Klassen schrijven

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

Inleiding

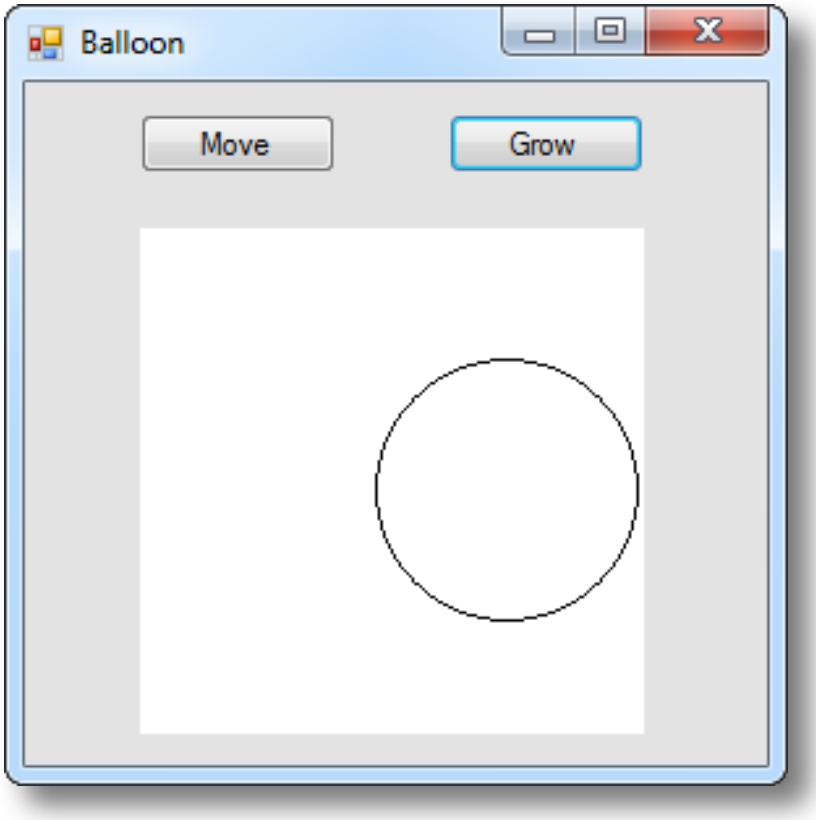
- *Wat is een klasse?*
- *Wat is een object?*
- *Hoe maak je een object?*
- *Welke klassen ken je tot hiertoe in C#?*

Een klasse

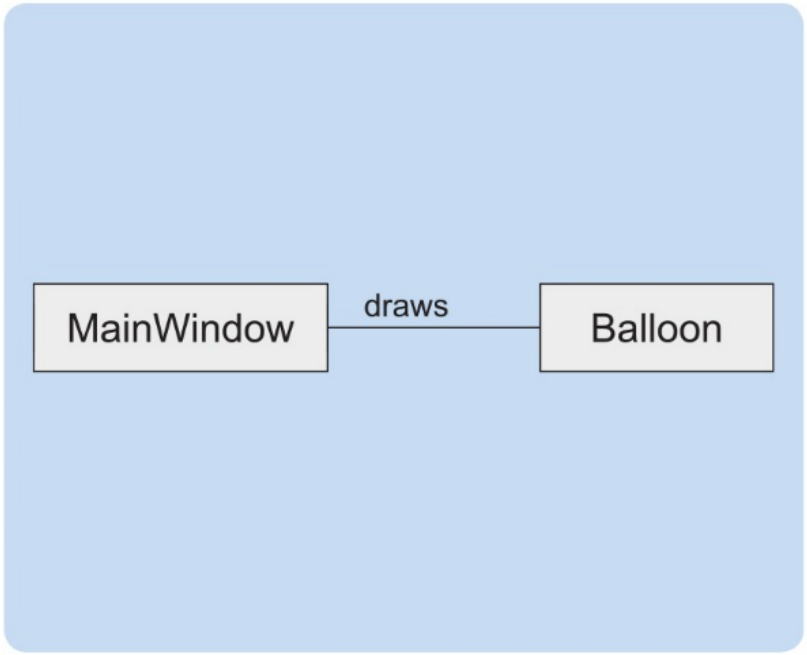
- **Heeft `private` member variabelen**
  - **Toestand**
- **Heeft (een) constructormethode(s)**
  - **Dezelfde naam als de klasse**
  - **Gebruik: `new`**
- **Heeft `public` methoden**
  - **Aanspreekpunt voor de buitenwereld**
- Heeft properties
  - Gecontroleerde toegang tot de toestand
- Heeft `private` methoden
  - Hulpmethoden enkel beschikbaar binnen het object

Een eigen klasse ontwerpen

- Een programma dat `Balloon` objecten kan tekenen
- Demo



Klassediagram

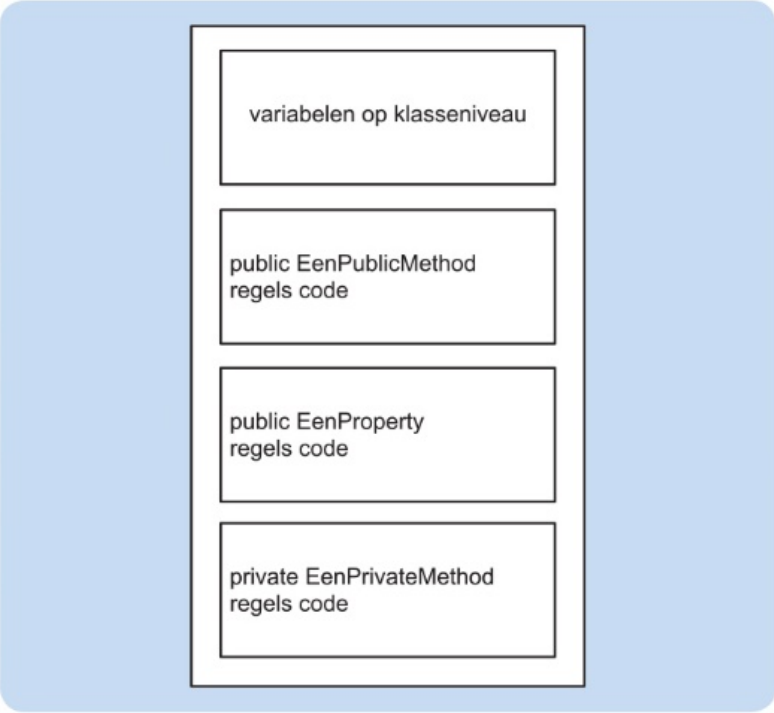


Figuur 10.2 Klassediagram dat de twee klassen in het ballonprogramma laat zien

Je krijgt onmiddellijk een beeld van de structuur van de applicatie

klassediagram

- Structuur van een klasse
  - Klassenaam
  - Member variabelen
  - Methodes
  - Properties
- Voor een variabele, methode of property staan er modifiers:
  - Een – betekent `private`
  - Een + betekent `public`
  - Een # betekent `protected`
- Lijnen tussen klassen tonen verbanden (associaties)



Figuur 10.3 Structuur van een object of klasse gezien door de ogen van de programmeur die het programma schrijft

Balloon.cs

```
public class Balloon
{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);

    public void MoveRight(int xStep)
    {
        x = x + xStep;
    }

    public void ChangeSize(int change)
    {
        diameter = diameter + change;
    }

    public void Display(Graphics drawArea)
    {
        drawArea.DrawEllipse(pen, x, y, diameter, diameter);
    }
}
```

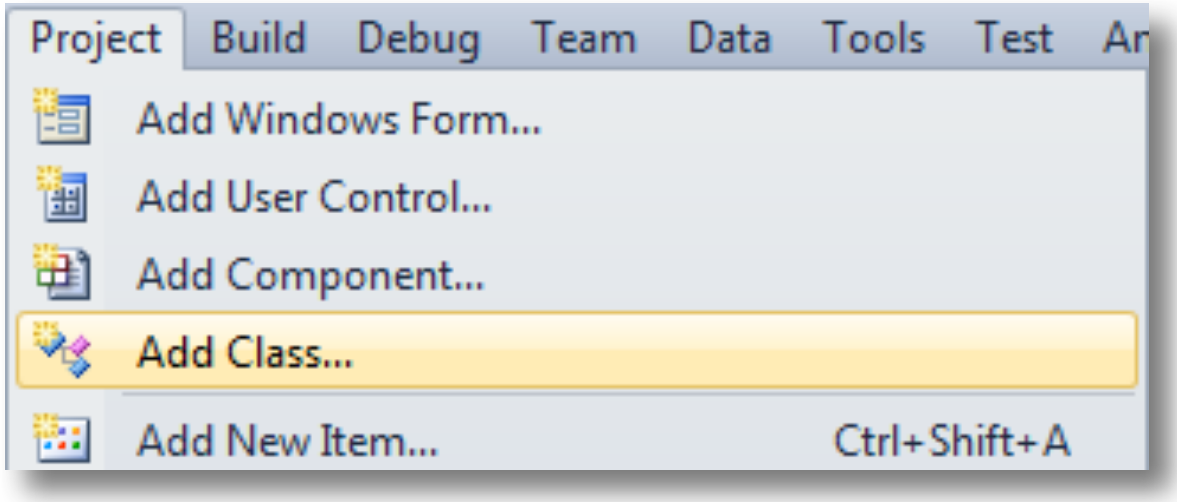
Klassenaam begint met hoofdletter

Lege lijnen + inspringen (leesbaarheid)

1 klasse per bestand (= goede gewoonte)

Demo

- Voeg de klasse `Balloon` toe
- **Add Class** in het **Project menu**



In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

private variabelen

```
public class Balloon
{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);

    public void MoveRight(int xStep)
    {
        x = x + xStep;
    }

    public void ChangeSize(int change)
    {
        diameter = diameter + change;
    }

    public void Display(Graphics drawArea)
    {
        drawArea.DrawEllipse(pen, x, y, diameter, diameter);
    }
}
```

`private` variabelen

- Beschrijven de toestand van elk `Balloon` object
- Zijn niet toegankelijk voor de buitenwereld
- Variabelen `public` maken is een slechte programmeerstijl
- Wel: `public` methodes en properties manipuleren de waarden van `private` variabelen.

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

public methoden

```
public class Balloon
{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);

    public void MoveRight(int xStep)
    {
        x = x + xStep;
    }

    public void ChangeSize(int change)
    {
        diameter = diameter + change;
    }

    public void Display(Graphics drawArea)
    {
        drawArea.DrawEllipse(pen, x, y, diameter, diameter);
    }
}
```

`public` methoden

- Beschrijven het gedrag van elk `Balloon` object
- Vormen de connectie, interface met de buitenwereld
- Zeer vaak manipuleren `public` methodes de interne toestand van een object op een gecontroleerde manier

Inkapseling

- Informatie verbergen
- Engels: encapsulation
- *Zeer belangrijk OO principe!*
- Elk object heeft een toestand die verborgen blijft voor de buitenwereld
- Elk object vertoont een bepaald gedrag die deze toestand beïnvloedt
- De buitenwereld kan enkel het object manipuleren via `public`
  - Methodes
  - Properties

Inkapseling

Standpunt ontwerper

```
public class Balloon
{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);

    public void MoveRight(int xStep)
    {
        x = x + xStep;
    }

    public void ChangeSize(int change)
    {
        diameter = diameter + change;
    }

    public void Display(Graphics drawArea)
    {
        drawArea.DrawEllipse(pen, x, y, diameter, diameter);
    }
}
```

Inkapseling

Standpunt gebruiker

```
public class Balloon
{
    [redacted]

    public void MoveRight(int xStep)
    [redacted]

    public void ChangeSize(int change)
    [redacted]

    public void Display(Graphics drawArea)
    [redacted]
}
```

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

Properties

- Zorgen voor een gecontroleerde toegang tot member variabelen
  - Lezen: *get access*

```
name = textBox1.Text;
```

  - Schrijven: *set access*

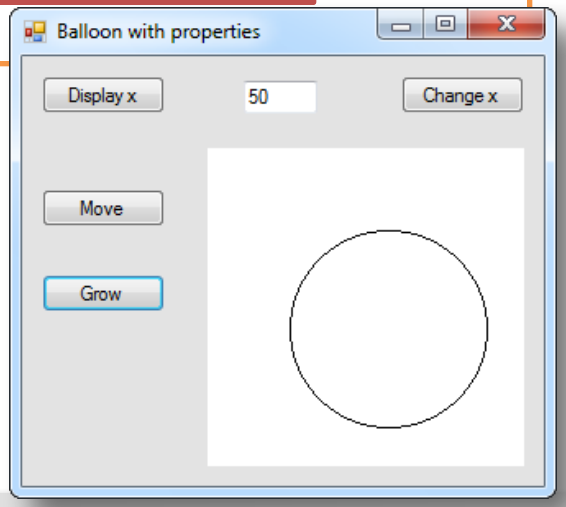
```
textBox1.Visible = false;
```

- Dit *lijkt* alsof je rechtstreeks member variabelen verandert, maar je doet dit via (property-)methoden!

Voorbeeld, Balloon p185

```
private void changeXButton_Click(object sender, EventArgs e)
{
    balloon.XCoord = Convert.ToInt32(xCoordTextBox.Text);
    drawArea.Clear(Color.White);
    balloon.Display(drawArea);
}

private void displayXButton_Click(object sender, EventArgs e)
{
    xCoordTextBox.Text = Convert.ToString(balloon.XCoord);
}
```



Property definitie

```
public int XCoord
{
    get
    {
        return x;
    }
    set
    {
        x = value;
    }
}
```

```
public int XCoord
{
    get { return x; }
    set { x = value; }
}
```

```
public int XCoord
{ get; set; }
```

Compacte indentatie

VS tip: propfull <tab> <tab>

```
xCoordTextBox.Text = Convert.ToString(balloon.XCoord);
balloon.XCoord = Convert.ToInt32(xCoordTextBox.Text);
```

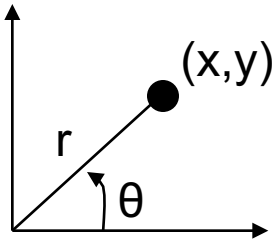
Property met enkel get of set

```
public int XCoord
{
    get { return x; }
}
```

```
public int XCoord
{
    set { x = value; }
}
```

Waarom properties?

- De interne voorstelling wordt verborgen
  - Bv: stel dat je in plaats van met een (x,y) stelsel met een polair assenstelsel (r,θ) zou werken, dan moeten de property methods de omrekening doen naar x en y
  - Onderscheid tussen `get` en `set` is mogelijk
  - Toegangscontrole is mogelijk, bv enkel x en y tussen -100 en 100 is toegestaan




```
public int XCoord{
   get { if(x<100 && x>-100)
            return x; }
  }
```

Autoproperties

- Vaak wil je een member variabele enkel instellen en uitlezen.
- Nooit `public` member variabelen gebruiken! Altijd via properties werken.
- Autoproperties (automatische property): handige schrijfwijze voor property met member variabele achter de schermen.
- VS: `prop` <tab><tab>

```
public Color FillColor { get; set; }
```

Autoproperties

```
public Color FillColor { get; set; }
```

- Deze property definieert een private member variabele van type Color, zonder expliciete naam: dus alternatief van

```
private Color c;
public Color FillColor { get{return c;}
                         set{c=value;}  }
```

- Meer controle, bv testen of kleur van ballon is toegelaten
  - Property voluit definiëren

Methode of property?

- Methode = actie, gedrag, vaardigheid, …
  - Naam = werkwoord
- Property = toestand, informatie, eigenschap, …
  - Naam = zelfstandig naamwoord
- Soms is het onderscheid een kwestie van smaak en/of stijl
  - `ChangeColor()` in plaats van `Color`

Oefening 10.6

- Klasse Bankrekening
- Methode of property?
  - TotaalBijgeschreven
  - TotaalAfgeschreven
  - HuidigSaldo
  - BerekenRente
  - Naam

Oefeningen

- Breid Balloon object uit met variabele die kleur beschrijft
- Schrijf methode MoveUp die ballon een aantal pixels omhoog beweegt door een aangegeven parameter
- Schrijf methode die kleur van ballon verandert
- Herschrijf methode Display zodat de aangepaste gekleurde ballon wordt getoond
- Voeg property diameter aan toe. Houdt er rekening mee dat diameter maximum 100 kan zijn

```
public class Balloon
{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);

    public void MoveRight(int xStep)
    {
        x = x + xStep;
    }

    public void ChangeSize(int change)
    {
        diameter = diameter + change;
    }

    public void Display(Graphics drawArea)
    {
        drawArea.DrawEllipse(pen, x, y, diam
        eter, diameter);
    }
}
```

```
public class Balloon{
    private int x = 50;
    private int y = 50;
    private int diameter = 20;
    private Pen pen = new Pen(Color.Black);
    private Color c= Color.Black;

    public void MoveRight(int xStep){x = x + xStep;}

    public void MoveuP(int yStep){y = y + yStep;}

    public void ChangeColor(Color c1){c = c1;}

    public void ChangeSize(int change){diameter = diameter + change;}

    public void Display(Graphics drawArea){
        Pen.Color=c;
        drawArea.DrawEllipse(pen, x, y, diam
        eter, diameter);
    }
    public int Diameter { get{return diameter;}
                          set{if (value<100) diameter = value;}  }

}
```

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

Constructor

- Speciale methode om een object te creëren.
- Deze methode heeft dezelfde naam als de klasse.
- Parameters zijn mogelijk.
- Overladen van Constructor is mogelijk (zie verder)
- Aanroep met new in andere klasse

Constructor

- Als je geen constructor definieert, dan is er toch (impliciet) een default constructor zonder parameters. Deze initialiseert alle member variabelen op hun default waarde.
  - Indien member variabele niet geïnitialiseerd is, C# geeft zelf standaardwaarde
    - Getallen=0, bool type=false, objecten=null, string=“”
  - Initialiseer zelf
- Van zodra je zelf één (of meer) constructors schrijft, dan vervalt deze default constructor.

Constructor: voorbeeld

```
public Balloon(int initialX,
               int initialY,
               int initialDiameter)
{
    x = initialX;
    y = initialY;
    diameter = initialDiameter;
}
```

```
Balloon balloon1 = new Balloon(10, 10, 50);

// wanneer compileert dit niet?
Balloon balloon2 = new Balloon();
```

Overloading of default waarden

- Vaak maak je meerdere versies van een constructor
- Telkens met een variatie van de parameters
- = overloading
- Default waarden kunnen hier vaak voor kortere code zorgen

Overloading constructor

```
public Balloon()
{
    x = 50;
    y = 50;
    diameter = 20;
}

public Balloon(int initialX)
{
    x = initialX;
    y = 50;
    diameter = 20;
}
...
```

Voor elke combinatie van parameters een overloaded constructor

```
public Balloon(int initialX,
               int initialY)
{
    x = initialX;
    y = initialY;
    diameter = 20;
}

...

public Balloon(int initialX,
               int initialY,
               int initialDiameter)
{
    x = initialX;
    y = initialY;
    diameter = initialDiameter;
}
```

Default waarden

```
public Balloon(int initialX = 50,
               int initialY = 50,
               int initialDiameter = 20)
{
    x = initialX;
    y = initialY;
    diameter = initialDiameter;
}
```

Dit is equivalent met de vorige slide

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

`private` methoden

- Interne hulpfuncties (binnen het object)
- Als je deze methodes binnen andere methodes van het object gebruikt, hoef je geen object ervoor te zetten. Eventueel mag het woord `this`

private methoden

```
public double Area
{
    get
    {
        return this.CalcArea();
        // return CalcArea();
    }
}

private double CalcArea()
{
    double radius;
    radius = diameter / 2.0;
    return Math.PI * radius * radius;
}
```

Bewerkingen op objecten

- Primitieve types: int, bool, double
  - Aanmaak zonder new, gewoon gelijkstellen =
    - int a= 5;
  - De bewerkingen liggen vast
- Object types: Balloon,…
  - Aangemaakt explicitiet met new
    - Balloon b = new Balloon(10,20,50)
  - De bewerkingen worden bepaald door de methodes die op deze objecten bestaan

Bewerkingen met Balloon

- Geeft lijst van bewerkingen van Balloon, p185?

Objecten vernietigen

```
Balloon balloon = new Balloon(10, 10, 50);

// Dit maakt het eerste object niet meer toegankelijk
// garbage collector zal dit dan verwijderen
balloon = new Balloon(20, 10, 30);
```

- Garbage collection is een proces dat uitgevoerd wordt door de “Common Language Runtime” van .NET
- Het zorgt ervoor dat niet meer gebruikte objecten worden verwijderd, zodat het geheugen weer vrijkomt voor nieuwe objecten
- Je kan als programmeur een hint geven dat je een object niet meer nodig hebt door aan de variabele de waarde `null` toe te kennen
  - balloon = null;

In dit hoofdstuk …

- Klassestructuur
- Private variabelen
- Publieke methoden (`public`)
- Properties
- Constructormethodes
- Private methodes (`private`)
- Static methodes en properties

`static` methoden en properties

- Soms heeft het logisch gezien geen nut om van een klasse objecten te maken
- Typische voorbeelden zijn bibliotheken van functies
- De methoden en properties spelen dan op klasse niveau ipv op object niveau
- Sleutelwoord: `static`

static methoden en properties

```
public class Math
{
    public static double Sqrt(double value)
    {
    ...
    }
}
```

```
...
double x = Math.Sqrt(64);
...
```

Static methoden en properties

Voorbeeld static methoden

- Convert.ToInt32 en Convert.ToString

Voorbeeld static property

- Color.Black, Color.White

Waarom Static methoden en properties

- Alles wordt geschreven onder klasse, geen alternatief
- Veel static methoden in bib
- Zelf static methoden schrijven als beginner niet vaak

Partial klasse

- public partial class Form1
  - Header van klasse Form1
  - Klassedefinitie is niet compleet, aangevuld met items uit ontwerpmodus

- Application.Run(new Form1());
  - Opdracht uit runtime systeem
  - Object uit klasse Form1 wordt gemaakt als parameter van static methode Application.Run() die programma start
  - In static methode Main(), in automatisch aangemaakte klasse Program.cs (=startpunt van programma)

Samenvatting

Samenvatting

- Verschil Klasse en object
- Onderdelen van klasse
  - Private leden
  - Constructor
  - Properties
  - Methoden
- Static methoden

Oefening: Zoek fout

```
private Balloon redBalloon;
redBalloon.Display(drawArea);
```

Oefening: 10.3, p203

BankRekening