Delegates und Events

Delegate = Methodentyp

Deklaration eines Delegate-Typs

```
delegate void Notifier (string sender);      // normale Methodensignatur
                                             // mit Schlüsselwort delegate
```

Deklaration einer Delegate-Variablen

```
Notifier notify;
```

Zuweisung einer Methode an eine Delegate-Variable

```
void SayHello(string sender) {
    Console.WriteLine("Hello from " + sender);
}

notify = new Notifier(SayHello);
```

Aufruf der Delegate-Variablen

```
notify("Max");                               // Aufruf von SayHello("Max") => "Hello from Max"
```

Zuweisung unterschiedlicher Methoden

Jede passende Methode kann einer Delegate-Variablen zugewiesen werden

```
void SayGoodBye(string sender) {
    Console.WriteLine("Good bye from " + sender);
}

notify = new Notifier(SayGoodBye);

notify("Max");          // SayGoodBye("Max") => "Good bye from Max"
```

Bemerkungen

- Einer Delegate-Variablen darf *null* zugewiesen werden.
- Delegate-Variable darf nicht aufgerufen werden, wenn sie keinen Delegate-Wert enthält (sonst gibt es eine Exception).
- Delegate-Variable kann in Datenstruktur gespeichert, als Parameter übergeben werden etc.

Erzeugen von Delegate-Werten

new DelegateType (obj.Method)

- *obj* kann *this* sein (und somit fehlen)
- *Method* kann *static* sein. In diesem Fall steht statt *obj* der Klassenname.
- *Method* darf nicht *abstract*, wohl aber *virtual*, *override*, *new* sein.
- *Method*-Signatur muß mit *DelegateType* übereinstimmen
  - gleiche Anzahl von Parametern
  - gleiche Parametertypen (inklusive Return-Typ)
  - gleiche Parameterarten (ref, out, value)

Delegate-Variable speichert Methode *und* Empfängerobjekt !

Multicast-Delegates

Delegate-Variable kann mehrere Werte gleichzeitig aufnehmen

```
Notifier notify;
notify = new Notifier(SayHello);
notify += new Notifier(SayGoodBye);

notify("Max");                    // "Hello from Max"
                                  // "Good bye from Max"

notify -= new Notifier(SayHello);

notify("Max");                    // "Good bye from Max"
```

Beachte

- Wenn ein Multicast-Delegate eine Funktion ist, wird letzter Funktionswert geliefert.
- Wenn ein Multicast-Delegate *out*-Parameter hat, wird Parameter des letzten Aufrufs geliefert. *ref-Parameter* werden durch alle Methoden durchgereicht.

Ähnliches in Java


```
interface Notifyer {                          // entspricht Delegate-Typ
    void notify(String sender);
}

class HelloSayer implements Notifier {        // entspricht Delegate-Wert
    public void notify(String sender) {
        System.out.println("Hello from" + sender);
    }
}

Notifier n = new HelloSayer();                // entspricht Initialisierung der Delegate-Variablen

n.notify("Max");
```

- Interface übernimmt Rolle des Delegates.
- Rund um die aufzurufende Methode braucht man eine Klasse, die das Interface implementiert.
- Keine Aufrufe statischer Methoden möglich.
- Es wird nur die Methode, aber nicht der Empfänger gespeichert.
- Multicasts erfordern zusätzlich Registrierungsmechanismus und Aufrufschleife.

Events = spezielle Delegate-Variablen


```
class Model {
    public event Notifier notifyViews;
    public void Change() { ... notifyViews("Model"); }
}
```

```
class View1 {
    public View1(Model m) { m.notifyViews += new Notifier(this.Update1); }
    void Update1(string sender) { Console.WriteLine(sender + " was changed"); }
}
class View2 {
    public View2(Model m) { m.notifyViews += new Notifier(this.Update2); }
    void Update2(string sender) { Console.WriteLine(sender + " was changed"); }
}
```

```
class Test {
    static void Main() {
        Model m = new Model(); new View1(m); new View2(m);
        m.Change();
    }
}
```

Warum Events statt normaler Delegate-Variablen?

- Nur die Klasse, die das Event deklariert, darf es auslösen (bessere Kapselung).
- event-Felder dürfen von außen nur mit += oder -= modifiziert werden (nicht mit =)

Event-Konventionen in .NET

Delegates für das Event-Handling sollten folgende Signatur haben

```
delegate void SomeEvent (object source, MyEventArgs e);
```

- Rückgabetyp void
- 1. Parameter = Sender des Events (Typ *object*)
- 2. Parameter = Event-Parameter (Unterklasse von *System.EventArgs*)

```
public class EventArgs {
    public static readonly EventArgs Empty;
}
```

Beispiel für .Net-Events


```
public delegate void KeyEventHandler (object sender, KeyEventArgs e);

public class KeyEventArgs : EventArgs {
    public virtual bool  Alt { get {...} }                  // true if Alt-Key was pressed
    public virtual bool  Shift { get {...} }                // true if Shift-Key was pressed
    public         bool  Control { get {...} }              // true if Ctrl-Key was pressed
    public         bool  Handled { get{...} set {...} }     // indicates if event was already handled
    public         int   KeyValue { get {...} }             // the typed keyboard code
    ...
}
```

```
class MyKeyEventSource {
    public event KeyEventHandler KeyDown;
    public KeyPressed() {
        KeyDown(this, new KeyEventArgs(...));
    }
}
```

```
class MyKeyListener {
    public MyKeyListener(...) { keySource.KeyDown += new KeyEventHandler(HandleKey);}
    void HandleKey (object sender, KeyEventArgs e) {...}
}
```