Grundlagen II

In dieser Kurseinheit vertiefen Sie Ihre Kenntnisse der objektorientierten Programmierung mit C#. Sie lernen mit den Delegaten und den darauf aufbauenden Ereignissen zwei Schlüsselkonzepte für die Entwicklung von performanten und interaktiven Systemen kennen. Sie erfahren, wie Sie Laufzeitfehler erkennen und behandeln können und wie Sie von C# und von Visual Studio beim Debugging unterstützt werden.

Vererbung, Polymorphie und Interfaces

Lesen Sie bitte **Kapitel 4** des Lehrbuchs.

Die objektorientierte Programmierung baut auf drei Säulen auf: Datenkapselung, Vererbung und Polymorphie. Nachdem Sie in Kapitel 3 bereits mit den Eigenschaftsmethoden eine wichtige Möglichkeit zur Datenkapselung kennengelernt haben, lernen Sie in diesem Kapitel die anderen beiden Konzepte kennen. Auch hier gibt es einige Unterschiede im Vergleich zu Java.

1. In Java kann jede Methode überschrieben werden, außer sie ist mit »sealed« markiert. In C# kann keine Methode überschrieben werden, außer sie ist mit »virtual« oder »abstract« markiert.
2. Eine ableitende Klasse hat in C# drei Alternativen, wenn in der Oberklasse eine Methode als »virtual« deklariert wird: Sie erbt die Methode ohne eine eigene, typspezifische Implementierung vorzusehen oder sie verdeckt die geerbte Methode mit »new« oder sie überschreibt die geerbte Methode mit »override«.
3. Ein mit »protected« geschützter Member ist innerhalb seiner Klasse und für abgeleitete Klasseninstanzen zugreifbar. In Java hingegen wird damit nicht nur Subklassen der Zugriff erlaubt, sondern auch allen Klassen aus demselben Package.
4. Auf ein »internal« Element darf nur innerhalb der aktuellen Assembly zugegriffen werden. Eine Assembly in .NET Framework entspricht in etwa einer JAR-Datei in Java. Eine vergleichbare Einschränkung für JAR-Files in Java gibt es nicht.
5. Schnittstellen in C# können Methoden, Eigenschaften, Ereignisse und Indexer vorschreiben. Die ersten beiden Elemente kennen Sie schon. Ereignisse werden in Kapitel 5 und Indexer in Kapitel 10 behandelt. Konventionsgemäß wird dem Bezeichner einer Schnittstelle ein »I« vorangestellt.

Delegaten und Ereignisse

Lesen Sie bitte **Kapitel 5** des Lehrbuchs.

Ereignisse in C# basieren auf dem Observer-Pattern und dienen vorrangig der Interaktion zwischen einer Benutzeroberfläche und dem Anwender. Die Klasse, die das Ereignis sendet (oder auslöst), wird als *Herausgeber* bezeichnet und die Klassen, die das Ereignis empfangen (oder behandeln), werden als *Abonnenten* bezeichnet. Ereignisse verfügen über folgende Eigenschaften:

1. Der Herausgeber bestimmt, wann ein Ereignis ausgelöst wird; die Abonnenten bestimmen, welche Maßnahme als Reaktion auf das Ereignis ergriffen wird.

2. Ein Ereignis kann mehrere Abonnenten haben. Ein Abonnent kann mehrere Ereignisse von mehreren Herausgebern behandeln.
3. Ereignisse, die keine Abonnenten haben, werden nie ausgelöst.
4. Ereignisse dienen normalerweise zur Signalisierung von Benutzeraktionen wie das Klicken auf Schaltflächen oder Auswählen von Menüs in der grafischen Benutzeroberfläche.
5. Wenn ein Ereignis mehrere Abonnenten hat, werden die Ereignishandler synchron aufgerufen, wenn ein Ereignis ausgelöst wird.
6. In der .NET Framework-Klassenbibliothek basieren Ereignisse auf dem EventHandler-Delegaten und der EventArgs-Klasse.

Ein Delegat ist ein Objekt, das den Zeiger auf die Methode eines anderen Objekts enthält. Man kann Delegaten daher als typisierte Funktionszeiger bezeichnen. Jede Methode einer beliebigen verfügbaren Klasse oder Struktur, die mit dem Delegattyp übereinstimmt, kann dem Delegaten zugewiesen werden. Bei der Methode kann es sich um eine statische Methode oder um eine Instanzenmethode handeln. Diese Fähigkeit, als Parameter auf eine Methode zu verweisen, macht Delegaten ideal für das Definieren von Rückrufmethoden. Delegaten verfügen über folgende Eigenschaften:

1. Delegaten ähneln C++-Funktionszeigern, sind aber typsicher.
2. Delegaten ermöglichen es, Methoden als Parameter zu übergeben.
3. Delegaten können zum Definieren von Rückrufmethoden verwendet werden.
4. Delegaten können miteinander verkettet werden. So können beispielsweise mehrere Methoden für ein einziges Ereignis aufgerufen werden.
5. In C#, Version 2.0, wurde das Konzept der anonymen Methoden eingeführt, durch die anstelle einer separat definierten Methode Codeblöcke als Parameter übergeben werden können. In C# 3.0 wurden Lambda-Ausdrücke als eine präzisere Methode zum Schreiben von Inlinecodeblöcken eingeführt.

Delegaten spielen in der .NET-Programmierung eine wichtige Rolle und werden Ihnen daher im weiteren Verlauf immer wieder begegnen. Erfahrungsgemäß muss man sich als Java-Entwickler am Anfang jedoch an dieses Konzept gewöhnen, da es in Java nichts Vergleichbares gibt.

Strukturen und Enumerationen

Lesen Sie bitte **Kapitel 6** des Lehrbuchs.

Eine Struktur ist ein Konstrukt, das aus C stammt und in Java nicht verfügbar ist. Strukturen können zwar Konstruktoren, Konstanten, Felder, Methoden, Eigenschaften, Indexer, Operatoren und geschachtelte Typen enthalten, werden aber meist lediglich zum Kapseln von Gruppen zugehöriger Felder verwendet. Da es sich bei Strukturen im Gegensatz zu Klassen um Werttypen handelt, sind sie in den meisten Fällen performanter. Strukturen unterscheiden sich von Klassen jedoch darin, dass sie nicht abstrakt sein können und keine Implementierungsvererbung unterstützen.

Eine Enumeration in C# ist Gruppierung mehrerer Konstanten vom gleichen Datentyp. Als Datentypen sind dabei nur `byte`, `long`, `short`, `int` und `long` zu gelassen. Enumerationen in Java können deutlich komplexer sein und auch Konstruktoren, Methoden und Felder enthalten, wie das Tutorial zu Enum Types zeigt.

Fehlerbehandlung und Debugging

Lesen Sie bitte **Kapitel 7** des Lehrbuchs.

In nahezu jedem Programm können zur Laufzeit zum Beispiel durch unzulässige Eingaben oder durch sonstige Probleme (Ausfall des Netzwerks oder der Datenbankanbindung) Fehler auftreten. Hinzu kommen die logischen Fehler, die wir als Entwickler bei der Implementierung selbst verursachen. In diesem Kapitel lernen Sie, wie man solche Fehler zur Laufzeit erkennen und behandeln kann. Bei der Fehlersuche (Debugging) werden Sie auch von Visual Studio tatkräftig unterstützt.

Ausnahmen werden genauso behandelt wie in Java. Es gibt nur eine Besonderheit. Die Ausnahme muss im catch-Block keiner Variablen zugewiesen werden. Dies geschieht entweder durch das Weglassen des Namens der Variable hinter dem Typ der Ausnahme oder durch das komplette Weglassen der runden Klammern hinter dem Schlüsselwort »catch« (letzter catch-Block).

Fragen zum Lernstoff

Delegaten versus Schnittstellen

Sowohl Delegaten als auch Schnittstellen ermöglichen es einem Klassendesigner, Typdeklarationen und Implementierung voneinander zu trennen. Eine bestimmte Schnittstelle kann von jeder Klasse oder Struktur geerbt und implementiert werden. Ein Delegat kann für eine Methode jeder beliebigen Klasse erstellt werden, solange die Methode mit der Methodensignatur des Delegaten übereinstimmt. Ein Schnittstellenverweis oder ein Delegat kann von jedem Objekt verwendet werden, ohne über Informationen zu der Klasse zu verfügen, die die Schnittstelle oder die Delegatmethode implementiert.

Wann sollte ein Klassendesigner in Anbetracht dieser Gemeinsamkeiten einen Delegaten verwenden, und wann sollte eine Schnittstelle verwendet werden?

Ereignisse und Multicast-Delegaten

Ereignisse sind im Grunde nichts anders als Multicast-Delegaten, wie das folgende Beispiel zeigt.

```
namespace Demo
{
    public class Publisher
    {
        public delegate void MyDelegate();
        public MyDelegate EventX;
        public event MyDelegate EventY;

        public void fireEventX()
        {
            if (EventX != null) EventX();
        }

        public void fireEventY()
        {
            if (EventY != null) EventY();
        }
    }


    class Program
    {
        static void Main(string[] args)
        {
            Publisher publisher = new Publisher();
            publisher.EventX += delegate () {
                Console.WriteLine("Event-Handler for EventX");
            };
            publisher.EventY += delegate () {
                Console.WriteLine("Event-Handler for EventY");
            };
            publisher.fireEventX(); // Event-Handler for EventX
            publisher.fireEventY(); // Event-Handler for EventY
            Console.ReadLine();
        }
    }
}
```

Welcher kleine aber entscheidende Unterschied besteht zwischen Ereignissen und Multicast-Delegaten?

Antworten

Sie finden die Antworten auf die obigen Fragen im OLAT-Kurs im Kursbaustein »Lehrmaterial«.