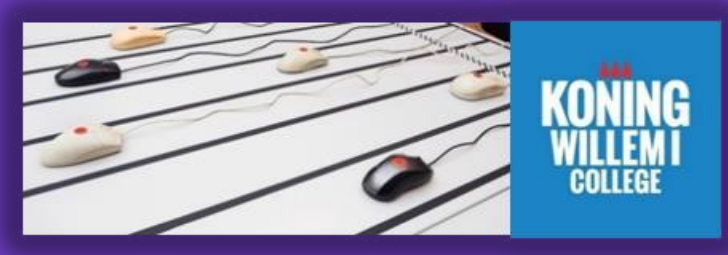
Basis standalone

Realiseren

hoofdstuk

5

Windows Forms



Algemene informatie

Onderwerp	Windows Forms
Leerdoel(en)	1. Het verschil kunnen aangeven tussen GUI en CUI 2. Het kunnen benoemen van de 9 ontwerpprincipes voor gebruikersvriendelijkheid 3. Het aanmaken van een Windows Forms applicatie 4. Het verschil kunnen aangeven tussen Code en Designer modus binnen Visual Studio. 5. Het kunnen aanmaken van de basiscomponenten: a. Basis (Label, Panel, Groupbox, Image, Picture) b. Input (TextBox, Checkbox, ComboBox, Radio, Button) c. Lijsten (Combobox, ListView) 6. Het kunnen aanpassen van de volgende Properties: a. Name b. Title / Tekst / Font / TextAlign c. Enabled / Visible / d. Anchor e. Dock f. Tag 7. Het gebruiken van de volgende events: a. Click b. Load c. Resize d. SelectedIndexChanged
Vereiste voorkennis	1. De student heeft de voorgaande readers bestudeert
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Tot nu toe heb je alleen maar console applicaties gemaakt. Bij een console applicatie verloopt de invoer haast altijd via het toetsenbord en gaat de uitvoer naar de console en verschijnt dan op het scherm.

Console applicaties zijn over het algemeen niet echt gebruikersvriendelijk. Een console applicatie heeft een zogenoemde CUI-interface (Character User Interface). Vanaf nu ga je naast console applicaties ook zogenaamde Forms-applicaties maken. Dit zijn Windowsapplicaties zoals je gewend bent dat ze eruitzien. Een Windowsapplicatie is een programma met een grafische interface dat uitgevoerd wordt op je eigen computer. Zo'n grafische weergave noemen we een GUI (Graphical User Interface).

Inhoud

1. Introductie Graphical User Interface (GUI)

Tot nu toe heb je alleen maar console applicaties gemaakt. Bij een console applicatie verloopt de invoer haast altijd via het toetsenbord en gaat de uitvoer naar de console en verschijnt dan op het scherm.

Console applicaties zijn over het algemeen niet echt gebruikersvriendelijk. Een console applicatie heeft een zogenoemde CUI-interface (Character User Interface).

Vanaf nu ga je naast console applicaties ook zogenaamde Forms-applicaties maken. Dit zijn Windowsapplicaties. Een Windowsapplicatie is een programma met een grafische interface dat uitgevoerd wordt op je eigen computer. Zo'n grafische weergave noemen we een GUI (Graphical User Interface).

Een GUI heeft de volgende functies, in willekeurige volgorde:

- Een grafisch overzicht geven van de beschikbare programmafuncties
- Het bestuurbaar/bedienbaar maken van een applicatie
- Gebruikersinput vereenvoudigen
- Output weergeven op een grafische wijze.

In principe zou elke GUI deze functies moeten hebben. Als een GUI goed is gemaakt, dan zorgt deze ervoor dat de applicatie hierdoor gebruikersvriendelijker wordt. Bij alle applicaties die je voor deze opleiding dient te maken is gebruikersvriendelijkheid zeer belangrijk.

Benieuwd waar de computer zoals jij hem kent (Muis, iconen, buttons, etc) vandaan komt? In 1982 kwam Xerox met een revolutionair systeem. Bekijk zeker de onderstaande video:
https://www.youtube.com/watch?v=Cn4vC80Pv6Q

2. Negen Principes Gebruikersvriendelijkheid

Je weet nu dat je rekening moet gaan houden met gebruikersvriendelijkheid, maar wanneer is iets nu gebruikersvriendelijk. Gebruikers willen snel en makkelijk met een programma kunnen werken. Maar hoe doe je dit dan?

Het eerste wat een gebruiker ziet is de interface van het programma. In jullie geval is dit een Grafische User Interface (GUI). De eerste indruk is belangrijk maar niet bepalend. Als de gebruiker een mooi menu ziet maar er gebeurt niets als hij/zij op een menuoptie klikt, is het programma NIET gebruikersvriendelijk.

Om je te helpen een programma gebruikersvriendelijk te maken geven we je een aantal ontwerpprincipes. De negen ontwerpprincipes zijn hieronder kort weergegeven met wat voorbeelden. Probeer bij het ontwerpen van de GUI hier goed rekening mee te houden.

1. Taakgeschiktheid

De gebruikersinterface moet aansluiten op de werkzaamheden van de gebruiker. Een gebruikersinterface is er vóór de gebruiker en niet andersom.

Wanneer een timmerman een spijker in de muur slaat met een hamer, denkt hij niet na over de bediening van de hamer; hij concentreert zich op de spijker. Een goede gebruikersinterface laat zich vergelijken met de hamer.

De gebruiker kan de interface intuïtief bedienen waardoor hij zich volledig kan concentreren op zijn werk.

2. Consistentie

In onze samenleving is praktisch iedereen in staat om een doosje lucifers op de juiste wijze te bedienen. Een Indiaan uit het Amazonegebied die nog nooit met onze cultuur in aanraking is geweest, zal echter grote moeite hebben om met het doosje vuur te maken. Zo is het ook met gebruikersinterfaces. Een gebruiker die nog nooit met een grafische gebruikersinterface heeft gewerkt, zal meer moeite hebben om zich een nieuwe applicatie eigen te maken dan een gebruiker die vertrouwd is met de concepten van grafische gebruikersinterfaces.

Consistentie is niet alleen tussen verschillende interfaces belangrijk maar ook binnen een interface.

3. Bestuurbaarheid

De gebruiker moet het idee hebben dat hij de gebruikersinterface bestuurt en niet dat de gebruikersinterface hem bestuurt. De gebruiker hoort een actieve rol te spelen in de bediening, niet een reactieve.

4. Herkenbaarheid

De gebruikersinterface moet herkenbaar zijn voor de gebruiker. De gebruiker moet de structuur en werking als het ware zelf kunnen afleiden. Dit is mogelijk door een goed conceptueel model te hanteren. Een conceptueel model biedt de gebruikers een zodanige abstractie van de onderliggende techniek dat de gebruikers begrijpen hoe ze de gebruikersinterface moeten bedienen.

5. Terugkoppeling / feedback

Vergelijk een elektrisch fornuis met een gasfornuis. Bij het koken op een elektrisch fornuis is er altijd sprake van een zekere vertraging tussen het moment waarop de gebruiker de temperatuur van een kookplaat instelt en het moment waarop de kookplaat de ingestelde temperatuur bereikt. Bij een gasfornuis daarentegen krijgt de gebruiker vrijwel direct terugkoppeling wanneer hij het gas hoger of lager draait. Over het algemeen wordt dit als prettiger ervaren.

Een ander goed voorbeeld is een printfunctie. Vaak duurt het enige tijd voordat er iets uit de printer komt. Ongeduldige gebruikers hebben dan al meerdere malen op de printknop gedrukt.

6. Eenvoud

Kijk naar de ontwikkeling van de afstandsbediening van televisies. De eerste afstandsbedieningen waren apparaten met veel knoppen die bovendien allemaal dezelfde grootte hadden. De huidige generatie afstandsbedieningen kenmerkt zich door weinig knoppen. De belangrijkste knoppen (kanaal vooruit/achteruit, volume harder/zachter) zijn groter uitgevoerd dan de andere. Daarnaast zijn specialistische knoppen (om bijvoorbeeld de kanalen te programmeren en de helderheid en het contrast in te stellen) vaak verborgen achter een menu.

7. Aantrekkelijkheid

Vergelijk de vormgeving van de website van de ene krant eens met die van een andere. De manier van presenteren komt overeen met die van de papieren versie. Of vergelijk het uiterlijk van de ene auto met dat van een andere. Beide auto's hebben een specifieke doelgroep voor ogen. Een goede interface roept positieve gevoelens op bij de doelgroep.

8. Tolerantie

Een gebruiker moet het resultaat van een commando eenvoudig ongedaan kunnen maken. Als de gebruiker per ongeluk een verkeerd gegeven invoert, moet hij dit zonder veel moeite kunnen herstellen. De gebruikersinterface dient de gebruiker zodanig te sturen dat deze geen of weinig fouten kan maken.

9. Flexibiliteit

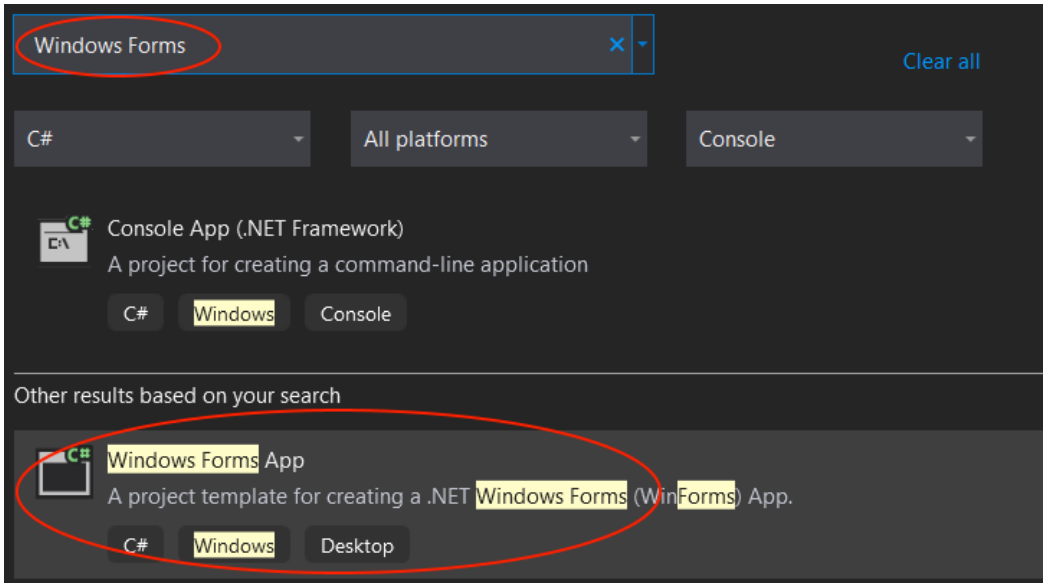
Iedereen heeft zijn eigen favoriete werkwijze. Voor het knippen en plakken in een tekstverwerker gebruikt de één de knoppen in het lint, de ander de toets combinaties Ctrl+X gevolgd door Ctrl+V en weer een ander sleept de tekst met de muis naar de gewenste positie. Een goede interface ondersteunt verschillende werkwijzen. Daarnaast biedt een goede interface de gebruiker de mogelijkheid de interface aan te passen aan zijn persoonlijke wensen.

Het artikel hieronder gaat dieper in op goede en slechte UI / User Interactie (UX) designs.
https://www.portent.com/blog/cro/9-fundamental-ux-principles-will-boost-conversions.htm

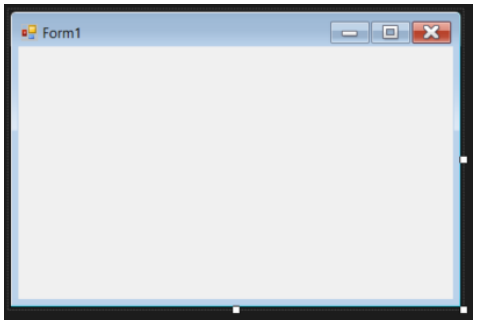
3. Starten met Windows Forms

Doordat we werken met Windows Forms komen er nieuwe tools kijken zoals de Designer en Toolbox. Laten we deze gaan ontdekken, door onderstaande stappen te volgen:

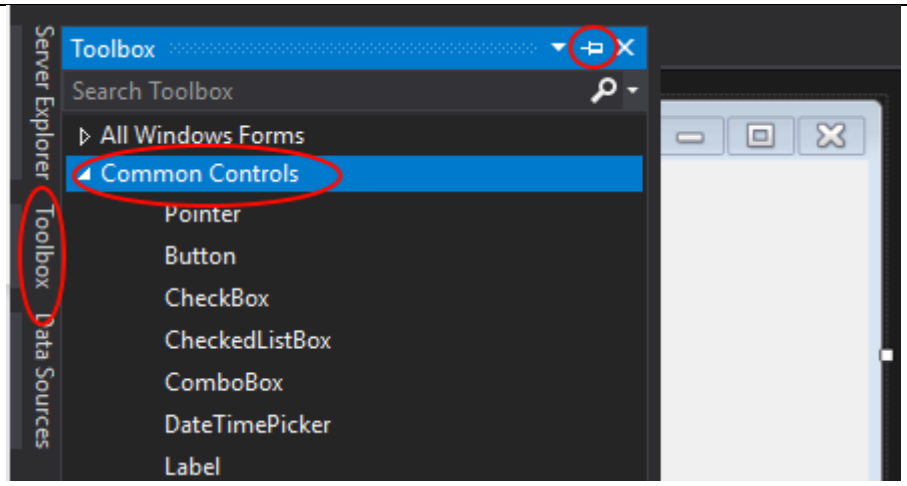
- ☑ Maak een nieuw Project aan van het type Windows Forms App. Noem deze **MyFirstWindowsFormsApp.**



- ☑ Dubbelklik op Form1.cs. Ziet nu een leeg formulier. Dit heet de **Designer Modus**:

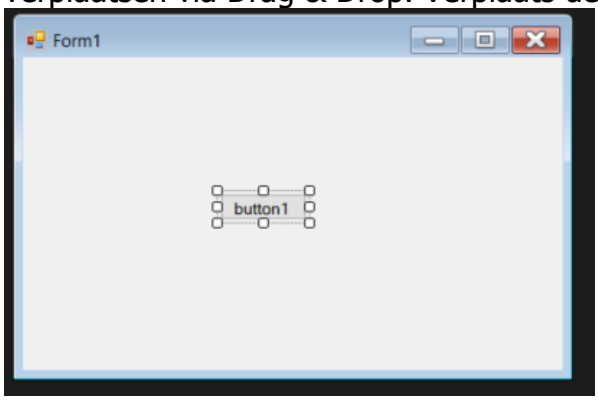


- ☑ Open de **ToolBox** (aan de linkerkant van je scherm). Open het kopje "Common Controls":

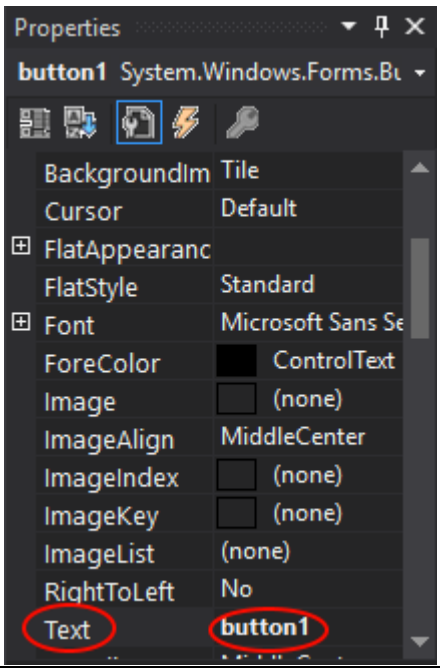


- Je ziet nu de meest voorkomende **Controls** die je kunt gebruiken voor je applicatie. Je kunt ervoor kiezen om via het Pinnetje (zie rode cirkel rechtsboven) dit menu vast te zetten. Je zult het veel gebruiken.

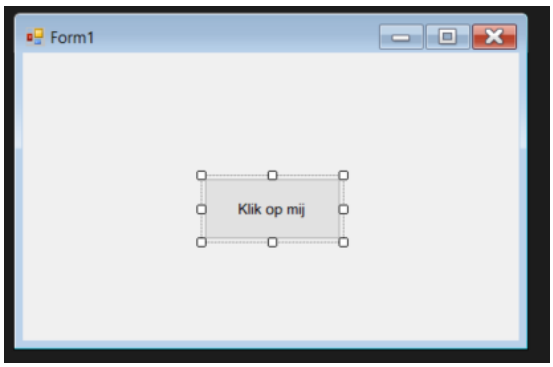
- Dubbelklik op een Button. Je ziet dat deze nu toegevoegd is aan je Form. Deze kunt je verplaatsen via Drag & Drop. Verplaats de Button naar het midden van het scherm:



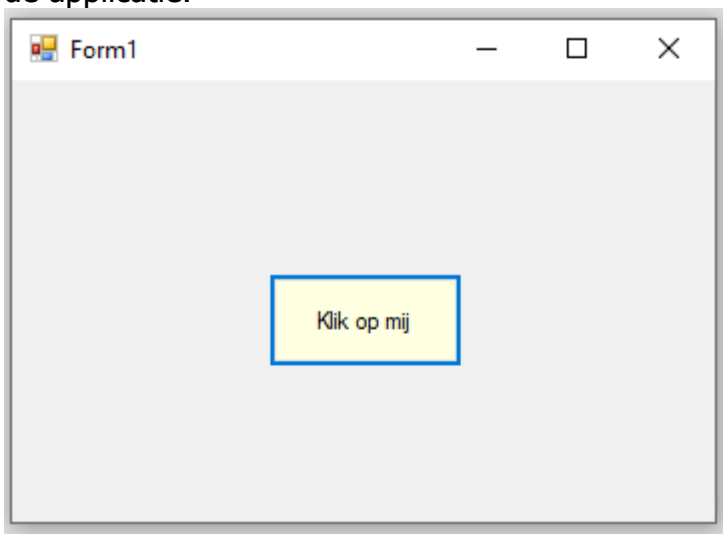
- Klik éénmaal op de Button om hem te selecteren. Bekijk nu het "**Properties**" menu (meestal rechtsonder). Zoek naar "Text":



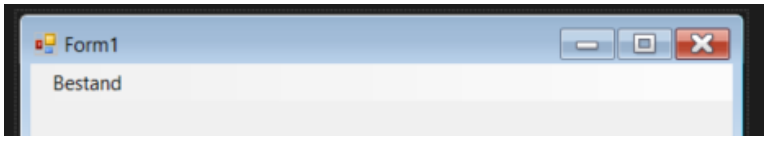
- Verander de **Tekst** button1 in "Klik op mij".
- Maak de button ook wat breder en hoger via de vierkantjes rondom de button:



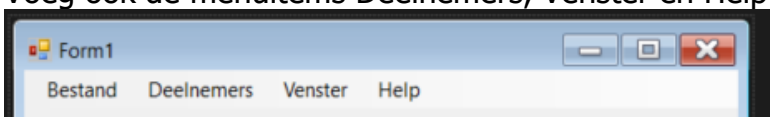
- Verander in het Properties menu ook de **BackColor** van de button naar "Info" en start de applicatie.



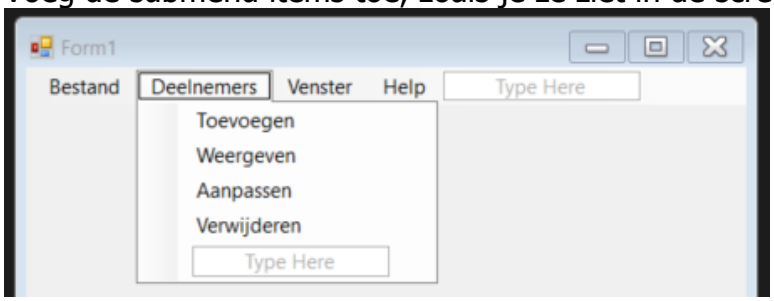
- Het resultaat van deze applicatie zie je hierboven.
- We gaan nu een menubalk toevoegen aan onze applicatie. Dit heet een **MenuStrip**. Deze kun je vinden onder "Menus & Toolbars" in je Toolbox-menu. Voeg een MenuStrip toe door erop te dubbelklikken.
- Voeg waar staat "Type Here" de tekst "Bestand" toe. Zodat onderstaand resultaat ontstaat.



- Voeg ook de menuitems Deelnemers, Venster en Help toe:



- ☑ Klik op het item "Deelnemers", zodat je Dropdownelementen kunt toevoegen.
- ☑ Voeg de submenu items toe, zoals je ze ziet in de screenshot:

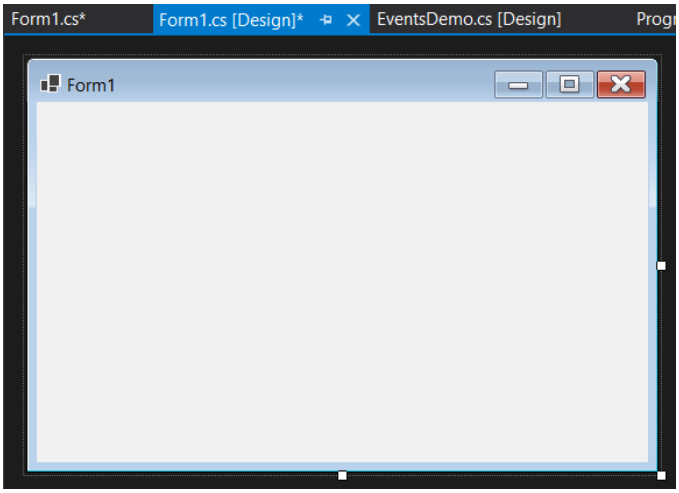


- ☑ Maak in deze applicatie ook minimaal 3 Labels, 3 Textboxes, 1 DateTimePicker en nog één Button aan. Zorg ervoor, dat alles netjes uitgelijnd is.
- ☑ Geef de **Labels** en **Buttons** een **Tekst** (zelf bedenken).
- ☑ Voeg ook een Panel toe en geef deze de rode achtergrondkleur
- ☑ Geef één van de Labels het lettertype "Berlin Sans FB" via de Property "**Font**". Maak hem ook de grootte 18.
- ☑ Maak de 2e button disabled (onklikbaar) via de property "**Enabled**" op false te zetten. Bekijk wat er gebeurt als je de applicatie start.
- ☑ Geef de DateTimePicker de property "**Value**" van de dag waarop jij geboren bent. Start de applicatie en kijk of dit werkt.

4. Windows Forms Theorie

4.1 Wat is een Form

De gehele UI van één specifiek scherm valt binnen een zogeheten *Form*. Een *Form* is dus de hoogste versie van een component. Binnen de *Form* bevinden zich alle componenten. Een *Form* kun zowel bewerken via de Designer modus, als via Code. Je bewerkt op beidde manieren hetzelfde form.

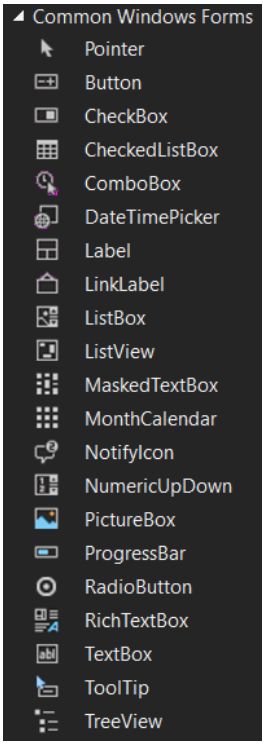


De Designer modus van Form1

```
public partial class Form1 : Form
{
    1 reference | 0 changes | 0 authors, 0 changes
    public Form1()
    {
        InitializeComponent();
    }
}
```

De Code behind van Form1

4.2 Standaard Componenten



Hier links zie je een screenshot van de Toolbox kopje "Common Windows Forms". Dit zijn de meest gebruikte Componenten die je zult gaan gebruiken (echter missen we hier de Panel, waarover later meer). Lees ze even door, en probeer ze maar eens uit.

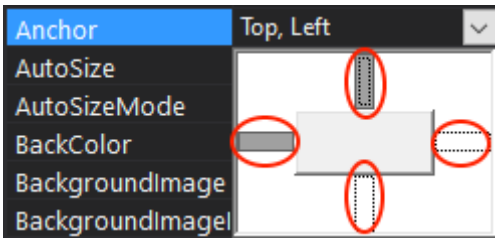
General

Je kunt veel gebruikte Componenten slepen naar het "General" kopje. Sleep bijvoorbeeld de Label, TextBox, Button, ListView en Panel maar eens naar de General.

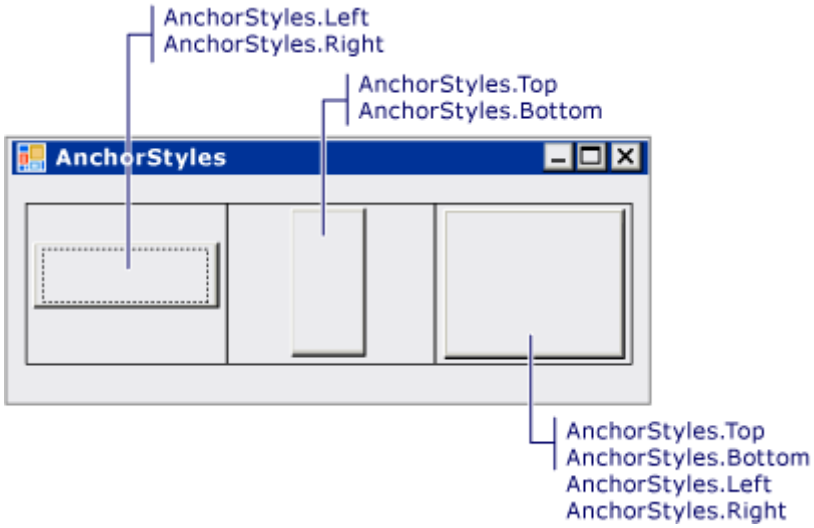
4.3 Positioneren van componenten via Anchor

Binnen Windows Forms kun je componenten laten positioneren doormiddel van de **Anchor** property te veranderen. Zodra een component een Anchor heeft, zal hij zich vastklemmen (verankeren) aan die positie (inclusief de marges) en zodra een formulier groter of kleiner wordt (resize genoemd) zich aan de hand van zijn Anchors bewegen of resizen. Dit is vooral handig bij Responsive Applicaties, waarbij de scherm groottes kunnen veranderen.

Je hebt voor een Anchor 4 opties, die je kunt selecteren*: Top, Right, Left, Bottom*. Dat selectiescherm ziet er als volgt uit:



Het is mogelijk om meerdere opties tegelijk te selecteren.



Uitwerking van verschillende Anchors

4.4 Componenten vooraf positioneren via Dock

De **Dock** property lijkt erg op de Anchor, alleen is de Dock velen malen agressiever. Via Dock kun je forceren, dat een component zich **ALTIJD** volledig links, rechts of boven vastklemt aan zijn *Parent* component, zonder rekening te houden met tussenruimtes (marges).

 Ga meer informatie over Anchor en Docking naar:
https://www.codeproject.com/Articles/2231/Working-with-Anchoring-and-Docking-Properties

4.5 Een nieuw Form aan gebruiker tonen

Je kunt op elk gewent moment in je applicatie een bepaald formulier opstarten en laten zien aan de gebruiker. Omdat een *Form* een reguliere *Class* is kun je hem declareren en initialiseren. Daarna kun vertoon je het *Form* aan de gebruiker door de methode **.Show()** aan te roepen op het object.

Zie voorbeeldcode hieronder.

```
// Declareren en initialiseren van een EventsDemo Object
EventsDemo frmEventsDemo = new EventsDemo();

// Tonen (aan de gebruiker) van dit Form
frmEventsDemo.Show();
```

Het declareren en initialiseren en daarna tonen van een Formulier

 Je kunt nu **oefening 5.1** maken.

5. ListViews

Maar hoe laat je de gegevens uit een lijst aan een gebruiker zien? Daar hebben we het Windows Forms Control genaamd **ListView** voor!

Hieronder zie je een screenshot van zo'n ListView:

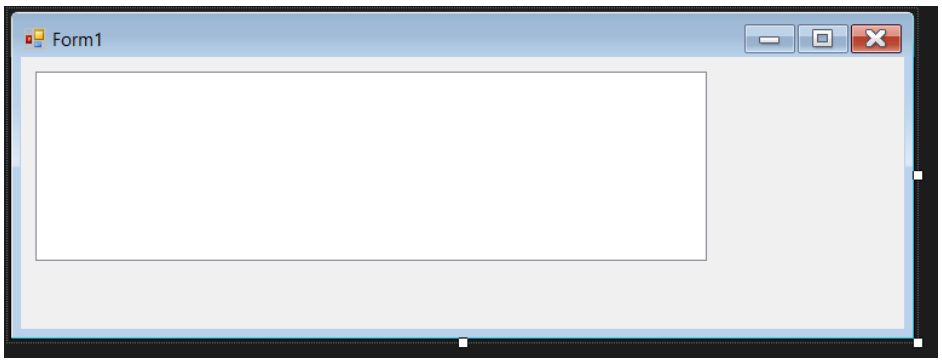
Deelnemers:

Voornaam	Achternaam	Woonplaats
Ron	Spierings	Den Bosch
Jan	Keesens	Vlijmen
Arjen	Nieuwkoop	Noordwijk
Maria	Jetten	Sprang Capelle

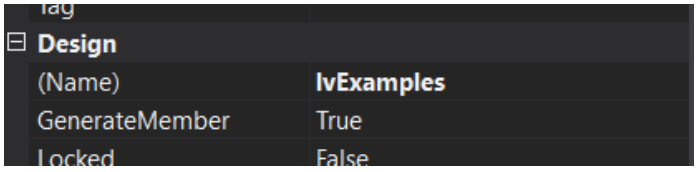
Je ziet hier een lijst met met daarin 3 verschillende kolommen (Voornaam, Achternaam en Woonplaats). De lijst zelf bevat 4 items (Ron, Jan en Arjen en Maria). Dit is een redelijk standaard manier van lijstweergaves en voelt voor veel gebruikers als erg intuïtief aan.

Laten we zo'n ListView gaan programmeren. Volg onderstaande stappen.

- Maak een nieuw Windows Forms project aan, noem deze **ListViewExample.**
- Voeg aan het formulier een **ListView** toe vanuit de **Toolbar:**

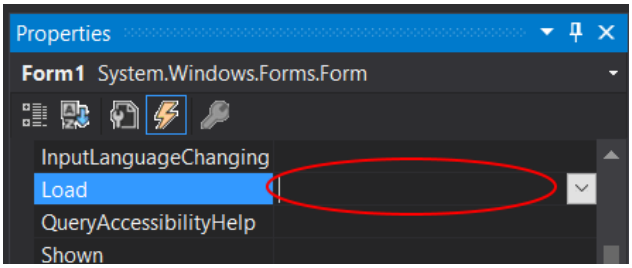


- Geef de **ListView** de property **Name** lvExamples.



- Nu we een ListView hebben, kunnen we (via code) de ListView columns gaan geven en een bepaalde View modus. De code hiervoor dient uitgevoerd te worden op het moment dat het formulier klaar met laden is. Hier hebben we het **Load** event voor.

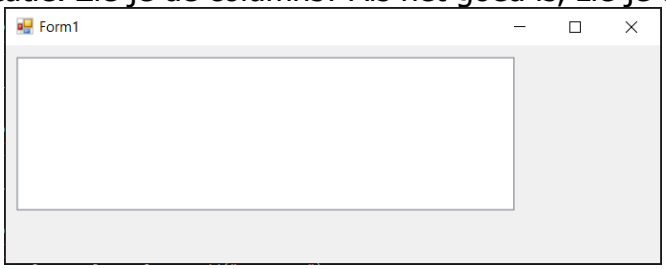
Ga naar de events van het formulier (Property menu) en dubbelklik op het **Load** event om een Load event aan te maken.



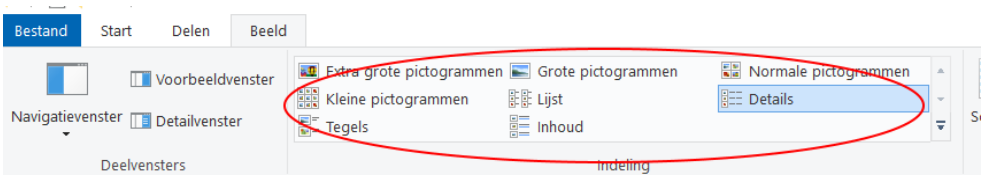
- We kunnen nu Columns toevoegen aan onze ListView via de property genaamd **Colums.** De property Columns is ook een **List<string>**, dus kunnen we via de **.Add()** methode hier items aan toevoegen. Neem onderstaande code over.

```
private void Form1_Load(object sender, EventArgs e)
{
    lvExamples.Columns.Add("Voornaam");
    lvExamples.Columns.Add("Achternaam");
    lvExamples.Columns.Add("Woonplaats");
    lvExamples.Columns.Add("Aanmeldtijd");
}
```

- Start de applicatie. Zie je de columns? Als het goed is, zie je deze (nog) niet.



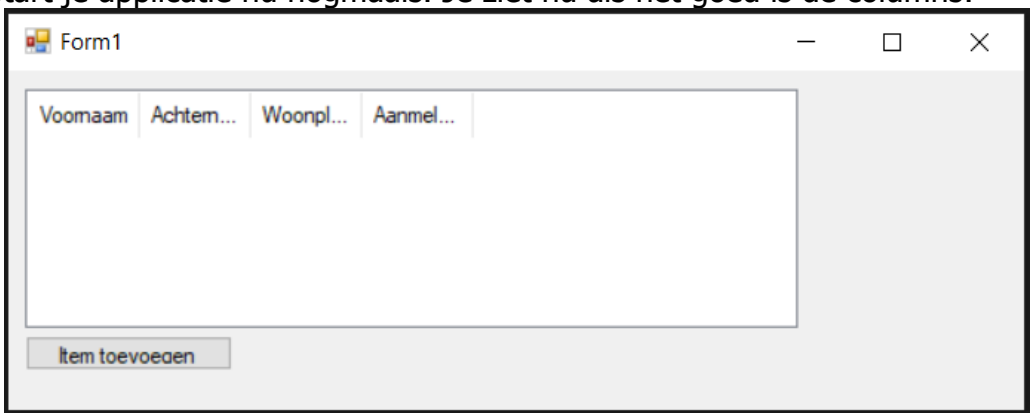
- Een ListView kan namelijk meerdere zogeheten View modus hebben. Zo kun je werken met Iconen (groot / klein), Tiles, Detail lijst en nog enkele meer. Deze Views kun je erg vergelijken met de opties die je hebt om je Windows Verkenner in te stellen.



- Om de columns te zien, moeten we de View op **Details** zetten via onderstaande code:

```
private void Form1_Load(object sender, EventArgs e)
{
    lvExamples.Columns.Add("Voornaam");
    lvExamples.Columns.Add("Achternaam");
    lvExamples.Columns.Add("Woonplaats");
    lvExamples.Columns.Add("Aanmeldtijd");

    lvExamples.View = View.Details;
}
```

- Start je applicatie nu nogmaals. Je ziet nu als het goed is de columns:



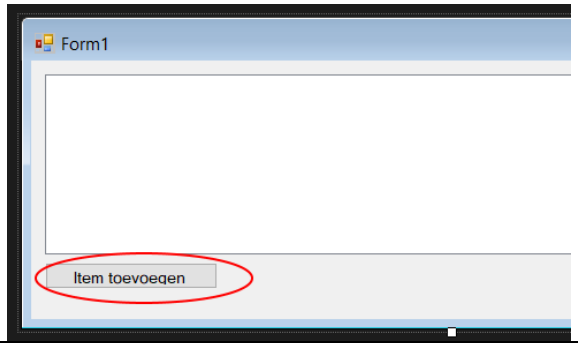
- Je hebt een lege ListView gemaakt met enkele Columns.

5.1 Vullen van een ListView

We gaan nu de eerder aangemaakte ListView vullen met gegevens. Hiervoor heeft een ListView de property **Items.** De property Items is van het type **List<ListViewItem>**, wat betekend dat het een lijst met **ListViewItems** bevat. Om dus een nieuw item toe te voegen aan een ListView, moeten we eerst een **ListViewItem** object aanmaken, vullen en daarna toevoegen aan de ListView.

Volg onderstaande stappen om dit te doen.

- Maak binnen het formulier waar ook je ListView staat een nieuwe button aan. Geef deze de Tekst "Item toevoegen":

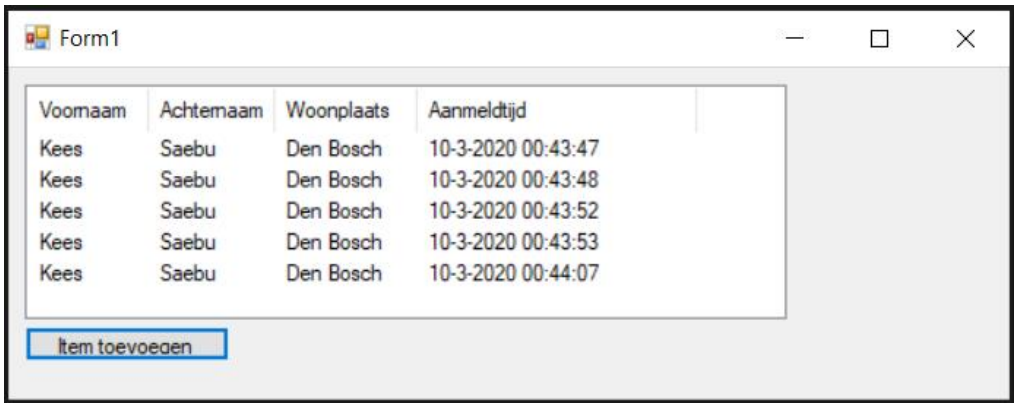


- Maak een **Click** event aan voor deze button en voeg hier de onderstaande code aan toe:

```
private void button1_Click(object sender, EventArgs e)
{
    // De eerste column van je ListView vul je tijdens het initialiseren
    ListViewItem item = new ListViewItem("Kees");

    // Alle andere columns vul je via de property SubItems
    item.SubItems.Add("Saebu");
    item.SubItems.Add("Den Bosch");
    item.SubItems.Add( DateTime.Now.ToString() );

    lvExamples.Items.Add(item);
}
```

Belangrijk is om te zien, dat de eerste column (Voornaam in ons geval) anders gevuld dient te worden, dan de overige Columns. De tekst van de eerste column dien je op te geven tijdens het aanmaken van het nieuwe **ListViewItem**. De overige columns kun je invoeren via de property **SubItems.** Ook dit is weer een List, die je dus via **.Add()** kunt vullen.

- Start je applicatie en druk een paar keer op Item toevoegen. Maak de columns breder door de rand wat te slepen. Wat zie je?



Je kunt nu **oefening 5.3** maken, om zo de ListView nog eens te oefenen.

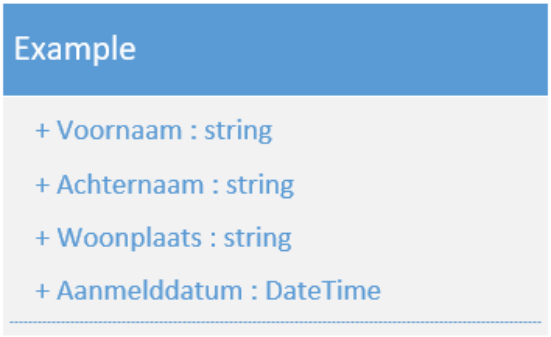
5.2 Vullen van een ListView via een List

We hebben zojuist gezien hoe je handmatig één item kan toevoegen aan een ListView. In de praktijk zul je merken dat je ListViews meestal vult aan de hand van een Array of List, die je ontvangt vanuit bijvoorbeeld een Database of webrequest.

Daarom is het belangrijk om te leren hoe je vanuit een Array of List een ListView vult via een **foreach** loop. Volg hiervoor onderstaande stappen.

We gaan ditmaal de ListView vullen met gegevens die komen uit een **List** met **Classes** van een eigen gedefinieerde Class. We gaan dus Object-Oriented Programming toepassen. Mocht je deze stappen niet snappen, neem dan de reader over OOP (Object-Oriented Programming) nogmaals door.

- Voeg aan je project **ListViewExample** een nieuwe Class genaamd "Example.cs" toe. Deze klasse moet eruit zien, zoals onderstaande klasse diagram. De code voor de klasse kun je vinden onder het klasse diagram:



Het klassediagram van de klasse Example.

```
public class Example
{
    // Properties declareren
    public string Voornaam;
    public string Achternaam;
    public string Woonplaats;
    public DateTime Aanmeldtijd;

    // Constructor om de properties
    public Example()
    {
        Voornaam = "Onbekend";
        Achternaam = "Onbekend";
        Woonplaats = "Onbekend";
        Aanmeldtijd = DateTime.Now;
    }
}
```

De code van de class Example

- We gaan nu **3 objecten** maken van de klasse **Example**. Dat doen we in de constructor van het formulier. Daarna gaan we de properties van de 3 objecten vullen met data. Zie onderstaande code:

```
public Form1()
{
    // Teken de componenten op het scherm
    InitializeComponent();

    // Objecten declareren en initialiseren
    Example example1 = new Example();
    Example example2 = new Example();
    Example example3 = new Example();

    // Properties vullen
    example1.Voornaam = "Ron";
    example2.Voornaam = "Martijn";
    example3.Voornaam = "Ralph";

    example1.Achternaam = "Spierings";
    example2.Achternaam = "van de Wetering";
    example3.Achternaam = "Gijsbrechts";

    example1.Woonplaats = "Den Bosch";
    example2.Woonplaats = "Zijtaart";
    example3.Woonplaats = "Tilburg";
}
```

- Vraag: Waarom vullen we de property **Aanmelddatum** niet?
  Antwoord: Omdat deze in de Constructor van de class Example al ingevuld wordt.

- We hebben nu 3 objecten (*example1, example2* en *example3)*. Deze objecten gaan we toevoegen aan een nog te maken **List** object genaamd *exampleList*. Deze List dient items van het type **Example** te bevatten. Voeg onderstaande code toe aan je formulier:

```
public partial class Form1 : Form
{
    // Declareren en intialiseren van de List met Examples
    List<Example> exampleList = new List<Example>();
```

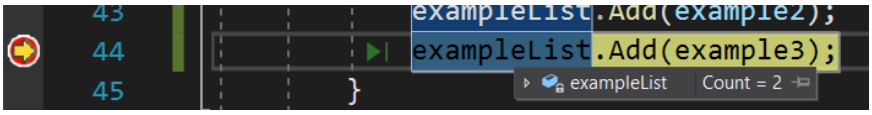
Omdat we het object *exampleList* direct in de class Form1 plaatsen (en dus niet in een methode of constructor) is deze een zogeheten **global** object geworden. Dit betekendt dat binnen de class Form1 door alle methoden benaderd kan worden.

- Voeg onderaan de constructor van Form1 de volgende code toe om de 3 items aan het object *exampleList* toe te voegen:

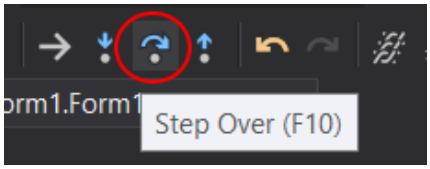
```
    example3.AchterNaam = "Gijsbrechts";

    example1.Woonplaats = "Den Bosch";
    example2.Woonplaats = "Zijtaart";
    example3.Woonplaats = "Tilburg";

    // Toevoegen van items aan exampleList
    exampleList.Add(example1);
    exampleList.Add(example2);
    exampleList.Add(example3);
}
```

- De List *exampleList* is nu gevuld met 3 items van het type Example. Dit kun je checken door een **Breakpoint** (via een dubbelklik) te plaatsen bij de laatste .Add() methode aanroep en de applicatie te starten:

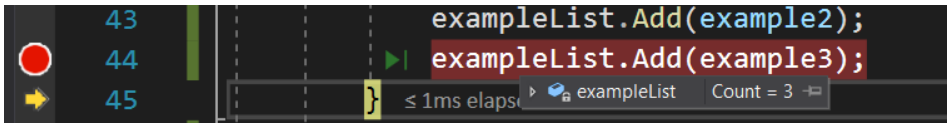
Een breakpoint geplaatst op regel 44.



Na het starten is de BreakPoint bereikt en is de code gepauzeerd op regel 44.
Je ziet dat exampleList een Count van 2 items heeft. Waarom slechts 2 (en niet 3?).

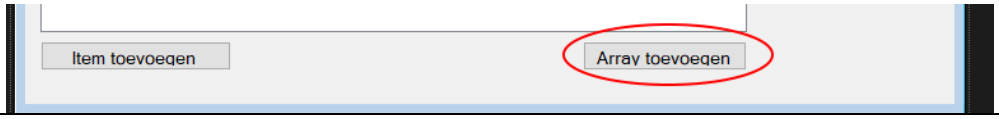


Gebruik Step Over om ook de laatste regel alsnog uit te voeren, waardoor de Count nu naar 3 gaat.



De Count van het object exampleList is nu 3 geworden.

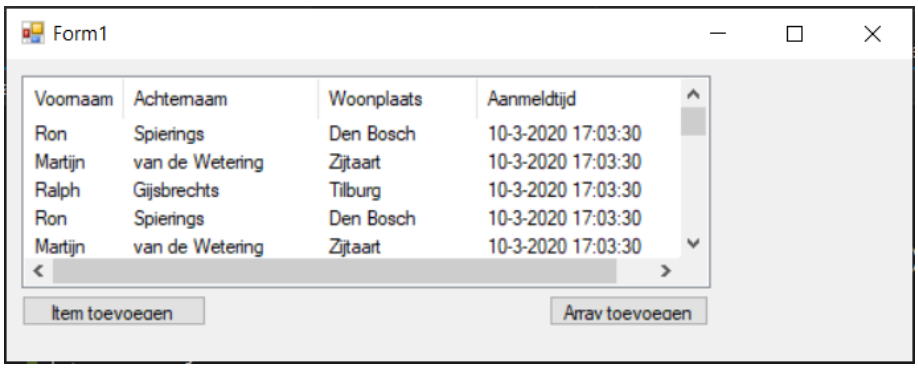
- Plaats een nieuwe button op het formulier. Geef de tekst "Array toevoegen":



- Je kent de **foreach** loop (zie hoofdstuk 2), de **List** en ook de **ListView**. Nu is het tijd om deze zaken bij elkaar te voegen, om via de ListView Geef het **Click** event de volgende code:

```
private void button2_Click(object sender, EventArgs e)
{
    foreach(Example exampleItem in exampleList)
    {
        ListViewItem newLvItem = new ListViewItem(exampleItem.Voornaam);
        newLvItem.SubItems.Add(exampleItem.Achternaam);
        newLvItem.SubItems.Add(exampleItem.Woonplaats);
        newLvItem.SubItems.Add(exampleItem.Aanmeldtijd.ToString());

        lvExamples.Items.Add(newLvItem);
    }
}
```

- Start de applicatie en druk een aantal keer op "Array toevoegen". De lijst dient zichzelf te vullen, zoals je in onderstaande screenshot kunt zien:



- Gefeliciteerd. Je hebt een applicatie gebouwd, waarin je een ListView op 2 verschillende manieren vult (stuk voor stuk) of via een **List**. Ook hebben we een aparte klasse (Example) geschreven, waardoor we op een Object Oriented manier onze code hebben gegroepeerd.

5.3 ListView leeg maken van een ListView via Item.Clear()

Het is soms nodig om een ListView leeg te maken, zodat je deze opnieuw kunt vullen (met nieuw data). Dit kan gedaan worden via de **.Clear()** methode op de de propertie **Item.** Volg onderstaande stappen om dit te doen.

- Maak een nieuwe Button aan in het formulier. Geef deze de tekst "Lijst leegmaken".

- Voeg in het **Click** event de volgende code toe:

```
private void button3_Click(object sender, EventArgs e)
{
    lvExamples.Items.Clear();
}
```

- Test de applicatie door items aan de ListView toe te voegen en daarna op de button te klikken. Als het goed is, verdwijnen de huidige items uit de ListView.

Je kunt nu **oefening 5.4** maken, waarin je jow kennis over ListViews en Object Georienteerd programmeren gaat combineren.

6. Events

Op elk component wat je plaatst op in je applicatie, kan de gebruiker een interactie mee aangaan. Het meest bekende voorbeeld is natuurlijk het klikken van een button of bewegen van een muis. Maar ook typen in een TextBox of sluiten van een venster (Een *Form* is ook slechts een component!).

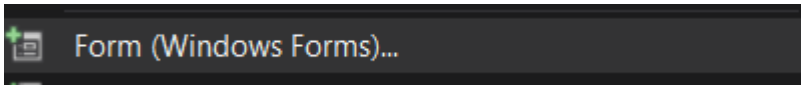
Iedere **Control** dat je aanmaakt, kan op de een of andere manier gebeurtenissen afvangen. Echter blijft het niet bij deze simpele voorbeelden, maar zijn er nog veel meer verschillende Events.

Voorbeelden van Events zijn:

- Click
- Load / Paint
- Resize
- KeyPress / KeyUp / KeyDown (toetsenbord)
- MouseEnter / MouseMove / MouseLeave
- SelectedIndexChanged (Listview)

Al deze interacties vinden plaats via **Events**. Jij als programmeur kunt code schrijven, die pas uitgevoerd wordt zodra zo'n **Event** plaats vindt. We gaan een voorbeeld van zo'n Event hieronder maken. Volg onderstaande stappen:

- ☑ Open in het project **MyFirstWindowsFormsApp.**
- ☑ Maak een nieuw formulier aan genaamd EventsDemo via Rechtermuisklik → Add → Form (Windows Forms).



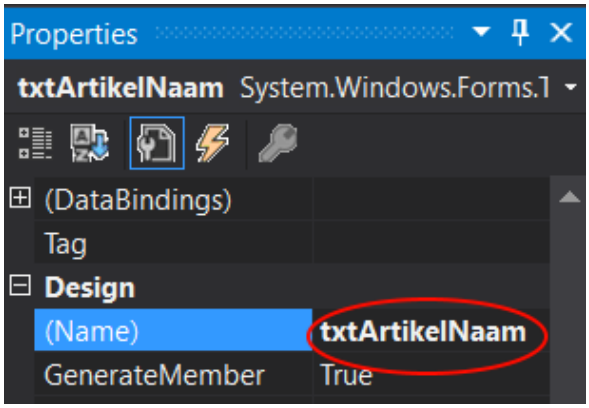
- We moeten er nu voor zorgen, dat het programma het **Form** *EventsDemo* opstart in plaats van *Form1*. Open daarom *Program.cs.*

- En vervang binnen de *Main* de initialisatie (die gebeurt op de regel van Application.Run) van het **Object** *Form1* naar *EventsDemo*. Zie onderstaande voorbeeld:

```
static void Main()
{
    Application.SetHighDpiMode(HighDpiMode.SystemAware);
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new EventsDemo());
}
```

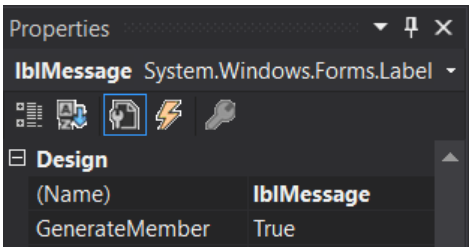
- Maak onderstaand scherm na, zodat we 3 Textboxen, een Button en een aantal Labels hebben:

Omdat we de Textbox willen gaan uitlezen, dienen we de Textboxen een leesbare unieke naam te geven om later via deze naam dus de Textbox te kunnen benaderen. Dit doen we door de Property **Name** aan te passen.



- Pas de property **Name** aan bij de Textboxes, zodat deze worden (in volgorde):
  - *txtArtikelNaam*
  - *txtArtikelID*
  - *txtOpmerking*

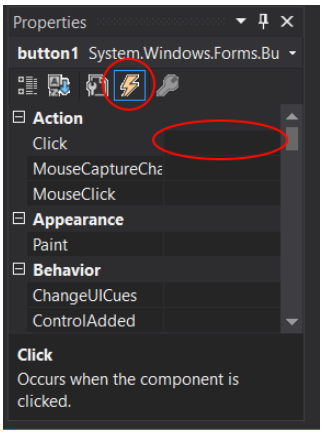
- Doe hetzelfde voor de property **Name** voor Label4, maak hier *lblMessage* van:



We gaan nu een **Click Event** aanmaken op de button. Deze gaat daarna code uitvoeren.

- Selecteer de Button door in de Designmodus er éénmaal op te klikken.

- Klik in het Propertymenu nu op de bliksem om het **Event** menu te openen:



- Dubbelklik naast het "Click" event om het Event aan te maken en automatisch de **Event Listener Methode** te laten genereren:

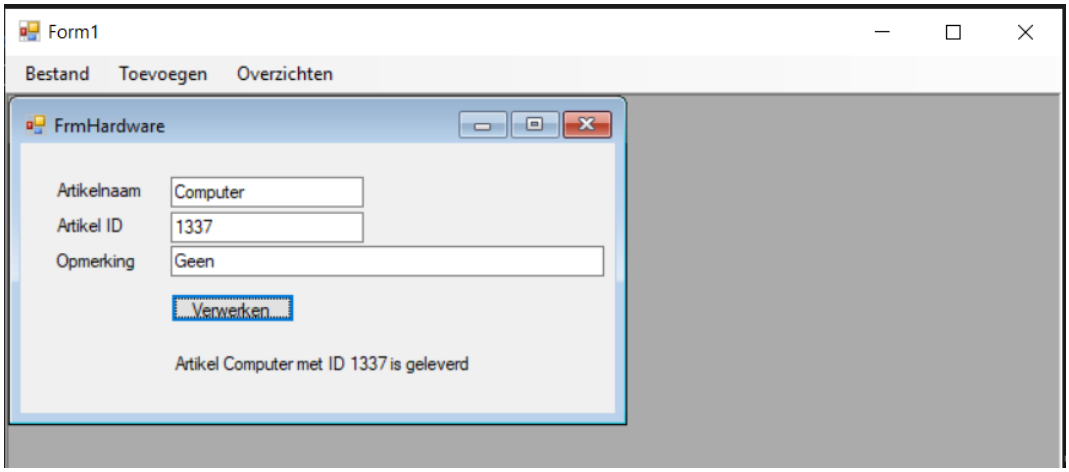
```
private void button1_Click(object sender, EventArgs e)
{

}
```

Deze **Event Listener Methode** zal uitgevoerd worden, zodra iemand op de knop klikt. Hierbinnen kunnen we onze programmacode schrijven.

- ☑ Neem onderstaande code over binnen deze Event Listener Methode:

```
private void button1_Click(object sender, EventArgs e)
{
    string artikelNaam = txtArtikelNaam.Text;
    string artikelID = txtArtikelID.Text;
    string opmerking = txtOpmerking.Text;

    lblMessage.Text = "Artikel " + artikelNaam + " met ID " + artikelID + " is geleverd";
}
```

- ☑ Start je programma. Je zou iets als onderstaand resultaat moeten krijgen:



- ☑ Beantwoord de volgende vragen:

  - Welke events kun je vinden binnen een Textbox? En binnen een Label?
  - Waarom dient de Textboxt via *txtArtikelNaam.Text* uitgelezen te worden?

Je kunt nu **oefening 5.2** maken.