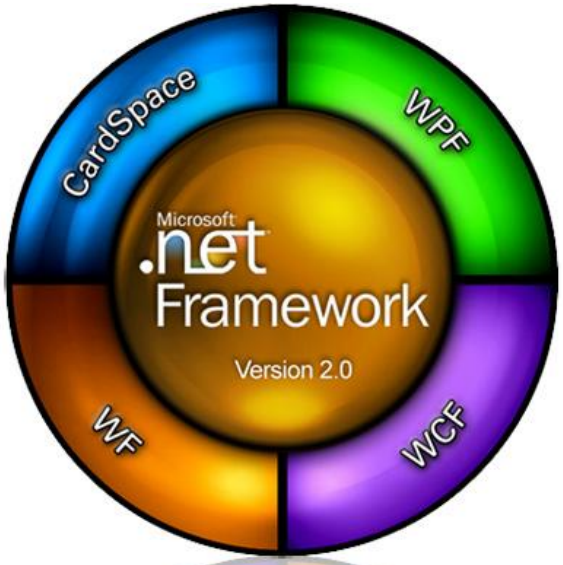


# C# Programmierung

Eine Einführung in das .NET Framework

Di

Datentypen
Arrays
Operatoren
Parameter
Überladen
Heap / Stack
Klassen
(un)Boxing

# Tag 2

## Grundlagen und Einstieg OOP

Ausbau der Fähigkeiten vom Vortag mit einem sanften Einstieg in die tolle Welt der OOP

# Möglichkeit für Fragen

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

# Weitere Grundlagen

- Konstanten und Sammlung dieser (**Enumeration**)
- Zusammenfassen von Datentypen in einer **Struktur**
- Warum sind Strukturen *keine* Klassen? **Unterschiede**!
- Unterscheidung zwischen Werttyp (*Stack*!) und Referenztyp (*Heap*!) – **fundamentale Eigenschaft**
- Überladen von Funktionen (*Signatur einer Funktion*)

# Beispiel

01 – Weitere Grundlagen

# Arrays in C#

- Statische Arrays wie in C++ (**Jagged**) oder fixed
- Syntax leicht unterschiedlich zu C: int[ ] a = new int[3]; oder int[ ] a = new int[ ]{1, 2, 5}; etc.
- Dynamische Arrays mit *ArrayList*
- Interessante Liste mit *hashtable*
- Weitere Typen (sehr spezialisiert): Queue, Stack

# Beispiel

02 - Arrays

# Neue Operatoren

- Dienen v.a. für **OOP** (Häufige Verwendung später)
- typeof() liefert uns den **Typ** eines Objektes
- Verkürzte Schreibenweise für if(obj.GetType() == typeof(Klasse)) mit „**is**“ – if(obj is Klasse)
- **Referenzen** casten mit „**as**“ – Klasse a = obj as Klasse;
- Null-Referenz behandeln mit „**??**“: b = a ?? new Klasse;

# Beispiel

03 - Neues

# Weitere Features

- unsafe { /* … */ } gibt uns Chance für Pointer (wie in C)
- Referenzieren von *Werttypen* mit ref, oder out
- Letzteres hebt **Teamcharakter** von C# hervor
- Dank **Eigenschaften** weg von Methoden wie „*int getLength()*“ und „*void setLength(int n)*“
- Können nun schreiben: int Length{ get; set; }

# Beispiel

04 - Weiteres

# Jetzt wird's ernst…

- Erstellen einer Klasse (Bauplan für jedes Objekt!)
- Instanziieren einer Klasse (wieso?) – Operator new
- Wie geht das erben von **einer** Klasse? (wie geht das mit Mehrfachvererbung bzw. wieso nicht von ***n*** Klassen?)
- Casten von Klassen – sog. *boxing* und *unboxing*
- Methoden **implementieren** und **überschreiben**

# Klassen näher betrachtet

- Wieso sind Modifizierer so hilfreich bzw. was bringen internal | private | protected gegenüber public?
- Vorhandene Methoden verstecken [**don't**] mit new
- Zusammenhang Eigenschaften, Modifizierer und Klassen (*OOP at its finest*!) – Aufbau von sog. *Design Pattern*
- Nochmal betrachtet: **Heap** (Klassen) und **Stack** (Strukturen) – essentieller Unterschied!

# Beispiel

05 – Mehr zu Klassen

# Wozu dient OOP?

- Robusterer und leichter erweiterbarer Code
- Verringern von Schreibarbeit
- Definieren von Schnittstellen
- Verhindern von Jenga-Code

# Abschließendes Beispiel

- Legen neue Klasse an mit Array (verzweigt):
- Arrayzugriff nur über Funktionen bzw. Eigenschaft
- Konstanten über Enumerationen verwalten
- Nicht eingebaut: Ausnutzen von ref bzw. out aus
- Dran denken: Heap (**class**) und Stack (**struct** )