Zusammenfassung Advanced Programming

APROG HS22

Joel von Rotz & Andreas Ming

01.01.2023

Inhaltsverzeichnis

1 C# und .Net-Framework

1.1 Vergleich C & C#

	C (POP)	*C# (OOP)*
	Prozedurale **O**rientierte **P**rogrammierung	**O**bjekt **O**rientierte **P**rogrammierung
Compilation	Interpreter	Just-in-time *(CLR)*
Execution	Cross-Platform	.Net Framework
Memory handling	`free()` after `malloc()`	Garbage collector
Anwendung	Embedded, Real-Time-Systeme	Embedded OS, Windows, Linux, GUIs
Execution Flow	Top-Down	Bottom-Up
Aufteilung in	Funktionen	Methoden
Arbeitet mit	Algorithmen	Daten
Datenpersistenz	Einfache Zugriffsregeln und Sichtbarkeit	Data Hiding (`privat`, `public`, `protected`)
Lib-Einbindung	*.h* File mit `#include`	*namespaces* mit `using`

1.2 Struktur C#-Programm

1.2.1 Klassen

1.2.2 Namespace

```
namespace { ... }
```

`namespace` dient zur Kapselung von Methoden, Klassen, etc., damit zum Beispiel mehrere Klassen/Methoden gleich benannt werden können.

```
namespace SampleNamespace {
  class SampleClass {...}
  struct SampleStruct {...}
  enum SampleEnum {a, b}
  namespace Nested {
    class SampleClass {...}
  }
}

namespace NameOfSpace {
  class SampleClass{...}
  ...
}
```

1.3 Keywords

1.3.1 Zugriffs

1.3.2 using

`using` wird zum Abkürzen von Namen

```
System.Console.WriteLine("Hello World!");

// using Namespace
using System;
...
Console.WriteLine("Hello World!");
```

1.3.3 `class`

1.3.4 `struct`

> **!** Unterschied `struct` & `class`
>
> `structs` sind *value* Typen und übergeben jeden Wert/Eigenschaften. `classes` sind *reference* Typen und werden als Referenz übergeben.
>
> - `class` → call by reference (Übergabe als Reference)
> - `struct` → call by value (Übergabe als Wert)

1.4 Datentypen

1.4.1 `string`

Strings werden mit dem folgender Deklaration

> **!** Wichtig
>
> Strings können nicht verändert werden -> sind **read-only**
>
> ```
> string s = "Hallo Welt";
>
> s[1] = 'A'; // ERROR
> ```

Parameter in String einfügen

Parameter/variablen können in Strings direkt eingefügt werden.

1.4.2 Bildschirmausgabe

Variante 1 - C Style:

```
Console.WriteLine("The sum of {0} and {1} is {2}",a,b,result);
```

Variante 2 - C# Style:

```
Console.WriteLine("The sum of" + a + "and" + b + "is" + result);
```

Variante 3 - new C# Style:

```
Console.WriteLine($"The sum of {a} and {b} is {result}");
```

2 Konzepte C#

2.1 Overloading

2.1.1 Konstruktor Overloading

```
class Point {
  private int pos_x;
  private int pos_y;

  public Point(int x, int y) {
    this.pos_x = x;
    this.pos_y = y;
  }

  public Point() : this(0,0) {}
}
```

Mit `this` nach dem Konstruktor (unterteilt mit `:`) kann der Aufruf auf einen anderen Konstruktor weitergeleitet werden. Der Inhalt des vorherigen Konstruktors wird erst nach dem Ablauf des `this`-Konstruktors (im Beispiel `Point(int x, int y)`).

Konstruktor Aufruf-Reihenfolge

```
using System;

class Point {
  private int pos_x;
  private int pos_y;

  public Point(int x, int y) {
    this.pos_x = x;
    this.pos_y = y;
    Console.WriteLine($"Point {this.pos_x}, {this.pos_y}");
  }

  public Point(int x) : this(x, 0) {
    Console.WriteLine("x-only");
  }

  public Point() : this(0,0) {}
  Console.WriteLine("no value");
}
```

Wird nun `Point(4)` aufgerufen erhält man folgendes auf der Konsole

```
Point 4, 0
x-only
```

2.1.2 Methoden Overloading

2.2 Default Parameter (implizit Overloading)

2.3 Multithreading `System.Threading`

```
static void Main(string[] args) {
  Thread t = new Thread(Run);
  t.Start();
  Console.ReadKey();
}

static void Run() {
  Console.WriteLine("Thread is running...");
}
```

2.4 Parametrisierter Thread

Falls ein Parameter übergeben werden muss, kann die delegierte `ParameterizedThreadStart`-Signatur verwendet werden. Der Thread wird normal aufgesetzt und bei `.Start()`

```
static void Main(string[] args)
{
  //...
  TcpClient client = listener.AcceptTcpClient();
  Thread t = new Thread(HandleDateTimerequest);
  t.Start(client);
  // ...
}

// must be of ParameterizedThreadStart signature
private void HandleDateTimerequest(object _object)
{
  TcpClient client = (TcpClient)_object;
  // ...
}
```

3 Linux & Raspberry Pi 4

4 Windows Presentation Foundation

> **i** Unterschied zwischen WPF & Console Application
>
> WPF ist

5 Weitere Konzepte

5.1 TCP / UDP

5.2 MQTT

6 Notes

6.1 Overflows Integer

Im folgenden Code wird eine Variable `i` mit dem maximalen Wert eines `int` geladen und folgend inkrementiert.

```
int i = int.MaxValue;
i++;
```

Wird aber dies direkt in der Initialisierung eingebettet (`...+ 1`), ruft der Compiler aus, da er den Overflow erkennt. (Einsetzung von Compilern)

```
int i = int.MaxValue + 1; // COMPILE-FEHLER
i++;
```

> **Vorsicht**
>
> Dieser Overflow-Fehler gilt nur bei **konstanten** Werten bei der Initialisierung. Wird eine separate Variable mit dem Maximalwert initialisierit und an `i` hinzuaddiert, gibt es keinen Fehler.
>
> ```
> int k = int.MaxValue;
> int i = k + 1; // KEIN Fehler
> ```

7 Glossar

- **Timeslicing**: Bei Computersystemen wird *timeslicing* verwendet, damit mehrere Prozesse "parallel" verlaufen können. Jedem Prozess/Thread wird ein fixer Zeitslot gegeben, in dem es sein Code abarbeiten kann,
- **Präventiv/kooperativ**: Ein *präventives* Betriebsystem unterbricht ein Prozess, wenn dieser sein Time-Slot verbraucht hat. Ein *kooperatives* BS unterbicht die Prozesse nicht und die Prozesse geben an, wann es fertig ist.