DELEGATEN

WAS SIND DELEGATEN?

In den letzten Videos ging es um das Thema Delegaten. Fassen wir das Ganze mal kurz zusammen!

Ganz einfach gesagt handelt es sich bei Delegaten um Funktionszeiger. Man kann mithilfe von Delegaten eigene Typen definieren, die eine Referenz zu einer Methode speichern können. Delegaten ermöglichen also das Speichern von Methodenreferenzen in einer Variable, über welche man diese dann jederzeit aufrufen kann.

DIE DEFINITION EINES DELEGATEN

Bei der Definition eines Delegaten wird der Rückgabe-Typ und die Methodensignatur angegeben. Damit eine Methode dann in den Delegaten gespeichert werden kann, muss deren Signatur mit der des Delegaten übereinstimmen. Hier mal ein Beispiel einer Delegaten-Definition:

```
public delegate int Berechne(int zahl1, int zahl2);
```

Abbildung 1. Eine Delegaten-Definition

Wie du siehst, ähnelt die Definition eines Delegaten der Kopfzeile einer normalen Methode. Der einzige Unterschied ist, dass wir keinen Methodenkörper schreiben und das Schlüsselwort „**delegate**“.

Um nun eine Methode in diesem Delegaten speichern zu können, muss diese natürlich kompatibel sein. Hier mal ein Beispiel für eine mit dem oben genannten Delegaten kompatible Methode:

```
static int Addition(int zahl1, int zahl2)
{
    return zahl1 + zahl2;
}
```

Abbildung 2. Eine kompatible Methode zum "Berechne" Delegaten aus Abbildung 1

Diese Methode kann man in einer Variable unseres „Berechne“ Delegaten speichern. Hier siehst du wie das Ganze aussehen könnte:

```
static void Main(string[] args)
{
    Berechne rechenoperation = new Berechne(Addition);
    int ergebnis = rechenoperation(10, 20);

    Console.WriteLine(ergebnis);
    Console.ReadKey();
}
```

Abbildung 3. Die Nutzung des "Berechne" Delegaten mit der Methode "Addition()"

Wie du siehst erstellen wir zuerst eine Variable vom Typ „Berechne“. Bei der Erstellung des „Berechne“ Objekts wird der Konstruktor aufgerufen und diesem übergeben wir unsere gewünschte Methode, die wir in das Objekt

speichern wollen. Über die Variable unseres Delegaten-Typs können wir nun jederzeit die Methode aufrufen, die wir darin gespeichert haben.

WANN VERWENDET MAN DELEGATEN?

Jetzt stellt sich natürlich noch die Frage, wann man Delegaten denn überhaupt verwendet. Man verwendet Delegaten eigentlich hauptsächlich in Verbindung mit den sogenannten Events, auf welche ich im nächsten Modul zu sprechen komme. Allerdings eignen sich Delegaten auch dann sehr gut, wenn zur Zeit der Programmierung noch nicht genau bekannt ist, welche Methode man überhaupt zur Laufzeit aufrufen möchte.

MUTLICAST DELEGATEN

Jetzt gibt es neben den einfachen Delegaten auch noch die Multicast Delegaten. Diese sind eigentlich das exakt selbe wie normale Delegaten, nur mit dem kleinen Unterschied, dass sie mehrere Methoden gleichzeitig beinhalten und diese beim Aufruf des Delegaten auch alle hintereinander ausführen. Um aus einem normalen Delegaten einen Multicast Delegaten zu machen, müssen wir lediglich mit dem **„+=“** Operator eine weitere Methode zu diesem Hinzufügen.

```
public delegate void Ausgabe();
```

Abbildung 4. Definition eines Delegaten

```
static void Begrüßung()...
static void Smalltalk()...
static void Verabschiedung()...
```

Abbildung 5. Definition kompatibler Methoden

```
static void Main(string[] args)
{
    Ausgabe gespräch = new Ausgabe(Begrüßung);
    gespräch += Smalltalk;
    gespräch += Verabschiedung;

    gespräch();
    Console.ReadKey();
}
```

Abbildung 6. Erstellung eines Delegaten-Objektes

In diesem Beispiel werden die Methoden „Begrüßung()“, „Smalltalk()“ und „Verabschiedung()“ hintereinander ausgeführt.