Events

Ingo Köster

Diplom-Informatiker (FH)

Events

- Objekte reagieren auf Anstöße von außen
- Beispiel
  - Von einem Objekt vom Typ `Auto` wird die Methode `Fahren()` aufgerufen
- Objekte können auf diese Anstöße ihrerseits selbst reagieren

Events

- Ein Methodenaufruf ist ein Verfahren, mit dem ein Aufrufer einem Objekt einen Anstoß gibt, damit dieses eine bestimmte Verhaltensweise zeigt
- Als Konsequenz eines Methodenaufrufs könnte das Objekt seinerseits bei seinem Aufrufer eine Reaktion auslösen
  - Und/Oder bei anderen Objekten
- Diese Reaktion wird als Ereignis (Event) bezeichnet

Beispiel

- Beispiel
  - Klasse: `Aktie`
  - Attribute: `Name` und `Preis`

- Preis kann über Property geändert werden

Beispiel

```
class Aktie
{
  string name;
  int preis;

  public int Preis
  {
    get { return preis; }
    set { preis = value; }
  }
}
```

Beispiel

- Andere Objekte möchten darüber informiert werden, wenn sich der Preis der Aktie ändert
- Jedes Objekt möchte individuell auf die Preisänderung reagieren können
- Anlegen eines Delegates (außerhalb der Klasse)
  - `delegate void PreisÄnderungsEventHandler(int alterPreis, int neuerPreis);`
- Anlegen eines Events (innerhalb der Klasse)
  - `public event PreisÄnderungsEventHandler KursÄnderung;`

Beispiel - Property

```
public event PreisÄnderungsEventHandler KursÄnderung;

public int Preis
{
  get { return preis; }
  set
  {
    if (preis == value) { return; }
    // Auslösen des Events bei Preisänderung
    KursÄnderung(preis, value);
    preis = value;
  }
}
```

Broadcaster & Subscriber

- Broadcaster
  - Objekt das ein Delegate enthält
  - Schickt ggf. einen Broadcast, indem das Delegat aufgerufen wird

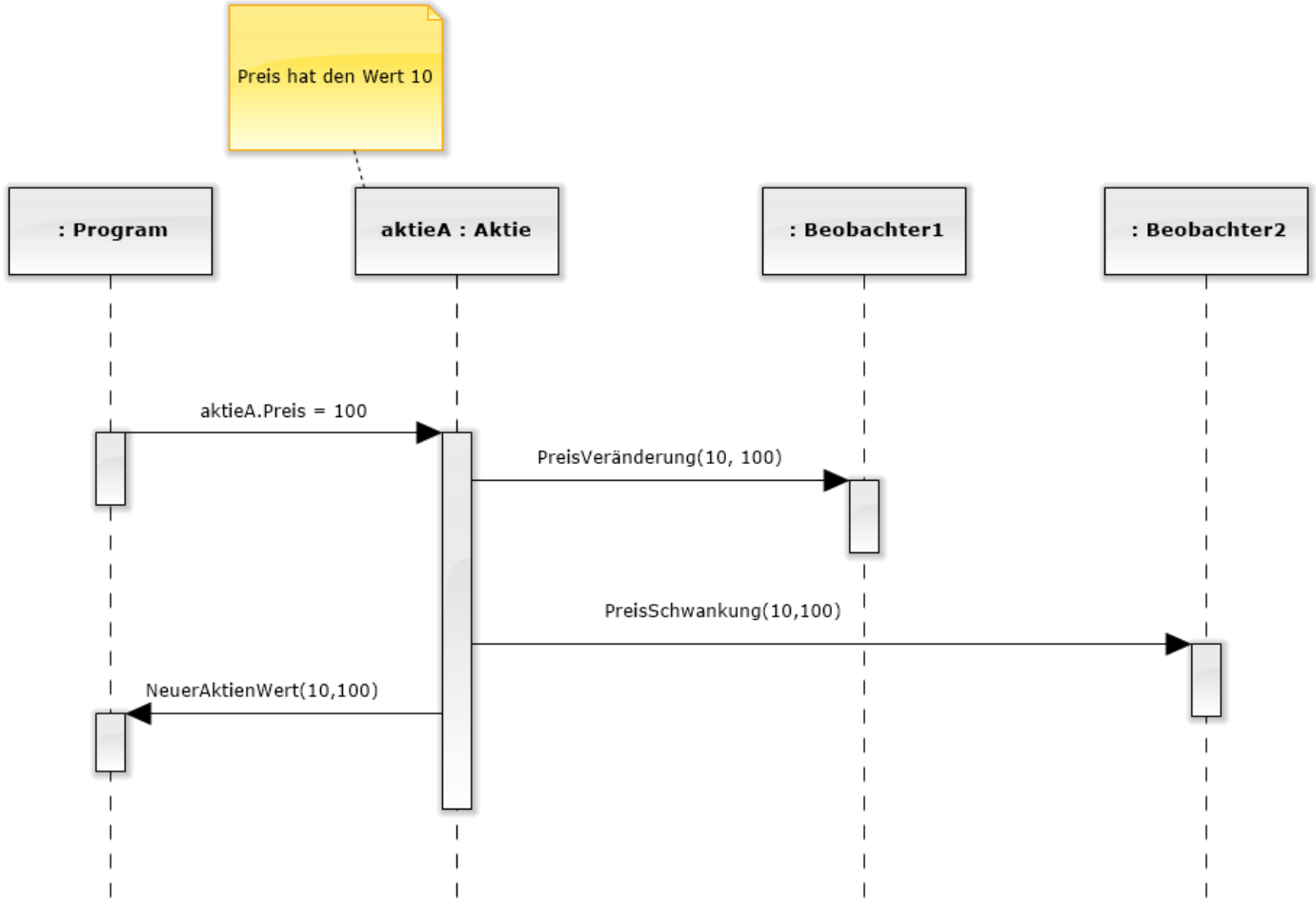
- Subscriber
  - Zielmethodenempfänger
  - Subscriber entscheidet von wann bis wann er zuhört
    - Mittels += und -=

Subscriber

```
static void Main(string[] args)
{
  Aktie aktieA = new Aktie("A", 10);
  aktieA.KursÄnderung += NeuerAktienWert;
  aktieA.Preis = 190;
}

static void NeuerAktienWert(int alt, int neu)
{
  Console.WriteLine("Preis Alt: {0} Neu {1}", alt, neu);
}
```

Methodenaufruf durch ausgelösten Event



Vorteile eines Events

- Probleme mit Delegaten:
  - Ersetzen anderer Subscriber durch erneutes Zuweisen mittels **=**
    - anstatt mit += (Multicast Delegate)
  - Entfernen aller Subscriber indem Delegat auf `null` gesetzt wird
  - Erzeugen eines Broadcasts durch Aufruf des Delegaten von einem Subscriber

Keine Behandlung des Events

› Beim Aufrufer muss das von einem Objekt ausgelöste Ereignis nicht zwangsläufig an einen Ereignishandler gebunden werden

› Vor der Auslösung eines Ereignisses sollte daher geprüft werden, ob ein Ereignisempfänger überhaupt die Absicht hat, auf das Ereignis zu reagieren

```
if (KursÄnderung != null)
{
   KursÄnderung(preis, value);
}
```

Ereignisse mit Übergabeparameter

- Die Referenz auf das auslösende Objekt ist im Ereignishandler nicht bekannt

- Im Beispiel Aktie:
  - In Methode `GeänderterPreis()` ist die auslösende Aktie nicht bekannt

- Ein Ereignishandler kann von mehreren Objekten benutzt werden
  - Auch von unterschiedlichen Typen

Ereignisse mit Übergabeparameter

```
public delegate void InvalidRadiusEventHandler(Circle einKreis);
...
public double Radius {
  get { return radius; }
  set {
    if(value >= 0) { radius = value; }

    else if(InvalidRadius != null)
    {
      InvalidRadius(this);
    }
  }
}
```

Ereignisse mit Übergabeparameter

› Der Ereignishandler stellt einen Parameter bereit, in dem das ereignisauslösende Objekt die Referenz auf sich selbst übergibt

```
public void kreis_InvalidRadius(Circle sender)
{
  Console.WriteLine("Negativer Radius");
  Console.Write("Neueingabe: ");
  sender.Radius =  Convert.ToDouble(Console.ReadLine());
}
```

Ereignishandler im .NET Framework

- Alle Ereignishandler im .NET Framework weisen zwei Parameter auf:
  - 1. Parameter: Das auslösende Objekt
  - 2. Parameter: Ereignisspezifische Daten

- Der erste Parameter ist immer vom Typ `Object`
  - Ereignisse sollen mehreren unterschiedlichen Ereignisdefinitionen zur Verfügung stehen

Ereignishandler im .NET Framework

› Für ereignisspezifische Daten werden spezielle Klassen benötigt, die von der Klasse EventArgs abgeleitet werden

```
class InvalidRadiusEventArgs : EventArgs
{
  private double radius;
  public double Radius { get { return radius; } }
  public InvalidRadiusEventArgs(double radius)
  {
    this.radius = radius;
  }
}
```

Ereignishandler im .NET Framework

```
delegate void InvalidRadiusEventHandler(Object sender,
InvalidRadiusEventArgs e);


public double Radius {
  get { return radius; }
  set {
    if (value >= 0) { radius = value; }
    else {
      if (InvalidRadius != null) {
        InvalidRadius(this, new InvalidRadiusEventArgs(value)); }
    }
  }
}
```

Ereignishandler im .NET Framework

```
void einKreis_InvalidRadius(object sender,
InvalidRadiusEventArgs e)
{
  Console.WriteLine("Ein Radius von {0} ist nicht
zulässig.", e.Radius);


 Console.Write("Neueingabe: ");


  ((Circle)sender).Radius =
Convert.ToDouble(Console.ReadLine());
}
```

Namenskonventionen

- Delegatenname für einen Event endet auf `EventHandler`
  - Wird großgeschrieben
  - `delegate void PreisÄnderungs`**`EventHandler`**`(…`
- Eventname wird großgeschrieben
  - `public event PreisÄnderungsEventHandler` **`KursÄnderung;`**
- EventArgs Klassename endet mit `EventArgs`
  - `public class InvalidRadius`**`EventArgs`** `: EventArgs`