5
Abstrakte Klassen, Interfaces. Collections, generische Klassen, anonyme Funktionen

Jörn Loviscach

Versionsstand: 21. März 2014, 22:57

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos: http://www.j3L7h.de/videos.html

Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

1 Abstrakte Klassen

Oft benötigt man eine Mutterklasse nur, um davon zu erben, nicht aber, um wirklich Instanzen dieser Klasse anzulegen. Eine solche Mutterklasse sollte dann als `abstract` deklariert werden. Dann verbietet der Compiler, dass von der Klasse Instanzen erzeugt werden, dass also `new` dafür aufgerufen wird.

Beispiel im Objektkatalog: `Microsoft.Win32.FileDialog` stellt als abstrakte Mutterklasse diverse Hilfsmittel für `Microsoft.Win32.OpenFileDialog` und `Microsoft.Win32.SaveFileDialog` bereit.

Außerdem kann in abstrakten Klassen „rein virtuelle“ [pure virtual] Methoden anlegen. Diese haben in der Mutterklasse keinen Funktionsrumpf, sondern *müssen* in Kindklassen überschrieben werden – es sei denn, die Kindklassen sind

ebenfalls `abstract`. In C# schreibt man in der Deklaration einer rein virtuellen Methode `abstract` statt `virtual`. Auch in Java steht da `abstract`; in C++ steht `=0` hinter dem Namen der Methode. Beispiel: die `Remind`-Methode von `Meeting` rein virtuell machen:

1

In einem UML-Klassendiagramm verwendet man Kursivschrift für den Namen einer abstrakten Klasse und für rein virtuelle Methoden:

2

2 Interfaces

Anders als C++ verbieten Java und C# die Mehrfachvererbung, also dass eine Klasse mehrere Eltern (nicht Mutter und Großmutter!) hat und damit von mehreren Klassen gleichzeitig erbt. Dieses Verbot sorgt für saubereren Code, hat aber zur Folge, dass man eine Situation wie diese nicht mehr mit normalen Klassen lösen kann:

3

Das ist aber typischerweise kein Problem, weil es in C# und Java zur Hilfe eine Art Mini-Klasse gibt: `interface` (Schnittstelle). Ein Interface ist eine Sammlung von Methoden (Instanzmethoden, keine statischen Methoden) – aber ohne Implementierung, also ohne Schweifklammern mit Befehlen dahinter – und anderen Elementen, allerdings *keinen* Attributen. Alle Elemente des Interface gelten

automatisch als `public`. Eine Klasse kann von keiner oder einer Mutterklasse erben und gleichzeitig beliebig viele Interfaces implementieren. In C# schreibt man Mutterklasse und Interfaces mit Kommas getrennt in die Liste hinter dem Doppelpunkt. In Java steht statt dessen `extends ... implements ....`

Klassen stellt man sich meist als Baupläne für Maschinen vor, Interfaces dagegen als Bescheinigungen für Fähigkeiten – wie einen Führerschein oder einen schwarzen Gürtel. Deshalb haben Interfaces gerne Namen, die auf `...able` enden. In C# sollen die Namen von Interfaces außerdem mit `I` beginnen. Beispiele: `ICloneable`, `IComparable`. In UML kann man zum Beispiel einen Lollipop mit dem Namen des Interfaces an die implementierende Klasse zeichnen.

3 Collections, generische Klassen, anonyme Funktionen

Neben den schlichten Arrays es in allen nennenswerte Klassenbibliotheken diverse Möglichkeiten, Daten abzulegen: Collections (in C++: STL-Container). In .NET hat man zum Beispiel die folgenden im Namespace `System.Collections.Generic`:

4

Hier ein Beispiel für eine komplexe Collection: `Dictionary< , >`. Dies funktioniert wie ein Wörterbuch oder ein Telefonbuch: Jeder Wert wird unter einem Schlüssel [key] abgelegt und wiedergefunden. Man gibt zwei Typen an: den Typ der Schlüssel und den Typ der Datenwerte.

5

Die meisten interessanten Collections sind „generische Klassen“ [generics]: Sie sind nicht fertig gebaut, sondern verlangen noch die Angabe eines oder mehrerer Typen in spitzen Klammern. Diese Schreibweise ist in Java und C++ gleich. In C++ redet man von Templates. Diese Konstruktionen lassen sich auch übelst verschachteln: `List<Tuple<int,Queue<string>>>`.

Für `List< >` und viele andere Collections gibt es in .NET einen Satz hilfreicher Funktionen zum Summieren, Suchen des kleinsten/größten Werts, Suchen eines Elements, Sortieren usw. Die Details davon lassen sich – unter anderem – mit anonymen Funktionen („Lambda-Ausdrücken“) einstellen. Sie sehen in C# wie Abbildungsvorschriften $x \mapsto x^2 + 7$ in der Mathematik aus:

6

Statt der üblichen `for`-Schleife kann man für Collections die `foreach`-Schleife benutzen:

7

In Java schreibt man statt dessen `for(Meeting m : c)`. In C++ sieht das neuerdings auch fast so aus wie in Java.