exceptions

- Fouten die tijdens het uitvoeren van het programma optreden kan je opvangen in een try catch structuur.

```
string text = "test";  
text = text + text;  
int a;  
try  
{  
    a = Convert.ToInt32(text);  
}  
catch (Exception e)  
{  
    a = 10;  
}
```



Exceptions

- Beschrijf de exception zo specifiek mogelijk om de fout beter af te handelen.

```
string text = "test";
text = text + text;
int a;
try {
    a = Convert.ToInt32(text);
}
catch (FormatException e) {
    Console.WriteLine("Een format probleem is opgetreden");
    a = 20;
}
catch (Exception e) {
    Console.WriteLine("Er is iets fout gelopen");
    a = 10;
}
```



Exceptions

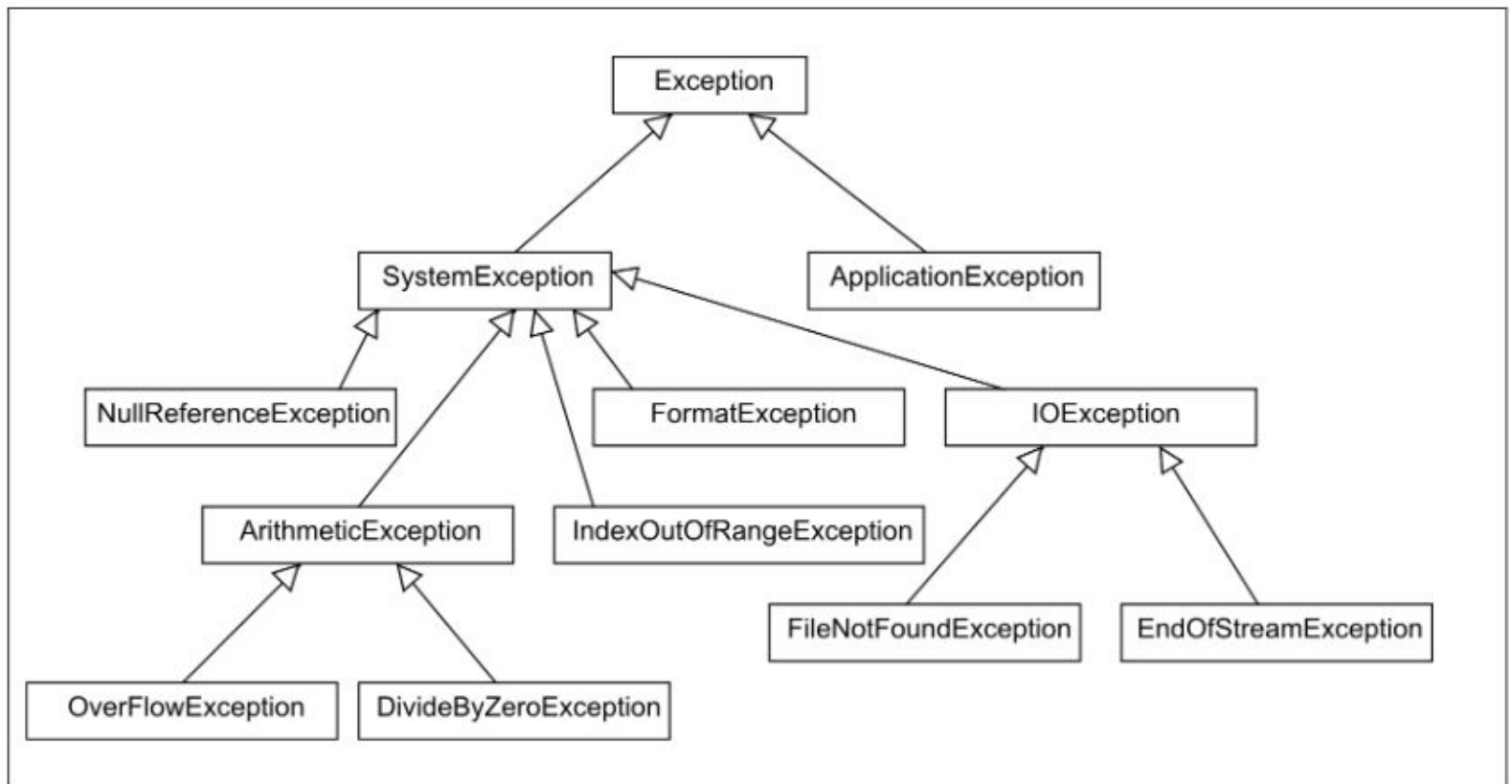
- Na een try catch kan **finally** volgen. De code in finally wordt altijd uitgevoerd na een try catch ongeacht de try block succesvol was.

```
List<string> regels = new List<string>() { "hallo", "ok" };
try {
    regels.Add("Ja");
    Console.WriteLine(regels[4]);
}
catch (ArgumentOutOfRangeException e) {
    Console.WriteLine("Het vijfde element is niet gevonden");
}
finally {
    // Wordt altijd uitgevoerd na try catch
    regels.Remove("Ja");
}
```



Exceptions

- Behandel specifieke fouten voor algemene
- Classificatie klassenhiërarchie:



throw

- Soms wil je dat het programma een fout genereert:
 - Wanneer de gebruiker de applicatie incorrect gebruikt.
 - Wanneer het gedrag binnen de applicatie afwijkt van wat er verwacht is.

```
Console.WriteLine("Geef een getal dat kleiner is dan 10");  
if (int.TryParse(Console.ReadLine(), out int inputVanGebruiker)) {  
    if (inputVanGebruiker >= 10) {  
        throw new FormatException(  
            "Het ingegeven getal is niet kleiner dan 10");  
    }  
}
```



Defensief programmeren

- Het is beter om defensief te programmeren.
 - Probeer te vermijden dat je applicatie fouten genereert.
 - Maak gebruik van validatietechnieken
 - Gebruik exceptionhandling wanneer je verwacht dat een runtime fout uitzonderlijk voorkomt.

```
if (File.Exists("C:\\Users\\CSharp\\CSharp.png"))
{
    picCSharp.Image = Image.FromFile(@"C:\Users\CSharp\CSharp.png"); // verbatim tekenreeks:
                                                                    geen \\ nodig in pad
}
else
{
    MessageBox.Show("Kan het bestand niet openen van drive D.");
}
```

