C# OO Programmeren

LES 8

JOHAN DONNÉ

Tips:

```
if (File.Exists(…) == true)  DoSomething();
```

=>

```
if (File.Exists(…)) DoSomething();
```

Tips:

```
if (points.Length < 3)
{
    throw new ArgumentException("…");
}
```

=> `if (points.Length < 3) throw new ArgumentException("…");`

Tips:

```
If (File.Existst(...))
{
    ...
}
else
{
    throw new ArgumentException()
}
```

=>

```
If (!File.Exists) throw new ArgumentException();
...
```

Overzicht

Delegates
Lambda expressions
Events

Delegates

Delegates

Delegate = ‘methode type’:

Abstracte beschrijving van een methodesignatuur.
Variabele van een delegate type: kan een methode in opgeslagen worden.

Analogie:

klasse : blauwdruk voor gelijkaardige objecten

delegate : blauwdruk voor gelijkaardige methodes

⇨ o.a. gebruikt bij:

- functioneel programmeren
- events
- multithreading, concurrent programmeren

Delegate - voorbeeld

```
public delegate int Calculate(int i1, int i2);

...

  private int Sum(int x, int y)
  {
     return x + y;
  }

  private int Product(int x, int y)
  {
     return x * y;
  }
...

// a delegate variable can be instantiated in two ways:
Calculate operation = Product;
var anotherOperation = new Calculate(Sum);

// ... and be used in the same way as the method
int x = operation(2, 3);
```

Delegate – als parameter

```
public delegate int Calculate(int i1, int i2);

...

private void WriteResult(string message, Calculate calculate)
{
   int result = calculate(a, b);
   Console.WriteLine($" {message} : {result}");
}

...

// instantiate the delegate
Calculate operation = Sum;

// pass the delegate instance as a parameter
WriteResult("Sum", operation);

// a suitable method can be passed immediately as a parameter
WriteResult("Product", Product);
```

Predefined delegates

Meeste delegates (=methode definitie) volgen standaard patroon.

⇨ Meest gebruikte versies zijn vooraf gedefinieerd in ‘System’:

```
public delegate void Action();

public delegate void Action<T>(T arg);

public delegate void Action<T1, T2>(T1 arg1, T2 arg2);

...

// tot 16 parameters
```

Predefined delegates

Voorbeeld:

```
 private string text;

 public void DisplayText(Action<string> display)
 {
    display(text);
 }

...

 private void SimpleWrite(string message)
 {
    Console.WriteLine(message);
 }

...

 DisplayText(SimpleWrite);
```

Predefined delegates

⇨ Wanneer er een waarde teruggegegeven wordt:

```
public delegate TResult Func<Tresult>();

public delegate TResult Func<T,Tresult>(T arg);

public delegate TResult Func<T1,T2,Tresult>( T1 arg1, T2 arg2);

...

// tot 16 parameters
```

Delegate – demo

Demo 01

Lambda expressions

Lambda expressions

Een variabele van een delegate type bevat een methode
(alsof het data zou zijn).

Waarde van delegate instance moet een geschikte methode zijn.
'geschikt' : signatuur stemt overeen met delegate definitie (types!)

Er moet dus een geschikte methode voorhanden zijn.

⇨ Telkens volledige Methode op voorhand schrijven kan omslachtig zijn.

'Lambda expression' = manier om methode 'on the fly' te declareren

Lambda expression

= functie definitie:

(parameter list) => expression / statement block

Voorbeelden:

(x) => x + x

(a, b) => { return a + b;}

```
Calculate test = (a, b) => a + b;

Calculate test = (a, b) => { return a + b; };
```

Opmerking: geen types nodig voor de parameters als die afgeleid kunnen worden uit de delegate declaratie:

```
public delegate int Calculate(int i1, int i2);

Func<int,int,int> calculate
```

Lambda expression als parameter

```
public delegate int Calculate(int i1, int i2);

...

private void WriteResult(string message, Calculate calculate)
{
   int result = calculate(a, b);
   Console.WriteLine($" {message} : {result}");
}

...


WriteResult("Sum", (a, b) => a + b);


WriteResult("Product", (x, y) => {
                                    var z = x * y;
                                    return z;
                                 });
```

Delegates, lambda expressions

==> Les08 LambdaDemo

Klasse om alle integers in een array op dezelfde manier te bewerken.

Zie code…

Events

Events

Situatie:

Wanneer een bepaald ‘event’ gebeurt, wil je vanuit één object andere objecten op de hoogte kunnen brengen.

⇨ ‘Observer’ pattern:

‘publisher’: genereert ‘event’ notifications.

‘subscribers’ : reageren op ‘event’ notifications via ‘event handler’

Voorbeeld: ‘OnClick’ event voor Button in WinForm.

Events in C#

'Publisher':

```
public class Counter
{
    public delegate void CountChangedEvent (int count);

    public event CountChangedEvent CountChanged;

    private int count = 0;

    public int Count
    {
        get { return count; }
        set
            {
                count = value;
                if (CountChanged != null) CountChanged(value);
            }
    }
}
```

Events in C#

'subscriber':

```
public class CountObserver
{
    public CountObserver(Counter c)
    {
        c.CountChanged += CountChangedEventHandler;
    }

    public void CountChangedEventHandler(int count)
    {
        Console.WriteLine($" new count value: {count}");
    }
}
```

Events in C#

Gebruik:

```
var counter = new Counter();

var countObserver = new CountObserver(counter);

counter.Count += 1;
counter.Count = 5;
```

```
new count value: 1
new count value: 5
```

Events in C#

Opmerkingen:

- Je kan verschillende event handlers koppelen aan een event
  (typisch vanuit verschillende klassen)
  ⇨ alle gekoppelde handlers worden sequentieel aangeroepen.
- Omgekeerde bewerking van += is -= (‘unsubscribe’)
- Events kunnen enkel vanuit hun eigen klasse geactiveerd worden
- Nuttig in lagenmodel voor communicatie op initiatief van odnerliggende laag
- Ook hier kunnen de voorgedefinieerde delegates gebruikt worden:

```
public event Action<int> CountChanged;
```

Events in C#

Opmerkingen:

- Als er geen subscribers zijn, is event = null!
- Als je in de 'Publisher' dan het event oproept ==> NullReferenceException !!

```
if (CountChanged != null) CountChanged(value);

// or

CountChanged?.Invoke(value);
```

Events in C#: MS patroon

- Declareer een eigen ‘EventArg’ subklasse

```
public class CountChangedEventArgs : EventArgs
{
    public int NewCount { get; set; }
}
```

- Optioneel, declareer eigen Eventhandler delegate:

```
public delegate void CountChangedEventHandler(object sender, CountChangedEventArgs e);
```

- Declareer je event:

```
public event EventHandler<CountChangedEventArgs> CountChanged;
// public event CountChangedEventHandler CountChanged;
```

- En roep die op:

```
CountChanged?.Invoke(this, new CountChangedEventArgs{ NewCount = count});
```

Bringing it all together

Demo beeldverwerking

Beeld filteren