

Delegates



Anonymous Methoden

- **Anonymous methods** zijn methoden zonder naam
 - Kunnen parameters aanvaarden en return values geven
 - Kunnen gedeclareerd worden via **delegate** keyword
- Voorbeeld:

```
class SomeClass
{
    delegate void SomeDelegate(string str);

    static void Main()
    {
        SomeDelegate d = delegate(string str)
        {
            MessageBox.Show(str);
        };
        d("Hallo");
    }
}
```

Wat zijn Delegates?

- **Delegates** zijn speciale .NET types die een referentie naar een method kunnen bevatten
- Beschrijft de **signatuur** van een bepaalde methode
 - Aantal en types van de parameters
 - De return-type
- De “waarden” van delegates zijn methods
 - Deze methods voldoen aan het signatuur (parameters types en return type)
- ◆ Delegates zijn reference-types

Wat zijn Delegates? (2)

- Delegates zijn analoog aan function pointers in C en C++
 - Sterk-getypeerde pointer (reference) naar een methode
 - Pointer (address) naar een callback function
- Kan verwijzen naar static of instance method
- Kan verwijzen naar een reeks van methods
 - Zogenaamde “multicast” delegates
- Worden gebruikt om **callback** aanroepen te kunnen doen
- Implementeren het "publish-subscribe" model (design pattern)

Delegates – Voorbeeld

```
// Declaratie van een delegate
public delegate void EenvoudigeDelegate(string param);

public class DelegateVoorbeeld
{
    static void TestMethode(string param)
    {
        Console.WriteLine("Ik ben aangeroepen via een delegate.");
        Console.WriteLine("Ik heb parameter met waarde: {0}.", param);
    }

    static void Main()
    {
        // Instantieer de delegate
        EenvoudigeDelegate d = new EenvoudigeDelegate(TestMethode);
        // Aanroep van de methode, verwijzing via delegate
        d("test");
    }
}
```

Eenvoudige Delegate

Demo



Oefeningen Delegate

1. Declareer een niet-generic delegate dat een referentie kan bijhouden naar deze methods:

```
public static int Max(int getal1,int getal2){...}
```

```
public static int Min(int getal1,int getal2){...}
```

Test de aanroepen naar deze 2 methods via de delegate vanuit de Main.

2. Declareer een generic delegate die een referentie kan bijhouden naar deze methods:

```
public static bool IsGelijk(int getal1,int getal2)
```

```
public static bool IsGelijk(double getal1, double getal2)
```

```
public static bool IsGelijk<T>(T param1, T param2)
```

Generic en Multicast Delegates

- Een delegate kan generiek zijn bv:

```
public delegate void SomeDelegate<T>(T item);
```

- Voorbeeld van gebruik van generic delegate:

```
public static void Notify(int i) { ... }  
SomeDelegate<int> d = new SomeDelegate<int>(Notify);
```

- De code hierboven kan vereenvoudigd worden als volgt:

```
SomeDelegate<int> d = Notify;
```

- Delegates zijn **multicast** (kunnen referenties naar meerdere methoden tegelijk bijhouden), toegekend via de **+=** operator, Bijvoorbeeld:

```
d += Notify;
```


Multicast Delegate – Voorbeeld


```
delegate int StringDelegate<T>(T value);
public class MultiDelegates
{
    static int PrintString(string str)
    {
        Console.WriteLine("String waarde: {0}", str);
        return 1;
    }
    int PrintStringLength(string value)
    {
        Console.WriteLine("Lengte: {0}", value.Length);
        return 2;
    }
    public static void Main()
    {
        StringDelegate<string> d = MultiDelegates.PrintString;
        d += new MultiDelegates().PrintStringLength;
        int result = d("een string");
        Console.WriteLine("Teruggegeven resultaat: {0}", result);
    }
}
```

Multicast Delegate – Voorbeeld

```
delegate int StringDelegate<T>(T value);  
public class MultiDelegates  
{  
    static int PrintString(string str)  
    {  
        Console.WriteLine("String waarde: {0}", str);  
        return 1;  
    }  
    int PrintStringLengte(string value)  
    {  
        Console.WriteLine("Lengte: {0}", value.Length);  
        return 2;  
    }  
    public static void Main()  
    {  
        StringDelegate<string> d = MultiDelegates.PrintString;  
        d += new MultiDelegates().PrintStringLengte;  
        int result = d("een string");  
        Console.WriteLine("Teruggegeven resultaat: {0}", result);  
    }  
}
```

Multicast Delegate – Unsubscribe

```
delegate int StringDelegate<T>(T value);
public class MultiDelegates
{
    static int PrintString(string str)
    {
        Console.WriteLine("Str: {0}", str);
        return 1;
    }
    int PrintStringLength(string value)
    {
        Console.WriteLine("Length: {0}", value.Length);
        return 2;
    }
    public static void Main()
    {
        StringDelegate<string> d = MultiDelegates.PrintString;
        d += new MultiDelegates().PrintStringLength;
        d -= MultiDelegates.PrintString;
        int result = d("some string value");
        Console.WriteLine("Returned result: {0}", result);
    }
}
```



Oefening Multicast Generic Delegate

Schrijf **een delegate (generic)** die referenties naar meerdere methodes bijhoudt.

De methods waarnaar hij tegelijk moet refereren zijn :

```
static void SchrijfText(string text)
```

```
static void SchrijfTextDubbel(string text)
```

Deze methoden schrijven respectievelijk de text en 2 keer na elkaar de text naar de console

Zorg dat deze 2 methods worden uitgevoerd met één enkele aanroep van de delegate-variabele

Hergebruik dezelfde generic delegate om de volgende methoden tegelijk aan te roepen:

```
static void SchrijfDubbel(int getal)
```

```
static void SchrijfKwadraat(int getal)
```

Deze schrijven resp het dubbel en het kwadraat van het getal naar de console

Built-in Delegates

- In C# zijn een aantal Built-in delegates voorzien die kunnen gebruikt worden :
- Niet generieke Built-in delegate **Action** uit System namespace. Kan gebruikt worden met methoden die geen terugkeerwaarde hebben en geen parameters aannemen:
- Generieke Built-in delegates:
 - **Action<>**
 - **Predicate<>**
 - **Func<>**

Built-in Generic Delegates

- **Action<>**

Generic delegate die gebruikt kan worden voor methoden die minstens 1 parameter nemen en geen terugkeerwaarde geven

- **Func<>** (Func<TResult>, Func<T, TResult>, Func<T1, T2, TResult>, ...)

Voor methoden die al dan niet parameters hebben én terugkeerwaarde

Predicate<>

- Voor methoden die 1 input param nemen en een bool terugkeerwaarde geven

Action

- Kan gebruikt worden om verwijzing bij te houden naar methoden die geen terugkeerwaarde geven en minstens 1 input parameter aannemen

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

- Voorbeeld: Max is een methode die 2 inputwaarden van het type int aanvaardt en een void teruggeeft (schrijft rechtstreeks de maximum waarde naar de console)

```
Action<int, int> maxNumber = Max;  
maxNumber(getal1, getal2);
```

Predicate

- Predicates zijn built-in generic delegates met de volgende signatuur:

```
public delegate bool Predicate<in T>(T obj);
```

- Kan gebruikt worden om een Boolean criterium te verifiëren
- Is gelijkaardig aan `Func<T, bool>`
- Bv. `MyClassIsSleeping` geeft een bool terug en neemt als input param een enum `DayOfWeek` waarde (dagen van de week)

```
Predicate<DayOfWeek> mySleepingPredicate = MyClassIsSleeping;
```

- Wordt veel gebruikt bij Lambda expressies om een element op te zoeken in `Array` of `List<T>`
 - Bv `List<T>.FindAll(...)` Geeft alle elementen die voldoen aan een bepaald criterium
 - Bv.

```
List<int> list = new List<int>() { 1, 2, 3, 4 };  
List<int> evenNumbers = list.FindAll(x => (x % 2) == 0);
```


Func

- Func<> is een built-in generic delegates met de volgende signatuur:

```
public delegate TResult Func<in T1,..., out TResult>(T1 arg,...);
```

- Kan gebruikt worden om verwijzingen bij te houden naar methoden met een terugkeerwaarde en 0,1, of meerdere input parameters
- Voorbeeld:
IsGelijk is een methode die 2 double input parameters aanneemt en een bool teruggeeft:

```
Func<double, double, bool> isGelijkDouble = IsGelijk;
```

Built-in Generic Delegates voorbeelden

```
static void MyIntMethod(int i){
    Console.WriteLine($"MijnIntMethode param {i}");
}

static void MyStringMethod(int i, string s) {
    Console.WriteLine($"MijnStringMethode param {i} en param {s}");
}

static bool MyClassIsSleeping(DayOfWeek weekday) {
    if (DateTime.Now.Hour > 12 && weekday != DayOfWeek.Saturday)
        return true;
    else
        return false;
}

static void Main(string[] args) {
    //Gebruik maken van built-in generic delegate Action<>
    Action<int> myIntAct = MyIntMethod;
    Action<int, string> myIntStringAct = MyStringMethod;
    myIntAct(10);
    myIntStringAct(5, "Hallo");

    //built-in generic delegate Func<> bv: zonder param en met return waarde
    Func<DayOfWeek, bool> mySleepingFunc = MyClassIsSleeping;
    Console.WriteLine("Slaapt op zaterdag = " + mySleepingFunc(DayOfWeek.Saturday));
    Predicate<DayOfWeek> mySleepingPredicate = MyClassIsSleeping;
    Console.WriteLine("Slaapt vandaag = " + mySleepingPredicate(DateTime.Now.DayOfWeek));
    Console.Read();
}
```

Built-in Generic Delegates en lambda expressies

- Een lambda expressie is een **anonymous methode**
- Lambda expressies
 - Gebruiken de lambda operator **=>**
 - Lees als “**gaat naar**”
 - Aan de linkerkant van **=>** worden de input parameter(s) gespecificeerd
 - Aan de linkerkant van **=>** staat een expressie of statement
 - Generic Built-in Delegates kunnen een Lambda-methode bijhouden: bv:

```
Func<bool> boolFunc = () => true;  
Func<int, bool> intFunc = (x) => x < 10;  
if (boolFunc() && intFunc(5))  
    Console.WriteLine("5 < 10");
```

Referenties

- C# Programming @ Telerik Academy
 - ♦ csharpfundamentals.telerik.com
- ♦ Telerik Software Academy
 - ♦ academy.telerik.com

