Einführung in C#

Björn Lohrmann

FG Komplexe und Verteilte IT-Systeme

mit Material von Dr. Ulf Rerrer-Brusch

Einführung in C#

- Streng typisierte objekt-orientierte Programmiersprache
- Wird übersetzt in Intermediate Language (IL), ähnlich Java-Bytecode
- Wird ausgeführt von Common Language Runtime (CLR) – ähnlich JVM
- Anforderung: Architekturunabhängigkeit
- Vorbilder: Java und C++
- viele Bücher, Tutorials und Online-Materialien verfügbar

Gemeinsamkeiten von C# und Java

- Keine Header-Dateien
- Mehrfachvererbung von Schnittstellen (nicht von Implementierungen)
- Keine globalen Funktionen oder Konstanten (alles in Klassen)
- Arrays und Strings mit festen Längen und Zugriffskontrolle
- Alle Variablen müssen vor der ersten Verwendung initialisiert werden
- Alle Objekte erben von **`Object`**-Klasse
- Objekte werden auf dem Heap erzeugt (mit dem Schlüsselwort **`new`**)
- Garbage Collector
- Thread Unterstützung, Synchronisation

C# Basics

- Einfache Typen
  - **sbyte, short, int, long, float, double, bool, char**
  - vorzeichenlos: **byte, ushort, uint, ulong**
- Enumerationen

```
enum Color {red=1, blue=2, green=4}
```

- Konstanten (wie in C/C++)

```
const int var = 3;
```

- Arrays

```
int[] array = new int[3];
int[] array = new int[] {1, 2, 3};
int[] array = {1, 2, 3};

int x = a[2];
int y = a.Length;
Array.Copy(b, a, 2);
Array.Sort(b);
```

Strings

- Strings

```
string s1 = „Hello“;
string s2 = string.Copy(s1);

if (s1 == s2) { ... }
```

- Ein-/Ausgabe (Console)

```
using System;
Console.Write(„kein Newline am Ende“);
Console.WriteLine(„diesmal mit Newline“);
Console.WriteLine(„Hello {0}!“, „students“)
String test = Console.ReadLine();
```

- andere Operationen
  - `s.ComparesTo(s1),`
  - `s.Substring(from, length),`
  - `s.ToUpper(),`
  - `s.StartsWith(s1)`, …

Klassen und Vererbung

- Übernommen von C++; Syntax identisch für Klassen und Interfaces

```
class D : B, C {...}
```

- Jedoch keine Vererbung unter Angabe von Zugriffsrechten
- Eine Klasse kann max. von *einer* anderen Klasse erben, aber *mehrere* Interfaces implementieren
- Wurzelklasse ist **`Object`**, von der alle anderen Klassen abgeleitet sind

```
class C {...}
// bedeutet implizit
class C : Object {...}
```

- Mit **`GetType()`** kann der Typ der Klasse abgefragt werden

```
Object obj = new C();
Type type = obj.GetType();
Console.WriteLine (type.Name);
```

Interfaces

- Explizite Implementierung eines Interfaces:

```
public interface ITeller {
        void Next();
}
public interface IIterator {
        void Next();
}
public class Clark: ITeller, IIterator {
        void ITeller.Next() {}
        void IIterator.Next() {}
}
```

- Vermeidung von Namenskonflikten bei mehreren Interfaces:

```
Clark clark = new Clark();
((ITeller)clark).Next();
```

Flusskontrolle

- **`if`**, **`while`**, **`for`** wie in Java
- **`foreach`**

```
foreach (Object o in collection ) { ... }
foreach (int i in array) { ... }
```

  - **`foreach`**-Anweisung arbeitet auf allen Objekten, die das Interface **`System.Collections.IEnumerable`** implementieren
- **`switch`**
  - Kontrollfluss muss explizit festgelegt werden
  - **`break`**, **`return`**, **`goto`**, oder **`throw`** muss am von **`case`** stehen

```
switch(name) {
        case „Zaphod Beeblebrox“:
                Console.WriteLine(„Hello Zaphod“);
                break;
        case „Ford Perfect“:
                Console.WriteLine(„Hi“);
                break;
```

Assemblies, Namespaces und Zugriffslevel

- Namensräume (namespaces) wie in Java: Trennung mit „.“
  - Namensräume importieren mit **`using`**-Schlüsselwort

- Eine Assembly besteht aus mehreren Dateien (einem Projekt), die zu einer .exe- (Executable) oder .dll-Datei (Bibliothek) kompiliert werden.
  - definieren ihren eigenen Namensraum
  - verschiedene Versionen einer Assembly können parallel existieren

- Fünf Zugriffslevel
  - **`private`** (Zugriff nur innerhalb der Klasse, wie in Java)
  - **`internal`** (Zugriff innerhalb der Assembly)
  - **`protected`** (Zugriff innerhalb der Klasse und abgeleiteter Klassen)
  - **`internal protected`** (wie **`protected`**, zusätzlich im Assembly)
  - **`public`** (Zugriff immer erlaubt)

Entwicklungsumgebungen

- Microsoft Visual Studio 2008/2010
  - sehr groß
  - sehr mächtig
  - gute Hilfe (MSDN Library)
- Beschaffung über
  - über MSDN-AA der TUB
    https://irb.eecs.tu-berlin.de/tubit_login/
  - zusammen mit der
    MSDN-Library

msdn academic alliance