Delegaten

Einführung

delegate ist ein Objekt, das

delegiert == weiterleitet

d. h. eine Tätigkeitsanforderung wird weitergeleitet

- **Weiterleitung erfolgt an eine Funktion**
- **diese Funktion(en) wurde dem Delegat zugeordnet**
- **Klient stellt Anforderung an Delegaten**

Delegaten

Begriffsklärung

- ist Methodentyp (Referenztyp), der der Deklaration von Variablen dient, die Verweise auf Methoden enthalten
- ähnelt einem Funktionszeiger in C++

```
                              // Definition der Delegate-Variablen
public delegate double RechenMethode(double d);
public static double quadrat(double d){return d*d;}
. . .
RechenMethode rechne = new RechenMethode(quadrat); // Init.
Double erg = rechne(5.1);     // Aufruf
```

- Der Variablen vom Delegatentyp kann jede (auch statische) Funktion zugewiesen werden, die den gleichen Rückgabetyp und die gleichen Übergabeparameter hat einschließlich der Parameterarten (params, out, ref)

Delegaten

Verwendung

- Deklaration eines Delegaten („Funktionszeiger“)

```
delegate Rückgabetyp DelBezeichner(Übergabeparameter);
  // Definition des Typs mit Schlüsselwort delegate
  // kann auf Methode mit gleichen Übergabeparametern
  // und gleichem Rückgabetyp verweisen
```

- Erzeugung einer Delegate-Variablen:

```
DelBezeichner delVar;
```

- Zuweisen einer passenden Methode

```
delVar = new DelegateTyp(Method);
 mit
   Method ist statische Methode
  oder Instanzmethode
     this.Method oder Klasse.Method
```

Delegaten

Delegate vs. Interface

- Interface
  - definiert Über- und Rückgabeparameter und Methodennamen
  - wird über Ableitung (Vererbung) verwendet
    => enge Kopplung

- Delegate
  - definiert nur Über- und Rückgabeparameter
  - entkoppelt deklarierende und verwendende Klasse
  - jede Methode (Klassen- oder Instanzmethode),die erforderliche Signatur erfüllt kann (zur Laufzeit) verwendet werden

Delegaten

Multicast Delegaten

- eine Variable des Delegatentyps kann auf mehrere Funktionen verweisen (Multicast)
- beim Aufruf werden alle Methoden nacheinander ausgeführt (Ergebniswerte werden bis auf letzten verworfen)
- Delegatvariablen, die null enthalten werfen beim Aufruf eine NullReferenzException
- der Funktionswert des Aufrufs ist der Rückgabewert der letzten Funktion (analog bei out Parametern)
- Anhängen erfolgt mit dem Operator +=
- Entfernen erfolgt mit dem Operator -=
- Multicast Delegaten werden für Event-Handling benutzt (Publish-Subscriber Pattern)
- Ausnahmen in Multicast-Delegaten werden beachtet

Delegaten

Klasse MulticastDelegate

- Ist Delegate mit Aufrufliste für mehrere Methoden
- Alle Delegattypen sind hiervon abgeleitet
- MulticastDelegate wiederum ist von der Klasse Delegate abgeleitet

- Klassenelemente:
  - GetInvocationList(): Gibt Aufrufliste zurück
  - Target: Instanz des aktuellen Delegaten (falls nicht statisch)
  - Method: Methode des aktuellen Delegaten
  - . . .

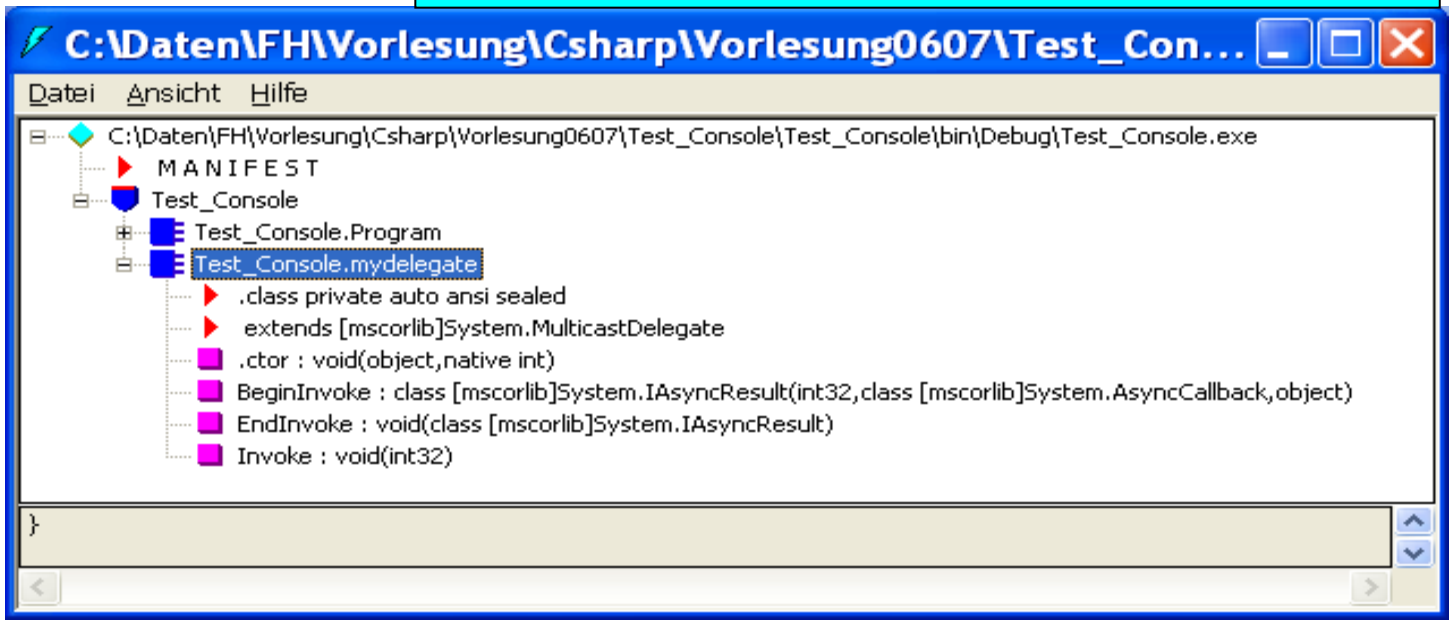
- Mit GetInvocationList() können gezielt einzelne Delegaten aus der Liste aufgerufen werden

Delegaten

Delegaten intern

- ildasm von:

```
delegate void mydelegate(int i);
```



- Compiler erzeugt eine vom MulticastDelegate abgeleitete Klasse

Delegaten

Anonyme Methoden

- sind eine Art Kurzschrift zur Durchführung von Funktionen, die mit einem Delegaten verbunden sind

- bieten im Wesentlichen eine Möglichkeit, einen Codeblock als Delegatparameter zu übergeben

- durch anonyme Methoden wird der Codierungsaufwand des Instanziierens von Delegaten für den Entwickler reduziert, weil keine separaten Methoden mehr erstellt werden müssen
- der Leistungszuwachs liegt nur in der Produktivitätssteigerung des Entwicklers aber nicht bei der Ausführung der Anwendung

Delegaten

Anonyme Methoden

```
class Program
{
   delegate void MyDelegate(int i);
   static void Main(string[] args)
   {  MyDelegate d1 = delegate(int i){Console.WriteLine(i);};
      d1(8); // Aufruf der namenlosen Methode
      MyDelegate d2 = delegate {Console.WriteLine("text");};
      d2(8); // ohne Parameter der Rückrufmethode

   }
```

- Delegate ermöglicht inline-Definition einer Methode ohne Namen und Zuweisung an Delegatinstanz
- Es brauchen keine Parameter für die Rückrufmethode definiert werden, falls diese nicht verwendet werden
- in Java ab Version 8 auch verfügbar

Delegaten

Lambda Ausdrücke

- sind Konstrukt aus der funktionalen Programmierung

- erlauben Notation einer anonymen Funktion in kompakter, prägnanter Form

```
. . .
delegate int myDelegate (int i, int j);
myDelegate add1 = (i,j) => i + j;   // Lambda-Ausdruck
// Parametertypen werden automatisch hergeleitet
// entspricht:
myDelegate add2 = (int i,int j) => {return i + j;}
```

- Syntax: Eingabeparameter => Ausdruck
  Beispiel: - Mathematik f(x,y) = x + y
  - Lambda-Ausdruck (x,y) => x + y
- intensive Verwendung in LINQ

Delegaten

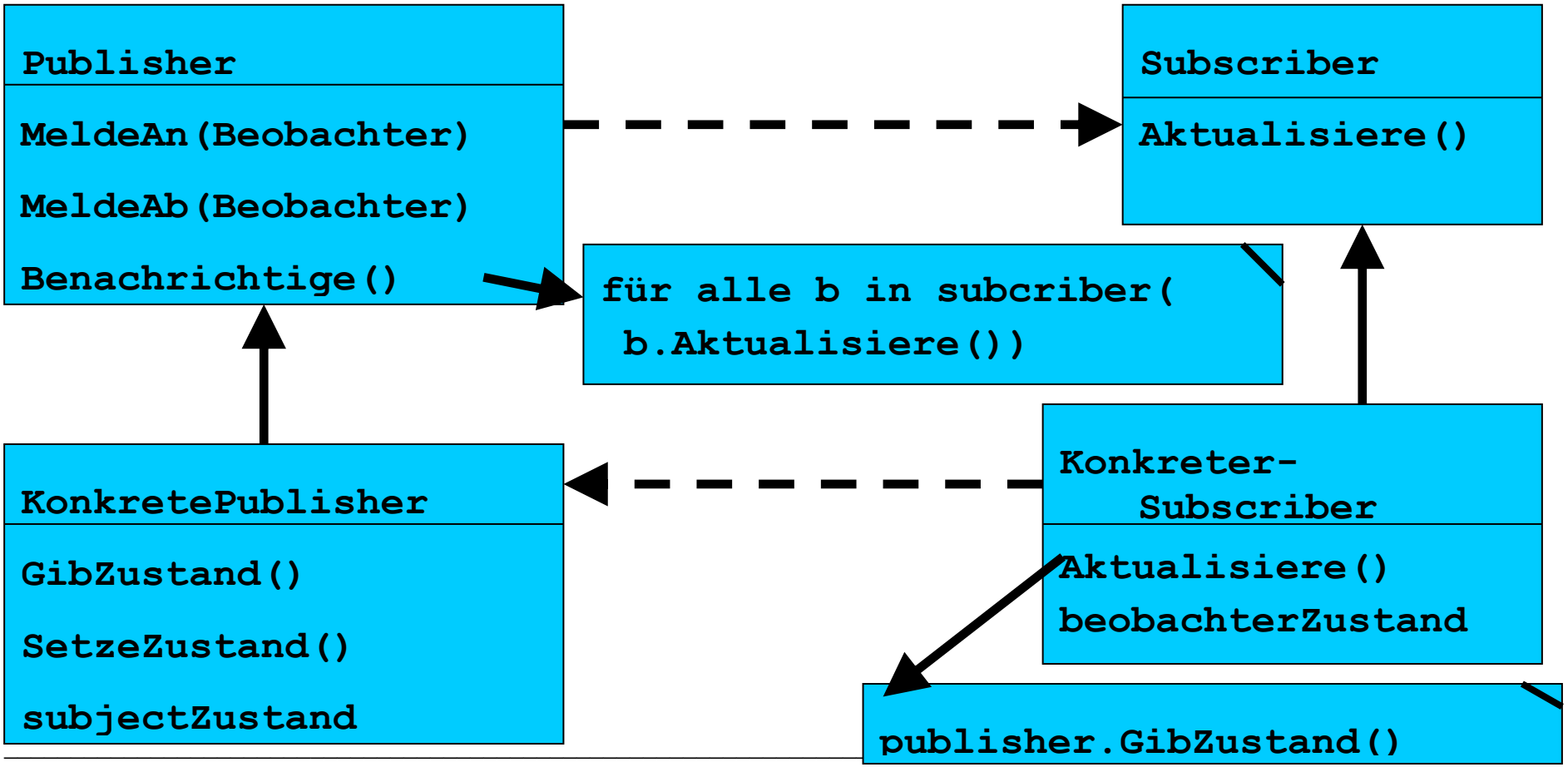
Ko- und Kontravarianz bei Delegaten

- Delegaten die als Rückgabetyp den Basistyp erwarten können auch auf Methoden zeigen die einen abgeleiteten Typ zurückgeben
  (Kovarianz)

- Delegaten die als Argument einen abgeleiteten Typ erwarten können auch auf Methoden zeigen welche als Argument den Basistyp erwarten
  (Kontravarianz)

Delegaten

Beobachter (Publish-Subscriber)



Delegaten

Events

- sind Delegate-Variablen, die als Felder deklariert werden
- Eventhandling basiert auf dem Beobachtermuster und wird mit Multicast-Delegaten gehandbabt
- Schlüsselworte: delegate und event
- mit dem Schlüsselwort event wird ein Feld für ein Ereignis definiert
- event veranlasst den Compiler Einschränkungen des Delegaten für Ereignisse zu überprüfen,
  Einschränkungen:
  - Ereignis kann nur von der umschliessenden Klasse ausgelöst werden (Schutzmechanismus!)

  Aber: ersetzt man im Code event durch delegate so bleibt das Programm ausführbar