1
Überblick über das zweite Semester

Jörn Loviscach

Versionsstand: 1. April 2011, 23:06

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

1 Objektorientierung

Die Sprache C aus dem vergangenen Semester steht für drei miteinander verwandte Konzepte („Paradigmen“):

[1]

Der nächste Schritt ist, zusammengehörige Daten und Funktionen auch wirklich zusammenzufassen und vor dem Rest der Welt weitgehend zu verbergen: Verkapselung [encapsulation]. So entstehen „Objekte“. Sie wirken wie Maschinen mit klaren Schaltern und Reglern, Ein- und Ausgaben. Die Details sind im Gehäuse verborgen und sollten nur den Hersteller interessieren. Probleme in solche Einheiten aus Daten und Funktionen zu zerlegen und dann Software aus entsprechenden Einheiten zusammenzubauen, macht das Paradigma Objektorientierung [object orientation] aus:

[2]

Objekte sind in den meisten objektorientierten Programmiersprachen *Instanzen* von *Klassen*:

[3]

Neben dem „Black Box“-Prinzip der Verkapselung gibt es weitere Grundideen der Objektorientierung. Die wichtigsten davon sind Vererbung [inheritance] und Polymorphie (Vielgestaltigkeit) [polymorphism]:

[4]

Die wesentlichen Nutzen der Objektorientierung sind:

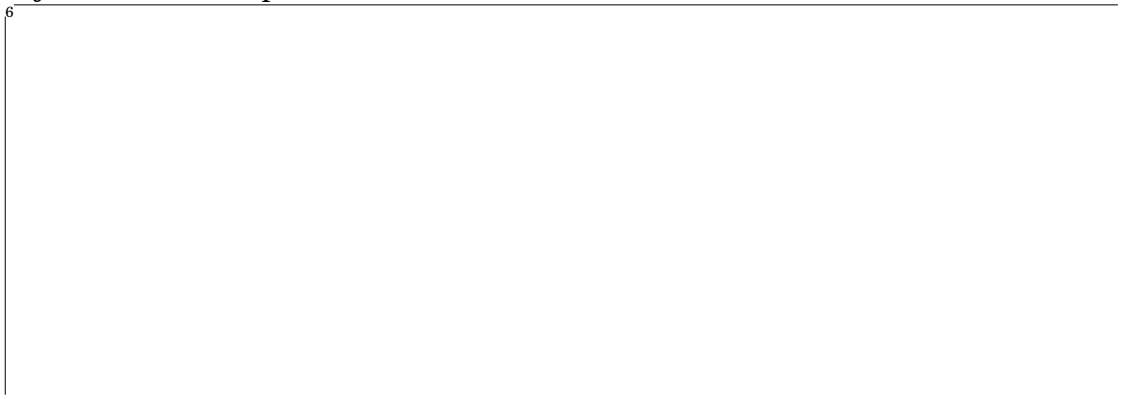
5

Die wesentlichen Ideen der Objektorientierung stammen aus den 60er Jahren des vergangenen Jahrhunderts. Als ein grundlegendes Problem erweist sich, parallele Vorgänge mit den bisherigen objektorientierten Programmiersprachen abzubilden. Mehr dazu im Abschnitt Multithreading. Ein Paradigma, das damit leichter umgehen kann, ist die funktionale Programmierung – auch wenn sie aus den 30er Jahren stammt. Im Kern stehen hier Funktionen, die sich so verhalten wie in der Mathematik, also insbesondere keine Nebeneffekte haben. In Visual Studio gibt es mit der Sprache F# eine aktuelle Vertreterin dieses Konzepts.

2 C# und .NET

In der Familie an Sprachen, die aus C hervorgegangen sind, finden sich viele objektorientierte Spachen:

6



C++ ist auf Grund seines Alters mit einigem historischen Ballast befrachtet. In wenigen Monaten lässt sich kaum lernen, ein einziges wasserdichtes C++-Programm zu schreiben. Moderne Sprachen wie Java und C# sind deutlich einfacher, um die Konzepte der Objektorientierung zu verstehen und robuste eigene Software zu schreiben. Mit einem fundierten Hintergrund kann man sich dann später andere Sprache wie C++ sauberer aneignen.

Java- und C#-Programme sind im Großen objektorientiert, sehen aber im Kleinen aus wie C: Typen, Funktionen, `if`, `else`, `for`, `while`, `switch` und auch die

Hauptfunktion `main` finden sich wieder, letztere allerdigs groß geschrieben. Demo in C#.

Zu dem Ökosystem um C# gehört nicht nur Microsofts Entwicklungsumgebung Visual Studio mit ihren vielen Automatiken, sondern vor allem auch die .NET-Klassenbibliothek. Sie unterstützt praktisch alles, was Microsoft Windows bietet. Drei Beispiele:

Sprachausgabe (Text-to-Speech): Einen Verweis auf `System.Speech` hinzufügen. Am Anfang der C#-Datei `using System.Speech.Synthesis;` ergänzen. Zum Beispiel auf Klick dies ausführen:

```
SpeechSynthesizer sp = new SpeechSynthesizer();
sp.SelectVoice("LH Anna");
sp.SpeakAsync("Dies ist ein dummer Satz.");
```

Die deutschen Stimmen LH Anna und LH Stefan stehen erst nach Installation von Microsoft Reader (Link) und dem deutschen Text-to-Speech-Package (Link) zur Verfügung.

Spracheingabe: Einen Verweis auf `System.Speech` hinzufügen. Am Anfang der C#-Datei `using System.Speech.Recognition;` ergänzen. Am Beginn des Programms dies ausführen:

```
Choices theChoices = new Choices();
theChoices.Add("Bielefeld");
theChoices.Add("Gütersloh");
theChoices.Add("Minden");
SpeechRecognizer recognizer = new SpeechRecognizer();
recognizer.LoadGrammar(new Grammar(new GrammarBuilder(theChoices)));
recognizer.SpeechRecognized += speechRecognized;
recognizer.Enabled = true;
```

Und diese Funktion ergänzen:

```
void speechRecognized(object sender, SpeechRecognizedEventArgs e)
{
    Title = e.Result.Text;
}
```

Stifteingabe: In der XAML-Datei `<InkCanvas Name="theInkCanvas" />` ergänzen. Für die Schrifterkennung `using System.Windows.Ink;` und Verweise ergänzen auf:

```
C:\Program Files\Reference Assemblies\Microsoft\Tablet PC\
                                                v1.7\IAWinFX.dll
C:\Program Files\Reference Assemblies\Microsoft\Tablet PC\
                                                 v1.7\IACore.dll
C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\IALoader.dll
```

Weil diese Bibliotheken älter sind, eine Anwendungskonfigurationsdatei `App.config` hinzufügen, in der als Konfiguration steht:

```
<startup useLegacyV2RuntimeActivationPolicy="true">
<supportedRuntime version="v4.0" />
</startup>
```

Und im Code:

```
InkAnalyzer ia = new InkAnalyzer();
ia.AddStrokes(theInkCanvas.Strokes);
AnalysisStatus status = ia.Analyze();
string text = ia.GetRecognizedString();
```

3 Grundlegende Typen und Operationen. Arrays

Viele der grundlegenden Datentypen wie `int` und `double` finden sich wieder. Die mathematischen Operationen werden mit `Math.` aufgerufen. Demo.

`Color` stellt eine Farbe dar – samt einiger Operationen damit. Einen Zeichenkette `string` kann man nach ihrer Länge fragen oder sich von ihr Teile geben lassen. Eine Zeitspanne[c1] `TimeSpan` lässt sich zum Beispiel aus Sekunden oder aus Tagen bilden oder zu einer anderen Zeitspanne addieren. Ein Zeitpunkt `DateTime` lässt sich zum Beispiel vom der aktuellen Zeit bilden. Die Differenz zweier Zeitpunkte ist eine Zeitspanne.

[c1] jl: TimeSpan, nicht DateTime

Bei Arrays ist die Schreibweise des Typs etwas anders: `int[] a = new int[42];` Auch Arrays lassen sich nach ihrer Länge fragen und mit `Array.` zum Beispiel sortieren oder in der Reihenfolge umkehren.

4 Klassen und Objekte

Wie C kennt C# Datenstrukturen `struct`, allerdings ohne Semikolon am Ende der Definition. Im Sinne der Objektorientierung können Strukturen auch Funktionen enthalten – wie die mit `class` gebildeten Klassen. Diese funktionieren trotz gleichen Aussehens etwas anderes als Strukturen und sind der Kern der Objektorientierung.

Neu gegenüber C ist auch, dass man Teile von Strukturen und Klassen vor der Außenwelt verbergen kann: Zum Beispiel ist, was mit `public` markiert ist, für alle sichtbar, was mit `private` markiert ist, aber nur für aus den Funktionen der jeweiligen Struktur oder Klasse heraus. Hiermit lässt sich ein Gehäuse um die Innereien bauen:

[7]

5 Datenstrukturen

Neben dem Array kennen wir schon die Warteschlage `Queue` und den Stapel `Stack`. Diese und viele andere bilden die in .NET eingebauten Collections. Zum Beispiel ist `Queue<string>` eine Warteschlange für Zeichenketten. Die übliche Collection ist `List`. Es wirkt wie ein größenveränderliches Array.

6 WPF: Windows Presentation Foundation

Das Windows Presentation Foundation ist die aktuelle Methode, grafische Oberflächen für Windows zu entwickeln. Es kann mit Stift- und Toucheingabe umgehen, aber auch mit Hilfe der Grafikkarte Bilder um 3D-Objekte wickeln.

Im ersten Seminar hatten wir eine Ellipse in die XAML-Datei eingetragen und per Programm bewegt. Man kann Grafiken auch komplett per Code erzeugen, zum Beispiel für Diagramme:

```
Line c = new Line();
c.X1 = 12;
c.X2 = 23;
c.Y1 = 45;
c.Y2 = 67;
c.Stroke = Brushes.Red;
c.StrokeThickness = 5.0;
theCanvas.Children.Add(c);
```

Auf der Grafikkarte lassen sich Effekte anwenden, zum Beispiel ein Weichzeichner:

```
System.Windows.Media.Effects.BlurEffect be
          = new System.Windows.Media.Effects.BlurEffect();
be.Radius = 10;
theCanvas.Effect = be;
```

7 Dateien und Datenströme

.NET besitzt sehr einfache Methoden, mit Dateien umzugehen: Zum Beispiel kann man eine Textdatei einfach in den Speicher laden: `string[] lines = System.IO.File.ReadAllLines("Pfad_der_Datei");` Das ist allerdings keine gute Idee, wenn man zum Beispiel eine Datensammlung von mehreren Gigabytes in einer Datei hat und nur einige Datensätze daraus lesen will. Mit einem `Stream`, einem Datenstrom, geht das geschickter. Hier kann man mit einem Cursor durch die Datei gehen und dort gezielt lesen und schreiben:

[8]

Objekte zu speichern und wieder einzulesen, ist ebenfalls simpel: Das übernimmt die Objektserialisierung von .NET auf Wunsch selbst. Man schreibt praktisch nur `[Serializable]` vor die Definition der Klasse.

8 Exceptions

Im wahren Leben können immer wieder Ausnahmesituationen (daher der Begriff „Exceptions“) auftreten, die den normalen Ablauf stören: eine Datei lässt sich überschreiben, der Speicher wird knapp, die Internet-Verbindung reißt ab, in einen Formularfeld steht ein Buchstabe statt einer Zahl. Eine elegante Art der meisten objektorientierten Programmiersprachen, mit solchen Situationen umzugehen, sind Exceptions. Sie helfen, die Ausnahmebehandlung aus dem normalen Code herauszuhalten:

[9]

Demos: Versuch, eine nicht vorhandene Datei zu öffnen; Versuch, einen Buchstaben als Zahl zu lesen.

9 Multithreading

Wenn man schon Prozessoren hat, die dem Betriebssystem melden, dass sie acht Sachen gleichzeitig können (Quadcore mit Hyperthreading), möchte man das auch ausnutzen. Ein Programm kann sich dazu in mehrere Threads (wörtlich: Fäden) aufspalten, die nebeneinander laufen.

Wenn einfach nur etwas unbeaufsichtigt nebenbei passieren soll, kann man das mit `System.Threading.ThreadPool.QueueUserWorkItem` in Auftrag geben. Viel schwieriger wird es dagegen, wenn Threads zusammenarbeiten sollen. Es darf nicht der eine Thread in Datenstrukturen schreiben, aus denen der andere Thread gerade liest – außer, die Datenstrukturen sind raffiniert gebaut.

Sobald der eine Thread auf den anderen warten muss, verschenkt man Rechenleistung. Im schlimmsten Fall (Deadlock) wartet der eine auf den anderen sowie der andere auf den einen – und das Programm hängt. All dies ist in gängigen Programmiersprachen noch hakelig.

10 Datenbanken. Kommunikation per Netzwerk

C# besitzt eingebaute Möglichkeiten, um Daten aus Datenbanken zu holen und in Datenbanken zu schreiben. Dies ist für Unternehmensdaten ebenso interessant wie für Sammlungen von Messwerten.

Datenbanken liegen oft auf Servern im Netz. Ebenso lassen sich mit .NET leicht Daten von Webservern oder FTP-Servern holen, Programme auf mehreren Rechnern verbinden oder eigene Server einrichten. Hier das Holen der ersten 10.000 Zeichen einer Webseite:

```
using System.IO;
using System.Net;
//...
byte[] buffer = new byte[10000];
WebRequest request = WebRequest.Create("http://www.j3L7h.de/");
WebResponse response = request.GetResponse();
Stream resStream = response.GetResponseStream();
resStream.Read(buffer, 0, buffer.Length);
```

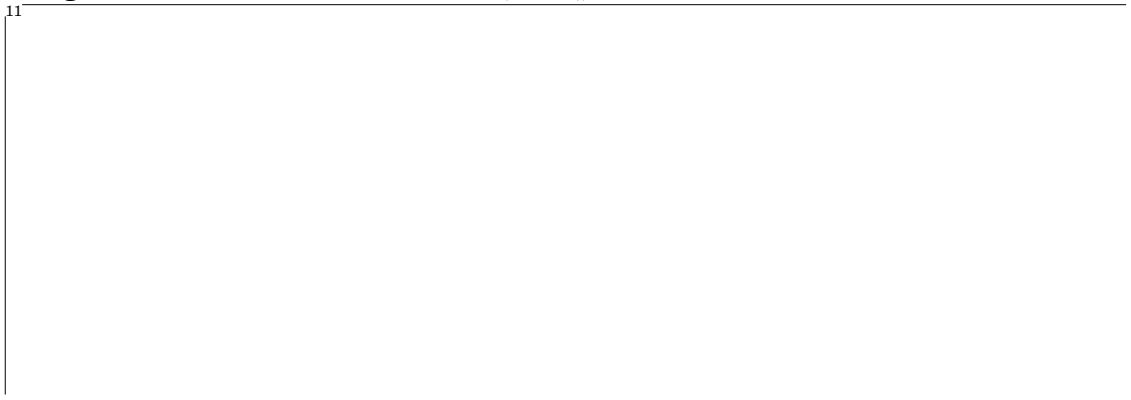
```
response.Close();
string text = Encoding.ASCII.GetString(buffer);
```

11 Design Patterns

Im Laufe der Jahrzehnte haben sich nicht nur typische Datenstrukturen wie Array und Warteschlange entwickelt. Vielmehr sind auch größere Programmstrukturen zu Standards geworden: Entwurfsmuster [design patterns]. Hier spielen gleich mehrere Klassen auf standardisierte Weise zusammen.

Ein Beispiel haben wir schon im ersten Seminar gesehen: Die Verarbeitung von Ereignissen durch Funktionsaufrufe, das „Observer Pattern“:

11



12 .NET von innen

.NET ist nicht nur eine Klassenbibliothek, sondern eine komplexe Laufzeitumgebung für Software. Die Programme werden – wie bei Java – in einer vereinfachten Zwischensprache ausgeliefert und erst „just in time“ in die jeweilige Maschinensprache übersetzt. Zur Laufzeit des Programms lässt sich abfragen, welche Klassen mit welchen Fähigkeiten vorhanden sind („Reflexion“). Wie Java kommt C# damit ohne die Header-Dateien von C und C++ aus. Der Compiler schaut selbst nach, statt die Informationen aus Header-Dateien lesen zu müssen.

Auf .NET laufen diverse Programmiersprachen; vier sind schon bei Visual Studio dabei. Es lassen sich Teile desselben Programms in verschiedenen Sprachen schreiben. Das geht im Prinzip auch mit Java, ist allerdings dort nicht offiziell vorgesehen.