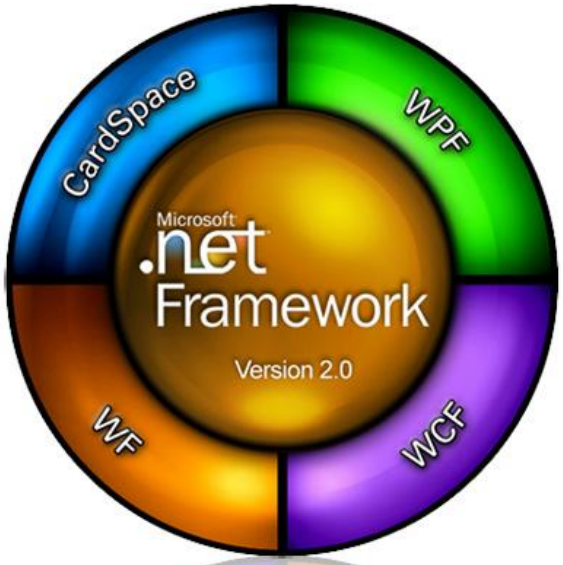


C# Programmierung

Eine Einführung in das .NET Framework

Do

- Events
- Delegates
- Operatoren
- Exceptions
- Namespaces
- Bibliotheken
- .NET Klassen
- Serialisierung

Tag 4

.NET und OOP

OOP abschließen und das Wissen über die Möglichkeiten mit dem .NET Framework erweitern.

Möglichkeit für Fragen

- Kurze Wiederholung des Vortags
- Fragen zu den Aufgaben?
- Fragen zu der Vorlesung von gestern?
- Anmerkungen und Wünsche?

Etwas delegieren

- Neuer Datentyp: **delegate**
- OOP Typ für *Funktionszeiger* aus C/C++ - hält das .NET Framework konsistent und gibt Übersicht!
- Syntax ist z.B.

```
public delegate void Evaluate(int x);
```

Modifizierer

Schlüsselwort

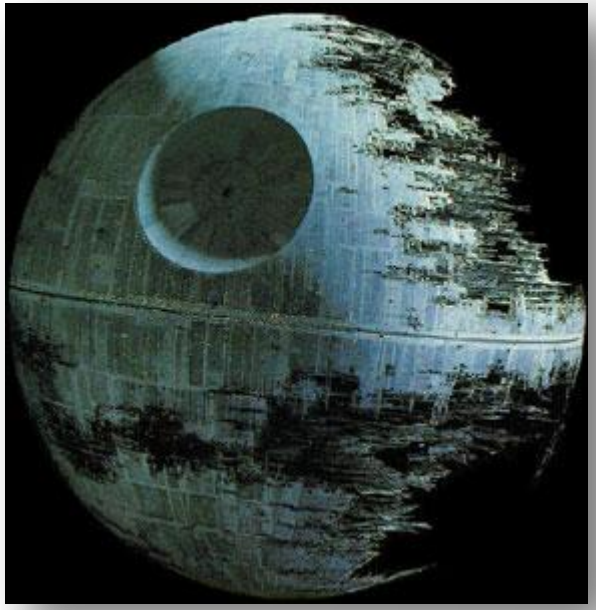
Rückgabe

Name des Delegaten

Parameter

Delegate Factories

- Durch Delegaten setzt man Funktionspointer
- Dies geschieht aber kontrolliert
- Somit kann man sich Megamoths sparen (sehr großen Methoden)
- Dies reduziert wiederrum die Gefahr von Bugs und steigert die Debugging-Fähigkeit des Codes



Wozu etwas delegieren?

- Delegaten sind auch nur Prototypen, d.h. man braucht etwas konkretes, z.B.

```
Evaluate ev = new Evaluate(foo);
```

- Geht nur wenn wir eine *Funktion* foo haben!
- Alle asynchronen Jobs, Ereignisse, Reflection und vieles mehr im .NET bauen auf den Delegaten auf
- Delegaten sind sehr viel **strikter** und **weniger fehleranfällig** als Funktionspointer in C

Beispiel

01 - Delegate

Ereignisse

- Um etwas als *Event* (Ereignis) zu deklarieren braucht man eigentlich nur 2 Dinge:

1. (Syntax) Das Schlüsselwort event
2. (Objekt) Einen Delegaten wie z.B. EventHandler

- Man erstellt dann in der Klasse eine Variable über

```
public event EventHandler myEvent;
```

- Jetzt ist das Element immer sicht- und nutzbar (**Blitz**)!

Beispiel

02 – Eigene Ereignisse

Operatoren überladen

- Ein wichtiges Thema für **OOP** – hier sind drei Typen von Operatoren wichtig (alle static):

1. Implizite und Explizite Casts ((?)this, …)
2. Mathematische / Logische Operatoren (+, &&, …)
3. Indexoperatoren (this[?], …)

- Alle drei funktionieren ein wenig anders, aber bei genauerer Betrachtungsweise genauso wie man sie naiv erwarten würde

Beispiel

03 – Operatoren überladen

Ausnahmen auslösen

- Ganz wichtiges Kapitel für **OOP** sind Ausnahmen
- eigene Ausnahmen sollten geschrieben werden um Fehler genau spezifizieren zu können
- Auslösen von Fehlern über das Schlüsselwort throw:

```
throw new Exception(„Das ist ein Fehler!“);
```

- Abfangen über den aus C++ schon bekannten try-catch Block (jetzt noch mit finally!)

Ausnahmen abfangen

- In einem **try**-Block führen Fehler nur zum Abbruch des Anweisungsblocks
- Sollte ein **catch**-Block vorhanden sein, so wird dieser im *Falle eines abgebrochenen* try-Blocks aufgerufen
- Der **finally**-Block wird bei Anwesenheit immer *(also auch falls return; verwendet wurde)* aufgerufen – eignet sich zum Schließen von noch offenen Dateien und freigeben von Handles etc.

Ausnahmen erstellen

- Sehr leicht möglich – muss im OOP Stil nur von „*Exception*“ (oder einer Tochterklasse davon) **geerbt** werden

- Damit kann im catch-Block der **Fehler spezifiziert** werden – Struktur ähnelt hier stark einer switch-case Block z.B.

```
try { … }
catch(SpecialException1 ex) { … }
catch(SpecialException2 ex) { … }
catch(MoreGeneralException ex) { … }
catch(Exception ex) { …}
```

Beispiel

04 - Ausnahmen

Pokémon Handling

- *Vermeiden* in einem größeren Block **alles auf einmal** abfangen zu wollen (sog. Catch'em all)

- **Immer** differenzieren und nur dort handeln wo es auch wirklich passieren kann

Ein genauerer Blick auf…

- Namensräume! Und… Wofür ist using noch gut?
- Wie kann man Subnamensräume erstellen?
- Kann man auch in existierende Namensräume eingreifen?
- Wozu braucht man Namensräume nochmal?
- **Beispiel**: Einbinden einer weiteren Bibliothek!

Was ist noch drin im .NET

- Leichteres Listen-Management mit Collections
- Die Math-Klasse – siehst du den Sinus?
- Auf Prozesse zugreifen (oder welche erstellen)
- Mit Streams umgehen (MemoryStream, FileStream)
- Auf das Dateisystem zugreifen (IO)
- Und noch (sehr sehr sehr …) vieles mehr…

Beispiel

05 – Weiteres im Dotnet

Serialisieren

- Sehr wichtiges Konzept. Frage dahinter:

  Wie krieg ich meine sehr abstrakte Klasseninstanz in ein binäres Objekt, mit dem ich machen kann, was ich will (*Manipulation, Speicherung, Versand*)?

- Mit dem BitConverter können wir Objekte sehr leicht konvertieren – diese müssen nur über eine „Flag“ als [Serializable] markiert werden!

- Anderer Ansatz: Wir konvertieren unser Objekt in einen *XML-String*, so dass wir diesen sehr leicht lesen und manipulieren können!

XmlSerializer

- Sehr toller Ansatz – das .NET-Framework besitzt sowieso sehr viele XML Möglichkeiten!
- Es werden **alle Eigenschaften** der Klasse serialisiert
- Sehr einfache Möglichkeit *eigene Dateitypen* zu basteln
- Bequeme Handhabung und **sehr einfaches Debugging** möglich
- Immer beide Richtungen (*Ser.*, *Deser.*) möglich!

Beispiel

06 – Klassen serialisieren

Abschließendes Beispiel

- Schreiben einer Klasse für Vektoren mit **Collections**
- Überladen des Indexoperators und der **Operatoren** + und *, sowie Einbau von Events
- Erstellen von **Exceptions** und einbinden dieser
- Implementieren von **Mathefunktionen**
- Einbauen von *Laden* und *Speichern* von Vektoren