Dictaat C# klassen

versie juni 2019

1. Klassen

Een klasse is een soort blauwdruk. Zie het als een tekening van hoe iets er uit moet gaan zien en wat dat “ding” kan gaan doen zodra je het daadwerkelijk gaat maken. Van een boot maak je eerst een detaillistische tekening. Wat voor soort hout heb je nodig? Waar liggen de verbindingen? Wat voor een motor komt er in te liggen? Et cetera. Deze tekening, de blauwdruk, kan niet varen. Je kunt er niet op dobberen. Pas wanneer de boot wordt gemaakt kun je er iets mee gaan doen. Je kunt er zelfs meerdere boten van maken! Zo is het met klassen ook. Een klasse is de blauwdruk voor “iets”. Je kunt er pas mee aan de slag op het moment dat je de blauwdruk *instantieert*.

We gaan eerst naar wat voorbeelden kijken die je al kent. Daarna gaan we zelf een klasse maken.

2.1. Klassen gebruiken

Bij OIS11 heb je leren werken met de Random *klasse*. Dit was al een voorbeeld van zo’n blauwdruk. Hieronder zie je een voorbeeld.

```
Random dobbelsteen;
dobbelsteen = new Random();
int getal = dobbelsteen.Next();
```

Op de eerste regel wordt een variabele aangemaakt met de naam dobbelsteen. Deze is van het type Random. Nu heb je alleen nog maar gezegd dat de dobbelsteen van het type Random is. Stel je de boot weer voor. De dobbelsteen gaat aangemaakt worden volgens de Random blauwdruk. Je kunt er nu nog niets mee doen!

Op de tweede regel wordt de dobbelsteen aangemaakt aan de hand van de blauwdruk van de Random klasse. Vanaf dit punt in de code kan de dobbelsteen gebruikt worden. We zeggen dat dobbelsteen een *instantie* is van de Random klasse. Dobbelsteen noemen we dan een object (vandaar de term object-georiënteerd).

In de derde regel zie je dat de dobbelsteen gebruikt wordt om een willekeurig getal op te vragen. Hier gebruik je dus functionaliteit die beschreven staat in de Random klasse. Je gebruikt deze echter niet via de klasse zelf, maar via de instantie van de klasse, namelijk dobbelsteen.

Tip: Je krijgt een *NullPointerException* wanneer je een instantie van een klasse gebruikt zonder dat deze is aangemaakt met new.

Je form is een klasse. Aan elke klasse kun je field (variabelen) en methodes toevoegen. Bekijk het volgende voorbeeldje (laat *partial* en de toevoeging : Form even voor wat het is).

```
public partial class FormDemo : Form
{
    private Random dobbelsteen = new Random();
    public int GooiDobbelsteen()
    {
        return dobbelsteen.Next(1, 7);
    }
    private void knopDoeIets_Click(object ...)
    {
        int getal = GooiDobbelsteen();
    }
}
```

In het voorbeeld is een variabele aan form toegevoegd van het type Random en als naam dobbelsteen. Deze wordt direct, op dezelfde regel nog, geïnitialiseerd. In de rest van het programma kun je de dobbelsteen dus veilig gebruiken.

Er is ook een methode aan het form toegevoegd. Deze kan, net als de variabele, overal in deze FormDemo klasse gebruikt worden. Dit wordt ook gedaan bij de klik-*"event handler"* van de knop. Wanneer de gebruiker op de knop drukt zal er een dobbelsteen worden gegooid. Het willekeurige getal wordt door de GooiDobbelsteen methode teruggegeven. De output van deze methode wordt opgevangen in een integer met de naam getal.

Hieronder zie je een ander voorbeeld van het gebruik van een klasse.

```
StringBuilder welkom = new StringBuilder();
welkom.Append("Welkom ");
welkom.Append("bij FUN12");
MessageBox.Show(welkom.ToString());
```

Hier wordt gebruik gemaakt van de StringBuilder klasse. Probeer het bovenstaande stukje code ook even zelf uit. Op de eerste regel code wordt een variabele van het type StringBuilder aangemaakt. De variabele heet welkom. Deze wordt direct, op dezelfde regel nog, geïnitialiseerd. Via de Append methode kunnen er stukken tekst aan worden toegevoegd. Deze wordt in zijn geheel, door de ToString() methode aan te roepen, aan de gebruiker laten zien.

Probeer zelf ook wat fields en methodes toe te voegen aan je form. Kijk eens wat je er allemaal mee kunt doen.

2.2. Zelf klassen maken

Stel je een boot voor met een snelheid, een naam, een gewicht en een aantal bemanningsleden. De boot kan varen en kan het anker uitgooien. Hoe zou je dit maken in de software? Allerlei variabelen maken en losse methoden? Wat nou als er meerdere boten nodig zijn? Voor dit soort *complexe types* kunnen we gelukkig ook onze eigen klassen maken. Bekijk het volgende voorbeeld.

```
class Boot
{
    private int Snelheid;
    private string Naam;
    private int Gewicht;
    private int AantalBemanningsleden;
    // Hieronder staan de methodes.
    public int GetSnelheid() { return Snelheid; }
    public void SetSnelheid(int snelheid) { Snelheid = snelheid; }
    public void Varen(int snelheid) { ... }
    public bool AnkerUitgooien() { ... }
}
```

In dit voorbeeld zie je als het goed is alle aspecten terug die beschreven stonden in het stukje tekst hierboven. Zo maak je een klasse! Dit kan heel gemakkelijk in Visual Studio door met je rechter-muis-knop te klikken op je C# project in de Solution Explorer en vervolgens Add > Class te kiezen. Nu kun je een naam voor je klasse ingeven en klaar ben je. Nu kun je fields en methodes toe gaan voegen. Net zoals bij je form.

```
Boot boot = new Boot();
boot.Varen(100);
MessageBox.Show("De snelheid is " + boot.GetSnelheid());
```

In bovenstaand voorbeeld zie je hoe je een eigen gemaakte klasse kunt gebruiken. Eigen net zoals een variable van het type Random. Je maakt een variabele aan van het juiste type (in dit geval Boot) en initialiseerd deze met de constructor (dat is de methode met dezelfde naam als de class). Dit gebeurt allemaal op de eerste regel van bovenstaand stukje code. Nu kun je de variabele gebruiken! Je kunt er methodes van aanroepen, zoals Varen en GetSnelheid.

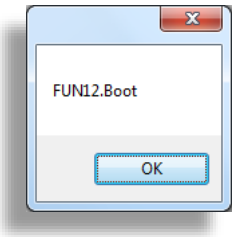
```
Boot flDutch = new Boot();
Boot titanic = new Boot();
flDutch.GetSnelheid();
titanic.GetSnelheid();
```

Tot slot kun je hierboven een van de meest krachtige aspecten van OOP zien. Elke klasse kun je gebruiken om meerdere instanties van te maken. Nu heb ik twee boten in mijn code! Elke boot met zijn eigen invulling. Zo kan ik heel gemakkelijk boten toevoegen in de code.

Probeer eens je eigen klasse te maken en te gebruiken! Wat kun je verzinnen en wat ga je er mee doen?

2.3. Je klasse als string

Soms wil je de instantie die je hebt gemaakt van je eigen klasse afbeelden als string. Bijvoorbeeld om in een ListBox te zetten. Gelukkig heeft C# daar de ToString methode voor bedacht. Elke klasse heeft deze methode. Probeer maar uit.




```
Boot b = new Boot();
string bootString = b.ToString();
MessageBox.Show(bootString);
```

Zoals je in de screenshot aan de rechterkant van de pagina al kunt zien is dit nog niet erg nuttig te noemen. Dit is wat de standaard ToString methode met je klasse doet. Gelukkig kunnen we daar zelf een meer bruikbare definitie aan geven. Bekijk het onderstaande stukje code.

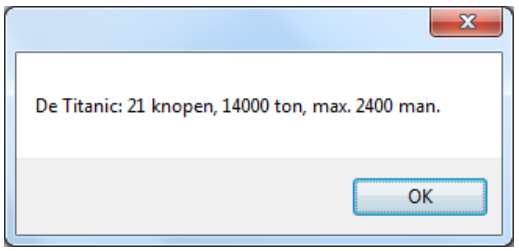
```
public override string ToString()
{
    StringBuilder sb = new StringBuilder();

    sb.AppendFormat("{0}: {1} knopen, {2} ton, max. {3} man.",
        Naam, Snelheid, Gewicht, AantalBemanningsleden);

    return sb.ToString();
}
```

Hier zie je een ToString methode die wat meer gegevens over de boot zal weergeven. Dit is ook te zien in onderstaande screenshot. Deze ToString methode kun je gewoon copy-pasten in je Boot klasse. Nadat je dat gedaan hebt kun je de nieuwe ToString methode gebruiken en uitproberen.

Let op dat je het `override` keyword ook meeneemt in de methode definitie. Daarmee weet C# dat je de standaard ToString methode wilt herschrijven. Die was toch al niet zo nuttig. Dus je eigen invulling is vast beter.



De ToString methode is ook handig om een andere reden. C# gebruikt die namelijk in de weergave van je objecten. Bekijk wat er is gebeurd in het volgende voorbeeld.

```
Boot b = new Boot();
  ...
lbBoten.Items.Add(b);
```

Op het moment dat je een boot toevoegd aan de ListBox zal C# voor jou de ToString methode aanroepen en deze gebruiken in de weergave. De ListBox zelf bevat de gehele instantie, niet alleen maar de string. Best handig dus!

Ga zelf aan de slag met de ToString methode en probeer deze uit in verschillende klassen die je zelf hebt gemaakt. Kun je deze dan ook in een ListBox zetten?

2.4. Voorbeelden

Voorbeeld	Uitleg

`class Aapje {`
`  private string Soort;`
`  private int Leeftijd;`
 	
`  public void SetSoort(string soort) {`
`  Soort = soort;`
`  }`
 	
`  public string MaakGeluid() {`
`  return "Oek oek oek";`
`  }`
 	
`  public override string ToString() {`
`  return Soort + " zegt " +`
`  MaakGeluid();`
`  }`
`}`
	Hier wordt een klasse *Aapje* gemaakt. Elk aapje wat je aanmaakt in de code is van een bepaalde soort en heeft een leeftijd. De leeftijd is een geheel getal (integer) en het soort aap is een string.
	
	Maak een aapje aan met:
	
	`Aapje kong = new Aapje();`
	`kong.SetSoort("Gorilla");`
	
	Elk aapje kan ook als string worden weergegeven met de ToString methode. Probeer het zelf eens uit:
	
	`Console.Out.WriteLine(kong);`

`class DierenVerzorger {`
`  private string Naam;`
`  private DateTime InDienstSinds;`
 	
`  public void SetNaam(string naam) {`
`  Naam = naam;`
`  }`
 	
`  public bool Verzorg(Aapje aap) {`
`  // Verzorg aapje...`
`  Console.Out.WriteLine(`
`  aap.MaakGeluid()`
`  );`
`  return true;`
`  }`
`}`
	Hier zie je een nieuwe class, *DierenVerzorger*. Deze heeft een naam en een datum van indiensttreding. Elke verzorger kan ook een aapje verzorgen. Zo zie je dat je ook andere klassen die je maakt kunt gebruiken in zelfgemaakte klassen:
	
	`DierenVerzorger verzorger =`
	`  new DierenVerzorger();`
	`verzorger.SetNaam("Henk");`
	`verzorger.Verzorg(kong);`
	
	Probeer zelf de verzorger eens een datum te geven voor zijn of haar indiensttreding. *DateTime* is ook een klasse.

```
class Speler {
  private string Naam;
  private int AantalLevens;
  private int Score;

  public void SetNaam(string naam) {
    Naam = naam;
  }

  public void SetAantalLevens(int levens) {
    AantalLevens = levens;
  }

  public void VerdienPunten(int punten) {
    if (IsGameOver() == false) {
      Score = Score + punten;
    }
  }

  public void VerliesLeven() {
    if (IsGameOver() == false) {
      AantalLevens = AantalLevens - 1;
    }
  }

  public bool IsGameOver() {
    if (AantalLevens <= 0) {
      return true;
    } else {
      return false;
    }
  }

  public override string ToString() {
      return Naam + ": " + Score;
  }
}
```

Hier links zie je een klasse voor een speler in een game. Elke speler heeft een naam, een huidige score en een aantal levens. Er zijn drie methoden. Met de VerdienPunten methode kan een bepaald aantal punten worden toegevoegd aan de score van de speler. Dit kan echter alleen maar wanneer de speler nog niet game over is.

Een speler is game over wanneer het aantal levens 0 of lager is. De speler verliest punten met de VerliesLeven methode. Deze zal het aantal levens met 1 verlagen wanneer de speler nog niet game over is.

Wat is de totaalscore van Scott in na het uitvoeren van het stukje code op de volgende pagina? Voer de code uit ter controle!

```
Speler s = new Speler();
s.SetNaam("Scott Pilgrim");
s.SetAantalLevens(3);

while (s.IsGameOver() != false)
{
    s.VerdienPunten(100);
    s.VerliesLeven();
}

Console.Out.WriteLine(s);
```