

## C# 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"
```

## C# 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

- 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.

## C# 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"
```

## C# Event-Konventionen in .NET

---

Delegates für das Event-Handling sollten folgende Signatur haben

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

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

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

## C# Beispiel für .NET-Events

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

public class EventArgs : 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 EventArgs(...));
    }
}
```

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