Übung 1 mit C# 6.0

MATTHIAS RONCORONI

Inhalt

1. Überblick über C#
2. Lösung der Übung 1
3. Code
4. Demo

C# allgemein

- aktuell: C# 6.0 mit .Net-Framework 4.6:
  - Multiparadigmatisch (Strukturiert, Objektorientiert, Funktional (v 3.0), Reflexiv)
  - Stark, statisch Typisiert (implizit (v 3.0) und dynamisch (v4.0) möglich)
  - Trend zu Platformunabhängigkeit (.net wird Quelloffen)

Syntax

- Ähnlich wie C++ und Java…

```
public class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Hallo Welt!");
    }
}
```

Syntax - Erweiterungsmethoden

- Fremde Klassen können ohne Vererbung erweitert werden

```
public static class Erweiterungen {
    public static int MalDrei(this int zahl) {
        return zahl*3;
    }
}
public static class Programm {
    public static void Main() {
        Console.WriteLine(5.MalDrei());
    }
}
```

Syntax - Partielle Klassendefinition

- Klassen können über mehrere Dateien verteilt werden

```
public partial class Class
{
    private string _name;
}
```

```
public partial class Class
{
    public string GetName()
    {
        return _name;
    }
}
```

Syntax - Eigenschaften

- Sieht aus wie eine Variable, arbeitet aber mit Getter und Setter

```
public partial class Class
{
    public string Name { get; private set; }
    public int Id { get; }
    public string IdString => Id.ToString();
    private int _nr;
    public int Nr { get { return _nr; }
        set { if (value == _nr) return;
            _nr = value;
            OnPropertyChanged(); }
    }
}
```

Syntax - Delegate

- Funktionspointer mit this und Typensicherheit

```
protected delegate void Handler(NetworkStream sender, Message message);


private void HandleShutdownMessage(NetworkStream sender, Message message)
{...}
Handler handler = HandleShutdownMessage;
Handler handler += HandleShutdownMessage1; // Beide werden aufgerufen!


Handler handler = delegate(NetworkStream sender, Message message) {...};
```

Syntax - Delegate / Lambda

- Verkürzte Schreibweise anonymer Methoden

```
Handler handler = (sender, message) => {...}
```

- Aufruf eines delegate / Lambda

```
handler.Invoke(sender, msg);
handler(sender, msg);
```

LINQ - Language Integrated Query

- Ähnlich wie SQL
- Bestandteil von .Net seit V 3.0
- Datenquelle egal (Objekte, SQL, Entities, XML, DataSet…)

```
List<int> Numbers = new List<int>() { 1, 2, 3, 4 };
var query = from x in Numbers select x;
Numbers.Add(5);
Console.WriteLine(query.Count()); //5
Numbers.Add(6);
Console.WriteLine(query.Count()); //6
```

Asynchrone Ausführung - async / await

- `async` markiert asynchrone Methoden
- Rückgabewert muss `Task` oder `Task<T>` sein
- `Await` ermöglicht den Zugriff auf das „synchrone“ Ergebnis
- `Task`/`Task<T>.ContinueWith(...)` ermöglicht weiteren (asynchronen) Code

Übung 1 - Übersicht

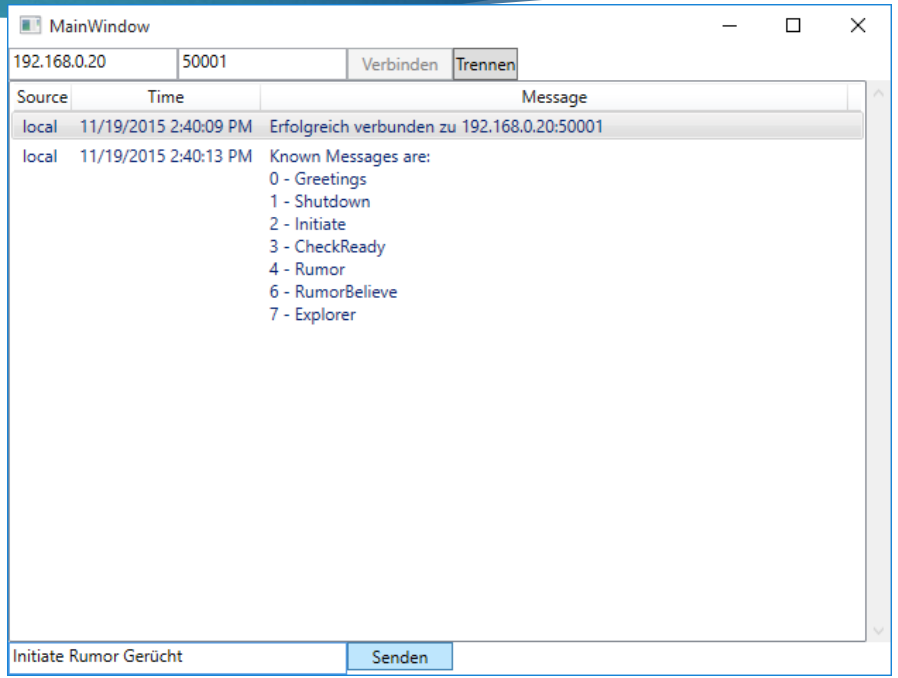
- Aufgeteilt in allgemeine Logik und Fachlogik
- Nachrichten als Typenhierarchie
- Attribute für Metabeschreibung der Nachrichten
- Factory findet zur Laufzeit alle Nachrichten in einer *.EXE (mit DLLs)
- ein grafischer Controller für alle Aufgabenstellungen

Übung 1 – Nachrichten / Factory

- Alle Nachrichten serializable
- Alle instanziierbaren Nachrichten
  - mit `MessageAttribute` markiert
  - nur Strings im Konstruktor (beliebige Anzahl)
- Factory durchsucht Assembly nach Nachrichten mit dem `MessageAttribute`
- Factory erstellt Nachrichten über Type, Id oder Name und übergibt Parameter an geeigneten Konstruktor

Übung 1 - Controller

- Interface für `MessageFactory`
- Nur mit einem Knoten verbunden
- Kann „Checks“ übers ganze Netz ausführen



Übung 1 - BaseNode / BaseExecutor

- implementieren Grundlegende Funktionalitäten
  - Komandozeilenargumente
  - Kommunikationsstruktur
  - erste Nachtichtentypen
- stellen Grundgerüst zur Spezialisierung

Übung 1 - RumorNode / RumorExecutor

- Implementieren nur speziell Aufgabe 4
  - zusätzliche Komandozeilenargumente
  - 2 zusätzliche Nachrichtentypen (Rumor, RumorBelieve)

Übung 1 - Fazit

- Hoher Aufwand bei Implementierung von `BaseNode` und `BaseExecutor`
- Zeitersparnis bei folgenden Übungen, da Spezialisierungen sehr einfach sind
- Keine Veränderungen an `MessageFactory`, `BaseNode`, `BaseExecutor` und Controller notwendig

Fragen?