

EVENTS

WAS SIND EVENTS?

In den letzten Videos ging es um Events. Diese stellen wirklich ein unglaublich nützliches Werkzeug bei der Programmierung dar und deshalb fassen wir das Ganze Thema nochmal zusammen.

Mithilfe von Events (auf Deutsch Ereignisse) kann man außenstehende Objekte darüber benachrichtigen, dass irgendwas in einem Objekt passiert ist. Events kann man innerhalb von Objekten definieren und andere außenstehende Objekte auf diese lauschen lassen. Wird das Event dann ausgelöst (also trifft das Ereignis ein), dann wird jedes lauschende Objekt darüber benachrichtigt und kann entsprechend mit einer eigenen Methode darauf reagieren. Diese Methode mit welchen man auf Events reagiert, nennt man **EventHandler**.

Das Event-System in C# basiert auf Delegaten und dem sogenannten Abonnenten/Publisher-Modell. Der Publisher ist dabei das Objekt, dass ein Event auslöst und die Abonnenten sind die Objekte, die auf das Event lauschen.

EVENTS DEFINIEREN UND AUSLÖSEN

Ein Event setzt sich aus mehreren Grundlegenden Dingen zusammen. Zuerst wäre da der Delegat für die EventHandler-Methoden. In diesem Delegaten werden später alle Methoden gespeichert, mit welchen auf das Event reagiert wird. Dann kommt die Definition des Events in der Auslöser-Klasse und am Ende wären da noch die EventHandler-Methoden in den einzelnen „Lauscher-Klassen“. Das Ganze ist also eigentlich leichter als es anfangs klingt.

In unserem Beispiel haben wir eine Klasse namens **Uploader**, welche für das Uploaden von Videodateien verwendet wird. In dieser Klasse möchten wir ein Event definieren, dass immer dann ausgelöst wird, wenn der Upload eines Videos erfolgreich abgeschlossen wurde. Zunächst benötigen wir für die Umsetzung einen Delegaten für die EventHandler:

```
public delegate void VideoUploadedEventHandler();
```

Abbildung 1. Der EventHandler-Delegat

Nun müssen wir unser Event in der Auslöser-Klasse namens **Uploader** definieren. Das geschieht mithilfe des Schlüsselworts „**event**“, dem anschließenden Zuweisen des EventHandler-Delegaten und dem vergeben eines Event-Bezeichners. Zu sehen ist das Ganze auf dem folgenden Codeschnipsel:

```
class Uploader
{
    //Event
    public event VideoUploadedEventHandler VideoUploaded;
}
```

Abbildung 2. Die Definition des Events in der Klasse "Uploader"

Wir haben nun also unseren EventHandler-Delegaten und das eigentliche Event definiert. Nun können wir dieses auch an einer beliebigen Stelle in der Klasse auslösen. Zum Beispiel am Ende einer Methode, welche ein Video hochlädt:

```

class Uploader
{
    //Event
    public event VideoUploadedEventHandler VideoUploaded;

    //Methoden
    public void VideoUpload()
    {
        Console.WriteLine("Upload läuft...");
        Console.WriteLine("Upload erfolgreich abgeschlossen!");
        VideoUploaded(); //Löst das Event aus
    }
}

```

Abbildung 3. Definition einer Methode welche das Event am Ende auslöst

Das Event ist nun also fertig definiert und wird auch ausgelöst. Der Publisher des Events hat nun also all seine Aufgaben erledigt, widmen wir uns dementsprechend noch den Abonnenten.

AUF EVENTS REAGIEREN

Andere Klassen können auf ein Event lauschen, indem sie eine Handler-Methode implementieren. Fügen wir zu unserem Beispiel noch eine weitere Klasse hinzu, in der wir genau das machen:

```

class Nachrichten
{
    public void VideoUploaded()
    {
        Console.WriteLine("Das Video wurde hochgeladen!");
    }
}

```

Abbildung 4. Definition einer neuen Klasse, die auf das Event lauschen wird

Die neue Klasse hat den Namen **Nachrichten** und sie besitzt eine Methode namens „**VideoUploaded()**“. Diese Methode ist der EventHandler, mit welcher auf das ausgelöste **VideoUploaded-Event** aus der Uploader-Klasse reagiert wird. Jetzt besteht zwischen dem Uploader und der Nachrichten-Klasse allerdings noch keine Verbindung. Um auf das Event reagieren zu können, muss dieses erst einmal von der „VideoUploaded()“ Methode aus der Nachrichten-Klasse abonniert werden. Das funktioniert folgendermaßen:

```

static void Main(string[] args)
{
    //Erstelle Objekte
    Uploader videoUploader = new Uploader();
    Nachrichten benachrichtiger = new Nachrichten();

    //Abonniere das Event
    videoUploader.VideoUploaded += benachrichtiger.VideoUploaded;
}

```

Wie du siehst, funktioniert das Abonnieren eines Events wie das Zuweisen einer Methode zu einem Multicast Delegaten. Wir müssen lediglich Objekte unserer Klassen erstellen und dann dem „VideoUploaded-Event“ aus der Uploader-Klasse unsere EventHandler-Methode aus der Nachrichten-Klasse hinzufügen. Das Ganze erfolgt mithilfe des „+=“ Operators. Wenn wir das Event nun auslösen, werden gleichzeitig alle EventHandler ausgelöst.

```
static void Main(string[] args)
{
    //Erstelle Objekte
    Uploader videoUploader = new Uploader();
    Nachrichten benachrichtiger = new Nachrichten();

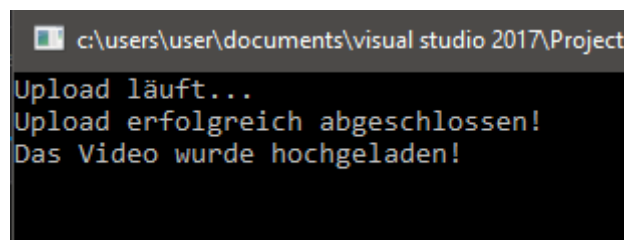
    //Abonniere das Event
    videoUploader.VideoUploaded += benachrichtiger.VideoUploaded;

    //Löse das Event aus
    videoUploader.VideoUpload();

    //Halte das Programm an
    Console.ReadKey();
}
```

Abbildung 5. Auslösen des Events

Das Ergebnis sieht dann so aus:



```
c:\users\user\documents\visual studio 2017\Projects
Upload läuft...
Upload erfolgreich abgeschlossen!
Das Video wurde hochgeladen!
```

Abbildung 6. Das Endresultat

SICHERERES AUSLÖSEN DES EVENTS

Im obigen Beispiel haben wir das Event auf eine sehr einfache und zugegebenermaßen unsaubere Art ausgelöst. Wir hatten eine „VideoUpload()“-Methode in der Uploader-Klasse und in dieser haben wir am Ende einfach das Event ausgelöst, ohne zu überprüfen, ob dieses überhaupt abonniert wurde. Das sollte man in der Regel vermeiden. Am besten ist es, wenn man das Event über eine eigene „Auslöser-Methode“ auslöst und in diese eine Sicherheitsabfrage miteinbaut. Wie man das macht, siehst du auf dem folgenden Codeschnipsel, welcher eine modifizierte Version unseres vorherigen Beispiels zeigt:

```

//Methoden
public void VideoUpload()
{
    Console.WriteLine("Upload läuft...");
    Console.WriteLine("Upload erfolgreich abgeschlossen!");
    OnVideoUploaded(); //Führe Eventauslöser aus
}

//Eventauslöser-Methode
protected virtual void OnVideoUploaded()
{
    if (VideoUploaded != null)
    {
        VideoUploaded(); //Löste Event aus
    }
}

```

Abbildung 7. Implementierung einer Auslöser-Methode

Anstatt das Event innerhalb der „VideoUpload()“-Methode auszulösen, machen wir dies nun in der „OnVideoUploaded()“-Methode. Eine Auslöser-Methode hat für gewöhnlich denselben Namen wie das Event, mit einem „On“ am Anfang. Innerhalb dieser „OnVideoUploaded()“-Methode befindet sich ein kleines If-Statement, in welchen gefragt wird, ob das Event „VideoUploaded“ auch abonniert wurde. Wir fragen also ab, ob „VideoUploaded“ ungleich **null** ist. Wenn es den Wert null hätte, dann würde das bedeuten, dass kein Objekt auf das Event lauscht und dementsprechend würde es auch nicht ausgelöst werden. Innerhalb des If-Statements wird das Event dann tatsächlich ausgelöst.