Delegates

Ingo Köster

Diplom-Informatiker (FH)

Delegate

- Englisches Wort für „Delegierter“
  - Jemand, der etwas weiterleiten soll
- Ein Delegate leitet einen Methodenaufruf an eine bestimmte Methode weiter
- Sozusagen eine Variable in der ein Verweis auf eine Methode gespeichert wird
- In der Sprache C auch als Funktionszeiger bekannt

Beispiel – Klasse mit Methoden

```
class Demo
{
  public static int Addition(int x, int y)
  {
    return x + y;
  }

  public static int Subtraktion(int x, int y)
  {
    return x - y;
  }
}
```

Beispiel – In der Main-Methode

› Delegaten Typen deklarieren (außerhalb von Klassendeklarationen)

```
delegate int CalculateHandler(int wert1, int wert2);
```

› Variable vom Typ des Delegaten anlegen (innerhalb der Klasse)

```
CalculateHandler calculate;
```

Beispiel – In der Main-Methode

› Delegaten-Variable zeigt auf Addition…

```
calculate = new CalculateHandler(Demo.Addition);
```

› …oder Subtraktion

```
calculate = new CalculateHandler(Demo.Subtraktion);
```

› Aufruf der gewählten Methode erfolgt über das Delegate

```
› int ergebnis = calculate(zahl1, zahl2);
```

Methodenverweis

- In einer Variablen einen Verweis auf die aufzurufende Methode speichern
  - `calculate = new CalculateHandler(Demo.Subtraktion);`

- Später an einer beliebigen Stelle im Code die Variable auswerten
  - `int ergebnis = calculate(zahl1, zahl2);`

Methodenverweis

- Ein Delegate kapselt einen Zeiger auf eine Methode
  - Steht prinzipiell für einen beliebigen Methodenaufruf
  - Jede Methode hat eine definierte Parameterliste mit Parametern eines bestimmten Typs

- Die Typen der Parameterliste der Methode, auf die das Delegat zeigt, müssen mit der Parameterliste der Delegat-Definition übereinstimmen

Aus dem Beispiel

- Methode
  - `public static int Addition(int x, int y) { … }`
- Delegate
  - `delegate int CalculateHandler(int wert1, int wert2);`

- Signaturen müssen übereinstimmen und die Methode muss zugreifbar sein
  - Daher kein Problem durch das `static`

Allgemeines Delegate

- Delegate auf alle Methoden die parameterlos sind und keinen Rückgabewert haben
  - `delegate void EinMethodenHandler();`

- Delegaten können das Schlüsselwort `params` verwenden

```
delegate void EinMethodenHandler(params int[] einArray);

…

static void Addition(params int[] summanden) { … }
```

Vereinfachter Aufruf eines Delegaten

- Anstelle von:
  - `CalculateHandler calculate = new CalculateHandler(Demo.Addition);`

- Wie folgt:
  - `CalculateHandler calculate = Demo.Addition;`

Multicast-Delegate

- Mehrere Delegaten können zu einem einzigen zusammengefasst werden
- Dadurch entsteht ein Delegaten-Verbund, der als Multicast-Delegate bezeichnet wird
- Durch den Aufruf eines Multicast-Delegate können mehrere Delegaten bzw. Methoden ausgeführt werden

Multicast-Delegate

- `delegate int BerechnungsHandler(int x, int y)`

- Methoden aus der Klasse Rechnen
  - `Add(int x, int y), Sub(…), Mul(…) und Div(…)`

- `BerechnungsHandler a = Rechnen.Add;`
- `BerechnungsHandler b = Rechnen.Sub;`
- `BerechnungsHandler c = a + b;`
- `BerechnungsHandler d = c + Rechnen.Mul;`

Multicast-Delegate

- Mittels += Operator weitere Delegaten einem Delegate hinzufügen
  - `BerechnungsHandler a = Rechnen.Add;`
  - `BerechnungsHandler b = Rechnen.Sub;`
  - `BerechnungsHandler c += a;`
  - `c += Rechnen.Add;`

- Mittels -= Operator Delegaten entfernen
  - `c -= Rechnen.Sub;`
    - Bei doppelten wird das Letzte entfernt
    - Entfernen von nicht vorhandenen Delegates wird ignoriert

Multicast-Delegate

```
BerechnungsHandler a = Rechnen.Add;
BerechnungsHandler b = Rechnen.Sub;
BerechnungsHandler c = a + b;
BerechnungsHandler d = c + Rechnen.Mul;
Console.WriteLine(d(5, 6));
```

- Ausgabe: 30
- Rückgabe nur von letztem Aufruf
  - Alle anderen Rückgabewerte werden verworfen (Lösen z.B. mittels out)

Delegate als Übergabeparameter

```
void Umwandeln(double[] werte, EinDelegateHandler
dieMethode)
{
  for (int i = 0; i < werte.Length; i++)
  {
    werte[i] = dieMethode(werte[i]);
  }
}
```

Delegate als Parameter

› Aufruf:

```
delegate double EinDelegateHandler(double x);
…
Rechnen.Umwandeln(zahlen, Rechnen.Wurzelziehen);
```

› Methode:

```
class Rechnen {
 public static double Wurzelziehen(double x) {
  return Math.Sqrt(x);
 }
}
```

Funktionen höherer Ordnung

- Die Methode `Umwandeln` aus dem Beispiel ist eine Funktion höherer Ordnung
  - *higher-order function*

- Funktionen höherer Ordnung nehmen andere Funktionen als Übergabeparameter entgegen oder liefern eine Funktion als Rückgabewert zurück

Methoden von Delegaten

- Jeder Delegate ist von der Klasse `Delegate` abgeleitet

- Methoden & Eigenschaften
  - `GetInvocationList()`
    - Liefert Aufrufliste eines Delegaten als Array
    - `Delegate[] liste = eineDelegate.GetInvocationList();`
  - `Target`
    - Liefert Objekt-Typ einer Methode im Delegat
  - `Method`
    - Liefert Signatur einer Methode im Delegat

Methoden von Delegaten (Beispiel)

```
› BerechnungsHandler a = Rechnen.Add;
› BerechnungsHandler b = Rechnen.Sub;
› BerechnungsHandler c = a + b;

› Delegate[] liste = c.GetInvocationList();
› liste[1].Target
› liste[1].Method
```

Anonyme Methoden

- Ggf. dienen Methoden nur dazu, sie mit einem bestimmten Delegate zu verknüpfen
- Code kann direkt mit einem Delegate verknüpft werden
- Code hat dann keinen Methodenbezeichner mehr und wird als anonyme Methode bezeichnet

Anonymen Methoden - Beispiel

```
delegate int CalculateHandler(int ivar1, int ivar2);
…
calculate = delegate(int x, int y)
{
  return x + y;
};    <- !!!
…
int result = calculate(zahl1, zahl2);
```

Anonymen Methoden - Beispiel

```
delegate void SimpleHandler();      // Deklaration

SimpleHandler einDelegate;          // Delegate-Objekt

// Keine runden Klammern
einDelegate = delegate
{
    Console.WriteLine("Hallo Welt");
};

einDelegate();                      // Aufruf
```

Multicast mit anonymen Methoden

```
calculate = delegate(int x, int y)
{
 return x + y;
};

calculate += delegate(int x, int y)
{
      return x * y;
};

// Ruft Addition und Multiplikation auf
calculate(5,5);
```