



GRADUAAT IN HET
PROGRAMMEREN

Visual C#.NET met WPF Syllabus

Cursus

C# Advanced

Opleidingsonderdeel

Informatica | Programmeren

Afdeling

Patricia Briers

Auteur



**DE HOGESCHOOL
MET HET NETWERK**

Overzicht

HOOFDSTUK 1	WERKEN MET TEKSTBESTANDEN.....	4
1.2	TEKSTBESTANDEN	4
1.3	STREAMWRITER EN STREAMREADER	4
1.4	FILESTREAM.....	5
1.5	STATISCHE KLASSE FILE.....	5
1.6	KLASSE FILEINFO.....	6
1.7	KOMMAGESCHEIDEN BESTANDEN (CSV FILES / COMMA-SEPARATED VALUES)	8
1.7.1	<i>Tekst wegschrijven</i>	8
1.7.2	<i>Tekst lezen met String.Split()</i>	8
1.8	BESTANDEN MET EEN VASTE LENGTE (FIXED-WIDTH TEXT FILES).....	8
1.8.1	<i>Tekst wegschrijven</i>	8
1.8.2	<i>Tekst lezen met Substring()</i>	8
1.9	BESTANDEN EN EXCEPTIONS	9
1.10	DIRECTORY	9
1.11	STANDAARDDIALOGVENSTERS	10
HOOFDSTUK 2	WERKEN MET MEERDERE VENSTERS	14
2.1	SOORTEN WINDOWS.....	14
2.1.1	<i>Werken met niet-modale vensters (modeless) / modale vensters</i>	14
2.2	GEGEVENS TUSSEN VENSTERS UITWISSELEN.	15
2.3	GEBEURTENISSEN BIJ VENSTERS.....	19
2.4	NIEUW VENSTER TOEVOEGEN	20
2.5	OPSTARTVOLGORDE BIJ MEERDERE VENSTERS.	20
HOOFDSTUK 3	EIGEN KLASSE MAKEN.....	22
3.1	.NET FRAMEWORK	22
3.1.1	<i>CLR (Common Language Runtime)</i>	22
3.1.2	<i>Class library</i>	22
3.2	ASSEMBLY	23
3.3	OBJECT GEORIENTEERD MODELLEREN.....	23
3.4	WERKEN MET KLASSE(N)	24
3.5	AUTO-IMPLEMENTED PROPERTIES EN EXPRESSION-BODIED PROPERTIES.	26
3.6	STATIC METHODEN EN PROPERTIES	30
3.7	PARTIAL CLASS	31
3.8	DECLARATIE VAN CLASS, EIGENSCHAPPEN OF PROCEDURES	31
3.9	OBJECT VERNIETIGEN	31
3.10	INSTALLEER HET ONDERDEEL CLASS DESIGNER	31
3.10.1	<i>Voeg een leeg klassendiagram toe aan een project</i>	32
3.11	VOEG EEN KLASSENDIAGRAM TOE OP BASIS VAN BESTAANDE TYPEN.....	32
3.12	DE INHOUD VAN EEN COMPLEET PROJECT WEERGEVEN IN EEN KLASSENDIAGRAM	33
HOOFDSTUK 4	OVERERVING EN POLYMORFISME VAN KLASSEN	34
4.1	HET GEBRUIK VAN CLASS DIAGRAM.....	34
4.2	OVERERVING.....	34
4.3	POLYMORFISME	36
4.3.1	<i>Sleutelwoord Base</i>	38
4.3.2	<i>Converteren van eigen objecten</i>	38
4.3.3	<i>Klasse en generieke collectie</i>	39

4.4	ABSTRACTE KLASSE	41
4.5	INTERFACE	44
4.6	SEALED KLASSE	46
4.6.1	Type Klasse	48
4.6.2	ClassLibrary	48
HOOFDSTUK 5	LINQ.....	51
5.1	DE QUERY.....	51
5.2	LINQ QUERY-BEWERKINGEN	53
5.2.1	WHERE.....	53
5.2.2	OrderBy	53
5.2.3	Distinct	54
5.2.4	GROUPBY.....	54
5.2.5	JOIN	55
HOOFDSTUK 6	WPF BESTURINGSELEMENTEN (VERVOLG).....	59
6.1	TABBLADEN (TABCONTROL)	59
6.2	STATUSBALK (STATUSBAR)	60
6.3	PASSWORDBOX	61
HOOFDSTUK 7	ADO.NET	62
	BELANGRIJKE KLASSEN IN ADO.NET ZIJN:	64
7.2	CONNECTION	64
7.3	SQLCOMMANDO	67
7.4	DATAREADER.....	69
7.4.1	ExecuteReader.....	70
7.4.2	ExecuteScalar	71
7.4.3	ExecuteNonQuery.....	71
7.4.4	SqlParameter.....	72
7.5	DATASET	73
7.5.1	DataTable.....	75
	• DataColumn	75
	• DataRow	75
7.6	SQLDATA ADAPTER	76
7.6.1	DataView.....	77
HOOFDSTUK 8	BIJLAGEN.....	80
B1	LIJST VAN CODE SNIPPETS IN C#	80
B2	LIJST VAN CLASS LIBRARY IN C#.....	81

Hoofdstuk 1 Werken met tekstbestanden

1.2 Tekstbestanden

Wanneer je gegevens naar een bestand *doorlopend* wilt schrijven of uit een bestand wilt lezen, maken de gegevens onderdeel uit van een *stream* (filestream). Een *stream* kan je voorstellen als een stroom of reeks bytes. We bekijken de tekstbestanden (*.txt) omdat vele toepassingen (waaronder tekstverwerkers, editors, databases) met deze bestanden werken. We gebruiken hiervoor de namespace **System.IO** die een aantal klassen omvat waar we mee kunnen werken.

Een aantal methoden van de **klasse File** om naar bestanden te schrijven of uit te lezen, gebruikt zogenaamde *Stream*-klasse. Voorbeelden zijn **FileStream** of **StreamReader/StreamWriter**. De **klasse FileInfo** maakt intensief gebruik van dit type klasse.

1.3 StreamWriter en StreamReader

```
//Namespace toevoegen.  
using System.IO;
```

```
// SCHRIJVEN  
// Tekstbestand creëren en openen (automatisch in standaarddirectory van je project \bin\Debug).  
StreamWriter sw = new StreamWriter("TestBestand.Txt");  
  
// Naar tekstbestand schrijven.  
sw.WriteLine("Volgorde van getallen :");  
for (int i = 0; i < 10; i++)  
    sw.Write(i + " ");  
sw.WriteLine(); // lege regel wegschrijven  
  
// Sluiten van bestand.  
sw.Close();
```

```
Volgorde van getallen :  
0 1 2 3 4 5 6 7 8 9
```

```
// LEZEN  
StreamReader sr = new StreamReader("TestBestand.Txt");  
  
// Inlezen van bestand per record.  
while (!sr.EndOfStream)  
{  
    Console.WriteLine(sr.ReadLine());  
}  
  
// Volledig bestand inlezen.  
Console.WriteLine(sr.ReadToEnd());  
  
// Inlezen van bestand per record in List  
List<string> namenLijst = new List<string>();  
while (!sr.EndOfStream)  
{  
    namenLijst.Add(sr.ReadLine());  
    Console.WriteLine(namenLijst[j]);  
    j++;  
}  
// Sluiten van bestand.  
sr.Close();
```

1.4 FileStream

Met de bijkomende FileStream kan je flexibeler werken dan als je enkel met StreamWriter/Reader werkt.

```
// SCHRIJVEN
FileStream fs = new FileStream("MyTest.Txt", FileMode.Append, FileAccess.Write);
StreamWriter sw = new StreamWriter(fs);

sw.WriteLine(" Volgorde van getallen:");
for (int i = 0; i < 10; i++)
    sw.Write(i + " ");
sw.WriteLine(); // Lege regel toevoegen.

// Sluiten van bestanden.
sw.Close();
fs.Close();
```

```
// LEZEN
FileStream fsRead = new FileStream("MyTest.Txt", FileMode.Open, FileAccess.Read);
StreamReader sr = new StreamReader(fsRead);

// Bestand tot einde lezen.
Console.WriteLine(sr.ReadToEnd());

// Inlezen van bestand per regel.
while (!sr.EndOfStream)
{
    TxtResultaat.AppendText(sr.ReadLine());
}
//Objecten sluiten
sr.Close();
fsRead.Close();
```

FileMode:	Open (bestaand bestand lezen/schrijven) Append (toevoegen aan bestand en eventueel Creatie) Create (bestand lezen/schrijven, bestaand bestand wordt overschreven) OpenNew ((bestand lezen/schrijven, bij bestaand bestand geeft een exception)
FileAccess:	Read (enkel lezen) ReadWrite (lezen en schrijven) Write (enkel schrijven)

1.5 Statische klasse File

File.Append(pad)	opent en voegt toe aan bestaand bestand
File.Create(pad)	creëert en overschrijft het bestand in opgegeven pad
File.CreateText(pad)	creëert en overschrijft een tekstbestand (UTF-8 encode)
File.Delete(pad)	verwijdert bestand
File.Copy(pad)	kopieert een bestand
File.Exists(pad)	test of bestand bestaat
File.OpenText(pad)	opent een bestaand tekstbestand (UTF-8 encoded)
File.OpenWrite(pad)	opent een bestaand tekstbestand om weg te schrijven

⁽¹⁾ UTF-8 encoded of Unicode/ISO 10646-teken: UTF-8 is een tekencodering met variabele lengte: niet elk teken gebruikt evenveel bytes. Afhankelijk van het teken worden 1 tot 4 bytes gebruikt, de eerste byte geeft aan hoeveel bytes er gebruikt worden. Voor het vastleggen van elk van de 128 ASCII-teken is slechts één byte nodig.

File.ReadAllLines(pad)	opent, lees in een array en sluit automatisch
File.ReadAllLines(bestand).Length;	geeft aantal regels in bestand
File.ReadAllText(pad)	leest het volledig bestand
File.WriteAllText(pad)	schrijft het volledig bestand weg
File.AppendAllText(pad,string)	voegt tekst aan bestand toe

```

string pad= "TestKlasseFile.Txt";
if (!File.Exists(pad))
{
    // Creëert een bestand om naar te schrijven (CreateText kan samenwerken met StreamWriter, niet Create)
    {
        sw.WriteLine("Programmeren - Lector: Patricia Briers");
        sw.WriteLine("Cisco - Lector: Toni Mini");
        sw.WriteLine("OS - Lector: Stijn Jacobs");
        sw.WriteLine("Web - Lector: Bram en Pieter");
    }
}

// Opent het bestand om te lezen.
using (StreamReader sr = File.OpenText(pad))
{
    while (!sr.EndOfStream)
    {
        Console.WriteLine(sr.ReadLine());
    }
}

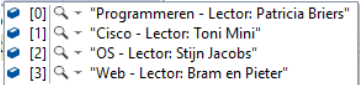
```

// OF dadelijk in array inlezen.

```

string[] gegevens = File.ReadAllLines(pad);

```



```

string bestand = @"C:..\Test.Txt";

if (!File.Exists(bestand))
{
    string boodschap = "Module A5: C# Programmatie" + Environment.NewLine;
    File.WriteAllText(bestand, boodschap);
}

// Tekst toevoegen
string toevoegen = "Klas: HI1-Dagopleiding" + Environment.NewLine;
File.AppendAllText(bestand, toevoegen);

// bestand openen
string leesTekst = File.ReadAllText(bestand);
Console.WriteLine(leesTekst);

```



Door gebruik te maken van de *using* wordt het bestand automatisch gesloten, zodat geen *Close()* nodig is. Bovendien wordt het bestand gesloten zelfs als het programma onverwacht sluit.

1.6 Klasse FileInfo

De klasse *FileInfo* werkt op dezelfde manier als de klasse *File*. Als je slechts enkele handelingen op het bestand doet, gebruik je best de klasse *File*. Als je **meerdere bewerkingen** op hetzelfde bestand uitvoert, gebruik je best *FileInfo*.

Wanneer je een bestand met *FileInfo* creëert, wordt de **beveiligingscontrole** slechts één keer uitgevoerd terwijl bij de klasse *File* wordt de controle bij elke bewerking uitgevoerd.

Toepassing: Huidige systeemdatum opvragen en wegschrijven

```
private void SchrijfData(FileInfo fi)
{
    //Gebruik Writer om huidige tijd weg te schrijven
    StreamWriter sw = fi.AppendText();
    sw.Write(DateTime.Now.ToString());
    sw.Close();
}

private void LeesData(FileInfo fi)
{
    //Gebruik Reader om data weer in te lezen
    StreamReader sr = fi.OpenText();
    TxtResultaat.Text = sr.ReadToEnd();
    sr.Close();
}

public void BtnDatumWegschrijven_Click(System.Object sender, System.EventArgs e)
{
    string pad = @"..\DatumBestand.Txt"; // relatief pad - onder \debug

    // In tegenstelling tot de statische klasse File moet je een instantie maken van FileInfo
    FileInfo fi = new FileInfo(pad);

    //Controle of bestand bestaat.
    if (fi.Exists)
    {
        string boodschap = string.Empty;;
        boodschap = " Het bestand" + pad + " bestaat al." + Environment.NewLine
            + "wilt u het bestand overschrijven?";

        //Vraag of bestand overschreven mag worden
        if (MessageBox.Show(boodschap, "Overschrijven", MessageBoxButtons.YesNo) ==
            DialogResult.Yes)
        {
            fi.Delete();
            SchrijfData(fi);
        }
    }
    else
    {
        SchrijfData(fi);
    }
    LeesData(fi);
}
```

`FileInfo fi = new FileInfo(pad);`

<code>fi.Exists</code>	testen of bestand bestaat
<code>fi.AppendText()</code>	maakt streamwriter en voegt tekst toe
<code>fi.CreateText()</code>	maakt streamwriter en schrijft tekst weg
<code>fi.OpenText</code>	maakt streamreader en leest tekst
<code>fi.OpenRead</code>	maakt een read-only bestand
<code>fi.OpenWrite</code>	maakt een write-only bestand
<code>fi.Delete()</code>	verwijdert bestand

1.7 Kommagescheiden bestanden (csv files / Comma-Separated Values)

Er zijn veel toepassingen of applicaties die gegevens opslaan in een kommagescheiden bestand (velden gescheiden door komma of ander scheidingsteken) of een bestand met velden van een vaste lengte.

1.7.1 Tekst wegschrijven

```
using (StreamWriter sw = new StreamWriter("KommaBestand.Txt"))
{
    // Gegevens wegschrijven
    sw.WriteLine("Patricia Briers; PCVO Limburg");
    sw.WriteLine("Anne Koninx; VJC Hasselt");
    sw.WriteLine("Jonas Vandeput; UHasselt Diepenbeek");
    sw.WriteLine("Paul Dox; PXL Hasselt");
}
```

1.7.2 Tekst lezen met String.Split()

```
string boodschap = string.Empty;
string[] velden;

using (StreamReader sr = new StreamReader("KommaBestand.Txt"))
{
    while (!sr.EndOfStream)
    {
        //Scheidingsteken opgeven, geeft array terug gescheiden door opgegeven karakter.
        velden = sr.ReadLine().Split(';');
        boodschap += $"{velden[0]} werkt in : {velden[1]}" + Environment.NewLine;
    }
}
MessageBox.Show(boodschap);
```

1.8 Bestanden met een vaste lengte (Fixed-width Text Files)

In bestanden met vaste lengte worden de velden tot een bepaalde lengte opgevuld met spaties. Om te werken met deze bestanden moet de eigenschap van *FieldType.FixedWidth* worden ingesteld. *FieldWidths* is een array van gehele getallen (integers). Voor elk veld is er een element in de array.

1.8.1 Tekst wegschrijven

```
using (StreamWriter sw = new StreamWriter("VastBestand.Txt"))
{
    // Gegevens wegschrijven
    sw.WriteLine($"{ "Briers", -20 } { "Patricia", -10 } { "Gent", -15 }");
    sw.WriteLine($"{ "Vandeput", -20 } { "Jonas", -10 } { "Diepenbeek", -15 }");
    sw.WriteLine($"{ "Nys", -20 } { "Karolien", -10 } { "Tongeren", -15 }");
    sw.WriteLine($"{ "Dox", -20 } { "Paul", -10 } { "Hasselt", -15 }");
}
```

1.8.2 Tekst lezen met Substring()

```
using (StreamReader sr = new StreamReader("VastBestand.Txt"))
{
    while (!sr.EndOfStream)
```



```

{
    string lijn = sr.ReadLine();
    string veld1 = lijn.Substring(0, 19).Trim();
    string veld2 = lijn.Substring(20, 9).Trim();
    string veld3 = lijn.Substring(30, 14).Trim();
    TxtResultaat.Text += $"{veld2} {veld1} werkt in {veld3}" + Environment.NewLine;
}

OF string lijn
while((lijn=sr.ReadLine())!=null) //geeft geen fout bij extra lege regel op einde van bestand.
{
    string veld1 = lijn.Substring(0, 19).Trim();
    string veld2 = lijn.Substring(20, 9).Trim();
    string veld3 = lijn.Substring(30, 14).Trim();
    TxtResultaat.Text += $"{veld2} {veld1} werkt in {veld3}" + Environment.NewLine;
}
}

```

1.9 Bestanden en exceptions

Bestanden zullen vaak een exception genereren omdat het bestand niet toegankelijk is. De *System.IO.FileNotFoundException* wanneer het bestand niet gevonden wordt of *IOException* wanneer er niet naar een bestand kan gelezen/geschreven worden of *OutOfMemoryException* wanneer er onvoldoende geheugen beschikbaar is,.....

Hier kan je best gebruik maken van een gestructureerde foutafhandeling.

```

string pad = "TryFile.Txt";
try
{
    // Opent het bestand om te lezen.
    using (StreamReader sr = File.OpenText(pad))
    {
        while (!sr.EndOfStream)
        {
            Console.WriteLine(s);
        }
    }
}

catch (FileNotFoundException ex)
{
    MessageBox.Show("Geef nieuwe naam!\r\n\r\n" + ex.Message, "Foutmelding",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
catch (Exception)
{
    MessageBox.Show("Kan bestand niet vinden.", "Foutmelding", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
}

```

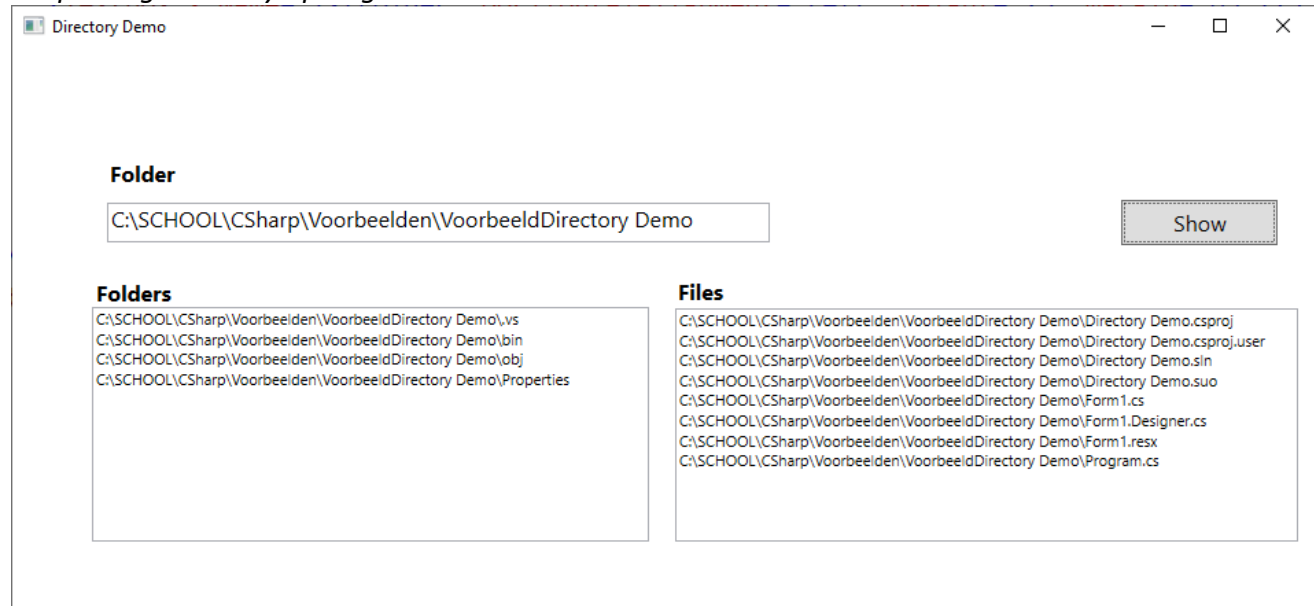
1.10 Directory

Voor het openen of bewaren van bestanden wordt naast de dialoogvensters *SaveFileDialog* en *OpenFileDialog* rechtstreeks met het pad en/of de bestandsnaam gewerkt. Als je geen bestand wilt

bewerken, maar enkel bestanden wilt opzoeken of de naam ervan wilt wijzigen, kan je met Directory werken.

```
// Bestand opvragen
string pad = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
string bestand = Path.Combine(pad, "MijnTekstbestand.Txt");
```

Toepassing directory opvragen



```
// Toont alle bestandsnamen
string[] bestanden = Directory.GetFiles(TxtMap.Text);

foreach (string file in bestanden)
{
    TxtBestanden.AppendText(file);
    TxtBestanden.AppendText(Environment.NewLine);
}

// Toont alle mappen
string[] mappen = Directory.GetDirectories(TxtMap.Text);

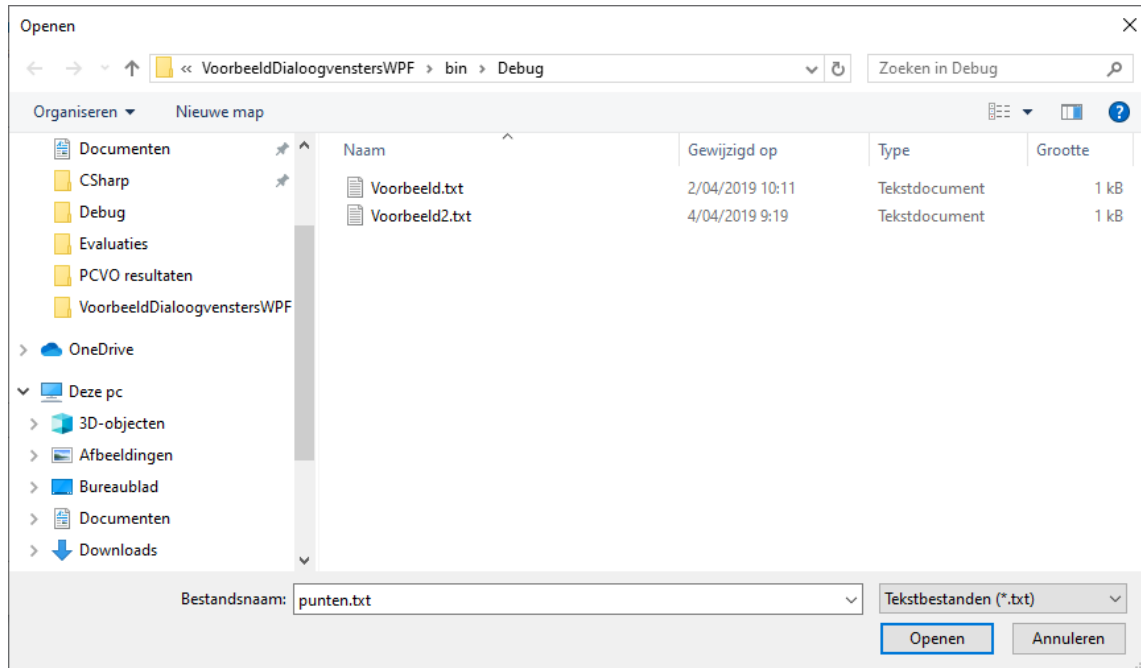
foreach (string dir in mappen)
{
    TxtMappen.AppendText(dir);
    TxtMappen.AppendText(Environment.NewLine);
}
```

1.11 Standaarddialogvensters

Indien we werken met bestanden is het prettig om je bestanden te kunnen openen/bewaren dmv de voorkomende dialoogvenster zoals vensters voor het selecteren van een bestand. De dialoogvenster zitten in de library Microsoft.Win32.

```
using Microsoft.Win32;
```

Open File Dialog



```
private void BtnOpenFile_Click(object sender, RoutedEventArgs e)
{
    OpenFileDialog ofd = new OpenFileDialog
    {
        Filter = "Alle bestanden (*.*)|*.*|Tekstbestanden (*.txt) |*.txt",
        FilterIndex = 2,
        FileName = "punten.txt",
        Multiselect = true,

        InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        // == OF ==
        InitialDirectory = System.IO.Path.GetFullPath(@"..\..\Bestanden"), // volledig pad
        // == OF ==
        InitialDirectory = Environment.CurrentDirectory // onder ..\Debug
    };

    if (ofd.ShowDialog() == true)
    {
        TxtFile.Text = $"PAD & FILE: {ofd.FileName}\r\n"; //volledig pad en bestandsnaam
        TxtFile.Text += $"PAD: {System.IO.Path.GetDirectoryName(ofd.FileName)}\r\n"; // enkel pad
        TxtFile.Text += $"FILE: {System.IO.Path.GetFileName(ofd.FileName)}\r\n\r\n"; // enkel
        bestandsnaam

        string pad = System.IO.Path.GetDirectoryName(ofd.FileName);
        string[] bestanden = System.IO.Directory.GetFiles(pad); //Lijst van bestanden.
        int i = 0;

        for (; i < bestanden.Length; i++)
        {
            TxtFile.Text += $"{bestanden[i]}\r\n"; // pad + bestandsnaam
        }
    }
}
```

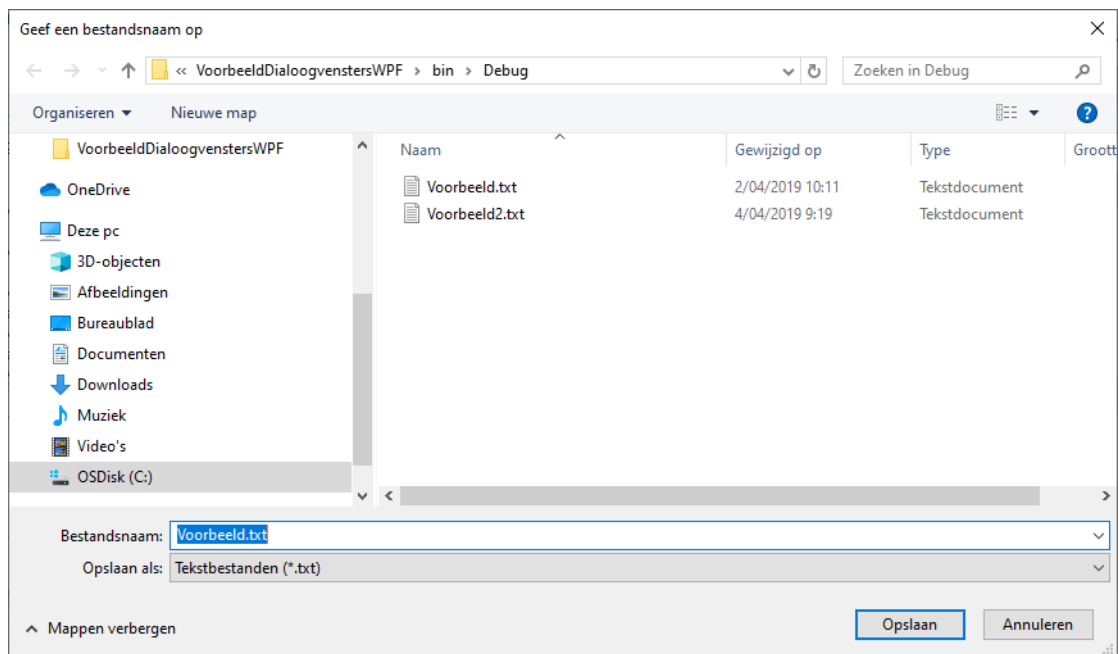
```

        TxtFile.Text += $"{System.IO.Path.GetFileName(bestanden[i])}\r\n"; //bestandnaam
    }
    TxtFile.Text += $"Aantal bestanden: {i}";

    // Selectie van meerdere bestanden afdrukken in listbox.
    foreach (string filename in ofd.FileNames)
    {
        LbxFiles.Items.Add(System.IO.Path.GetFileName(filename));
    }
}
}

```

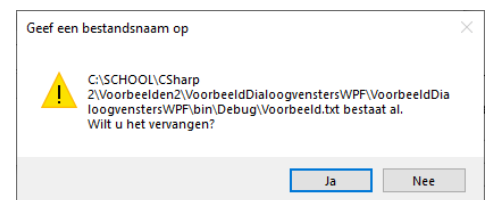
Save File Dialog



```

private void BtnSaveFile_Click(object sender, RoutedEventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog()
    {
        Filter = "Alle bestanden (*.*)|*.*|Tekstbestanden (*.txt)|*.txt",
        FilterIndex = 2,
        Title = "Geef een bestandsnaam op",
        OverwritePrompt = true, //Bevestiging wordt gevraagd bij overschrijven van een bestand.
        AddExtension = true, // Extensie wordt toegevoegd.
        DefaultExt = "txt",
        FileName = "Voorbeeld.txt",
        //InitialDirectory = @"C:\SCHOOOL\CSharp 2\Voorbeelden2", // Fysieke map
        InitialDirectory = Environment.CurrentDirectory // onder ..\Debug
    };
    sfd.ShowDialog();
}

```



```

    TxtFile.Text = $"PAD & FILE: {sfd.FileName}\r\n"; //volledig pad en bestandsnaam
    TxtFile.Text += $"PAD: {System.IO.Path.GetDirectoryName(sfd.FileName)}\r\n"; // enkel pad
    TxtFile.Text += $"FILE: {System.IO.Path.GetFileName(sfd.FileName)}\r\n"; // enkel
bestandsnaam
    TxtFile.Text += $"CURRENT DIRECTORY: {System.IO.Directory.GetCurrentDirectory()}"; //
C:..\Deb
    ug
}

```

Tip

Gebruik relatief pad

```

.\      pad waar .exe staat
..\     één map boven de map met .exe
..\..\  twee mappen boven de map met .exe

```

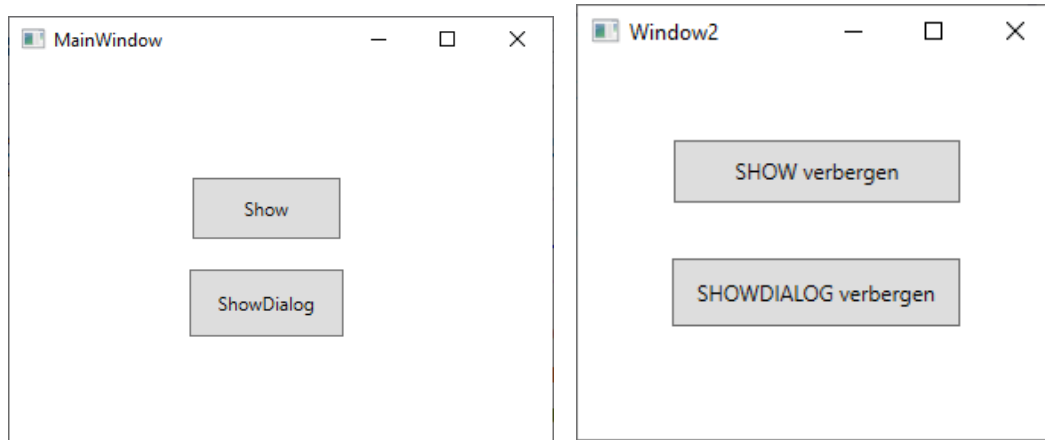
```
string bestand = System.IO.Path.Combine(@"..\..\Bestanden", LogoC.png);
```

Hoofdstuk 2 Werken met meerdere vensters

2.1 Soorten windows

Je kan op 2 manieren werken met vensters. We onderscheiden de niet-modale of **modeless** vensters of de **modale** vensters die fungeren als dialoogvenster. Een dialoogvenster moet je eerst sluiten vooraleer je terug naar het vorig venster kan.

2.1.1 Werken met niet-modale vensters (modeless) / modale vensters



MainWindow

```
private void BtnShow_Click(object sender, RoutedEventArgs e)
{
    // Declaratie van het tweede venster.
    Window2 w2 = new Window2();

    // Oproepen als niet-modale venster.
    w2.Show();
}

private void BtnShowDialog_Click(object sender, RoutedEventArgs e)
{
    // Declaratie van het tweede venster.
    Window2 w2 = new Window2();

    // Oproepen als dialoogvenster.
    w2.ShowDialog();
}
```

Window2

```
private void BtnShowHide_Click(object sender, RoutedEventArgs e)
{
    // Verbergen van huidig venster.
    Hide();

    // Sluiten van huidig venster.
    Close();
}
```

```

}

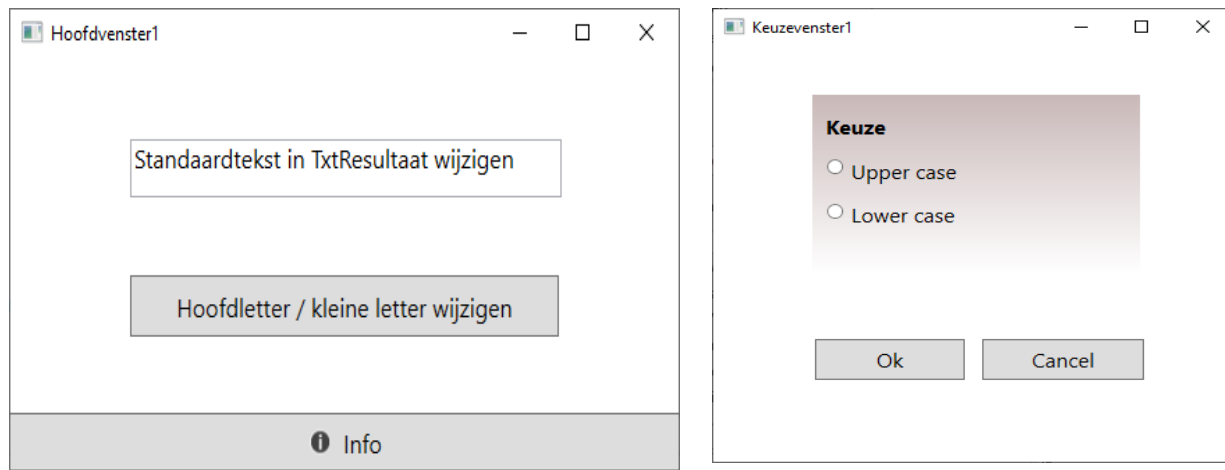
private void BtnShowDialogHide_Click(object sender, RoutedEventArgs e)
{
    // Verbergen van huidig venster.
    Hide();

    // Sluiten van huidig venster.
    Close();
}

```

2.2 Gegevens tussen vensters uitwisselen.

Vanuit het tweede formulier (keuzerondjes) wordt de tekst op het eerste formulier in hoofdletters of kleine letters getoond.



Er zijn verschillende manieren om gegevens tussen vensters uit te wisselen.

- **Uitwisseling via public TextBox** (deze oplossing verdient alvast geen schoonheidswedstrijd)

Hoofdvenster1

```
// Klasseniveau - public !!!!
public static TextBox txtWijzig = new TextBox();

private void BtnWijzigen_Click(object sender, RoutedEventArgs e)
{
    // Objecten worden aan elkaar doorgegeven.
    txtWijzig = TxtResultaat;

    // Declaratie van venster.
    Keuzevenster1 windowKeuze = new Keuzevenster1();

    // Modal form als dialoogvenster oproepen.
    windowKeuze.ShowDialog();

    //ShowDialog == True wanneer DialogResult.Ok en False wanneer DialogResult.Cancel.
    if (windowKeuze.DialogResult.HasValue && windowKeuze.DialogResult.Value)
    {
        MessageBox.Show("Knop Ok gedrukt.");
    }
    else
    {
        MessageBox.Show("Knop Cancel/Sluiten gedrukt.");
    }
}

private void BtnInfo_Click(object sender, RoutedEventArgs e)
{
    Info infovenster = new Info();
    infovenster.ShowDialog();
}
```

Keuzevenster1

```
private void BtnOk_Click(object sender, RoutedEventArgs e)
{
    string wijzigLetter = Hoofdvenster1.txtWijzig.Text;

    if (radUpper.IsChecked==true)
    {
        wijzigLetter = wijzigLetter.ToUpper();
    }
    else
    {
        wijzigLetter = wijzigLetter.ToLower();
    }

    Hoofdvenster1.txtWijzig.Text = wijzigLetter;

    // Doorgeven op OK geklikt.
    DialogResult = true;
}
```



```
private void BtnCancel_Click(object sender, RoutedEventArgs e)
{
    DialogResult = false;
}
```

- Uitwisseling via constructor van windows(tekstvak)

HoofdvensterTxt

```
private void BtnWijzigen_Click(object sender, RoutedEventArgs e)
{
    KeuzevensterTxt keuzevenster = new KeuzevensterTxt(TxtResultaat);

    //Modal venster oproepen
    keuzevenster.ShowDialog();

    //Opvangen op welke knop bij het keuzevenster geklikt is.
    if (keuzevenster.DialogResult.HasValue && keuzevenster.DialogResult.Value)
    {
        //Er is True geretourneerd.
        MessageBox.Show("Knop Ok gedrukt.");
    }
    else
    {
        //Er is False geretourneerd.
        MessageBox.Show("Knop Cancel/Sluiten gedrukt.");
    }
}

private void BtnInfo_Click(object sender, RoutedEventArgs e)
{
    Info infovenster = new Info();
    infovenster.ShowDialog();
}
```

KeuzevensterTxt

```
private TextBox tb;

public KeuzevensterTxt(TextBox txtResult)
{
    InitializeComponent();

    // Hier binnen het tekstvak definieren en de 2 objecten worden aan elkaar gekoppeld.
    tb = txtResult;
}

private void BtnOk_Click(object sender, RoutedEventArgs e)
{
    // === Via TextBox doorgegeven ===

    if (RadUpper.IsChecked == true)
    {
        tb.Text = tb.Text.ToUpper();
        tb.Background = System.Windows.Media.Brushes.BurlyWood;
        tb.FontWeight = FontWeights.Bold;
    }
}
```

```

        else
        {
            tb.Text = tb.Text.ToLower();
        }

        // Doorgeven op OK geklikt.
        DialogResult = true;
    }

    private void BtnCancel_Click(object sender, RoutedEventArgs e)
    {
        // Doorgeven op CANCEL geklikt.
        DialogResult = false;
    }

```

- **Uitwisseling via static klasse**

Indien je variabelen of objecten over meerdere vensters nodig hebt, kunnen de variabelen publiek gedeclareerd worden in een aparte klasse. Maak een static klasse DataTxt aan via ADD CLASS en formuleer je variabelen static (langere levensduur). We gebruiken de klasse static omdat de klasse niet overgeërfd kan worden (later meer hierover) en omdat ze enkel static variabelen omvat. Een statistische klasse moet bovendien nooit aangemaakt worden met een nieuwe instantie.

Klasse DataTxt

```

public static class DataTxt
{
    public static string Tekst = string.Empty;
}

```

HoofdvensterKlasse

```

public HoofdvensterKlasse()
{
    InitializeComponent();

    // Dadelijk inhoud van TxtResultaat toekennen aan Klasse DataTxt. Kan ook bij Form_Load.
    DataTxt.Tekst = TxtResultaat.Text;
}

private void BtnWijzigen_Click(object sender, RoutedEventArgs e)
{
    KeuzevensterKlasse keuzevenster = new KeuzevensterKlasse();
    keuzevenster.Show();
}

private void Window_Activated(object sender, EventArgs e)
{
    // Windows actief bij Load en terug visible.
    TxtResultaat.Text = DataTxt.Tekst;
}

private void BtnInfo_Click(object sender, RoutedEventArgs e)
{
    Info infovenster = new Info();
    infovenster.ShowDialog();
}

```

KeuzevensterKlasse

```
private void BtnOk_Click(object sender, RoutedEventArgs e)
{
    // === Via klasse doorgegeven ===

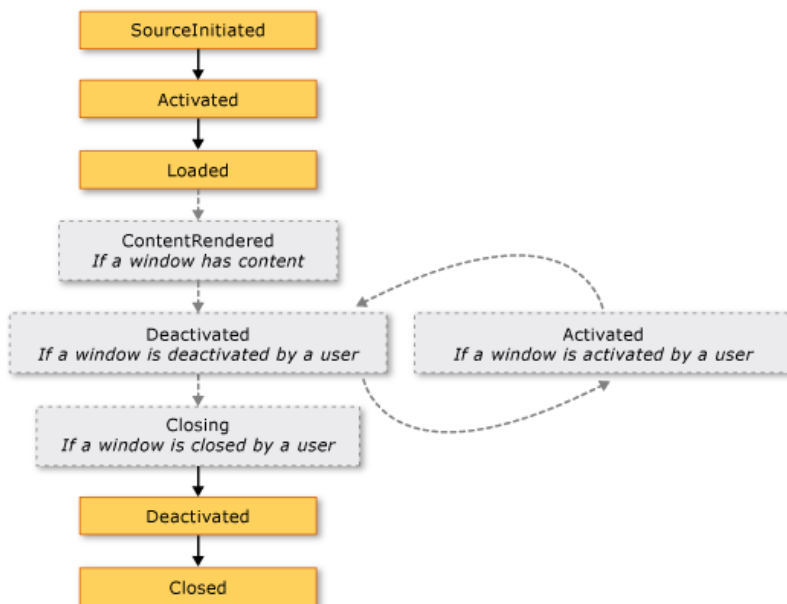
    if (RadUpper.IsChecked == true)
    {
        DataTxt.Tekst = DataTxt.Tekst.ToUpper();
    }
    else
    {
        DataTxt.Tekst = DataTxt.Tekst.ToLower();
    }

    //Formulier verbergen zodat HoofdvensterKlasse terug actief (Window_Activated) wordt.
    Hide();
}

private void BtnCancel_Click(object sender, RoutedEventArgs e)
{
    Close();
}
}
```

2.3 Gebeurtenissen bij vensters

Er kunnen diverse gebeurtenissen optreden bij vensters. De levensduur van een form is:



De volgorde bij het openen van vensters is:

- formulier wordt geladen dmv Window_Loaded
- InitializeComponent() waarbij XAML wordt doorgevoerd (objecten op het venster)

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    MessageBox.Show("Windows wordt geladen.");
}
```

```
public MainWindow()
{
    InitializeComponent();
}
```

De *volgorde* bij het sluiten van vensters is:

- Closing(): formulier gaat sluiten;
- Closed(): formulier is gesloten;

```
private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
{
    MessageBox.Show("Windows gaat sluiten.");
}

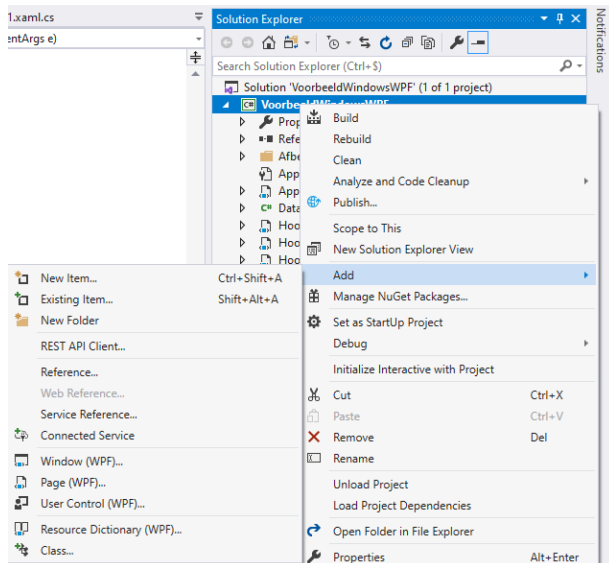
private void Window_Closed(object sender, EventArgs e)
{
    MessageBox.Show("Windows is gesloten.");
}
```

Het afsluiten van een venster (Shutdown) kan je op verschillende manieren:

- Toepassing volledig afsluiten (geen Windows_Closing mogelijk)
De volledige applicatie wordt afgesloten ongeacht of er andere vensters nog actief zijn.
Environment.Exit(0);
- Huidig venster afsluiten (Windows_Closing mogelijk)
Application.Current.Shutdown(); of **Close();**

2.4 Nieuw venster toevoegen

Klik met je rechter muisknop op je project (in vet) → Add → New Item → WPF Window



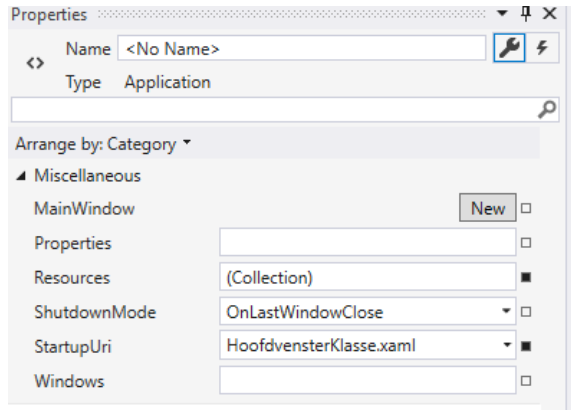
2.5 Opstartvolgorde bij meerdere vensters.

In App.xaml kan je de opstartvolgorde van vensters wijzigen.

```
// Opstarten van Hoofdvenster.
```

`StartupUri="HoofdvensterKlasse.xaml"`

<!--Klik op StartupUri en in je eigenschappenvenster kan je de verschillende vensters kiezen.-->



ShutdownMode:

- OnLastWindowClose: de applicatie sluit als het laatste venster gesloten is.
- OnMainWindowClose: de applicatie sluit volledig af als het MainWindow wordt gesloten ongeacht of er nog andere vensters open zijn.
- OnExplicitShutdown: sluit alleen bij een expliciete Shutdown (Close() of Application.Current.Shutdown())

StartUri:

- Uniform resource identifiers (URIs) wordt hier gebruikt om het startvenster aan te duiden. Dit venster wordt als eerste venster opgestart.

Hoofdstuk 3 Eigen klasse maken

3.1 .NET Framework

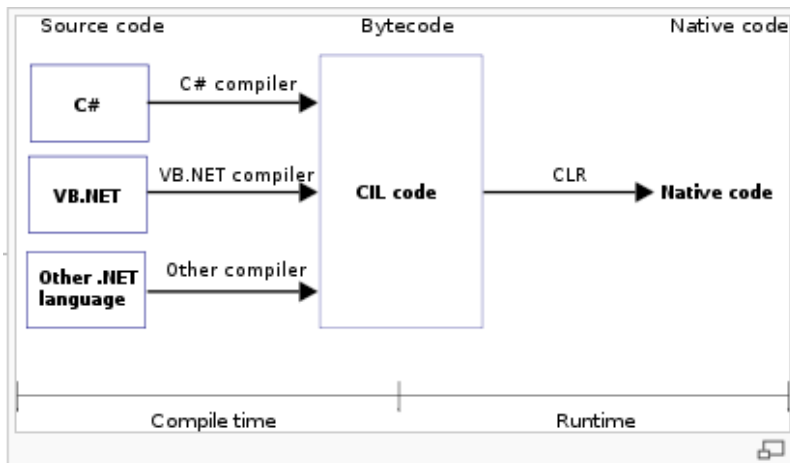
3.1.1 CLR (Common Language Runtime)

De programma's gemaakt in de verschillende talen (Visual C#, Java, Python,...) wordt gecompileerd naar een CIL code (Common Intermediate Language). Het is een objectgeoriënteerde assembleertaal dat door mensen nog gelezen kan worden. De oude benaming van CIL is MSIL (MicroSoft Intermediate Language).

Wanneer het programma wordt uitgevoerd dan wordt de CIL code door de JIT (Just In Time compilatie) compiler van de CLR omgezet naar machinetaal (broncode van het besturingssysteem). De CLR kan je zien als een virtuele machine (Virtueel Executiesysteem (VES)) en is een onderdeel van het .NET Framework. De JIT compiler vertaalt dus de CIL code naar machinetaal.

De CLR voert de CIL uit:

- Laden en uitvoeren van programmacode
- Meerdere toepassingen laten draaien
- Beheer van geheugen Windows Namespace
- Beveiligen
- Afhandelen van exception handling

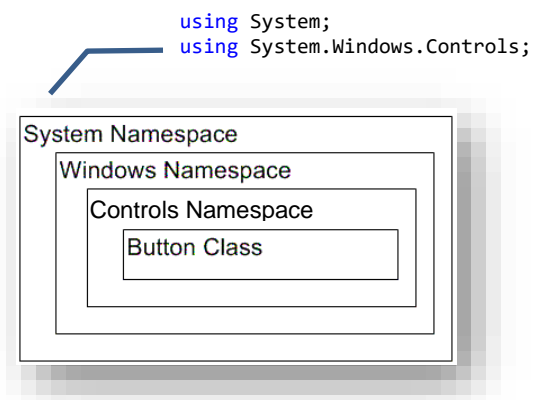


3.1.2 Class library

Het .NET Framework bevat een set van standaard class libraries. Een class library is georganiseerd in een hiërarchie van de **namespace**.

De meeste ingebouwde API's (application programming interface (API) zorgt voor de communicatie met een ander programma of onderdeel, meestal in de vorm van bibliotheken) maken deel uit van de *Microsoft* namespace of de *System* namespace. Deze class libraries implementeren een groot aantal gemeenschappelijke functies, zoals een bestand lezen en schrijven, dialoogvensters, interactie met de database,...

De .NET Framework class library is in tweeën opgedeeld: de **Base Class Library** en de **Framework Class Library**.



De *Base Class Library* (BCL) bevat een kleine deelverzameling van de hele class library en is de kern van de klassen die dienen als de basis API van de Common Language Runtime. De klassen in mscorlib.dll ¹ en enkele van de klassen in System.dll en System.core.dll worden beschouwd als een deel van de BCL.

Het Framework Class Library (FCL) is een superset van de BCL klassen en verwijst naar de hele class library die met het .NET Framework komt. Het bevat een uitgebreide set van libraries, met daarbij inbegrepen de Windows Controls, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation en Windows Communication Foundation.

3.2 Assembly

De assembly bevat:

- Gecompileerde programmacode (CIL)
- Ondersteunende bestanden (afbeeldingen, tekstbestanden,...)
- Metagegevens (info over methodes, type, klasse,...)
- Manifest (info over versie, bestanden,...)

3.3 Object georiëteerd modelleren

Om object georiëteerd (OO) te kunnen programmeren zijn **twee dingen** nodig. Ten eerste moet er een probleem zijn, dat object georiëteerd gemodelleerd kan worden en ten tweede dienen we over een programmeertaal te beschikken die object georiëteerd is. De hier gebruikte OO-programmeertaal is C#.

Het basis-idee van object georiëteerd modelleren is dat de wereld bestaat uit objecten die met elkaar communiceren door elkaar berichten (messages) te sturen. We kunnen b.v. studenten en lectoren prima modelleren als objecten die met elkaar communiceren door elkaar boodschappen toe te sturen.

De eerste twee vragen die je je moet stellen als je iets object georiëteerd wilt modelleren zijn:

- ✓ Wat zijn de objecten?
- ✓ Hoe communiceren de objecten met elkaar?

- Objecten kunnen geklassificeerd worden.

Studenten zijn objecten met naw-gegevens (naam,adres,woonplaats), inschrijvingsjaar, vakken, behaalde resultaten,... Deze beschrijving is voor alle studenten hetzelfde.

De gegevens in een klasse kunnen **ingekapseld** (encapsulation) worden met het **keyword class**.

```
public class Student
{
    ...
}
```

- Objecten moeten eerst gecreëerd worden.

Het resultaat van creatie heet een instantie (instance). Een klasse is slechts een beschrijving van hoe een object er uit gaat zien en wat er mee gedaan kan worden als het gecreëerd wordt. Iedere student is een instantie van de klasse Student en kan gecreëerd worden door:

```
Student stud = new Student()
```

 dat een nieuwe instantie maakt van de klasse Student.

¹ DLL (*D*ynamic *L*ink *L*ibrary) is de Softwarelibrary onder Windows behorend bij een bepaald programma, die een aantal logisch bij elkaar behorende functies bevat die tijdens de uitvoering van het programma worden aangeroepen en uitgevoerd. DLL-bestanden zijn te herkennen aan de extensie .DLL

- Objecten kunnen een toestand/velden hebben.

De toestand kan worden vastgelegd door lokale variabelen.

Vb. `private string` voornaam;
`private string` naam; ...

- Objecten **communiceren met zogenaamde eigenschappen en methoden**.

Methoden kunnen gebruikt worden om gegevens te bewerken of gegevens te bekijken of gegevens te veranderen,...

Eigenschap

```
public string Naam
{
    get { return familienaam; } geeft waarde terug
    set { familienaam = value; } ontvangt waarde
}
```

Method

```
public string ToonNaam()
{
    return voornaam + " - " + familienaam;
}
```

- De interne toestand van objecten kan onzichtbaar voor de buitenwereld gemaakt worden. Dit heet (*data hiding*). Indien we ervoor zorgen dat lokale variabelen of methoden niet publiekelijk (niet via de modifier `public`) zichtbaar zijn, worden ze ook niet direct zichtbaar voor de buitenwereld.

Vb. `private string` voornaam;

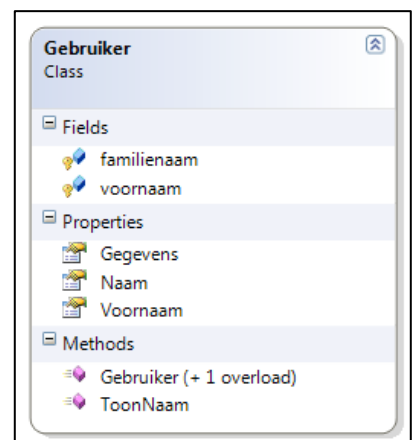
3.4 Werken met klasse(n)

Kies **Add Class** uit het projectmenu en voeg het class-sjabloon toe.

Klasse

```
public class Gebruiker
{
    //=====
    //  DECLARATIE VAN INSTANTIEVARIABLEN.
    //=====
    private string voornaam;
    private string familienaam;

    //=====
    //  DEFAULT CONSTRUCTOR ZONDER PARAMETERS
    //=====
    public Gebruiker()
    {
        //Initialisatie instantievariabelen.
        familienaam = string.Empty;
        voornaam = string.Empty;
    }
}
```




```

//=====
//  CONSTRUCTOR MET PARAMETERS
//=====
public Gebruiker(string name, string familyname)
{
    familienaam = familyname; // De property Voornaam/Naam krijgt een waarde.
    voornaam = name;
}

//=====
//  READ AND WRITE EIGENSCHAP VOORNAAM
//=====
public string Voornaam
{
    get { return voornaam; } // GetAccessor haalt waarde uit klasse.
    set { voornaam = value; } // SetAccessor ontvangt waarde.
}

//=====
//  READ AND WRITE EIGENSCHAP NAAM
//=====
public string Naam
{
    get { return familienaam; }
    set { familienaam = value; }
}

//=====
//  READ-ONLY EIGENSCHAP GEGEVENS
//  Voegt voornaam en naam in hoofdletter samen
//=====
public string Gegevens
{
    get { return voornaam + " " + familienaam.ToUpper(); }
}

//=====
//  FUNCTIEMETHOD TOONNAAM
//  Voegt voornaam en familienaam samen.
//=====
public string ToonNaam()
{
    return voornaam + " - " + familienaam;
}
}

```

N.B. De constructor moet altijd dezelfde naam als de klasse hebben!

Vanuit code

```
// Oproepen van constructor met parameters
Gebruiker gb = new Gebruiker("Patricia", "Briers");

// Klasse gebruiken
MessageBox.Show(gb.Voornaam, "Voornaam");
MessageBox.Show(gb.Gegevens, "Gegevens");

// Functie van klasse oproepen
MessageBox.Show(gb.ToonNaam(), "Naam van gebruiker");
```



Vergeet zeker niet met de **snippet** te werken, want deze levert snel de structuur van een eigenschap.

Ctor geeft de structuur van de constructor

prop geeft dadelijk : `public int MyProperty { get; set; }`

propfull geeft dadelijk:

```
public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
}
```

3.5 Auto-implemented properties en Expression-bodied properties.

Je kan ook gebruikmaken van auto-implemented properties. Hierdoor hoeft men voor de herhaaldelijke basisconstructie, waar de *Getter* de waarde van het gekoppelde veld oplevert, en de *Setter* de toegekende waarde aan het veld toekent, enkel nog een soort van signatuurregel definiëren: `public string Naam { get; set; }`

Vanaf C# 6.0 (Visual Studio 2015) kan je ook gebruik maken van de *Expression-bodied properties* die de `return` vervangen door het symbool `'=>'`.

Zo wordt onderstaande read-only eigenschap eens stuk korter geschreven:

```
public string Gegevens
{
    get { return Voornaam + " " + Naam.ToUpper(); }
}
```

wordt:

```
public string Gegevens => Voornaam + " " + Naam.ToUpper();
```

De functiemethode:

```
public string ToonNaam()
{
    return voornaam + " - " + familienaam;
}
```

wordt:

```
public string ToonNaam() => voornaam + " - " + familienaam;
```

Klasse

```
public string Naam {get; set;}
public string Voornaam {get; set;}

// READ-ONLY eigenschap Gegevens
public string Gegevens => Voornaam + " " + Naam.ToUpper();

// De functie Naam voegt voornaam en familienaam samen.
public string ToonNaam() => Voornaam + " - " + Naam;
```

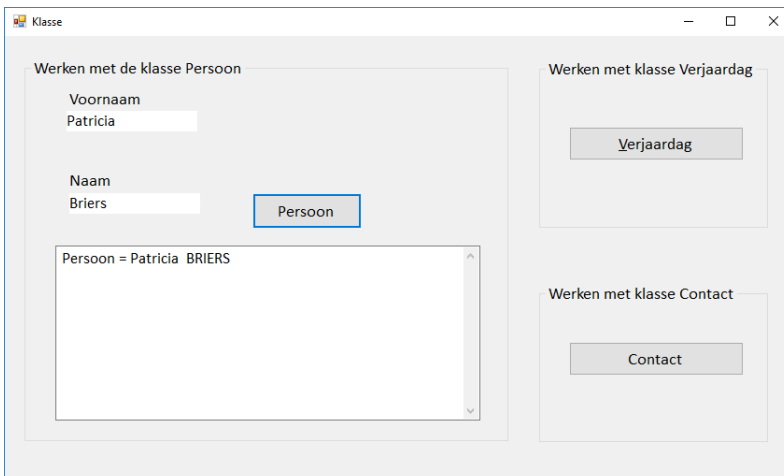
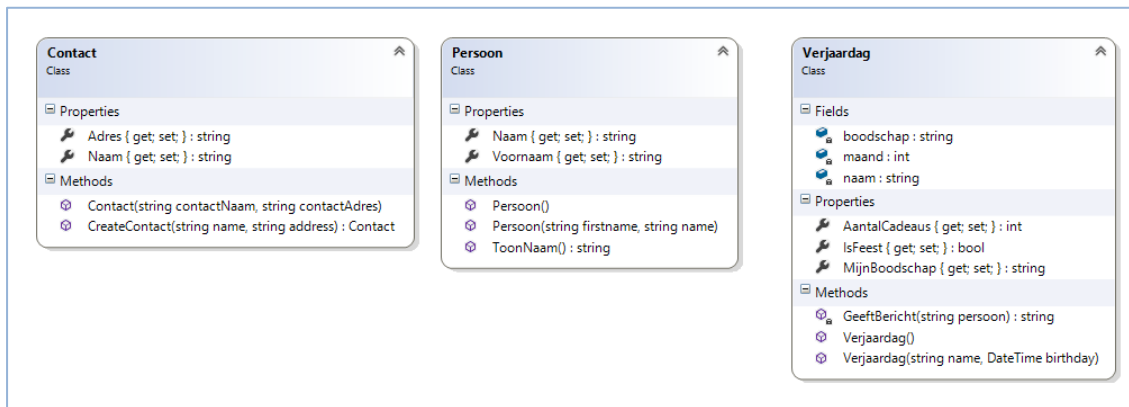
Vanuit code

```
// Oproepen van constructor zonder parameters. Bij new wordt automatisch een lege constructor
// gehanteerd.
Gebruiker gb = new Gebruiker();           === OF ===   Gebruiker gb = new Gebruiker
gb.Voornaam = TxtVoornaam.Text;           {   Voornaam = TxtVoornaam.Text,
gb.Naam = TxtNaam.Text;                   Naam = TxtNaam.Text
};

// Klasse gebruiken
MessageBox.Show(gb.Voornaam, "Voornaam");
MessageBox.Show(gb.Gegevens, "Gegevens");

// Functie van klasse oproepen
MessageBox.Show(gb.ToonNaam(), "Naam van gebruiker");
```

Voorbeeld 2



Programmeercode

```
private void BtnPersoon_Click(object sender, EventArgs e)
{
    // Constructor zonder parameter.
    Persoon iemand = new Persoon();
    TxtResultaat.Text = iemand.ToonNaam();

    // === of ===

    // Constructor met parameter.
    Persoon ik = new Persoon(TxtVoornaam.Text, TxtNaam.Text);
    TxtResultaat.Text = ik.ToonNaam();
}

private void BtnVerjaardag_Click(object sender, EventArgs e)
{
    // Constructor zonder parameters
    Verjaardag vj = new Verjaardag();

    vj.IsFeest = false;
    //vj.AantalCadeaus = 5;
    vj.MijnBoodschap = " Kristof";
    MessageBox.Show(vj.MijnBoodschap, "Verjaardagsfeest?");

    // === of ===

    // Constructor met parameters
    Verjaardag vjp = new Verjaardag(TxtVoornaam.Text, DateTime.Parse("1980-03-16"));
    MessageBox.Show(vjp.MijnBoodschap, TxtVoornaam.Text + " " + TxtNaam.Text);
}

private void BtnContact_Click(object sender, EventArgs e)
{
    string[] namen = {"Patricia Briers", "Lise Mercken", "Joachim Hemelaer",
                     "Pieter Kuppens", "Muriel Garcia"};
    string[] adressen = {"Elfde-Liniestraat 26 3500 Hasselt", "Genkerbaan 112 3600 Genk",
                        "Marktplein 45 3740 Rosmeer", "Keizel 18 3590 Diepenbeek", "Kapellestraat 28 3545 Halen"};

    // ===== Werken met klasse =====
    Contact iemand;
    for (int i = 0; i < namen.Length; i++)
    {
        iemand = new Contact(namen[i], adressen[i]);
        TxtResultaat.Text += $"{iemand.Naam}, {iemand.Adres}\r\n";
    }

    TxtResultaat.Text += Environment.NewLine;

    // === of ===

    // ===== Werken met geneste klasse =====

    for (int i = 0; i < namen.Length; i++)
    {
        // Oproepen constructor + geeft de eigenschappen Name en Address terug.
        var c = Contact.CreateContact(namen[i], adressen[i]);
        TxtResultaat.Text += $"{c.Naam}, {c.Adres}\r\n";
    }
}
```

Klasse Persoon

```
public class Persoon
{
    // === Read and write properties ===
    public string Naam { get; set; }
    public string Voornaam { get; set; }

    // ===== Constructor =====
    public Persoon()
    {
        Naam = "onbekend";
    }

    // ===== Constructor met parameters =====
    public Persoon( string firstname, string name)
    {
        Naam = name;
        Voornaam = firstname;
    }

    // ===== Method:gegevens tonen =====
    public string ToonNaam() => "Persoon = " + Voornaam + " " + Naam.ToUpper();
}
```

Je kan verhinderen dat er met een leeg object wordt begonnen door de default constructor rechtstreeks te verwijzen naar de geparametriseerde constructor.

Indien je geen constructor opneemt in je klasse, wordt automatisch een lege default constructor aangemaakt.

```
// == Constructor chaining: constructor met parameters wordt opgeroepen. ==
public Persoon() : this(null,"Onbekend")
{ }

// ===== Constructor met parameters =====
public Persoon( string firstname, string name)
{
    Naam = name;
    Voornaam = firstname;
}
```

Klasse Verjaardag

```
public class Verjaardag
{
    // ===== Klassevariabelen =====
    private int maand;
    private string boodschap, naam;

    // ===== CONSTRUCTOR ZONDER PARAMETERS =====
    public Verjaardag()
    { AantalCadeaus= 1; }

    // ===== CONSTRUCTOR MET PARAMETERS =====
    public Verjaardag(string name, DateTime birthday)
    {
        naam = name;
        maand = birthday.Month;
        AantalCadeaus = 5;
        if (maand == DateTime.Today.Month) IsFeest = true;
    }
}
```

```

        MijnBoodschap = naam;
    }

    // ==== READ AND WRITE PROPERTY ====
    public int AantalCadeaus { get; set; }
    public bool IsFeest { get; set; }

    public string MijnBoodschap
    {
        get { return boodschap; }
        set { boodschap = GeeftBericht(value); }
    }

    // ==== PRIVATE FUNCTION ====
    private string GeeftBericht(string persoon)
    {
        string bericht;

        if (IsFeest)
        {
            bericht = $"Gelukkige verjaardag! {persoon}\r\nAantal cadeaus = {AantalCadeaus}\r\nGeniet van het verjaardagsfeestje!";
        }
        else
        {
            bericht = $"Jammer {persoon} geen verjaardagsfeest!";
        }
        return bericht;
    }
}

```

```

// Een STATIC GENESTE KLASSE wordt gebruikt.
public class Contact
{
    // ==== Read-only properties ====
    public string Naam { get; private set; } // setter enkel voor intern gebruik!!
    public string Adres { get; private set; }

    // ==== Constructor ====
    public Contact(string contactNaam, string contactAdres)
    {
        Naam = contactNaam;
        Adres = contactAdres;
    }

    // ==== GENESTE KLASSE : maakt nieuwe instantie aan van klasse Contact ====
    public static Contact CreateContact(string name, string address)
    {
        return new Contact(name, address); // constructor wordt opgeroepen
    }
}

```

3.6 Static methoden en properties

Sommige methoden werken niet op een object. Een voorbeeld is de wiskundige vierkantswortelfunctie `Sqrt()`. Dergelijke methoden behoren tot een klasse (alles moet immers onder een klasse geformuleerd worden), maar worden beschreven als **Static**.

Wanneer je een van deze methode gebruikt, moeten ze voorafgegaan worden door de naam van de klasse. Vb. `Math.Sqrt()`

Andere voorbeelen zijn: `Convert.ToString`, `Convert.ToSingle()`, `String.Format()`,...

3.7 Partial Class

Als je klasse te groot of te ingewikkeld wordt, kan je de klasse opsplitsen in verschillende delen. Je gebruikt hierbij het sleutelwoord `partial`.

Contact.cs

```
public partial class Contact
{
    // ==== Read-only properties ====
    public string Naam { get; private set; }
    public string Adres { get; private set; }

    // ==== Constructor ====
    public Contact(string contactNaam, string contactAdres)
    {
        Naam = contactNaam;
        Adres = contactAdres;
    }
}
```

Contact.cs

```
public partial class Contact
{
    // In dit deel kunnen enkel zelfstandige methoden staan want de eigenschappen van deel 1
    // worden hier niet herkend!
    public static Contact CreateContact(string name, string address)
    {
        return new Contact(name, address);
    }
}
```

Je mag het tweede deel ook opslaan in een nieuw bestand (vb. `Contact2.cs`).

3.8 Declaratie van class, eigenschappen of procedures

<code>public</code>	binnen de assembly en de assembly die ernaar verwijst.
<code>private</code>	binnen dezelfde klasse of structuur
<code>protected</code>	binnen de klasse of afgeleide klasse
<code>internal</code>	enkel binnen de assembly
<code>protected internal</code>	binnen dezelfde assembly of binnen een afgeleide klasse

3.9 Object vernietigen

Een object houdt op te bestaan wanneer een programma stopt of wanneer je het object vernietigt door middel van: `object = null;`

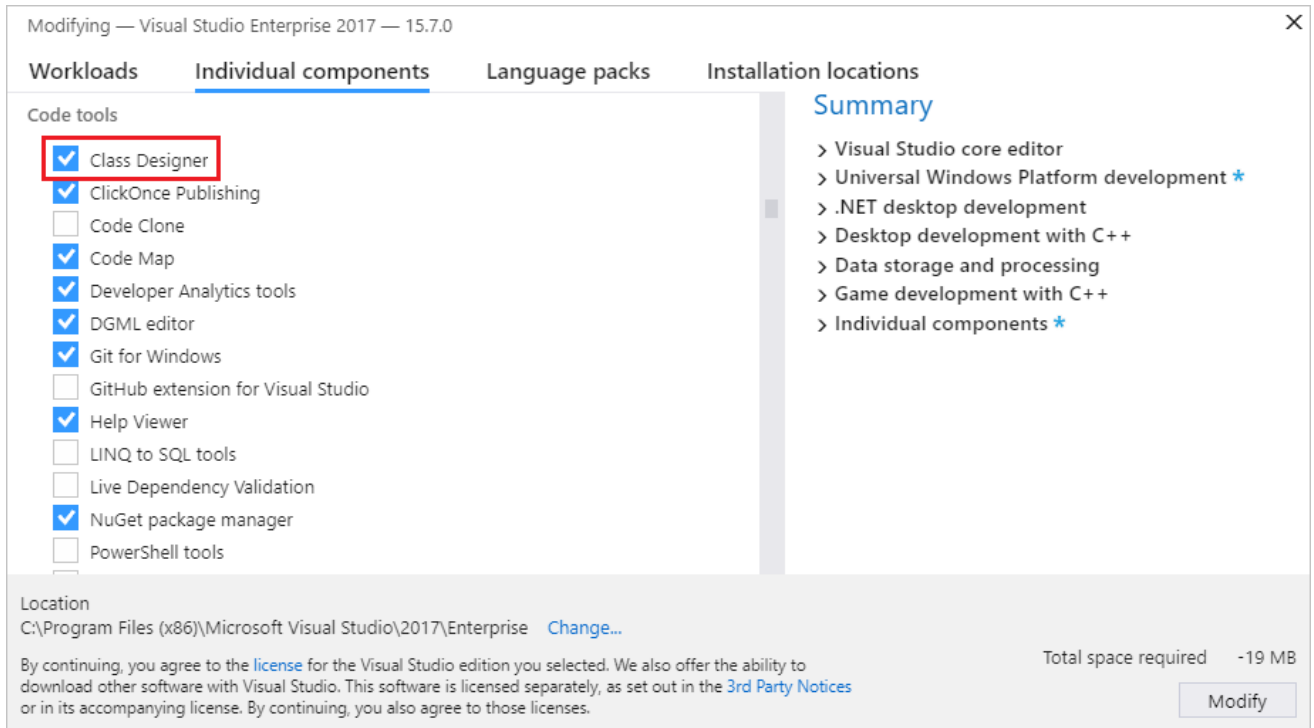
3.10 Installeer het onderdeel Class Designer

Als u het onderdeel **Class Designer** niet hebt geïnstalleerd, volgt u deze stappen om het te installeren.

1. Open **Visual Studio Installer** vanuit het Windows Start-menu of door **Tools > Get Tools and Features te selecteren** in de menubalk in Visual Studio.

Visual Studio Installer wordt geopend.

2. Selecteer het tabblad **Individuele componenten** en blader vervolgens naar de categorie Codetools.
3. Selecteer **Klasseontwerper** en selecteer vervolgens **Wijzigen** .



3.10.1 Voeg een leeg klassendiagram toe aan een project

1. Klik in **Solution Explorer** met de rechtermuisknop op het projectknooppunt en kies vervolgens **Toevoegen > Nieuw item** . Of druk op **Ctrl + Shift + A**.

Het dialoogvenster **Nieuw item toevoegen** wordt geopend.

2. Vouw **Algemene items > Algemeen uit** en selecteer vervolgens **Class Diagram** in de sjabloonlijst. Voor Visual C ++ -projecten, kijk in de categorie **Hulpprogramma** om de **Class Diagram**- sjabloon te vinden.

Het klassendiagram wordt geopend in Class Designer en wordt weergegeven als een bestand met de extensie **.cd** in **Solution Explorer** . U kunt vormen en lijnen vanuit de **Werkset** naar het diagram slepen.

Herhaal de stappen in deze procedure om meerdere klassendiagrammen toe te voegen.

3.11 Voeg een klassendiagram toe op basis van bestaande typen

Open in **Solution Explorer** het contextmenu van een klassebestand (klik met de rechtermuisknop) en kies vervolgens **View Class Diagram** . Of je opent in **Class View** de naamruimte of typ het contextmenu en kies vervolgens **View Class Diagram** .

Tip

Als **Class View** niet is geopend, opent u **Class View** vanuit het **View**- menu.

3.12 De inhoud van een compleet project weergeven in een klassendiagram

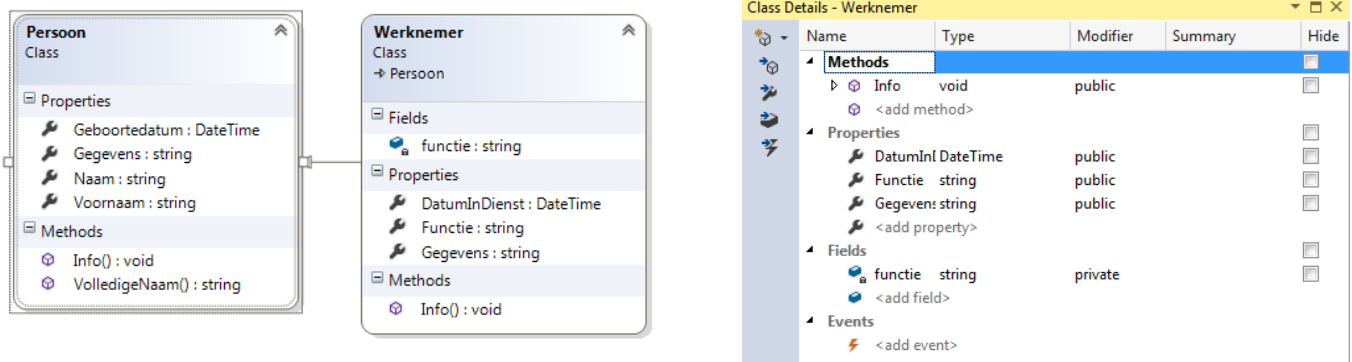
Klik in **Solution Explorer** of Class View met de rechtermuisknop op het project en kies **View** en vervolgens **View Class Diagram**. Er wordt een automatisch ingevuld klassendiagram gemaakt.

Hoofdstuk 4 Overerving en polymorfisme van klassen

4.1 Het gebruik van Class diagram

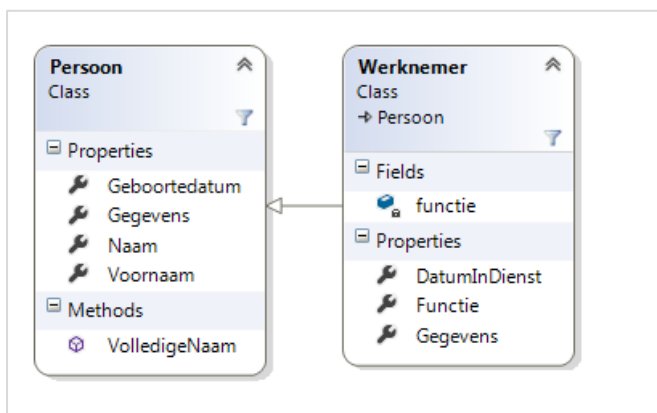
Menu: Project – New item – Class diagram

De diagram kan je gebruiken om te navigeren maar via het venster Class Detail kan je methode, eigenschappen of velden toevoegen/verwijderen. In het menu verschijnt een extra tab Class Diagram om eveneens methoden, eigenschappen of velden toe te voegen of te verwijderen.



4.2 Overerving

Eén van de concepten van OOP is de mogelijkheid om nieuwe klassen te maken op basis van bestaande klassen. De nieuwe klassen erven de eigenschappen, methodes en events van de basisklassen. Eerst moet in de basisklasse aangegeven worden dat de betreffende methode mag overschreven worden met behulp van het sleutelwoord **Virtual** en in de afgeleide klasse wordt de methode aangegeven door **Override**.



Klasse Persoon

```
public class Persoon
{
    public string Voornaam {get; set;}
    public string Naam {get; set;}
    public DateTime Geboortedatum {get; set;}

    // Virtual: eigenschap in de afgeleide klasse kan overschreven worden.
    public virtual string VolledigeNaam() => Voornaam + " " + Naam;
    public virtual string Gegevens => $"{Voornaam} {Naam}\r\nGeboortedatum: {Geboortedatum.ToLongDateString()}";
}
```

Klasse Werknemer

```
public class Werknemer : Persoon
{
    public DateTime DatumInDienst { get; set; }
    private string functie = "Lector";

    public string Functie
    {
        get { return functie; }
        set { functie = value; }
    }

    public override string Gegevens => $"{base.Gegevens} - {Functie}";
}
```

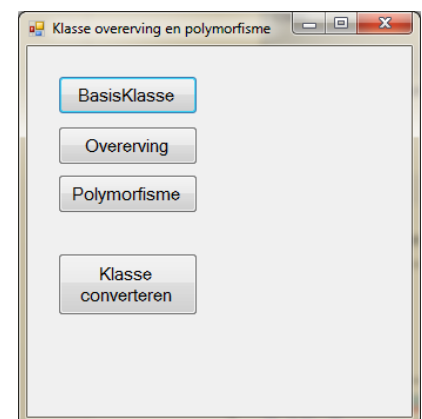
Venster

```
// Klassevariabelen.
Werknemer werknemer = new Werknemer();
Persoon persoon = new Persoon();

public void BtnBasisKlasse_Click(object sender, EventArgs e)
{
    // Initialiseren van klasse.
    persoon.Voornaam = "Patricia";
    persoon.Naam = "Briers";
    persoon.Geboortedatum = DateTime.Parse("27/11/1979");

    MessageBox.Show(persoon.VolledigeNaam(), " Volledige naam", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    MessageBox.Show(persoon.Gegevens, "Persoonlijke gegevens", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

public void tnOvererving_Click(object sender, EventArgs e)
{
    // Je kan de method VolledigeNaam() gebruiken uit de klasse Persoon.
    werknemer.Voornaam = "Leona";
    werknemer.Naam = "Bertels";
}
```

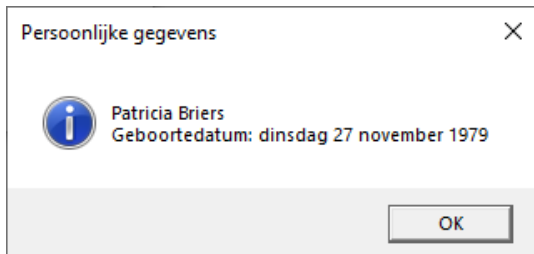


```

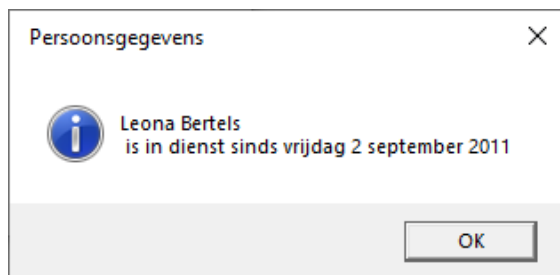
    werknemer.Geboortedatum = DateTime.Parse("27/11/1984");
    werknemer.DatumInDienst = DateTime.Parse("2/9/2011");

    string bericht = werknemer.VolledigeNaam() + "\r\n" + " is in dienst sinds " +
    werknemer.DatumInDienst.ToLongDateString();
    MessageBox.Show(bericht, " Persoonsgegevens", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}

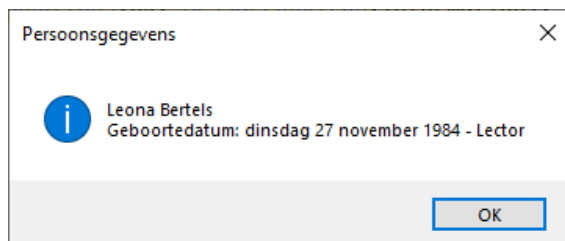
```



persoon.Gegevens



werknemer.VolledigeNaam()

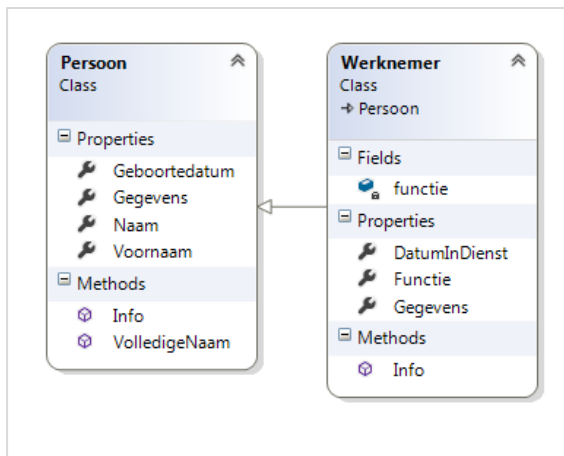


werknemer.Gegevens

4.3 Polymorfisme

Met polymorfisme worden methodes binnen een afgeleide klasse met dezelfde naam op een andere wijze geïmplementeerd.

Vb. Voidprocedure Info() wordt aan de klasse Persoon en Werknemer bijgevoegd.



Klasse Persoon

```

public virtual void Info(string tekst)
{
    string info = "Uw persoonlijke gegevens: ";
    MessageBox.Show(info + tekst, "Info Klasse Persoon", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

```

Klasse Werknemer

```

public override void Info(string tekst)
{
    string info = "Gegevens van de werknemer: ";
    MessageBox.Show(info + tekst, "Info Klasse Werknemer", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

```

Venster

```

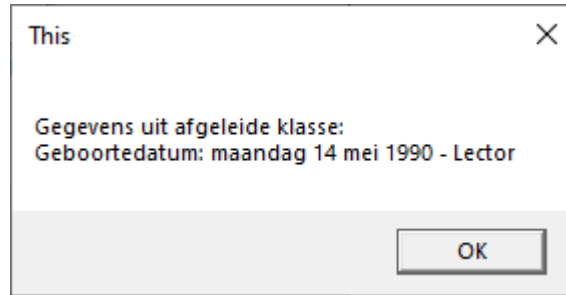
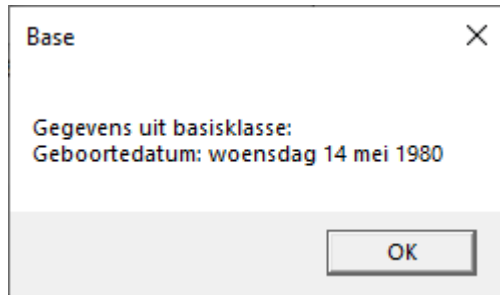
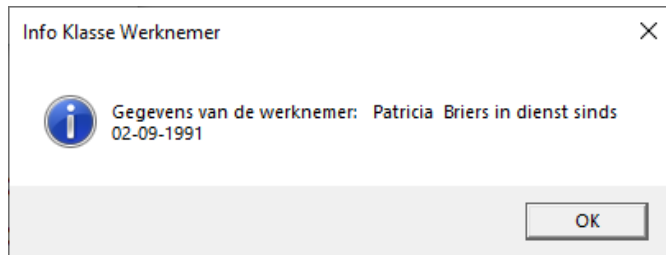
public void BtnPolymorfisme_Click(object sender, EventArgs e)
{
    Persoon persoon = new Persoon();
    persoon.Info("Maarten Clijsters" + " geboren in Hasselt");

    Werknemer werknemer = new Werknemer();
    werknemer.Info("Patricia Briers" + " in dienst sinds 02-09-1991");

    // of de eigenschappen Gegevens uit beide klassen.
    persoon.Geboortedatum = DateTime.Parse("14-05-1980");
    werknemer.Geboortedatum = DateTime.Parse("14-05-1990");
    MessageBox.Show("Gegevens uit basisklasse: " + persoon.Gegevens, "Base");
    MessageBox.Show("Gegevens uit afgeleide klasse: " + werknemer.Gegevens, "This");
}

```





4.3.1 Sleutelwoord Base

Indien je binnen een afgeleide klasse wilt refereren naar een eigenschap of methode in de basisklasse, dan kan je het sleutelwoord **base** gebruiken. Zolang de eigenschappen en methoden in de afgeleide klasse niet overschreven worden, verwijzen `MethodeNaam()/eigenschap` en `base.MethodeNaam()/base.eigenschap` naar dezelfde code en is de werking ervan gelijk.

```
public override string Gegevens => $"{base.Gegevens} - {Functie}";
```

4.3.2 Converteren van eigen objecten

Objecten van de basisklasse kunnen zonder problemen geconverteerd worden (verbredende conversie) naar de afgeleide klasse, maar bij een conversie van de afgeleide klasse naar de basisklasse kunnen gegevens verloren gaan (versmallende conversie).

```
public void BtnConverteren_Click(object sender, EventArgs e)
{
    persoon.Geboortedatum = DateTime.Parse("14-05-1980");
    werknemer.Geboortedatum = DateTime.Parse("14-05-1990");

    MessageBox.Show($"{BerekenLeeftijd(persoon)} jaar", "Geboortedatum van klasse persoon");
    → 39 jaar

    // De functie BerekenLeeftijd accepteert enkel een argument van het type Persoon!
    persoon = (Persoon) werknemer;

    MessageBox.Show($"{BerekenLeeftijd(persoon)} jaar", "Geboortedatum van klasse werknemer");
    → 29 jaar
}
```

```

public int BerekenLeeftijd(Persoon persoon)
{
    DateTime vandaag = DateTime.Today;
    int leeftijd = vandaag.Year - this.persoon.Geboortedatum.Year;
    if (this.persoon.Geboortedatum > vandaag.AddYears(-leeftijd)) leeftijd--;
    return leeftijd;
}

```

4.3.3 Klasse en generieke collectie

Een klasse die een eigen List *ModuleList<>* van verschillende datatypes of een klasse definieert.

Klasse *ModuleList*

```

public class ModuleList<T> // T wordt gebruikt voor elk datatype!
{
    private List<T> lst = new List<T>();

    // Methode ADD (toevoegen)
    public void Add(T item)
    {
        lst.Add(item);
    }

    // read-only eigenschap
    public int Count
    {
        get { return lst.Count; }
    }

    // ToString methode
    public override string ToString()
    {
        string listGeg = "";
        {
            for (int i = 0; i < lst.Count; i++)
            {
                listGeg += lst[i].ToString() + "\n";
            }
            return listGeg;
        }
    }
}

```

Klasse *Module*

```

public class Module
{
    public Module(string name, string school, string registration)
    {
        Naam = name;
        School = school;
        Registratie = registration;
    }

    public string Naam { get; set; }
    public string School { get; set; }
    public string Registratie { get; set; }
}

```

```

    public override string ToString()
    {
        return $"{Naam}    {Registratie} | {School}";
    }
}

```

Console toepassing

De eerste *ModuleList<int>* die 'ints' aan de list toevoegt, de *tweede ModuleList<string>* die alfanumerieke waarden toevoegt en de derde *ModuleList<Module>* die een list van klassen toevoegt.

```

class Program
{
    static void Main(string[] args)
    {
        // Numerieke waarden
        Console.WriteLine("List van gehele getallen");
        ModuleList<int> lst = new ModuleList<int>();
        int i1 = 100;
        int i2 = 200;
        lst.Add(i1);
        lst.Add(i2);
        Console.WriteLine(lst.ToString());
        Console.ReadLine();

        //Alfanumerieke waarden
        Console.WriteLine("List van alfanumerieke waarden");
        ModuleList<string> lst2 = new ModuleList<string>();
        string naam = "Programmeren in C#";
        string afdeling = "HI1 OKOT";
        lst2.Add(naam);
        lst2.Add(afdeling);
        Console.WriteLine(lst2.ToString());
        Console.Read();

        //uit klasse Module
        Console.WriteLine("List van de verschillende modules");
        ModuleList<Module> lst3 = new ModuleList<Module>();
        Module m1 = new Module("Programmeren 1 in C#", "PCVO Limburg Campus Hasselt", "2016");
        Module m2 = new Module("Programmeren 2 in C#", "PCVO Limburg Campus MaasMechelen",
"2017");
        lst3.Add(m1);
        lst3.Add(m2);
        Console.WriteLine(lst3.ToString());
        Console.Read();
    }
}

```


Afdruk

```
List van gehele getallen
100
200

List van alfanumerieke waarden
Programmeren in C#
Hi! OKOT

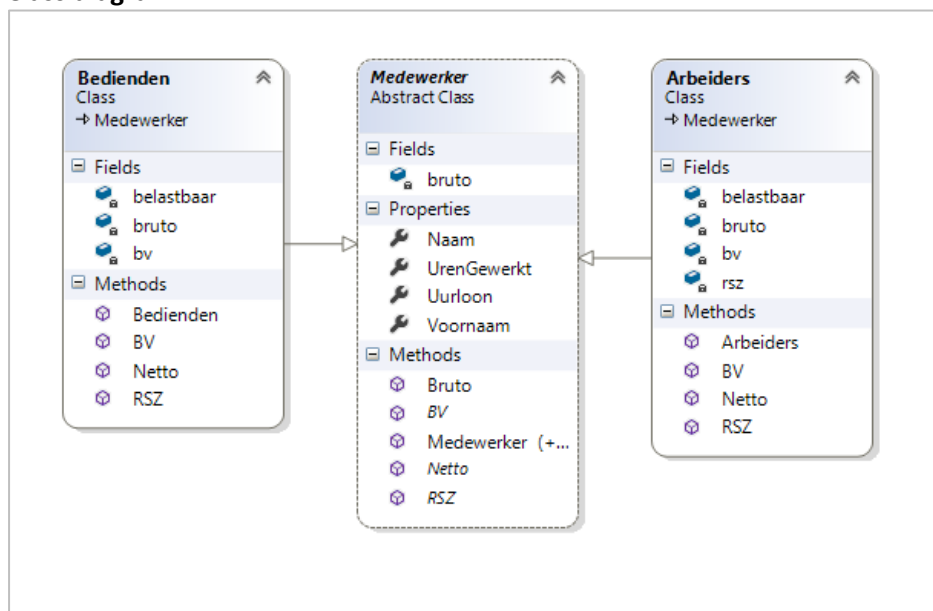
List van de verschillende modules
Programmeren 1 in C# 2016 : PCVO Limburg Campus Hasselt
Programmeren 2 in C# 2017 : PCVO Limburg Campus MaasMechelen
```

4.4 Abstracte klasse

Een abstracte klasse is ontworpen om te fungeren als een basisklasse en in de afgeleide klassen **moet** de basis worden overgeërfd. We spreken in dit verband ook van dynamisch polymorfisme ipv het statisch polymorfisme bij het overladen van methoden.

Je kan dus **geen instantie** creëren van dergelijke abstracte klasse, maar je kan slecht iets doen met de members van die abstracte klasse indien je van die klasse gaat overerven. Het is enkel mogelijk om instanties te creëren van concreet (dus niet abstracte) afgeleide types van dit abstracte type. Abstracte klassen worden bijna altijd gebruikt om iets abstract te omschrijven dat fungeert als concept.

Class diagram



```
public abstract class Medewerker
{
    private double bruto;

    // De klasse Werknemer is een abstracte klasse.
    public string Voornaam { get; set; }
    public string Naam { get; set; }
    public double Uurloon { get; set; }
    public double UrenGewerkt { get; set; }
    public double Bruto() => bruto;

    // Constructor.
    public Medewerker(string firstName, string familyName, double salary) // Bedienden
    {
        Voornaam = firstName;
        Naam = familyName;
        bruto = salary;
    }

    // Arbeiders
    public Medewerker(string firstName, string familyName, double hourlyWage, double hours)
    {
        Voornaam = firstName;
        Naam = familyName;
        bruto = hourlyWage * hours;
    }

    // Abstracte methode.
    public abstract double RSZ();
    public abstract double BV();
    public abstract double Netto();
}
```

```

public class Arbeiders : Medewerker
{
    private double rsz, belastbaar, bruto, bv;

    public Arbeiders(string firstName, string familyName, double hourlyWage, double
        hours):base(firstName,familyName,hourlyWage,hours)
    {
        bruto = Bruto();
    }

    public override double BV()
    {
        belastbaar = bruto - RSZ();

        if (belastbaar <= 50000)
        {
            bv= belastbaar * 0.45f;
        }
        else
        {
            bv = (bruto - 50000) * 0.5f + (50000*0.45f);
        }
        return bv;
    }

    public override double Netto() => belastbaar - bv;

    public override double RSZ()
    {
        rsz = bruto * 1.08 * 0.1307;
        return rsz;
    }
}

```

```

public class Bedienden: Medewerker
{
    private double belastbaar, bruto, bv;

    //Roept klasse Werknemer op.
    public Bedienden(string firstName, string familyName, double salary) : base(firstName,
        familyName,salary)
    {
        bruto = Bruto();
    }

    // De functies moeten overschreven worden.

    public override double RSZ() => bruto * 0.1307f;

    public override double BV()
    {
        belastbaar = bruto - RSZ();
        // Belastingsschijven If-structuur
        if (belastbaar <= 45000)
        {
            bv = belastbaar * 0.45f;
        }
        else

```

```

        {
            bv = (bruto - 50000) * 0.5f + (50000 * 0.45f);
        }
        return bv;
    }

    public override double Netto() => belastbaar - bv;
}

```

```

private void btnBediende_Click(object sender, EventArgs e)
{
    Bedienden med = new Bedienden(txtVoornaamBediende.Text, txtNaamBediende.Text,
        double.Parse(txtBruto.Text));
    txtSalaris.Text = $"{med.Voornaam} {med.Naam}\r\n\r\nBruto: {med.Bruto():c}\r\nRSZ:
        {med.RSZ():c}\r\nBedrijfsvoorheffing: {med.BV():c}\r\nNetto: {med.Netto():c}";
}

private void btnArbeider_Click(object sender, EventArgs e)
{
    Arbeiders med = new Arbeiders(txtVoornaamArbeider.Text, txtNaamArbeider.Text,
        double.Parse(txtUurloon.Text), double.Parse(txtAantalUren.Text));
    txtLoon.Text = $"{med.Voornaam} {med.Naam}\r\n\r\nBruto: {med.Bruto():c}\r\nRSZ:
        {med.RSZ():c}\r\nBedrijfsvoorheffing: {med.BV():c}\r\nNetto: {med.Netto():c}";
}

```

4.5 Interface

Een interface definieert enkel de members, zoals eigenschappen en procedures, die klassen en structuren kunnen implementeren. De interface beschrijft alleen de members van de afgeleide klassen en niet hun interne werking (implementatie).

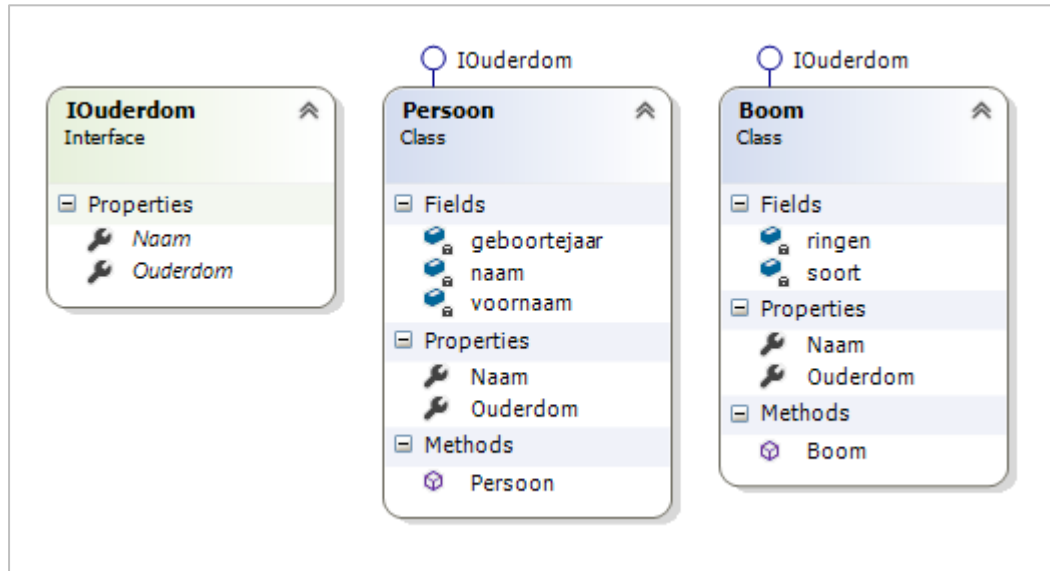
Interfaces definiëren dus een set van properties, methoden en events, net als classes. Maar in tegenstelling tot een klasse bevat een interface geen implementatie. Een interface kan je zien als een blauwdruk voor de afgeleide klassen. Een klasse die een interface implementeert, moet ieder aspect van deze klasse implementeren, precies zoals in de interface is vastgelegd.

Abstracte classes versus interfaces

Interface	Abstracte class
Kan alleen abstracte methoden bevatten (impliciet); keyword <code>abstract</code> mag niet	Kan abstracte en niet abstracte methoden bevatten
Niet bedoeld voor fields	Fields zijn mogelijk
Wordt geïntanceerd via haar implementatie	Wordt geïntanceerd via een subclass
Kan een class niet uitbreiden, kan wel een enkele of meerdere interfaces uitbreiden	Kan een enkele class uitbreiden en één of meerdere interfaces implementeren
Zijn geen onderdeel van de class-hiërarchie. Dezelfde interface (bijv <code>ILog</code>) kan gebruikt worden in classes die niets met elkaar te maken hebben (bijv. een database-class en een windows-forms class)	Abstracte classes zijn wel onderdeel van de class-hiërarchie.
Kennen geen constructor	Kunnen een constructor bevatten

Bieden een vorm van “multiple inheritance” omdat je meerdere interfaces kunt implementeren	Een class kan slechts erven van één andere class.
Alleen publieke methoden en constanten zijn mogelijk, zonder implementatie	Statische methoden, protected members en gedeeltelijke implementatie zijn allemaal mogelijk

Class diagram



```

public interface IOnderdom
{
    // Klassen die de interface IOnderdom implementeren,
    // moeten de eigenschappen Onderdom en Naam hebben.
    int Onderdom {get;}
    string Naam {get;}
}

```

```

public class Boom : IOnderdom
{
    // Instantievariabelen.
    private int ringen;
    private string soort;

    // Constructor.
    public Boom(string type, int plantjaar)
    {
        ringen = DateTime.Now.Year - plantjaar;
        soort = type;
    }

    // Property Onderdom - implementatie/interne werking van interface
    public int Onderdom => ringen;

    // Property Naam - implementatie/interne werking van interface
    public string Naam => "Boom: " + soort;
}

```

```

public class Persoon : IOuderdom
{
    // Instantievariabelen.
    private int geboortejaar;
    private string voornaam;
    private string naam;

    // Constructor.
    public Persoon(string firstName, string familyName, int yearOfBirth)
    {
        voornaam = firstName;
        naam = familyName;
        geboortejaar = yearOfBirth;
    }

    // Property Ouderdom - implementatie van interface
    public int Ouderdom => DateTime.Now.Year - geboortejaar;

    // Property Naam - implementatie van interface
    public string Naam => voornaam + " " + naam;
}

```

Windows Interface

```

public void BerekenOuderdom(IOuderdom age)
{
    MessageBox.Show(("Ouderdom: " + age.Ouderdom), age.Naam, MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}

private void BtnOuderdomPersoon_Click(object sender, EventArgs e)
{
    Persoon persoon = new Persoon(TxtVoornaam.Text, TxtFamiliennaam.Text,
        int.Parse(TxtGeboortejaar.Text));

    BerekenOuderdom(persoon);
}

private void BtnOuderdomBoom_Click(object sender, EventArgs e)
{
    Boom boom = new Boom(TxtSoort.Text, int.Parse(TxtPlantjaar.Text));
    BerekenOuderdom(boom);
}

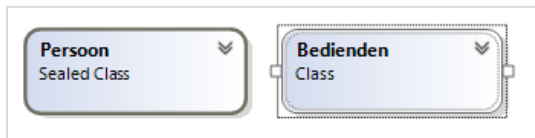
```

4.6 Sealed klasse

In tegenstelling tot een abstracte klasse dat overgeërfd moet worden, vormt de sealed (verzegeld) klasse een klasse dat niet overgeërfd kan worden. Je kan ook in een gewone klasse methoden verzegelen opdat ze niet overgeërfd kunnen worden.

Wanneer je het principe van een sealed klasse of sealed methode toepast, moet je zeker weten dat het niet overgeërfd kan worden omdat het kan leiden tot een geringere performantie.

Voorbeeld sealed klasse

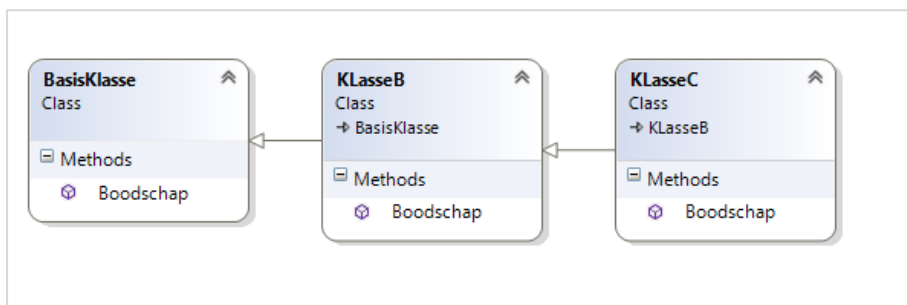


```
public sealed class Persoon
{
}
```

```
public class Bedienden: Persoon // Gaat niet!!
{
}
```

class sealedKlasse.Bedienden
'Bedienden': cannot derive from sealed type 'Persoon'

Voorbeeld sealed methods



```
public class BasisKlasse
{
    public virtual void Boodschap()
    {
        MessageBox.Show("Boodschap vanuit de basisklasse.", "Virtuele methode"
            , MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

public class KlasseB: BasisKlasse
{
    public sealed override void Boodschap()
    {
        MessageBox.Show("Boodschap vanuit de klasse die overerft van de basisklasse."
            , "Overschrijven en verzegelen van methode", MessageBoxButtons.OK
            , MessageBoxIcon.Information);
    }
}
```

```
public class KlasseC: KlasseB
{
    public override void Boodschap() // Geeft foutmelding omdat het niet kan overgeërfd worden.
    {
        void KlasseC.Boodschap()
        'KlasseC.Boodschap()': cannot override inherited member 'KlasseB.Boodschap()' because it is sealed

        MessageBox.Show("Boodschap vanuit de klasse die overerft van de basisklasse."
            , "Overschrijven en verzegelen van methode"
            , MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}
```

4.6.1 Type Klasse

Type klasse		Normaal	Static	Sealed	Abstract
Kan geïnstantieerd worden	:	Ja	Neen	Ja	Neen
Kan geërfd worden	:	Ja	Neen	Neen	Ja
Kan erven van anderen	:	Ja	Neen	Ja	Ja

4.6.2 ClassLibrary

- Class library zelf maken

Door zelf een class library te maken, kan je die klasse in meerdere projecten gebruiken door ze simpelweg toe te voegen aan je project.

Stap 1: Maak een nieuw project **BewerkingLib** en gebruik hiervoor het template **Class Library**.

Create a new project

Recent project templates

- WPF App (.NET Framework) C#
- Class Library (.NET Framework) C#**

Stap 2: Wijzig de namespace (geef een benaming dat weergeeft wat de klasse doet) en de klassenaam en vervul de klasse.

```
namespace ClassLibraryBewerking
{
    public class Bewerking
    {
        public float Som(float x, float y)
        {
            return x + y;
        }

        public float Min(float x, float y)
        {
            return x - y;
        }
    }
}
```

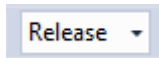


```

        public float Maal(float x, float y)
        {
            return x * y;
        }
    }
}

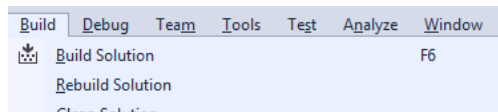
```

Stap 3: Wijzig in de standaard werkbalk de solution configuration van Debug in Release.



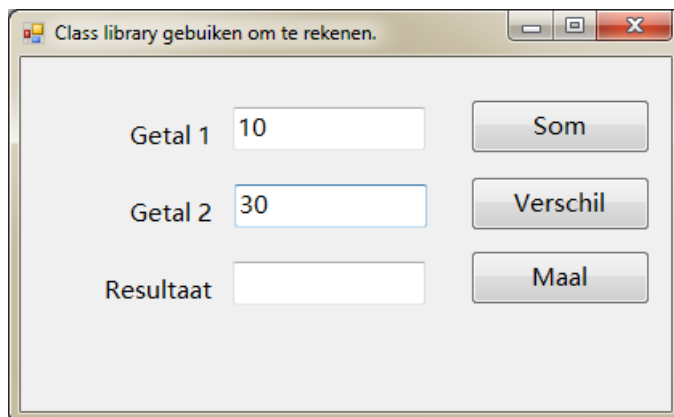
Daardoor wordt de debuginformatie niet in de library bewaard en wordt de DDL niet in de map Debug, maar in een nieuwe map Release bewaard.

Stap 4: Build Solution en onder de map Bin – Release wordt DDL toegevoegd.

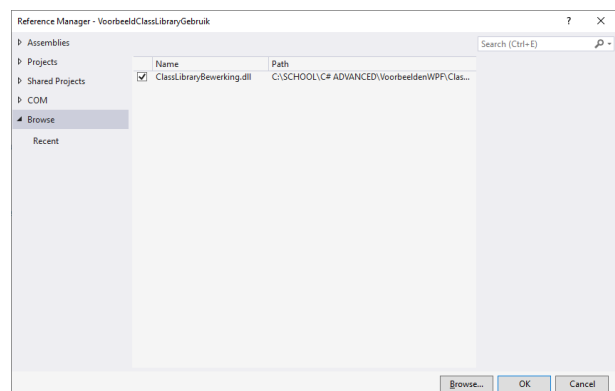
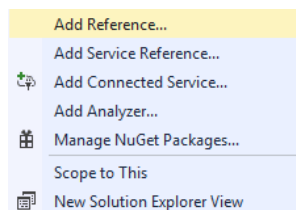


- **Class library gebruiken**

Stap 1: Maak een nieuwe Windows Application



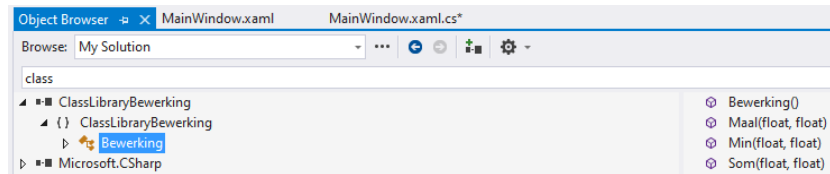
Stap 2: Voeg de Reference toe en browse naar je gemaakte class library.



Stap 3: Voeg toe: `using ClassLibraryBewerking;`

Stap 4: Maak gebruik van je Class Library.

Je kan in View -Object Browser je toegevoegde references bekijken.



```
private void BtnSom_Click(object sender, EventArgs e)
{
    Bewerking bw = new Bewerking();
    try
    {
        TxtResultaat.Text = $"{bw.Som(float.Parse(TxtGetal1.Text), float.Parse(TxtGetal2.Text))}";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void BtnVerschil_Click(object sender, EventArgs e)
{
    Bewerking bw = new Bewerking();
    try
    {
        TxtResultaat.Text = $"{bw.Min(float.Parse(TxtGetal1.Text), float.Parse(TxtGetal2.Text))}";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void BtnMaal_Click(object sender, EventArgs e)
{
    Bewerking bw = new Bewerking();
    try
    {
        TxtResultaat.Text = $"{bw.Maal(float.Parse(TxtGetal1.Text), float.Parse(TxtGetal2.Text))}";
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Hoofdstuk 5 LINQ

Language-Integrated Query (LINQ) bevat de query-mogelijkheden die rechtstreeks in de C#-taal gebruikt kunnen worden.

D.m.v. LINQ query's kan je bewerkingen op gegevensverzamelingen uitvoeren: gegevens selecteren en filteren, gegevens sorteren, (statistische) berekeningen op gegevens uitvoeren,...

Tot nu toe moest je, om met gegevens van een specifieke gegevensbron te werken, hiervoor telkens een specifieke taal gebruiken: Transact-SQL voor MS SQL Server, XPath of XQuery voor XML gegevens, geneste for/if statements voor arrays en collections, ...

Met LINQ wordt het allemaal eenvoudiger: je kan dezelfde C# syntax gebruiken voor elk type gegevensbron. Bovendien krijg je bij het schrijven van een LINQ query uitgebreide hulp (IntelliSense) en wordt de syntax van de code reeds gecontroleerd at compile-time.

De namespace **System.Linq** moet geïmporteerd zijn

Query-uitdrukkingen zijn geschreven in de *syntaxis van een declaratieve query*. Door de query-syntaxis te gebruiken, kunt u filteren, ordenen en groeperen op gegevensbronnen met een minimum aan code. Een *query* is een expressie die gegevens ophaalt uit een gegevensbron.

Alle LINQ-querybewerkingen bestaan uit drie verschillende acties:

- 1 Definiëren van de gegevensbron.
- 2 Opstellen van de query.
- 3 Uitvoeren van de query.

5.1 De query

De query specificeert welke informatie moet worden opgehaald uit de gegevensbron of bronnen. Optioneel specificeert een query ook hoe die informatie moet worden gesorteerd, gegroepeerd en gevormd voordat deze wordt geretourneerd. Een query wordt opgeslagen in een queryvariabele en geïnitieerd met een query-uitdrukking. Om het gemakkelijker te maken om query's te schrijven, heeft C# een nieuwe querysyntaxis geïntroduceerd.

Voorbeeld 1

```
// Drie delen van een LINQ Query:
// 1. Data source.
int[] getallen = new int[10] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// 2 Query maken met g als iteratievariabele.
var evenQuery1 =
    from g in getallen
    where g % 2 == 0
    select g;
```

```
// 3. Query uitvoeren.
foreach (int item in evenQuery1)
{
    Console.WriteLine($"{item} "); // Afdruk: 0 2 4 6 8
}
}
```

De *foreach* instructie is ook waar de queryresultaten worden opgehaald. In de vorige query bevat de iteratievariabele *g* elke waarde (één tegelijk) in de geretourneerde reeks.

De query in het vorige voorbeeld retourneert alle even getallen uit de gehele getallenrij. De query-uitdrukking bevat drie clauses: *from* , *where* en *select* . (Als u bekend bent met SQL, heeft u gemerkt dat de volgorde van de clauses is omgekeerd ten opzichte van de volgorde in SQL.) De *from* component geeft de gegevensbron aan, de *where* component past het filter toe en de *select* clause geeft het type de teruggestuurde elementen.

Het *var* sleutelwoord geeft de compiler opdracht om het type van een queryvariabele af te leiden door te kijken naar de gegevensbron die is opgegeven in de *from* component.

In plaats van de query volledig uit te werken kan je met behulp van Lambda expressie de query dadelijk integreren.

Voorbeeld 2

```
public static void Main(string[] args)
{
    // Drie delen van een LINQ Query:
    // 1. Data source.

    var getallen = new[] { 0, 2, 1, 4, 3, 5, 7, 6, 8, 9 };

    // 2 Query maken.
    Console.WriteLine("Getallen groter dan 3:");
    var getallenGroterDan3 = getallen.Where(getal => getal > 3);

    // ==== OF ====
    var getallenGroterDan3 =
        from g in getallen
        where g > 3
        select g;

    // 3. Query uitvoeren.
    foreach (var getal in getallenGroterDan3)
    {
        Console.WriteLine(getal);
    }

    // Query
    Console.WriteLine("Gesorteerde lijst:");
    var gesorteerdeGetallen = getallen.OrderBy(getal=>getal);

    // ==== OF ====
    var gesorteerdeGetallen =
        from sg in getallen
        orderby sg
        select sg;
}
```

```

// Uitvoering
foreach (var getal in gesorteerdeGetallen)
{
    Console.WriteLine(getal);
}

// Query en Uitvoering
Console.WriteLine("Aantal getallen groter dan 3:");
Console.WriteLine(getallen.Count(getal => getal > 3));
}

```

Als u wilt dat een query onmiddellijk wordt uitgevoerd en de resultaten in het cachegeheugen worden opgeslagen, kunt u de methoden `ToList` of `ToArray` gebruiken.

```

// Resultaten bewaren in List
List<int> numQuery2 =
    (from g in getallen
     where g % 2 == 0
     select g).ToList();

// Resultaten bewaren in Array
var numQuery3 =
    (from g in getallen
     where g % 2 == 0
     select g).ToArray();
OF
int[] numQuery5 =
    (from g in getallen
     where (g % 2) == 0
     select g).ToArray();

```

5.2 LINQ Query-bewerkingen

5.2.1 WHERE

Met de where-restrictor filter je de gegevens uit de data source.

```

int[] reeks = { 10, 20, 30, 40 };

// Alle getallen groter dan of gelijk aan 30.
var resultaat = from r in reeks
                where r >= 30
                select r;

// Alle getallen gelijk aan 10 of gelijk aan 40.
var resultaat = from r in reeks
                where r == 10 || r == 40
                select r;

// Alle getallen groter dan 10 en kleiner dan 40.
var resultaat = from r in reeks
                where r >= 10 && r < 40
                select r;

```

5.2.2 Orderby

Het is vaak handig om de geretourneerde gegevens te sorteren. De orderby clause zorgt ervoor dat de elementen in de reeks worden gesorteerd.

```
// Alle getallen worden in dalende volgorde gesorteerd.
var resultaat2 = from r in reeks
                 where r >= 10
                 orderby r descending // 40 30 20 10
                 select r;
```

Om de resultaten in stijgende volgorde te sorteren, wordt `orderby...ascending` gebruikt waarbij *ascending* weggelaten mag worden.

```
// Alle getallen worden in stijgende volgorde gesorteerd.
var resultaat2 = from r in reeks
                 where r >= 10
                 orderby r // 10 20 30 40
                 select r;
```

5.2.3 Distinct

Hiermee worden alle dubbele elementen in een verzameling verwijderd. Het retourneert alleen specifieke (of unieke) elementen.

```
// Declaratie van array.
int[] getallen = { 10, 20, 30, 20, 40 ,30 ,50 };

// Distinct.
var resultaat = getallen.Distinct();

// Afdruk.
foreach (int item in resultaat)
{
    Console.WriteLine(item); // Afdruk: 10 20 30 40 50
}
```

5.2.4 GROUPBY

De groepclausule retourneert nul of meer items die overeenkomen met de sleutelwaarde voor de groep.

```
int[] getallen = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// Group elements voor even getallen --> False: geeft 1 3 5 7 9
//                                     True: geeft 2 4 6 8

var resultaat = getallen.GroupBy(a => a % 2 == 0);

// === Of ===
var resultaat =
    from g2 in getallen
    group g2 by g2 % 2 == 0 into newg2
    select newg2;
```

```

// Item
foreach (var group in resultaat)
{
    // Display key for group.
    Console.WriteLine("Is even ? {0}:", group.Key);

    // Display values in group.
    foreach (var item in group)
    {
        Console.Write("{0} ", item);
    }

    // End line.
    Console.WriteLine();
}

// Afdruk
Is even ? False:
1 3 5 7 9
Is even ? True:
2 4 6 8

```

5.2.5 JOIN

Voorbeeld 1

```

// Array 1.
var getallen1 = new int[3] { 40, 20, 30 };

// Array 2.
var getallen2 = new int[3] { 50, 30, 20 };

var resultaat = from y in getallen1
                join x in getallen2 on (y + 10) equals x
                select y;

// Directe Join
var resultaat = getallen1.Join(getallen2,
    x => x + 10,
    y => y,
    (x, y) => x);

// Afdruk.
foreach (var item in resultaat)
{
    Console.WriteLine(item); // 40 20 want 40+10=50 en 20+10=30
}

```

Voorbeeld 2

```

class Klant
{
    public int Code { get; set; }
    public string Naam { get; set; }
}

```

```

class Order
{
    public int Productcode { get; set; }
    public string Product { get; set; }
}

static void Main(string[] args)
{
    // Klant
    var klanten = new Klant[]
    {
        new Klant{Code = 1055, Naam = "Karel"},
        new Klant{Code = 1086, Naam = "Danira"},
        new Klant{Code = 1007, Naam = "Lise"},
        new Klant{Code = 1038, Naam = "Francis"}
    };

    // Bestellingen
    var bestellingen = new Order[]
    {
        new Order{Productcode = 1055, Product = "Laptop HP"},
        new Order{Productcode = 1086, Product = "Laptop Acer"},
        new Order{Productcode = 1007, Product = "Laptop Lenovo"},
        new Order{Productcode = 1007, Product = "SSD 1TB"},
        new Order{Productcode = 1038, Product = "Laptop Apple"},
        new Order{Productcode = 1055, Product = "SSD 500GB"}
    };

    // Join
    var query = from c in klanten
                join o in bestellingen on c.Klantcode equals o.Productcode
                select new { c.Naam, o.Product };

    // Afdruk.
    foreach (var group in query)
    {
        Console.WriteLine($"{group.Naam} - {group.Product}");
    }

    // Afdruk.

    Karel - Laptop HP
    Karel - SSD 500GB
    Danira - Laptop Acer
    Lise - Laptop Lenovo
    Lise - SSD 1TB
    Francis - Laptop Apple
}

```


Directe uitvoering afdwingen

Query's die aggregatiefuncties uitvoeren over een reeks bronelementen, moeten eerst die elementen herhalen. Voorbeelden van dergelijke vragen zijn *Count*, *Max*, *Average* en *First*. Deze worden uitgevoerd zonder een expliciete foreach instructie, omdat de query zelf foreach moet gebruiken om een resultaat te retourneren. Merk ook op dat dit soort query's één waarde retourneert, geen *IEnumerable* verzameling. De volgende query retourneert een telling van de even nummers in de bronarray:

```
// Data source.
int[] scores = { 90, 71, 82, 93, 75, 82 };

// Query
int score80 = (from score in scores
               where score > 80
               select score).Count(); // Afdruk: 4

/// OF ///
var score80Plus = from score in scores
                  where score > 80
                  select score;

int aantal = score80Plus.Count(); // Afdruk: 4
```

5.2.6 Klassen

```
// Voorbeeld met klassen

class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public int Age { get; set; }
}

static void Main(string[] args)
{
    Student[] studenten = {
        new Student() { StudentID = 1, StudentName = "Joanna Pollers", Age = 28 },
        new Student() { StudentID = 2, StudentName = "Noah Janssens", Age = 31 },
        new Student() { StudentID = 3, StudentName = "Emma Vanschoenwinkel", Age = 35 },
        new Student() { StudentID = 4, StudentName = "Dries Gerris", Age = 30 },
        new Student() { StudentID = 5, StudentName = "Tiziana Delaet", Age = 41 },
        new Student() { StudentID = 6, StudentName = "Jonas Wellens", Age = 27 },
        new Student() { StudentID = 7, StudentName = "Joke Vermeersch", Age = 29 },
    };

    Console.WriteLine($"\\n\\n===== Studentengegevens =====");
    foreach (Student item in studenten)
    {
        Console.WriteLine($"Studentennummer: {item.StudentID} - Naam: {item.StudentName,-10} - leeftijd: {item.Age}");
    }

    // Zoekt de twintigers.
    Student[] student20 = studenten.Where(s => s.Age > 19 && s.Age < 30).ToArray();
}
```

```

Console.WriteLine($"\\n\\n===== Studenten tussen 20 en 30 =====");
foreach (Student item in student20)
{
    Console.WriteLine($"Studentennummer: {item.StudentID} - Naam: {item.StudentName,-10} -
leeftijd: {item.Age}");
}

// Zoekt Dries.
Student zoekNaam = studenten.Where(s => s.StudentName.Contains("Dries")).FirstOrDefault();
Console.WriteLine($"\\n\\n===== Studentengegevens van Dries =====");
Console.WriteLine($"Studentennummer: {zoekNaam.StudentID} - Naam: {zoekNaam.StudentName,-10} -
leeftijd: {zoekNaam.Age}");

// Zoekt studentennummer 5
Student student5 = studenten.Where(s => s.StudentID == 5).FirstOrDefault();
Console.WriteLine($"\\n\\n===== Studentengegevens met studentID=5 =====");
Console.WriteLine($"Studentennummer: {student5.StudentID} - Naam: {student5.StudentName,-10} -
leeftijd: {student5.Age}");

Console.ReadLine();
}

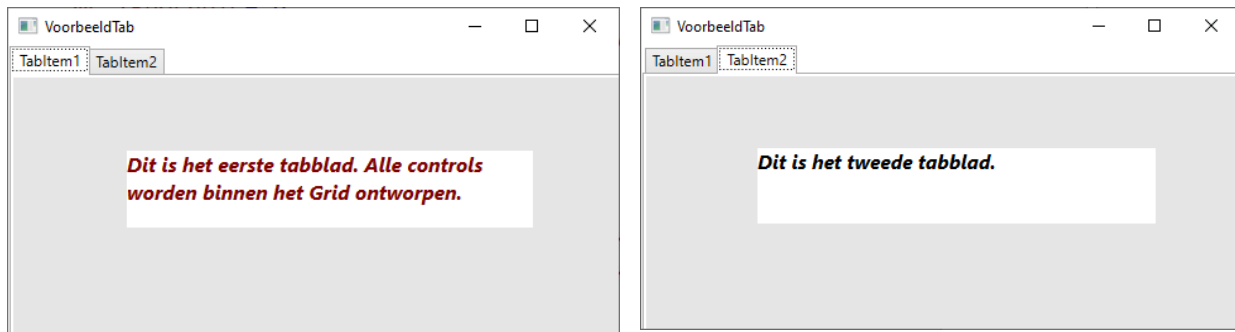
```

Hoofdstuk 6 WPF besturingselementen (vervolg)

6.1 Tabbladen (TabControl)

Het gebruik van tabbladen is een manier om grote hoeveelheden informatie op 1 formulier weer te geven. Via Ctrl+Tab kan je wisselen tussen de tabbladen.

- Eigenschappen: Via Items (Collection) kan je tabbladen toevoegen en de eigenschappen instellen.
TabStripPlacement: de tabknoppen bovenaan, links, rechts of onderaan zetten.
- Methods:
' Focus zetten op eerste tabblad
tabNaam.SelectedTab = TabPage1
tabNaam.SelectedIndex = 1

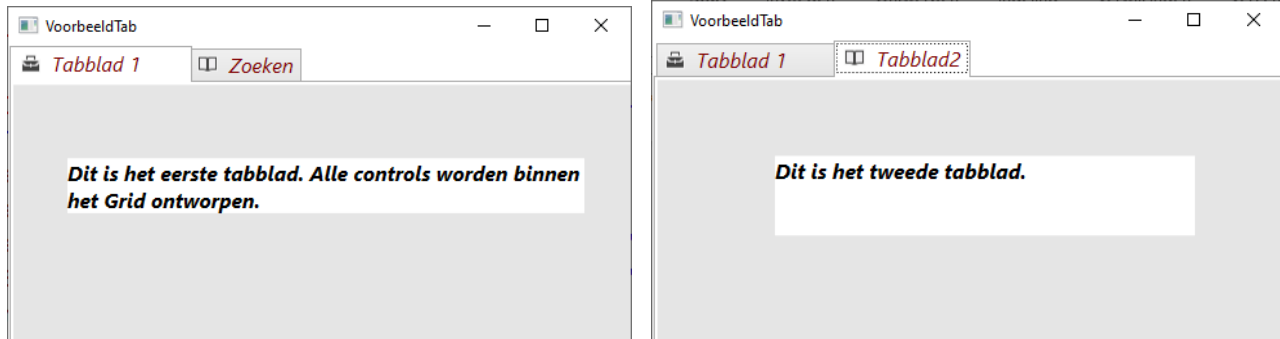


```
<TabControl HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
  <TabItem Header="TabItem1">
    <Grid Background="#FFE5E5E5">
      <TextBlock Text="Dit is het eerste tabblad. Alle controls worden binnen het Grid ontworpen." />
    </Grid>
  </TabItem>
  <TabItem Header="TabItem2">
    <Grid Background="#FFE5E5E5">
      <TextBlock Text="Dit is het tweede tabblad." />
    </Grid>
  </TabItem>
</TabControl>
```

Tip

Stel je de verticale en horizontale uitlijning in op Stretch zodat de tabcontrol mee verkleint of vergroot.

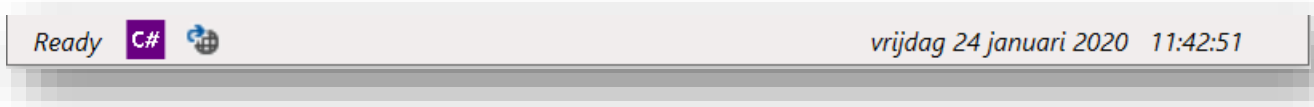
Je kan ook op de tabknoppen andere controls (Image, TextBlock,...) plaatsen. Gebruik hiervoor een StackPanel.



```
<TabControl HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
  <TabItem>
    <TabItem.Header>
      <StackPanel Orientation="Horizontal">
        <Image Source="Afbeeldingen/brief_Case.png" />
        <TextBlock Text=" Tabblad 1"/>
      </StackPanel>
    </TabItem.Header>
    <Grid Background="#FFE5E5E5">
      <TextBlock Text="Dit is het eerste tabblad. Alle controls worden binnen het Gridontworpen."/>
    </Grid>
  </TabItem>
  <TabItem>
    <TabItem.Header>
      <StackPanel Orientation="Horizontal">
        <Image Source="Afbeeldingen/book_Open.png" />
        <TextBlock Text=" Tabblad2"/>
      </StackPanel>
    </TabItem.Header>
    <Grid Background="#FFE5E5E5">
      <TextBlock Text="Dit is het tweede tabblad. "/>
    </Grid>
  </TabItem>
</TabControl>
```

6.2 Statusbalk (StatusBar)

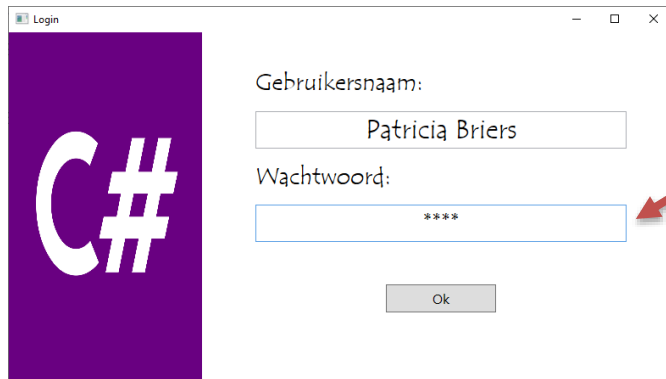
De **statusbalk** wordt gebruikt om een status van een toepassing weer te geven. De statusbalk bevindt zich doorgaans onderaan een venster. Je kan een statusbalk onderverdelen in verschillende controls en elk deel bevat bepaalde informatie over het programma.



```
<StatusBar DockPanel.Dock="Bottom" Height="30" VerticalAlignment="Top" >
    <TextBlock Text="    Ready" FontStyle="Italic" FontSize="16"/>
    <Image Source="Afbeeldingen/CSharpLogo.png" Margin="10,0,0,0"/>
    <Image Source="Afbeeldingen/FileFromWeb_6281.png" Margin="5,0,0,0"/>
    <TextBlock x:Name="TextBlockDatumTijd" Text="Datum en tijd" Margin="400,0,0,0"/>
</StatusBar>
```

6.3 PasswordBox

Met de PasswordBox kan je gemakkelijk paswoorden ingeven zoals een TextBox.



Eigenschappen:

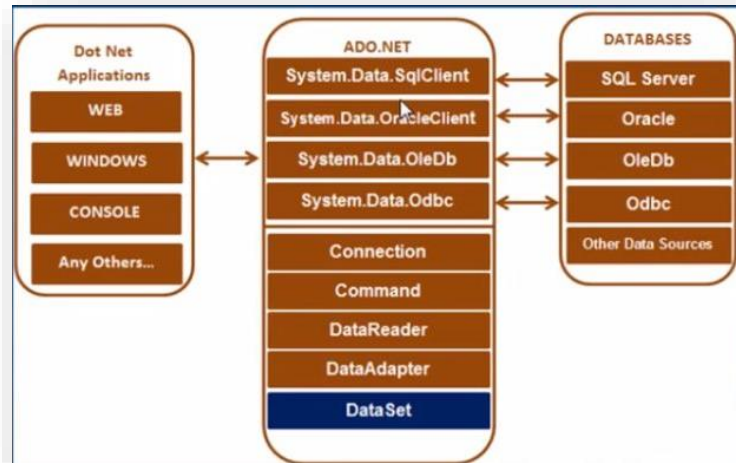
- PasswordChar: karakter dat weergegeven wordt tijdens de invoer.
- MaxLength: max. lengte dat ingevoerd kan worden.
- Password: de tekst dat ingevoerd is

```
<PasswordBox x:Name="PwdBoxLogin" MaxLength="4" PasswordChar="*" />
```

Hoofdstuk 7 ADO.NET

ADO.NET is ontworpen om ontwikkelaars te helpen efficiënt te werken met databanken. Voor dit onderdeel van de cursus heb je eveneens de SQL Server nodig.

ADO.NET is dus een set klassen (een framework) om te communiceren met gegevensbronnen zoals databases en XML-bestanden. ADO is de afkorting voor *ActiveX Data Objects*. Hiermee kunnen we verbinding maken met onderliggende gegevens of databases. Het heeft klassen en methoden om gegevens op te halen en te manipuleren.



ADO.Net objectmodel

De gegevenstoegangsklassen staan in de Namespace System.Data.

Het ADO.Net-objectmodel bestaat uit twee belangrijke componenten:

- Verbonden model (.NET Data Provider - een set componenten inclusief de objecten Connection, Command, DataReader en DataAdapter)
De *verbonden laag* omvat de klassen binnen ADO.Net waarbij verbinding met de database tijdens de applicatie verbonden blijft met de database.
- Niet-verbonden model (DataSet)
De *niet-verbonden laag* omvat de klassen binnen ADO.Net waarbij met een kopie van de datagegevens gewerkt wordt, hierbij is geen fysieke connectie meer met de database. De toepassing gebruikt tijdelijke gegevens aan de zijde van de toepassing die een DataSet bevat.

Disconnected klasse : is geworteld in de DataSet.

Klasse	Omschrijving
DataSet	Vertegenwoordigt een volledige verzameling tabellen, relaties en constraints (beperkingen). De namespaces System.Data.OleDb en System.Data.SqlClient hebben dit object van ADO.Net gemeen, waardoor het een kerncomponent van ADO.Net is. Niet alle programmeurs zijn voor het gebruik van DataSets.
DataTable	Vertegenwoordigt een gegevensbron die gegevens in rij- en kolomopmaak opslaat

DataColumn	Vertegenwoordigt een kolom in een DataTable
DataRow	Vertegenwoordigt een rij in een DataTable
Constraint	Vertegenwoordigt een opgelegde beperking in een DataTable
DataRelation	Vertegenwoordigt een relatie tussen twee velden van twee DataTables

Connected klasse: bestaat uit klassen die de Managed Providers omvatten.

Klasse	Omschrijving
DataAdapter	Vertegenwoordigt een databasequery of Stored Procedure die wordt gebruikt om het object DataSet te vullen, kan voorzien worden van een SelectCommand, InsertCommand, UpdateCommand en DeleteCommand.
DataReader	Snelle gegevenstoegang naar een database. Forward-only, read-only.
Command	Een SQL commando sturen naar de database
Connection	Verbinding met databank

Managed Providers

Managed Providers zijn een verzameling van klassen die in het *.Net Framework* de basis vormen van het ADO.Net programmeermodel. De Managed Data Providers bevatten klassen die voor het volgende kunnen gebruikt worden:

- toegang krijgen tot gegevens van **SQL Server**
- toegang krijgen tot andere OLE DB-providers (zoals **MS Access**, ...)
- toegang krijgen tot **Oracle** 8i en hoger.

Er zijn nog vele ander providers beschikbaar voor uiteenlopende databasesystemen.

De belangrijkste klasse om te werken met de SQL Server vind je hieronder.

ADO.Net klassen voor SQL Server - System.Data.SqlClient

Klasse	Omschrijving
SqlConnection	Vertegenwoordigt een open verbinding naar een SQL-server gegevensbron
SqlDataAdapter	Vertegenwoordigt een verzameling gegevensopdrachten en een databaseverbinding om het ADO.Net-object DataSet te vullen.
SqlCommand	Vertegenwoordigt een T-SQL instructie of een Stored Procedure die SQL Server uitvoert.
SqlParameter	Wordt gebruikt om parameters mee te geven aan het object SqlCommand.
SqlError	Verzamelt informatie over foutmeldingen die een ADO.Net toepassing tegenkomt.

Belangrijke klassen in ADO.NET zijn:

- Connection Class
- Command Class
- DataReader Class
- DataAdapter Class
- DataSet Class

Laten we deze klasse eens wat verder bestuderen.

7.2 Connection

Een SqlConnection-object vertelt ADO.Net met welke server en database je werkt en regelt de communicatie tussen de toepassing en SQL-server.

De informatie die dient om toegang te krijgen tot SQL-server (servernaam, databasenaam, gebruikersnaam, paswoord, ...) zit vervat in een tekenreeks die we de ConnectionString noemen. Om concreet te verbinden met een SQL Server database hebben we een dus een ConnectionString nodig (voor OLE DB was dit ook het geval).

String-onderdeel	Omschrijving
Data Source of Server	Identificeert de server: servernaam, IP-adres of (local) voor lokale server.
Initial Catalog of Database	Naam van de database op de server
Integrated Security	Stel in op SSPI (Security Support Provider Interface) om gebruik te maken van je Windows gebruikersgegevens
User ID (uid)	Gebruikersnaam
Password (pwd)	Paswoord voor de gebruikte User ID
AttachDbFilename	Voor SQL Server Express: pad naar het bestand, je kan ook met de aanduiding DataDirectory werken. Dan wordt de datamap, of standaard de root-map van het project aangewend.

De SqlConnection ConnectionString bevat volgende onderdelen: server=NaamPC of server=.\sqlexpress

Als je de verbinding maakt naar een echte SqlServer gebruik je de eerste versie (server=NaamPc) en vervang je NaamPC door de naam van de computer waarop de SqlServer draait.

Als je de verbinding maakt naar SqlExpress op je computer, gebruik je de tweede versie (server=.\sqlexpress). In de voorbeeldcode maak je een verbinding naar een SqlExpress op de eigen computer.

Als je de eerste versie gebruikt (verbinding met een echte SqlServer, en de SqlServer draait op dezelfde computer als jouw applicatie, mag je ook schrijven: server=(local)

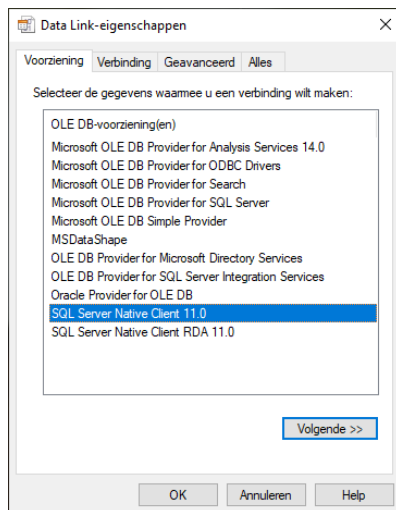
Voorbeeld 1

```
SqlConnection connect = new SqlConnection(@"Integrated Security = SSPI; Persist Security Info = False; Initial Catalog = Med; Data Source = PATRICIA_PC\SQLEXPRESS");
```


Een leuke manier om een connectiestring te zoeken is via een UDL file (bestand dat de connectie-informatie opslaat).

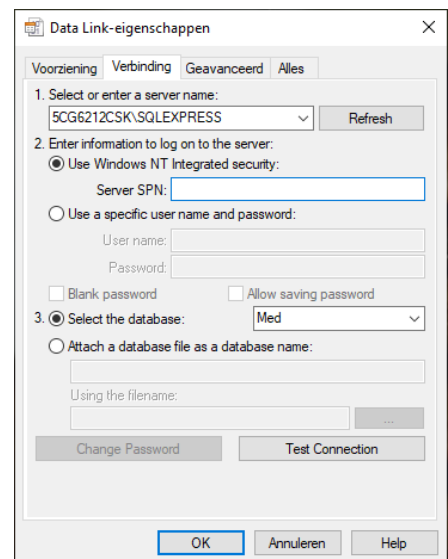
Maak een nieuw tekstbestand aan en wijzig de extensie naar UDL. Open je bestand en vul de Data Link-eigenschappen aan. Zorg dat je SQL Server actief is.

Stap 1 Kies je provider. Deze DLL zorgt ervoor dat we met de juiste database kunnen connecteren. We kiezen voor SQL Server Native Client. Je kan hier ook kiezen voor ODBC maar dit is geen goede keuze omdat een ODBC provider geschreven is om met meerdere databases te kunnen connecteren is deze ook een stuk trager.



Stap 2 Op het tweede tabblad vul je de server en de datasource in. In het volgende scherm van onze wizard moeten er opnieuw instellingen gedaan worden. Wanneer je niet weet hoe je SQL Server heet kan je best de MMS opstarten en daar de naam van de SQL server noteren.

In het tweede deel van het scherm ga je de rechten bepalen. Indien je SQL server is geconfigureerd met geïntegreerde security settings van (dus Windows inloggegevens). Dit is de meest beveiligde manier om in te loggen. Indien jullie server niet zo geconfigureerd is moet je hier de login en paswoord meegeven.



Vul vervolgens de database waarmee je verbinding wilt maken en dan kan je de connection eventueel testen.

Stap 3 Open deze udl file in een kladblok en je vindt je connectiegegevens terug.

[oledb]

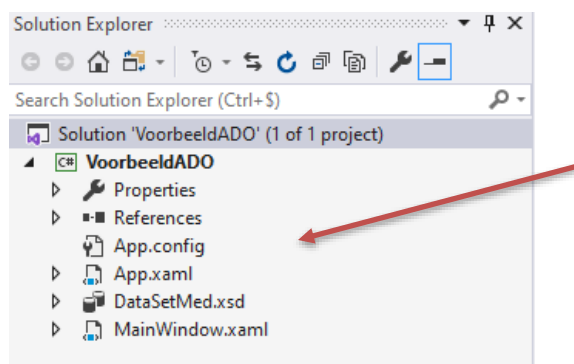
; Everything after this line is an OLE DB initstring

Provider=SQLNCLI11.1;Integrated Security=SSPI;Persist Security Info=False;User ID="";Initial Catalog=Med;Data Source=5CG6212CSK\SQLEXPRESS;Initial File Name="";Server SPN=""

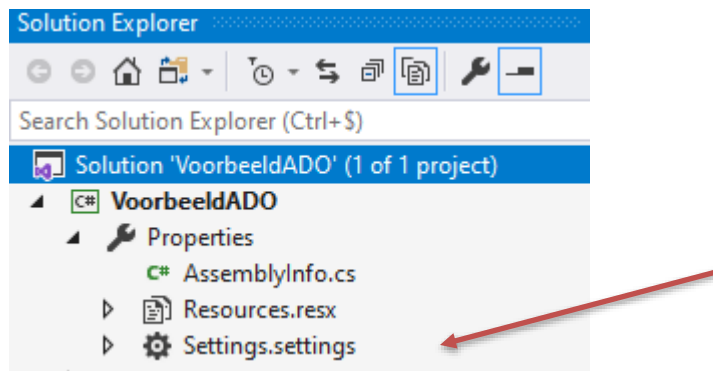
Je kan nu je connectiestring opnemen in je code. Indien een tabel verandert in de server moet je wel opnieuw een udl file aanmaken.

Voorbeeld 2

Je kan de connectiestring ook opnemen in App.Config in de Solution Explorer. Indien het niet aanwezig is, voeg je via ADD – New Item – Config file toe.



Open vervolgens de properties van je project en ga naar je settings.



Pas je setting aan (kan ook de rechter ellipsknop gebruiken voor de instelling van je value) en kijk daarna naar de toevoeging in je config file.

	Name	Type	Scope	Value
▶	CNstr	(Connectio...	Application	Integrated Security=SSPI;Persist Security Info=False;User ID="";Initial Catalog=Med;Data Source=5CG6212CSK\SQLEXPRESS

```
<connectionStrings>
    <add name="VoorbeeldADO.Properties.Settings.CNstr" connectionString="Integrated
Security=SSPI;Persist Security Info=False;User ID="";Initial Catalog=Med;Data
Source=5CG6212CSK\SQLEXPRESS" />
</connectionStrings>
```

Toepassing uittesten connectiestring

```
private void BtnConnectieTest_Click(object sender, RoutedEventArgs e)
{
    // Haalt de connectiestring uit de settings.
    string cn = Properties.Settings.Default.CNstr.ToString();

    SqlConnection sqlcn = new SqlConnection(cn);

    sqlcn.Open();

    if (sqlcn.State == System.Data.ConnectionState.Open)
    {
        MessageBox.Show("De connection is open.");
    }
    else
    {
        MessageBox.Show("De connection is NIET open.");
    }
}
```

7.3 SqlCommando

SqlCommand wordt gebruikt om gegevens uit de databank op te vragen. Je kan SQL-query's uitvoeren om gegevens in een DataSet of een DataReader te retourneren. De SqlCommand omvat Select, Update, Delete en Update.

Voorbeeld 1

```
private void BtnSqlCommand_Click(object sender, RoutedEventArgs e)
{
    // === 1ste manier ===
    SqlConnection connect = new SqlConnection(@" Integrated Security = SSPI; Persist Security
Info = False; Initial Catalog = Med; Data Source = 5CG6212CSK\SQLEXPRESS");

    SqlConnection cn = new SqlConnection();
    SqlCommand cmd = new SqlCommand();

    cmd.Connection = cn;
    string query = "select * from medewerkers";
    cmd.CommandType = CommandType.Text;
    cmd.CommandText = query;
    TxtResultaat.Text = cmd.CommandText; // Geeft: select * from medewerkers
}
```

Voorbeeld 2

```
private void BtnSqlCommand_Click(object sender, RoutedEventArgs e)
{
    // === 2de manier ===
    SqlConnection connect = new SqlConnection(@" Integrated Security = SSPI; Persist Security
Info = False; Initial Catalog = Med; Data Source = 5CG6212CSK\SQLEXPRESS");
    SqlConnection cn = new SqlConnection();
    string query = "select * from medewerkers";
    SqlCommand cmd = new SqlCommand(query);
    cmd.Connection = cn;
}
```

Voorbeeld 3

```
private void BtnSqlCommand_Click(object sender, RoutedEventArgs e)
{
    // === 3de manier ===
    SqlConnection connect = new SqlConnection(@" Integrated Security = SSPI; Persist Security
Info = False; Initial Catalog = Med; Data Source = 5CG6212CSK\SQLEXPRESS");
    SqlConnection cn = new SqlConnection();
    string query = "select * from medewerkers";
    SqlCommand cmd = new SqlCommand(query, cn);
}
```

Enkele waarde opvragen

- Maak een SqlCommand aan met de gewenste query
SqlCommand cmd = new SqlCommand("select max(mnr) from medewerkers", conn);
- Vraag de waarde op met de methode ExecuteScalar
short max = (short)cmd.ExecuteScalar();
Het gegevenstype short (Int16) uit C# komt overeen met smallint uit SQL-Server.

Data opvragen

- Maak een SqlCommand aan met de gewenste query.
SqlCommand cmd = new SqlCommand("select * from medewerkers", conn);
- Met de methode ExecuteReader haal je de resultaten op
SqlDataReader rdr = cmd.ExecuteReader();

Data invoegen

- Nieuw medewerker toevoegen
string insertString = @" insert into medewerkers(mnr, naam, voornaam)
values (7000, 'Briers', 'Patricia')";
SqlCommand cmd = new SqlCommand(insertString, conn);
- Uitvoeren van commando. Er worden geen rijen opgevraagd.
cmd.ExecuteNonQuery();

Data updaten

- Wijzig gegevens
string updateString = @" update medewerkers
set naam = 'Craig' , voornaam = 'Daniel'
where mnr = 7000";
SqlCommand cmd = new SqlCommand(updateString, conn);

```
cmd.ExecuteNonQuery();
```

Data verwijderen

- Gegevens verwijderen
string deleteString = @"delete from medewerkers where mnr = 7000";

```
SqlCommand cmd = new SqlCommand(deleteString, conn);  
cmd.ExecuteNonQuery();
```

7.4 DataReader

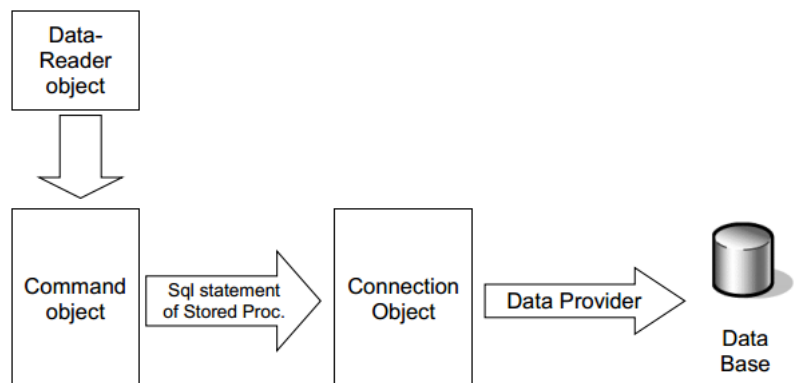
De klasse DataReader wordt gebruikt om uit een SQL select statement of een stored procedure data te lezen die meerdere rijen en/of kolommen omvat. Deze klasse gebruik je op het Command-object met de method ExecuteReader(). Deze methode geeft een DataReader object.

Een DataReader heeft volgende eigenschappen:

- Forward-only: je kan de rijen enkel van voor naar achter lezen. Je kunt niet terugkeren naar vorige rijen.
- Read-only: Je kan de rijen enkel lezen, maar niet wijzigen.
- De DataReader houdt slechts één rij (de rij die je nu leest) in het geheugen. Hij is dus spaarzaam op geheugengebruik.
- De DataReader is een performante manier om veel gegevens uit een database te lezen.

Er wordt dus geen DataSet gemaakt en er bevindt zich dus één rij informatie uit de databank tegelijk in het geheugen. Dit maakt de DataReader efficiënt bij het retourneren van grote hoeveelheden gegevens.

Als u echter een schema moet manipuleren of een aantal geavanceerde weergavefuncties, zoals automatisch oproepen, moet u een DataAdapter en DataSet gebruiken.



Belangrijkste **eigenschappen** van de DataReader:

- **FieldCount** het aantal kolommen in de opgegeven query
- **Item("kolomnaam")** geeft de waarde van die kolom uit de huidige opgehaald rij
- **Item(kolomnr)** geeft de waarde van de kolom met rangorde van kolomnr (Item(0) geeft inhoud van de eerste kolom uit de query).
- **HasRows** bevat True als je DataReader minstens één rij bevat, anders False.

Belangrijkste **methods** van DataReader:

- **Read()** Leest een rij uit de DataReader. Is true al er een rij is gelezen, in het andere geval false.
- **GetString(kolomnr)**, **GetInt32(kolomnr)**, **GetDecimal(kolomnr)**, ...

Leest de inhoud van de kolom uit de huidige rij. Deze Get...() functies zijn performanter dan de eigenschap Item.

- **IsNull(kolomnr)** geeft true als huidige rij leeg is (een NULL waarde), anders false. Het is belangrijk deze functie te gebruiken bij kolommen die leeg kunnen zijn want het uitvoeren van één van de Get...() functies op een lege kolom veroorzaakt een Exception.
- **GetOrdinal("kolomnaam")** geeft het kolomnummer van de opgevraagde kolom uit de query. Vb. de waarde 0 van 1^{ste} kolom uit query, waarde 1 uit 2^{de} kolom van query,...
- **GetName(kolomnr)** geeft de kolomnaam van de kolom met het specifiek kolomnummer.
- **Close()** sluit de DataReader. Dit is noodzakelijk want een DataReader blijft verbonden met het Connection object exclusief (voor zichzelf alleen) tot je de DataReader expliciet sluit met Close(). Als je dit vergeet te doen, kan je op hetzelfde Connection object geen andere handelingen meer uitvoeren. SqlServer bevat echter wel MARS (Multiple Active Result Sets) waarmee je meerdere DataReaders tegelijk kan openen. Stel dan bij de connectiestring: MultipleActiveResultSets=true. Bij het lezen uit één database werk je best met een using-statement zodat de connection automatisch sluit bij het einde van het using-statement (zie voorbeelden hieronder).

7.4.1 ExecuteReader

```
private void BtnExecuteReader_Click(object sender, RoutedEventArgs e)
{
    string con =@" Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = SCG6212CSK\SQLEXPRESS";

    string queryString = "select * from medewerkers";

    using (SqlConnection cn = new SqlConnection(con))
    {
        cn.Open();

        SqlCommand command = new SqlCommand(queryString, cn);
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            TxtResultaat.Text += $"{reader[0]} {reader[1]} {reader[2]} \r\n";
            // === OF ===
            TxtResultaat.Text += $"{reader.GetInt16(0)} {reader.GetString(1)}
{reader.GetString(2)}\r\n";
        }
        reader.Close();
    }
}
```



7369	CASPERS	JANA
7499	ALLARD	NELE
7521	DEFOUR	THOMAS
7566	JACOBS	EMMA
7654	MARTENS	RAF
7698	BRIERS	ANDREA
7782	CLERCKX	AN
7788	SWINNEN	CHRIS
7839	DE KONING	LIEVE
7844	DEN RUYTER	JOACHIM
7876	ELEUTEN	TOM

7.4.2 ExecuteScalar

Geeft de eerste kolom van de eerste rij uit de resultaatquery en een null reference als er niets opgehaald wordt.

De methode ExecuteScalar wordt gebruikt om een enkele waarde (bijvoorbeeld een verzamelwaarde) uit een database op te halen. Dit vereist minder code dan met de methode ExecuteReader om dan vervolgens aan de SqlDataReader door te geven.

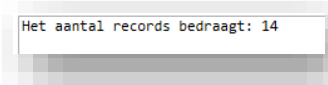
Een typische ExecuteScalar- query kan worden opgemaakt zoals in het volgende C # voorbeeld:

```
string query = "select count(mnr) from medewerkers";
```

```
private void BtnExecuteScalar_Click(object sender, RoutedEventArgs e)
{
    string con = @" Integrated Security = SSPI; Persist Security Info = False; Initial Catalog = Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    string query = "select count(mnr) from medewerkers";

    using (SqlConnection cn = new SqlConnection(con))
    {
        cn.Open();
        SqlCommand cmd = new SqlCommand(query, cn);

        // Aantal rijen tellen.
        int res = (int)cmd.ExecuteScalar();
        TxtResultaat.Text = $"Het aantal records bedraagt: {res}";
    }
}
```



7.4.3 ExecuteNonQuery

Geeft het aantal opgehaalde rijen uit de resultaatquery.

De methode ExecuteNonQuery wordt gebruikt om databaseobjecten zoals tabellen te maken of om de gegevens in een database te wijzigen (zonder het gebruik van DataSet) door UPDATE-, INSERT- of DELETE-instructies uit te voeren.

Voor UPDATE-, INSERT- en DELETE-instructies is de retourwaarde het aantal rijen waarop de opdracht betrekking heeft. Voor alle andere gevallen is de retourwaarde 0.

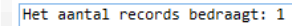
```
private void BtnExecuteNonQuery_Click(object sender, RoutedEventArgs e)
{
    string con = @" Integrated Security = SSPI; Persist Security Info = False; Initial Catalog = Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    string query = "update medewerkers set functie='TRAINER' where mnr=7369";
    using (SqlConnection cn = new SqlConnection(con))
    {
        cn.Open();
    }
}
```

```

SqlCommand cmd = new SqlCommand(query, cn);

// Aantal rijen tellen.
int res = (int)cmd.ExecuteNonQuery();
TxtResultaat.Text = $"Het aantal records bedraagt: {res}";
}
}

```



7.4.4 SqlParameter

Je kan ook parameters gebruiken om variabele waarde door te geven aan SqlCommand. Definieer de *ParameterName* (naam om te gebruiken) en vervolgens *Value* (de waarde waarmee je wenst te werken).

Voorbeeld 1 (query: medewerkers met een hoger salaris dan 3500)

```

private void BtnExecuteNQParameters_Click(object sender, RoutedEventArgs e)
{
    string con = @"Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    string query = "select voorn,naam,maandsal from medewerkers where maandsal >= @salaris";

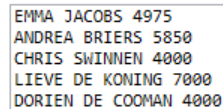
    // Parameter instellen.
    SqlParameter mnr = new SqlParameter();
    mnr.ParameterName = "@salaris";
    mnr.Value = "3500";

    using (SqlConnection cn = new SqlConnection(con))
    {
        // === 1ste manier Parameters doorgeven. ===
        cn.Open();

        SqlCommand cmd = new SqlCommand(query, cn);
        cmd.Parameters.Add(mnr);

        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())
        {
            TxtResultaat.Text += $"{reader[0]} {reader[1]} {reader[2]}\r\n";
        }
        reader.Close();
    }
}

```



Voorbeeld 2 (query: cursussen van type = DSG)

```

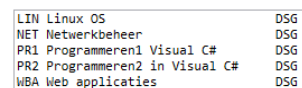
private void BtnExecuteNQParameters_Click(object sender, RoutedEventArgs e)
{
    string con = @"Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    string query = "select * from cursussen where type = @type";

    using (SqlConnection cn = new SqlConnection(con))
    {
        cn.Open();

        SqlCommand cmd = new SqlCommand(query, cn);
        cmd.Parameters.AddWithValue("@type", "DSG");

        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())

```




```

    {
        TxtResultaat.Text += $"{reader[0]} {reader[1],-30} {reader[2]}\r\n";
    }
    reader.Close();
}
}

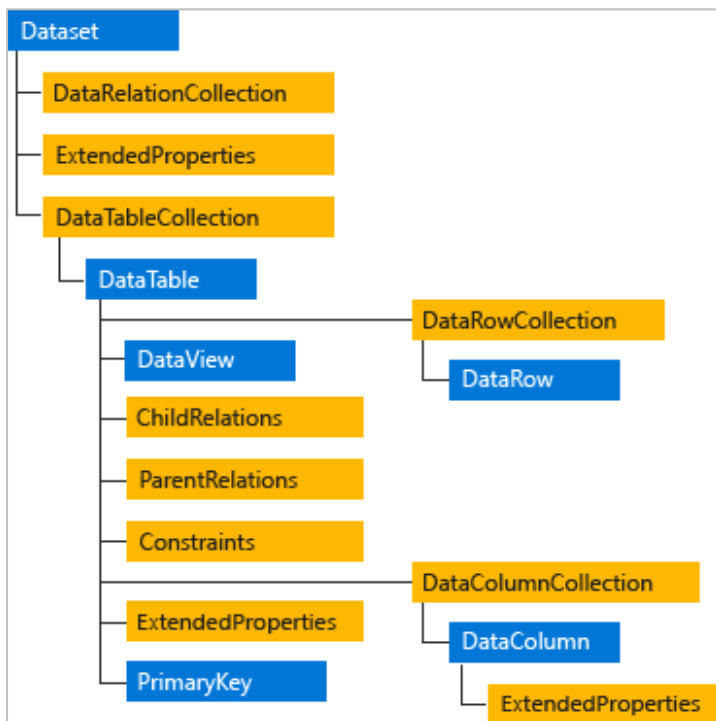
```

7.5 Dataset

Een van de belangrijkste technologie in de disconnected architecture is Dataset. Je kan hiermee gegevens uit je gerelateerde databank binnen halen en ermee werken in je toepassing. Het kan de performance aanzienlijk versnellen. Het gevaar met deze technologie is dat je echt alles in de dataset steekt vanwege het gebruiksgemak.

De niet-verbonden datatoegangsklassen worden uitgevoerd binnen de cliënt-toepassing en kunnen gebruikt worden zonder te verbinden met de database. Deze klassen kunnen ook nuttig zijn voor intern gegevensbeheer.

Klassenhierarchie



Voorbeeld

```
private void BtnDatasetCreate_Click(object sender, RoutedEventArgs e)
{
    DataSet ds = new DataSet();
    DataTable dt = new DataTable("Stud");

    // Kolommen declareren
    DataColumn dcStudId = new DataColumn("StudId", typeof(int));
    DataColumn dcNaam = new DataColumn("Naam", typeof(string));
    DataColumn dcOpleiding = new DataColumn("Opleiding", typeof(string));
    DataColumn dcGbDatum = new DataColumn("GbDatum", typeof(DateTime));

    // Kolommen toevoegen aan DataColumnCollection.
    dt.Columns.Add(dcStudId);
    dt.Columns.Add(dcNaam);
    dt.Columns.Add(dcOpleiding);
    dt.Columns.Add(dcGbDatum);

    // Extra kolom met verschillende kenmerken.
    DataColumn dcTel;
    dcTel = new DataColumn();
    dcTel.ColumnName = "Telefoon";
    dcTel.DataType = typeof(string);
    dcTel.Unique = true;
    dcTel.ReadOnly = false;
    dt.Columns.Add(dcTel);

    // Primaire sleutel aanduiden.
    DataColumn[] sleutel = { dcStudId }; // kunnen meerdere kolommen zijn => dus type array
    dt.PrimaryKey = sleutel;

    // Unieke constraint voor naam toepassen.
    UniqueConstraint uniek = new UniqueConstraint(dcNaam);

    // Rijen toevoegen aan DataTable
    dt.Rows.Add(new object[] { 1, "Kristof Palmaers", "Graduaat Programmeren",
"17/08/1980", "011775100" });
    dt.Rows.Add(new object[] { 2, "Paul Dox", "Graduaat Digitale vormgeving",
"17/03/1972", "011775101" });
    dt.Rows.Add(new object[] { 3, "Patricia Briers", "Graduaat Systemen en netwerken",
"17/10/1971", "011775102" });

    // Rij toevoegen met DataRow
    DataRow rij = dt.NewRow();
    rij[dcStudId] = 4;
    rij[dcNaam] = "Ann Das";
    rij[dcOpleiding] = "Graduaat Systemen en netwerken";
    rij[dcGbDatum] = "1990/12/15 18:35:10";
    rij[dcTel] = "011775103";
    dt.Rows.Add(rij);

    // Tabel toevoegen aan dataset
    ds.Tables.Add(dt);

    // Afdruk in component DataGrid
    DataView dv = ds.Tables["Stud"].DefaultView;
    DataGridStud.ItemsSource = dv;
}
```

StudId	Naam	Opleiding	GbDatum	Telefoon
1	Kristof Palmaers	Graduaat Programmeren	8/17/1980 12:00:00 AM	011775100
2	Paul Dox	Graduaat Digitale vormgeving	3/17/1972 12:00:00 AM	011775101
3	Patricia Briers	Graduaat Systemen en netwerken	10/17/1971 12:00:00 AM	011775102
4	Ann Das	Graduaat Systemen en netwerken	12/15/1990 6:35:10 PM	011775103

7.5.1 DataTable

Een DataTable is zoals een tabel in een database waar je kolommen (DataColumn) en rijen (DataRow) aan kan toevoegen.

```
DataTable dt = new DataTable("Stud");
```

- **DataColumn**

Je kan een DataColumn best vergelijken met een kolom in een tabel.

```
// Kolommen declareren (gegevens in constructor)
DataColumn dcStudId = new DataColumn("StudId", typeof(int));
DataColumn dcNaam = new DataColumn("Naam", typeof(string));
DataColumn dcGbDatum = new DataColumn("GbDatum", typeof(DateTime));

// Extra kolom met verschillende kenmerken toegekend aan de eigenschappen.
DataColumn dcTel;
dcTel = new DataColumn();
dcTel.ColumnName = "Telefoon";
dcTel.DataType = typeof(string);
dcTel.Unique = true;
dcTel.ReadOnly = false;
dt.Columns.Add(dcTel);

// Kolommen toevoegen aan DataColumnCollection.
dt.Columns.Add(dcStudId);
```

- **Primaire sleutelkolommen maken**

Aangezien een primaire sleutel van een tabel kan bestaan uit meerder kolommen, wordt de eigenschap PrimaryKey ingesteld op een Array van DataColumnns.

```
// Primaire sleutel toevoegen.
dt.PrimaryKey = new DataColumn[] {studId};
```

- **DataRow**

Het DataTable-object bevat een eigenschap Rows, die een DataRow-collectie bevat. Er zijn verschillende manieren om gegevens toe te voegen aan de Rows-collectie:

- De Rows-collectie heeft een methode Add die een DataRow ontvangt. Deze methode kan ook een Array van objecten ontvangen.
- De DataTable heeft een methode LoadDataRow. Deze methode kan je gebruiken om bestaande rijen te wijzigen of om nieuwe rijen toe te voegen. De methode LoadDataRow ontvangt een Array van objecten en een LoadOption enumeratie-waarde.

Je kan een DataRow best vergelijken met een rij in een tabel.

```
// Rijen toevoegen aan DataTable
dt.Rows.Add(new object[] { 1, "Kristof Palmaers", "Graduaat Programmeren",
"17/08/1980", "011775100" });
dt.Rows.Add(new object[] { 2, "Paul Dox", "Graduaat Digitale vormgeving",
"17/03/1972", "011775101" });
```

```
dt.Rows.Add(new object[] { 3, "Patricia Briers", "Graduaat Systemen en netwerken", "17/10/1971",
"011775102" });

// Rij toevoegen met DataRow
DataRow rij = dt.NewRow();
rij[dcStudId] = 4;
rij[dcNaam] = "Ann Das";
rij[dcOpleiding] = "Graduaat Systemen en netwerken";
rij[dcGbDatum] = "1990/12/15 18:35:10";
rij[dcTel] = "011775103";
dt.Rows.Add(rij);
```

Een DataRow kent verschillende RowStates:

RowState waarde	Beschrijving
Detached	De DataRow werd aangemaakt maar niet toegevoegd aan een DataTable
Added	De DataRow werd aangemaakt en toegevoegd aan een DataTable
Unchanged	De DataRow is niet veranderd sinds de laatste aanroep van de methode AcceptChanges. Wanneer de methode AcceptChanges wordt aangeroepen verandert de State van de DataRow in Unchanged.
Modified	De DataRow is bewerkt sinds de laatste aanroep van AcceptChanges
Deleted	De DataRow is verwijderd met de methode Delete van de DataRow

Uitvoer als XML

Je kan een DataTable heel eenvoudig als XML uitvoeren:

```
dt.WriteXml(@"..\stud.xml");
```

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Stud>
    <StudId>1</StudId>
    <Naam>Kristof Palmaers</Naam>
    <Opleiding>Graduaat Programmeren</Opleiding>
    <GbDatum>1980-08-17T00:00:00+02:00</GbDatum>
    <Telefoon>011775100</Telefoon>
  </Stud>
  <Stud>
    <StudId>2</StudId>
    <Naam>Paul Dox</Naam>
    <Opleiding>Graduaat Digitale vormgeving</Opleiding>
    <GbDatum>1972-03-17T00:00:00+01:00</GbDatum>
    <Telefoon>011775101</Telefoon>
  </Stud>
  <Stud>
    <StudId>3</StudId>
    <Naam>Patricia Briers</Naam>
    <Opleiding>Graduaat Systemen en netwerken</Opleiding>
    <GbDatum>1971-10-17T00:00:00+02:00</GbDatum>
    <Telefoon>011775102</Telefoon>
  </Stud>
  <Stud>
    <StudId>4</StudId>
    <Naam>Ann Das</Naam>
    <Opleiding>Graduaat Systemen en netwerken</Opleiding>
    <GbDatum>1990-12-15T18:35:10+01:00</GbDatum>
    <Telefoon>011775103</Telefoon>
  </Stud>
</NewDataSet>
```

7.6 SqlDataAdapter

De SqlDataAdapter werkt als een brug met behulp van Fill tussen een DataSet en SQL Server voor het ophalen en opslaan van gegevens.

Afhankelijk van het type wijziging wordt

Insert , Update of Delete uitgevoerd om de gewijzigde rij door te geven aan de databank.

Wanneer de SqlDataAdapter een DataSet vult, worden de tabellen en kolommen gemaakt als deze nog niet bestaan. Primaire sleutelinformatie wordt echter niet opgenomen behalve als de eigenschap MissingSchemaAction is ingesteld op AddWithKey .

DataAdapter voert vijf volgende stappen uit:

1. Create/open the connection
2. Haal de gegevens op volgens de opgegeven opdracht
3. Genereer XML-gegevensbestand
4. Fill data into DataSet.
5. Close connection.

```

DataTable tbl = null;
//string controlValue = string.Empty;
string con = @" Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";

using (SqlConnection cnn = new SqlConnection(con))
{
    cnn.Open();

    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cnn;
    cmd.CommandText = "select * from medewerkers";

    SqlDataAdapter da = new SqlDataAdapter(cmd);

    tbl = new DataTable();
    da.Fill(tbl);
}

```

7.6.1 DataView

Vertegenwoordigt een aangepaste weergave van een DataTable voor sorteren, filteren, zoeken, bewerken en navigeren in de databank. Wijzigingen in de gegevens van de DataView hebben invloed op de DataTable en wijzigingen in de gegevens van de DataTable zijn van invloed op alle bijbehorende DataView 's.

De DataTable heeft ook een eigenschap `DefaultView`. Hiermee wordt de standaardweergave voor de tabel geretourneerd. Als u bijvoorbeeld een aangepaste weergave op de tabel wilt maken, stelt u de `RowFilter` in op de DataView die wordt geretourneerd door de standaardweergave.

Voorbeeld 1

```

private void BtnDataView_Click(object sender, RoutedEventArgs e)
{
    DataTable tbl = null;
    //string controlValue = string.Empty;
    string con = @" Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    using (SqlConnection cnn = new SqlConnection(con))
    {
        cnn.Open();

        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cnn;
        cmd.CommandText = "select * from medewerkers";

        SqlDataAdapter da = new SqlDataAdapter(cmd);

        tbl = new DataTable();
        da.Fill(tbl);
    }

    DataView dv = new DataView(tbl, "Functie='MANAGER'", "functie", DataViewRowState.Unchanged);
    LbxFuncties.ItemsSource = dv;
    LbxFuncties.DisplayMemberPath = "naam";
}

```

JACOBS
BRIERS
CLERCKX

Voorbeeld 2

```
private void BtnDataView_Click(object sender, RoutedEventArgs e)
{
    DataTable tbl = null;
    //string controlValue = string.Empty;
    string con = @"Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    using (SqlConnection cnn = new SqlConnection(con))
    {
        cnn.Open();


        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cnn;
        cmd.CommandText = "select * from medewerkers";

        SqlDataAdapter da = new SqlDataAdapter(cmd);

        tbl = new DataTable();
        da.Fill(tbl);
    }

    DataView dv2 = new DataView(tbl);
    dv2.RowFilter = "naam like '%E%'";
    dv2.Sort = "naam";

    LbxNaamE.ItemsSource = dv2;
    LbxNaamE.DisplayMemberPath = "naam";
}
```



Voorbeeld 3

```
private void BtnDataView_Click(object sender, RoutedEventArgs e)
{
    DataTable tbl = null;
    //string controlValue = string.Empty;
    string con = @"Integrated Security = SSPI; Persist Security Info = False; Initial Catalog =
Med; Data Source = 5CG6212CSK\SQLEXPRESS";
    using (SqlConnection cnn = new SqlConnection(con))
    {
        cnn.Open();

        SqlCommand cmd = new SqlCommand();
        cmd.Connection = cnn;
        cmd.CommandText = "select * from medewerkers";

        SqlDataAdapter da = new SqlDataAdapter(cmd);

        tbl = new DataTable();
        da.Fill(tbl);
    }

    LbxNaamE.ItemsSource = dv2;
    LbxNaamE.DisplayMemberPath = "naam"; // kolomnamen respecteren!!!
                                         // 3de dataview

    DataView dv3 = new DataView(tbl);
    dv3.Sort = "functie desc";
    DataRowView[] drv = dv3.FindRows("MANAGER"); // zoekt op opgegeven sorteersleutel.
}
```

```
//DataRowView dr = null;  
foreach (DataRowView item in drv) // Geeft 1 rij dus daarom foreach  
{  
    TxtResultaat.Text += $"{item.Row["naam"]} {item.Row["functie"]}\r\n";  
}  
}
```

JACOBS	MANAGER
BRIERS	MANAGER
CLERCKX	MANAGER

Hoofdstuk 8 Bijlagen

B1 Lijst van Code Snippets in C#

#if	Code snippet for #if
#region	Code snippet for #region
attribute	Code snippet for attribute using recommended pattern
checked	Code snippet for checked block
class	Code snippet for class
ctor	Code snippet for constructor
~	Code snippet for destructor
cw	Code snippet for Console.WriteLine
do	Code snippet for do...while loop
else	Code snippet for else statement
enum	Code snippet for enum
equals	Code snippet for implementing Equals() according to guidelines
exception	Code snippet for exception
for	Code snippet for 'for' loop
foreach	Code snippet for foreach statement
forr	Code snippet for reverse 'for' loop
if	Code snippet for if statement
indexer	Code snippet for indexer
interface	Code snippet for interface
invoke	Code snippet for safely invoking an event
iterator	Code snippet for a simple iterator
iterindex	Code snippet for 'named' iterator/indexer pair using a nested class
lock	Code snippet for lock statement
mbox	Code snippet for MessageBox.Show
namespace	Code snippet for namespace
prop	Code snippet for an automatically implemented property
propg	Code snippet for an automatically implemented property with a 'get' access or and a private 'set' accessor
sim	Code snippet for int Main()
struct	Code snippet for struct
svm	Code snippet for 'void Main' method
switch	Code snippet for switch statement
try	Code snippet for try catch
tryf	Code snippet for try finally
unchecked	Code snippet for unchecked block
unsafe	Code snippet for unsafe statement
using	Code snippet for using statement
while	Code snippet for while loop

Gebruik het sleutelwoord en druk 2x keer op TAB om de structuur te verkrijgen.

B2 Lijst van Class library in C#

Deze verzameling classes is ingedeeld in namespaces.

Hieronder enkele van de belangrijkste namespaces:

- **System.Data**

Classes om databases (SQL Server, Oracle, Access, ...) te gebruiken.

- **System.XML**

Classes om XML (uitwisselbaar dataformaat tussen de meest voorkomende computers en programmeertalen) te gebruiken.

- **System.Diagnostics**

Classes om fouten in je programma op te sporen.

- **System.DirectoryServices**

Classes om Active Directory Services te gebruiken.

- **System.Messaging**

Classes om boodschappen te lezen en te verzenden over netwerken.

- **System.Timers**

Classes om op regelmatige tijdstippen (iedere dag, om de minuut, ...) code uit te voeren.

- **System.Globalization**

Classes om taal- en cultuurverschillen in je toepassing te verwerken (wij schrijven bvb. een datum als dag/maand/jaar, in de USA is dit maand/dag/jaar).

- **System.Net**

Classes om netwerken, het internet en hun protocollen te gebruiken.

- **System.Collections**

Classes om verzamelingen te maken die meer functionaliteit aanbieden dan arrays: lists, queues, arraylists, hashtables, dictionaries.

- **System.Collections.Generic**

Zelfde als bij System.Collections maar dan voor generic lists.

- **System.IO**

Classes om bestanden en directories te gebruiken.

- **System.Text**

Classes om verschillende karaktercoderingen (ANSI, Unicode, ...) te gebruiken.

- **System.Threading**

Classes om multithreading (gelijktijdige uitvoering van code) in je toepassing te gebruiken.

- **System.Reflection**

Classes om tijdens de uitvoering van het programma de eigenschappen en methods van Classes op te vragen.

- **System.Drawing**

Classes om punten, lijnen, rechthoeken, cirkels, afbeeldingen, tekst, ... te tekenen.

- **System.Windows.Forms**

Classes om Windows vensters in je programma te gebruiken met in die vensters alle populaire controls: tekstvakken, knoppen, menu's, ...