EVENTS

Vincent Uhlmann
IT-Akademie Dr. Heuer GmbH

EVENTS

- Events bauen auf Delegaten auf, indem sie eine zusätzliche Schicht der Abstraktion bieten
- Events ermöglichen es einer Klasse oder einem Objekt, andere Klassen oder Objekte über Ereignisse zu informieren
- Sender (Publisher) löst ein Event aus
- Empfänger (Subscriber) reagieren auf das Event
- Ein Event-Handler ist eine Methode, die aufgerufen wird, wenn das Event ausgelöst wird

VERGLEICH ZU DELEGATES

- Events erlauben eine lose Kopplung zwischen dem Publisher und Subscriber
- Der Publisher muss nicht wissen wer Subscribed hat
- Der Subscriber muss nicht wissen wer das Event ausgelöst hat
- Events steuern die Zugriffsberechtigung der Subscriber
- Durch die Einschränkung des Zugriffs auf den Delegaten von außen wird verhindert, dass externe Code Teile des Event-Handlers entfernt oder das Event unerwartet auslöst

SYNTAX

- Ein Event wird mit dem Schlüsselwort event und dem Delegate-Typ deklariert
- Der Delegate definiert die Signatur für den Event-Handler der Subscriber-Klasse

```
internal class Foo
{
    public event EventHandler MyEvent;
}
```

SYNTAX

- Ein Event wird durch Aufrufen des Delegaten ausgelöst
- Kann nur innerhalb der Klasse aufgerufen werden, die das Event angelegt hat
- Mit Invoke kann das Event aufgerufen werden
- Wenn ein Event keine Subscriber hat, wird beim Aufruf eine Null-Reference-Exception ausgelöst
- Kann durch den ? Operator verhindert werden

```
internal class Program
{
    public static event EventHandler? MyEvent;

    static void Main()
    {
        MyEvent?.Invoke(null, new EventArgs());
    }
}
```

SYNTAX

- Subscriber-Klassen registrieren sich für ein Event, indem sie einen Event-Handler bereitstellen
- Dies geschieht mit dem += Operator
- Die Methode TestMethod wird in diesem Beispiel aufgerufen, wenn das Event ausgelöst wird
- Zuweisung via = Operator ist im Vergleich zu Delegaten nicht möglich

```
public class Foo
{
    public Foo()
    {
        Program.MyEvent += TestMethod;
    }

    public static void TestMethod(object? sender, EventArgs e)
    {
        // Do something
    }
}
```

SYNTAX

- Subscriber können sich von einem Event abmelden, indem sie den -= Operator verwenden

```
public void Unsubscribe()
{
    Program.MyEvent -= TestMethod;
}
```

SYNTAX

- Ereignishandler können Parameter bereitstellen, die von dem ereignisauslösenden Objekt übergeben werden
- Ereignishandler in .NET weisen zwei Parameter auf
- 1. Das auslösende Objekt
- 2. Die Ereignisspezifischen Daten

```
public event EventHandler<EventArgs> Foo;

public void ExecuteFoo()
{
    Foo.Invoke(this, new EventArgs());
}
```

```
private void EreignisHandler(object? sender, EventArgs e)
{
}
```

SYNTAX

- Oftmals werden eigene Daten mit dem Event gesendet
- Die Daten werden in der Regel durch eine Klasse übergeben, die von EventArgs abgeleitet ist

```
public static event EventHandler<CustomEventArgs> MyEvent;

public class CustomEventArgs : EventArgs
{
    public int Result { get; set; }
    public string Message { get; set; }

    public CustomEventArgs(int result, string message)
    {
        Result = result;
        Message = message;
    }
}
```

EVENTS IN INTERFACES

- Events können in Interfaces deklariert werden
- Diese müssen von der Implementierenden Klasse implementiet werden

```
public interface IFoo
{
    event EventHandler? Foo;
}

public class Person : IFoo
{
    public event EventHandler? Foo;
}
```

ACCESSOREN

- Events ähneln Eigenschaften durch die Verwendung von Accessoren
- Add => fügt dem Delegaten einen Methode hinzu
- Remove => entfernt die Methode aus dem Delegaten
- Ermöglicht das Hinzufügen von Logik
- Beispiel: Eine Ausgabe wenn ein Event hinzugefügt bzw. entfernt wird

```
public class Person
{
    private EventHandler? _eventHandler;

    public event EventHandler Foo
    {
        add => _eventHandler += value;
        remove => _eventHandler -= value;
    }

    public void ExecuteFoo()
    {
        _eventHandler?.Invoke(this, new EventArgs());
    }
}
```

NAMENSKONVENTIONEN

- Delegatenname für ein Event endet mit EventHandler
- Eventname wird großgeschrieben
- EventArgs Klassenname endet mit EventArgs

```
public class Person
{
    public delegate void FooEventHandler(object? sender, FooEventArgs e);

    public event FooEventHandler? Foo;

    public void ExecuteFoo()
    {
        Foo?.Invoke(this, new FooEventArgs());
    }
}

public class FooEventArgs : EventArgs
{
    public string Name {get; set;}
    public FooEventArgs(string name)
    {
        Name = name;
    }
}
```