|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Technische Berichtsnr.: **3881**  Projektname: **Wifi Komplettausbildung**  Bearbeitet: Markus-Kragl  Geprüft: -  Datum: 2018\_12\_14\_ | | | | | | Lunzerstrasse 78  Postfach 28  4031 Linz, Austria  Phone: +43 (732) 6986-0 | | |
| **WifiKurs-3881-Software Developer in C# - Komplettausbildung** | | | | | | | | |
| **Zusammenfassung**  Dieser Bericht beschreibt: | | | | | | | | |
| Revision | .Bearbeitet | Datum | Geprüft | Datum | .Rev. folgender Seiten | | | |
| .  .  .  .  .  . | .  .  .  .  .  . | .  .  .  .  .  . | .  .  .  .  .  . | .  .  .  .  .  . | .  .  .  .  .  . | | | |
|  | | | | | | | Anzahl der Seiten  28 Pages |  |
| Seiten Nr.: | Seite 1 von 24 |

**Contents**

[1 Prüfung 4](#_Toc12717190)

[1.1 Prüfungsmodus 4](#_Toc12717191)

[1.2 Mögliche Prüfungsfragen 4](#_Toc12717192)

[2 .Net-Framework 5](#_Toc12717193)

[2.1 Symbole 5](#_Toc12717194)

[2.2 Aufbau 7](#_Toc12717195)

[3 C#-Referenz 8](#_Toc12717196)

[3.1 Schlüsselwörter 8](#_Toc12717197)

[3.2 Kontextabhängige Schlüsselwörter 8](#_Toc12717198)

[3.3 Typen 8](#_Toc12717199)

[3.4 Zugriffsmodifizierer 11](#_Toc12717200)

[3.5 Anweisungsschlüsselwörter 13](#_Toc12717201)

[4 C#-Programmierhandbuch 17](#_Toc12717202)

[4.1 Assemblies 17](#_Toc12717203)

[4.2 Namespaces 17](#_Toc12717204)

[4.3 Klassen 18](#_Toc12717205)

[4.4 Konstruktor und Destruktor 20](#_Toc12717206)

[4.5 Methode 22](#_Toc12717207)

[4.6 Eigenschaft 23](#_Toc12717208)

[4.7 Ereignis 25](#_Toc12717209)

[4.8 Schnittstellen 26](#_Toc12717210)

[4.9 Delegate 27](#_Toc12717211)

[4.10 Arrays 28](#_Toc12717212)

[4.11 Generika 29](#_Toc12717213)

[5 Mehrschichtenmodel 32](#_Toc12717214)

[5.1 Model View Controller 32](#_Toc12717215)

[5.2 Firmenframework 32](#_Toc12717216)

[6 WPF-Anwendung 34](#_Toc12717217)

[6.1 MVVM – Mehrschichtenmodel 34](#_Toc12717218)

[7 Xaml 36](#_Toc12717219)

[7.1 Databinding 36](#_Toc12717220)

[7.2 CLR Objects 36](#_Toc12717221)

[7.3 Converters 37](#_Toc12717222)

[7.4 Data Template 37](#_Toc12717223)

[7.5 XPath 38](#_Toc12717224)

[7.6 Relative Source 38](#_Toc12717225)

[7.7 Static Resources 38](#_Toc12717226)

[7.8 Dynamic Resources 38](#_Toc12717227)

[7.9 Style 39](#_Toc12717228)

[7.10 Triggers 39](#_Toc12717229)

[7.11 Control Template 41](#_Toc12717230)

[7.12 Transforms 41](#_Toc12717231)

[8 MySql 42](#_Toc12717232)

[8.1 LINQ 42](#_Toc12717233)

[9 Präprozessoranweisung 43](#_Toc12717234)

[10 Verwendung von “override” und “new” 44](#_Toc12717235)

[11 ESCAPE – SEQUENZEN 45](#_Toc12717236)

[12 Warum ist System.Random static 45](#_Toc12717237)

[13 Klassenbibliothek 45](#_Toc12717238)

[14 Assemblyversion 46](#_Toc12717239)

[15 Zeichenfolgeninterpolation 46](#_Toc12717240)

[16 Implizit typisierte lokale Variablen 46](#_Toc12717241)

[17 Zusatzinformation für eine Anwendung 48](#_Toc12717242)

# Prüfung

## Prüfungsmodus

1. Einheit – Prüfung

* 1.5 Stunden offene Fragen
* 1.5 Stunden Präsentation

2. Einheit – Prüfung

* 4 Programmbeispiele
* 1 kann gestrichen werden

## Mögliche Prüfungsfragen

Internal – Schlüsselwort

* Für Klassen ausserhalb der Assembly nicht sichtbar

String bearbeiten

* HÜ 21.03.2019
* https://docs.microsoft.com/de-de/dotnet/standard/base-types/basic-string-operations

Wie wird direkt auf den Speicher eines Arrays zugegriffen

* LINQ - durchlesen

Längsten Protokolleintragtyp zurückgeben

* 14.03.2019

Konstruktor über einen anderen aufrufen

* 14.03.2019

Typisches Prüfungsbeispiel: überschreiben von Methoden

HÜ 170119 🡪 Fenster XML Prüfungsbeispiel!!!

Prüfungsbeispiel: einfügen von Methode „Hinterlegen“ und implementieren

Prüfungsbeispiel: Eigenschaft programmieren (Aufgabenstellung in Kommentare schreiben!)

Prüfungsfrage 1303 (oder 1304?) 3.Frage (mit der Antwort: wenn private dann würde private dastehen daher eher protected)

Prüfungsfrage: Was bedeutet internal: möchte man die Klasse außerhalb des Assemblies nicht sehen dann verwendet man internal als Modifizierer

Prüfungsfrage: Was bedeutet SqlInjection: Hacker nutzen Schwachstellen wen nicht richtig mit Parametern zugegriffen wird d.h. keine Sql Anweisung im Code zusammensetzen 🡪 nur mit Procedures

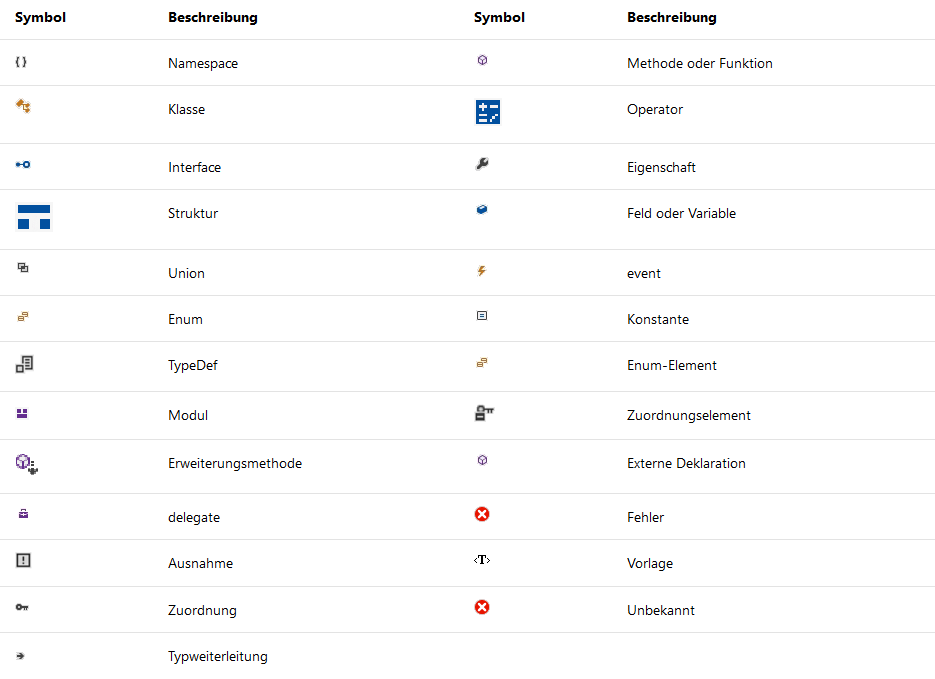
Immer Kommentare bei der Prüfung verwenden

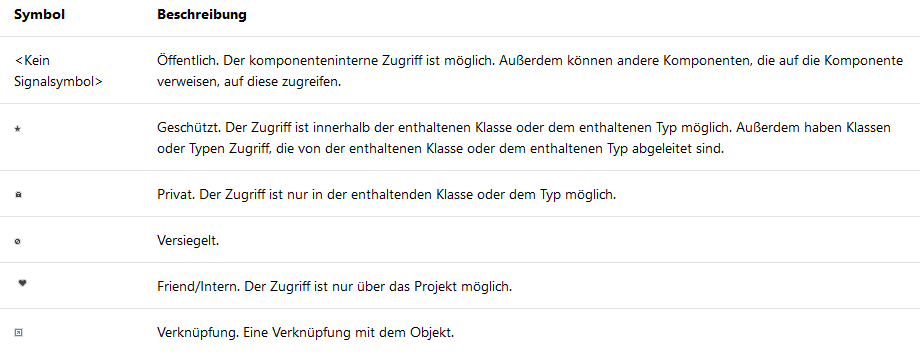
# .Net-Framework

Viele Klassen sind bereits im .Net-Framework beschrieben, damit muss nicht alles von neue beschrieben werden

<https://docs.microsoft.com/de-at/visualstudio/ide/class-view-and-object-browser-icons?view=vs-2017>

## Symbole





## Aufbau

⏹-⏹Assembly

{} Namespace

Symbol "Klasse" Klasse

Symbol "Deklaration" Methode

Symbol "Eigenschaften" Eigenschaft

Symbol "Ereignis" Ereignis

Schnittstellensymbol "Lollipop" Interfaces

Symbol "Struktur" Struktur

# C#-Referenz

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/>

## Schlüsselwörter

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/>

as

base

cachen

Multithreadings

Using!!!

Dispatcher

Async

Structure

Ereignis

## Kontextabhängige Schlüsselwörter

Ein Kontextschlüsselwort wird verwendet, um eine spezifische Bedeutung im Code bereitzustellen, es ist jedoch kein reserviertes Wort in C#

## Typen

|  |  |
| --- | --- |
| **Deklarierter Zugriff** | **Bedeutung** |
| [void](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/void) | Aus der Methode wird kein Wert zurückgegeben |
| [var](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/var) | Ab Visual Studio C# 3.0 können Variablen, die im Methodenbereich deklariert wurden, den impliziten „Typ“ var haben. Eine implizit typisierte lokale Variable ist stark typisiert, als hätten Sie den Typ selbst deklariert. Tatsächlich legt der Compiler den Typ fest. |

Das C#-Typensystem enthält die folgenden Kategorien:

* Werttypen
* Verweistypen
* Zeigertypen

Variablen, die Werttypen sind speichern Daten, und solche, die Verweistypen sind, verweisen auf die tatsächlichen Daten. Instanzen von Verweistypen werden auch als Objekte bezeichnet. Zeigertypen können nur im unsafe-Modus genutzt werden.

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/types>

### Werttypen

Eine Variable eines Werttyps enthält einen Wert des Typs. Beispielsweise kann eine Variable von Typ int den 42-Wert enthalten

Die einfachen Typen sind eine Reihe von vordefinierten Strukturtypen, die von C# bereitgestellt werden, und umfassen die folgenden Typen:

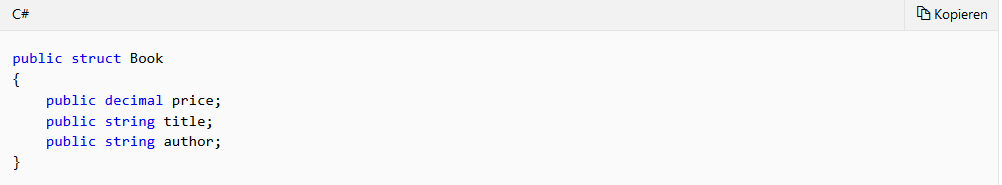
* Ganzzahltypen: ganzzahlige numerische Typen und der char-Typ, int-Typ
* Gleitkommatypen
* bool

Es gibt zwei Werttypen:

* Strukturen
* Enumerationen

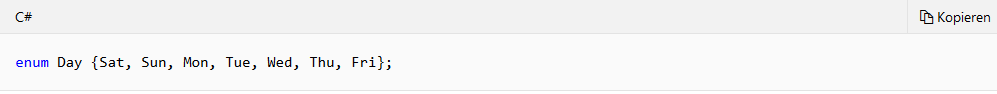
#### Strukturen

Ein struct-Typ ist ein ein Werttyp, der in der Regel verwendet wird, um eine kleine Gruppe verwandter Variablen zusammenzufassen



#### Enumerationem

Das Schlüsselwort enum wird zum Deklarieren einer Enumeration verwendet. Dies ist ein eigener Typ, der aus einer Gruppe benannter Konstanten besteht, die Enumeratorliste genannt wird.



### Verweistypen

Variablen von Verweistypen speichern Verweise auf ihre Daten (Objekte). Bei Verweistypen können zwei Variablen auf dasselbe Objekt verweisen. Daher können auf eine Variable angewendete Operationen das Objekt beeinflussen, auf das von der anderen Variablen verwiesen wird. Bei Werttypen besitzt jede Variable eine eigene Kopie der Daten, und auf eine Variable angewendete Operationen können die andere Variable nicht beeinflussen (außer im Fall von ref- und out-Parametervariablen, siehe in, ref und out-Parametermodifizierer).

Die folgenden Schlüsselwörter werden verwendet, um Verweistypen zu deklarieren:

* class
* interface
* delegate

C# enthält auch die folgenden integrierten Referenztypen:

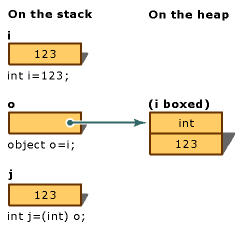
* dynamic
* object
* string

**dynamic**

Wird einstweilen nicht näher behandelt

**object**

Der object-Typ ist ein Alias für Object in .NET. Im vereinheitlichen Typsystem von C# erben alle Typen, vordefiniert und benutzerdefiniert sowie Verweis- und Werttypen, direkt oder indirekt von Object. Sie können Werte eines beliebigen Typs Variablen des Typs object zuweisen. Wenn eine Variable eines Werttyps in ein Objekt konvertiert wird, gilt es als geschachtelt. Wenn eine Variable eines Typobjekts in ein Wertobjekt konvertiert wird, gilt es als nicht geschachtelt. Weitere Informationen finden Sie unter Boxing und Unboxing (casten).



**string**

Wird einstweilen nicht näher behandelt

### Zeigertypen

Wird einstweilen nicht näher betrachtet.

## Zugriffsmodifizierer

Modifizierer werden zum Ändern von Typ- und Typmemberdeklarationen verwendet. In diesem Abschnitt werden die C#-Modifizierer beschrieben.

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/modifiers>

|  |  |
| --- | --- |
| **Modifizierer** | **Zweck** |
| [Zugriffsmodifizierer](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/access-modifiers) -[public](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/public) -[private](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/private) -[internal](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/internal) - [protected](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/protected) | Gibt den deklarierten Zugriff von Typen und Typmembern an. |
| [abstract](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/abstract) | Zeigt an, dass eine Klasse lediglich als Basisklasse anderer Klassen verwendet werden soll. |
| [async](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/async) | Zeigt an, dass die geänderte Methode, der Lambdaausdruck oder die anonyme Methode asynchron ist. |
| [const](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/const) | Gibt an, dass der Wert des Felds oder der lokalen Variablen nicht geändert werden kann. |
| [event](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/event) | Deklariert ein Ereignis. |
| [extern](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/extern) | Zeigt an, dass die Methode extern implementiert wurde. |
| [new](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/new) | Blendet einen von einer Basisklasse geerbten Member explizit aus. |
| [override](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/override) | Stellt eine neue Implementierung eines virtuellen Members bereit, der von einer Basisklasse geerbt wurde. |
| [partial](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/partial-type) | Definiert Teilklassen, Strukturen und Methoden innerhalb derselben Assembly. |
| [readonly](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/readonly) | Deklariert ein Feld, dem Werte nur als Teil einer Deklaration oder in einem Konstruktor derselben Klasse zugewiesen werden können. |
| [sealed](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/sealed) | Gibt an, dass eine Klasse nicht geerbt werden kann. |
| [static](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/static) | Deklariert einen Member, der zum Typ selbst und nicht zu einem bestimmten Objekt gehört. |
| [unsafe](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/unsafe) | Deklariert einen nicht sicheren Kontext. |
| [virtual](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/virtual) | Deklariert eine Methode oder eine Zugriffsmethode, deren Implementierung durch einen überschreibenden Member in einer abgeleiteten Klasse geändert werden kann. |
| [volatile](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/volatile) | Gibt an, dass ein Feld im Programm z. B. vom Betriebssystem, der Hardware oder einem gleichzeitig ausgeführten Thread geändert werden kann. |

### Zugriffsebenen

Verwenden Sie die Zugriffsmodifizierer public, protected, internal oder private, um eine der folgenden deklarierten Zugriffsebenen für Member anzugeben.

<https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/accessibility-levels>

|  |  |
| --- | --- |
| **Deklarierter Zugriff** | **Bedeutung** |
| [public](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/public) | Der Zugriff ist nicht beschränkt. |
| [protected](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/protected) | Der Zugriff ist auf die enthaltende Klasse oder auf Typen beschränkt, die von der enthaltenden Klasse abgeleitet sind. |
| [internal](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/internal) | Der Zugriff ist auf die aktuelle Assembly beschränkt. |
| [protected internal](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/protected-internal) | Der Zugriff ist auf die aktuelle Assembly oder auf Typen beschränkt, die von der enthaltenden Klasse abgeleitet sind. |
| [private](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/private) | Der Zugriff ist auf die enthaltende Klasse beschränkt. |
| [private protected](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/private-protected) | Der Zugriff ist auf die enthaltende Klasse oder auf Typen beschränkt, die von der enthaltenden Klasse innerhalb der aktuellen Assembly abgeleitet sind. Verfügbar seit C# 7.2. |

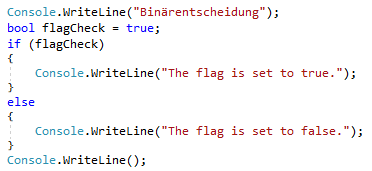
## Anweisungsschlüsselwörter

Anweisungen sind Instruktionen des Programms. Sie werden nacheinander ausgeführt.

| **Kategorie** | **C#-Schlüsselwörter** |
| --- | --- |
| Auswahlanweisungen | [if](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/if-else), [else](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/if-else), [switch](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/switch), [case](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/switch) |
| Iterationsanweisungen | [do](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/do), [for](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/for), [foreach](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/foreach-in), [in](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/foreach-in), [while](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/while) |
| Sprunganweisungen | [break](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/break), [continue](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/continue), [default](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/switch), [goto](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/goto), [return](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/return), [yield](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/yield) |
| Ausnahmebehandlungsanweisungen | [throw](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/throw), [try-catch](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/try-catch), [try-finally](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/try-finally), [try-catch-finally](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/try-catch-finally) |
| [Checked und unchecked](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/checked-and-unchecked) | [checked](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/checked), [unchecked](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/unchecked) |
| [fixed-Anweisung](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/fixed-statement) | [fixed](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/fixed-statement) |
| [lock-Anweisung](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/lock-statement) | [lock](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/lock-statement) |

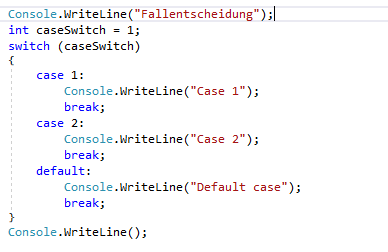
### If-else Binärentscheidung

Die if-Anweisung wählt eine Anweisung für die Ausführung anhand des Werts eines Boolean-Ausdrucks aus. Im folgenden Beispiel wird ein Boolean-Flag flagCheck auf true festgelegt und anschließend in der if-Anweisung geprüft. Die Ausgabe lautet



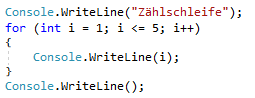
### Switch, Case Fallentscheidung

Die switch-Anweisung stellt eine Steuerungsanweisung dar, die mehrere Auswahlmöglichkeiten und Enumerationen behandelt, indem die Steuerung an eine der case-Anweisungen innerhalb ihres Texts übergeben wird.



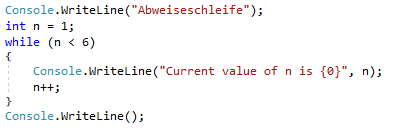
### For - Zählschleife

Mit der for-Schleife wird eine Anweisung oder ein Anweisungsblock wiederholt ausgeführt, bis ein bestimmter Ausdruck den Wert false liefert. Die for-Schleife ist nützlich zum Durchlaufen von Arrays und für sequenzielle Verarbeitung. Im folgenden Beispiel wird der Wert von int i in der Konsole ausgegeben, und i wird bei jedem Schleifendurchlauf um 1 erhöht.



### While - Abweiseschleife

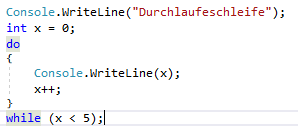
Mit der while-Anweisung wird eine Anweisung oder ein Anweisungsblock ausgeführt, bis ein bestimmter Ausdruck den Wert false liefert.



### Do – While - Durchlaufeschleife

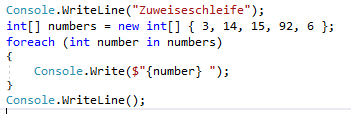
Mit der do-Anweisung wird eine Anweisung oder ein Anweisungsblock, eingeschlossen in {}, wiederholt ausgeführt, bis ein bestimmter Ausdruck den Wert false liefert. Im folgenden Beispiel werden die

Anweisungen in der do-Schleife solange ausgeführt, solange die Variable y kleiner als 5 ist.



### Foreach in - Zuweiseschleife

Schleife um einer Variable nacheinander alle Elemente einer Liste zuzuweisen.



# C#-Programmierhandbuch

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index>

## Assemblies

Beinhaltet alle Projektmappen

## Namespaces

Hier werden Gültigkeitsreiche für eine Gruppe von Objekten u. Codes deklariert.

Innerhalb eines Namespace können Sie 0 (null) oder mehr der folgenden Typen deklarieren:

* Class
* Interfaces
* Struct
* Enum
* Delegate
* Nested namespace

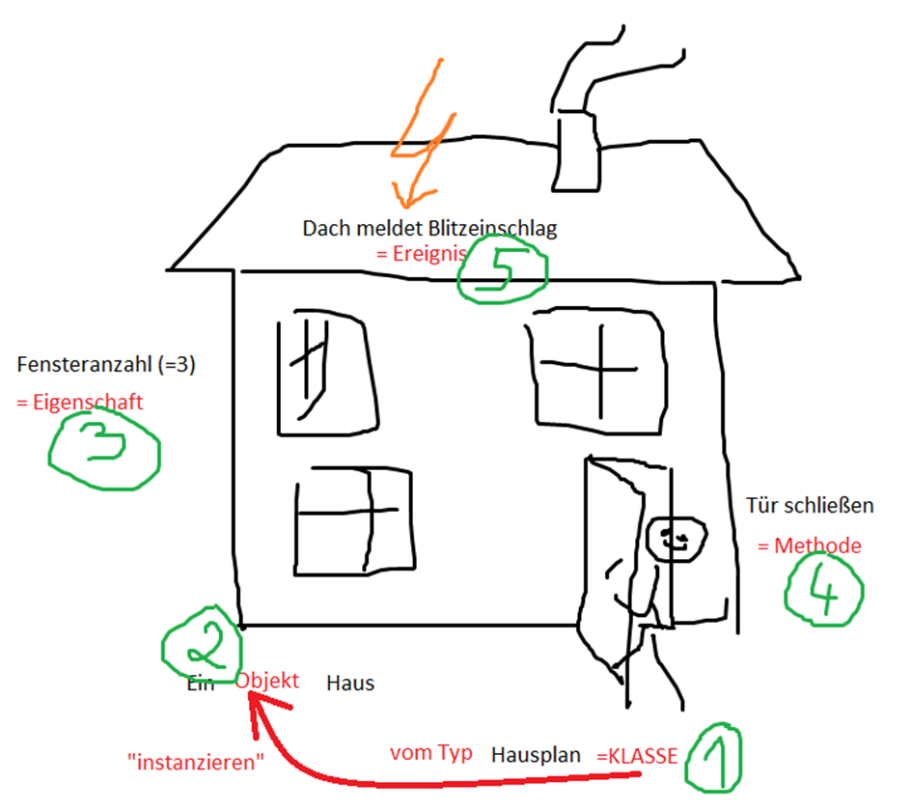
## Klassen

Die Klasse stellt den Haus Plan eines Programms dar. Klassen im westlichen durch

* Methoden => arbeitet
* Eigenschaften => beschreibt
* Ereignis => Mitteilungen

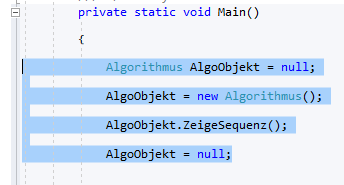
beschrieben werden.

Damit man mit einer Klasse arbeiten kann muss diese instanziiert werde also ein Objekt daraus erstellt werden.



### Instanzieren (Objekt erstellen)

Die Instanziierung der Klasse (Objekt Haus) erfolgt mit dem Schlüsselwort new.



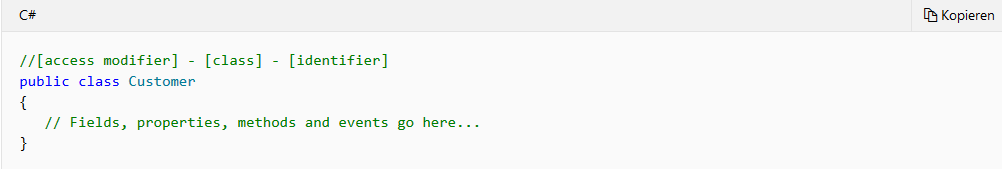
### Code - Klasse

public class Custommer

- Name der Methode

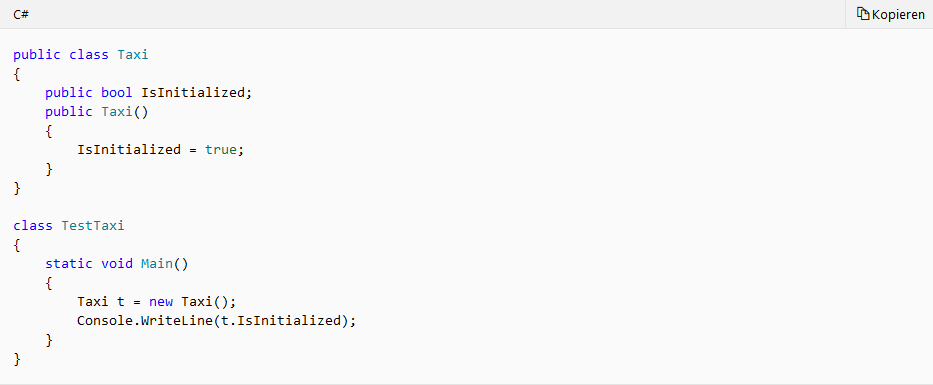
- Rückgabetyp

- Zugriffsmodifizierer

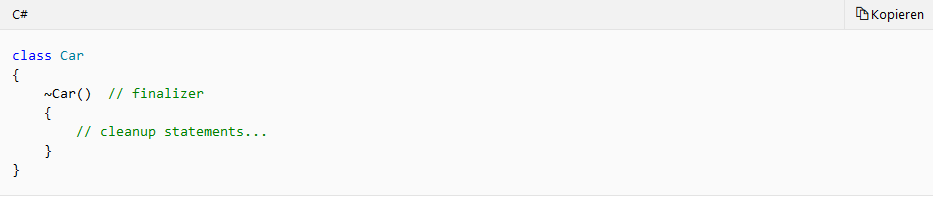


## Konstruktor und Destruktor

Der Konstruktor ist eine spezielle Funktion, welche bei der Initialisierung (also dem Erstellen) einer Klasse aufgerufen wird. Der Name der Konstruktor-Funktion entspricht immer dem Klassennamen. Die Funktion besitzt keinen Rückgabewert (auch nicht void), da der Konstruktor sozusagen das neu initialisierte Objekt zurückgibt. Wird kein Konstruktor in der Klasse erstellt, gibt es immer einen sogenannten Standard-Konstruktor, welcher keine weiteren Aktionen ausführt. Der Standard-Konstruktor besitzt keine Parameter. Jedoch kann dies bei der Implementierung eigener Konstruktoren geändert werden. Des Weiteren ist auch die Überladung des Konstruktors möglich. Wichtig zu wissen ist, dass sobald Konstruktoren in der Klasse deklariert werden, es den Standard-Konstruktor den C# automatisch erstellt, nicht mehr gibt. Der Zugriffsmodifizierer bei Konstruktoren ist normalerweise public. Wird als Zugriffsmodifzierer private verwendet, ist die Objektinitialisierung dieser Klasse nicht mehr möglich. Der private Konstruktor wird deshalb verwendet, um die Objektinitialisierung einer Klasse, welche lediglich statische Funktionen enthält, zu unterbinden.



Der Destruktor ist das Gegenteil des Konstruktors, da der Destruktor aufgerufen wird, wenn das Objekt zerstört / gelöscht wird. Wie bereits angesprochen, wird ein Objekt (welches auf dem Heap gespeichert ist) automatisch gelöscht, sobald das .NET Framework keine Referenzierung mehr findet. Dieses Löschen der Objekte geschieht automatisch durch den Garbage Collector (kurz GC), wenn das Programm aktuell keine Vorgänge zu bearbeiten hat oder der Speicherplatz kritisch wird. Des Weiteren ist es auch möglich, über die statische Funktion Collect() der Klasse GC das Löschen unreferenzierter Objekte manuell auszulösen. Der Name des Destruktors entspricht ebenfalls dem Klassennamen, jedoch mit einem vorangestellten Tilde-Zeichen (~). Beim Destruktor darf des Weiteren kein Zugriffsmodifizierer und Rückgabewert angegeben werden.



## Methode

Eine Methode ist ein Codeblock der eine Reihe von Anweisungen enthält

Die MAIN Methode ist der Einstiegspunkt.

### Code - Methode

public static void MethodenName ()

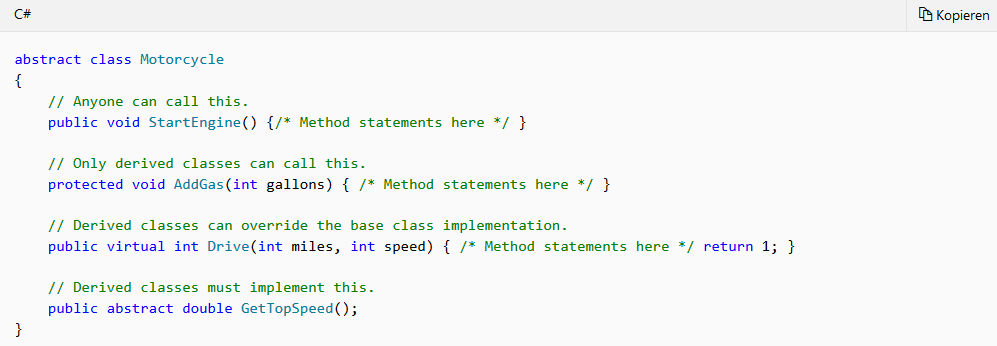
- Schnittstelle

- Name der Methode

- Rückgabetyp

- Zugriffsmodifizierer

Zugriffsmodifizierer



## Eigenschaft

Eine Eigenschaft ist ein Member, das einen flexiblen Mechanismus zum Lesen, Schreiben oder Berechnen des Werts eines privaten Felds bietet.

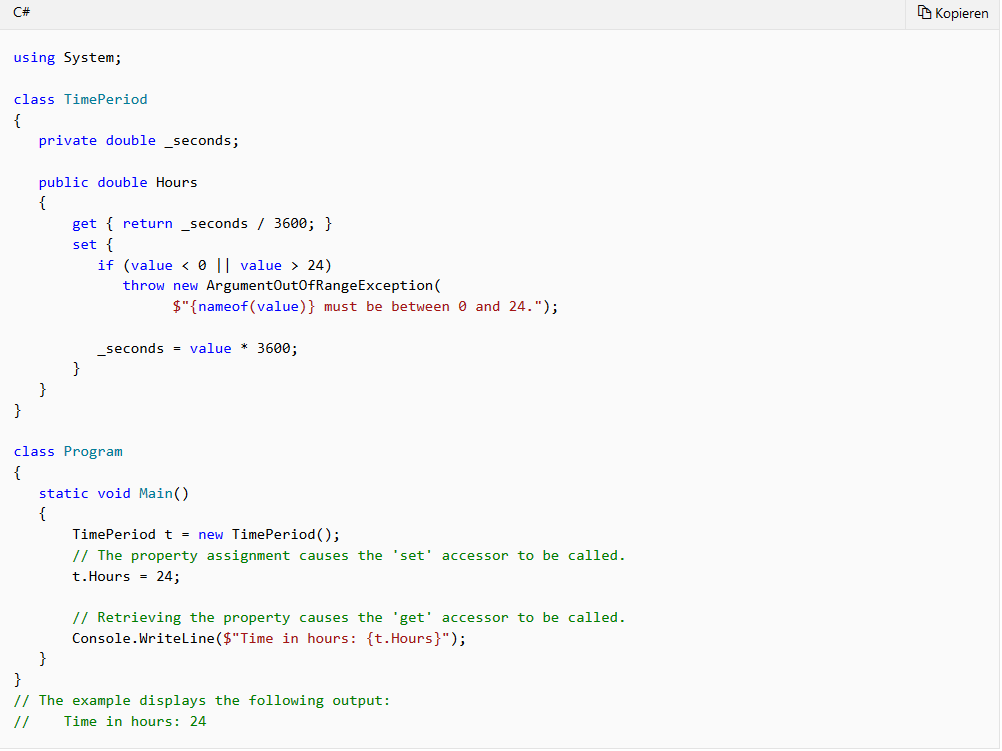
Mithilfe von Eigenschaften kann eine Klasse eine öffentliche Methode zum Abrufen und Festlegen von Werten verfügbar machen und dabei den Implementierungs- oder Verifizierungscode ausblenden.

Eine get-Eigenschaftenaccessor wird verwendet, um den Wert der Eigenschaft zurückzugeben. Ein set-Eigenschaftenaccessor wird verwendet, um einen neuen Wert zuzuweisen. Diese Zugriffsmethoden können über verschiedene Zugriffsebenen verfügen. Weitere Informationen finden Sie unter Einschränken des Accessorzugriffs.

Das **value**-Schlüsselwort wird verwendet, um den Wert zu definieren, der vom set-Accessor zugewiesen wird.

Eigenschaften können sein: Lesen/Schreiben (beide verfügen über einen get- und set-Accessor), schreibgeschützt (verfügen über einen get-Accessor, jedoch keinen set-Accessor), oder lesegeschützt (verfügen über einen set-Accessor, jedoch keinen get Accessor). Lesegeschützte Eigenschaften sind selten und werden am häufigsten verwendet, um den Zugriff auf vertrauliche Daten einzuschränken.

### Code - Eigenschaft



## Ereignis

### Code - Ereignis

## Schnittstellen

<https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/interfaces/>

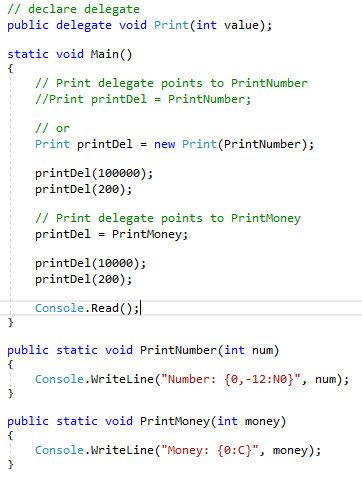
### Code - Schnittstelle

## Delegate

<https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/delegates/index>

Ein Delegat ist ein Typ, der Verweise auf Methoden mit einer bestimmten Parameterliste und dem Rückgabetyp darstellt. Nach Instanziierung eines Delegaten können Sie die Instanz mit einer beliebigen Methode verknüpfen, die eine kompatible Signatur und einen kompatiblen Rückgabetyp aufweist. Sie können die Methode über die Delegatinstanz aufrufen.

### Code - Delegate

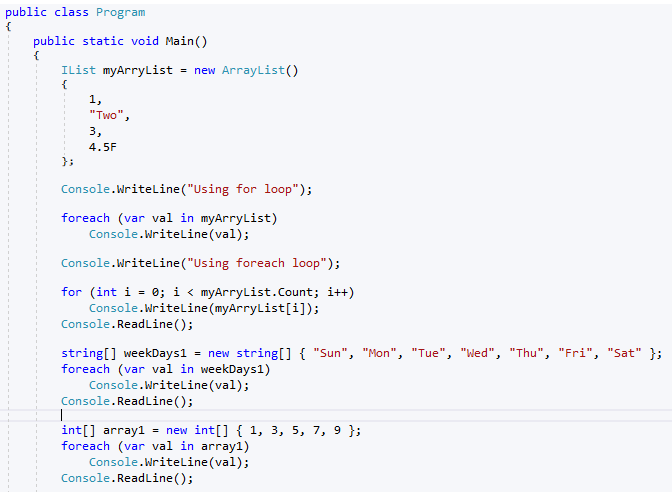


## Arrays

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/>

Wird nicht weiter behandelt.

### Code - Arrays



## Generika

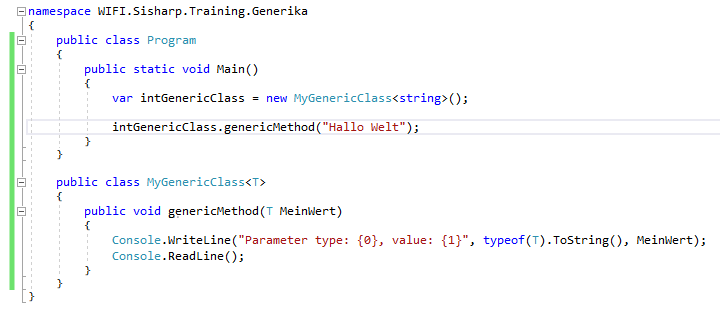
<https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/generics/>

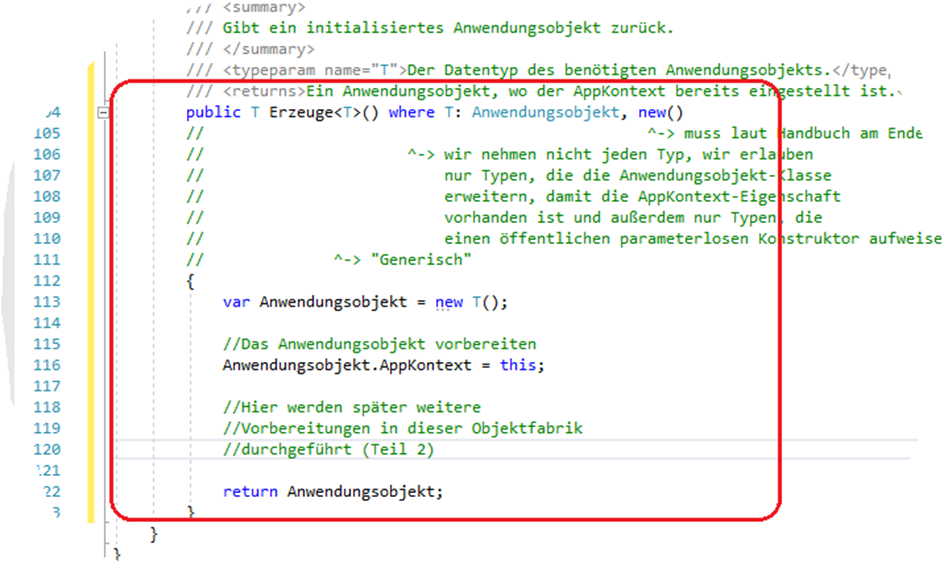
Generische Klassen und Methoden vereinen Wiederverwendbarkeit, Typsicherheit und Effizienz so, wie es ihre nicht generischen Gegenstücke nicht können. Generics werden am häufigsten für Auflistungen und deren Methoden verwendet.

Der Typparameter T wird an mehreren Stellen verwendet, wo für gewöhnlich ein konkreter Typ verwendet werden würde, um den Typ des Elements anzugeben, das in der Liste gespeichert wurde. Er wird wie folgt verwendet:

* Als Typ eines Methodenparameters in der Methode AddHead.
* Als Rückgabetyp der Eigenschaft Data in der geschachtelten Klasse Node.
* Als Typ des privaten Members data in der geschachtelten Klasse.

### Code – Generika



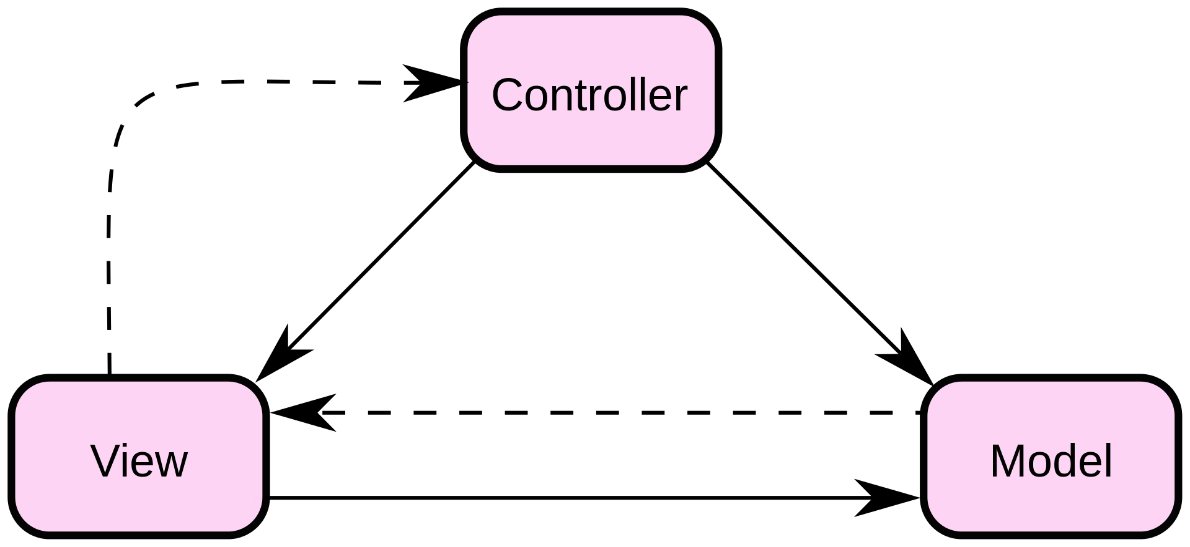


# Mehrschichtenmodel

Teil 1 setzt sich aus dem Model View Controller Mehrschichtenmodell zusammen, das die Datentransferobjekte und die Infrastruktur enthält.

<https://de.wikipedia.org/wiki/Model_View_Controller>

## Model View Controller



UserInterface (Windows Forms, WPF,…)

* Benutzeroberfläche, mit dem der User arbeiten kann

Controller

* Logik für den Zugriff auf die Daten bzw. dessen Erstellung

Geschäftslogik (Modell)

* Enthält Daten die von der Benutzeroberfläche dargestellt werden

## Firmenframework

Infrastruktur

* Hat zugriff auf alle Ebenen (ModelViewController) via Interfaces

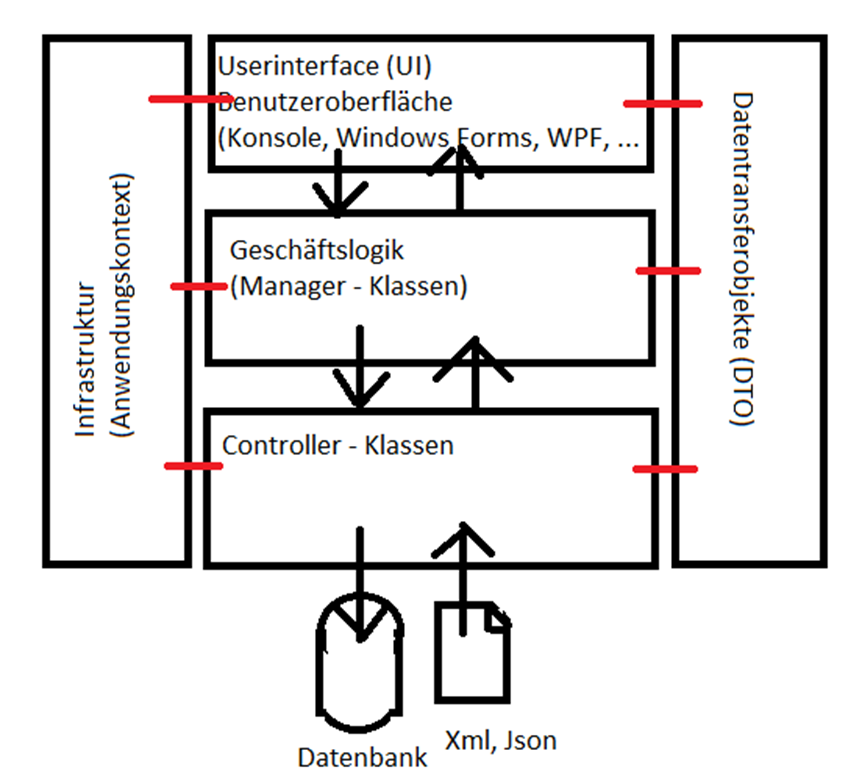
Datentranferobjekte (DTO)

* Hier werden die Daten aus dem Firmenframwork hinterlegt (Sprachen und Fenster)

Datenbank

* Hier sind alle relevanten Daten hinterlegt bzw. werden sie aus dieser abgerufen

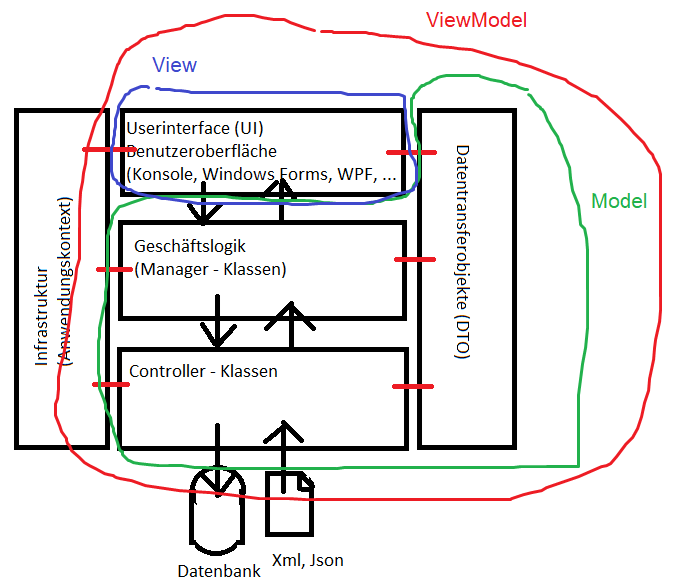
XML, Json



# WPF-Anwendung

<https://docs.microsoft.com/de-de/visualstudio/designers/getting-started-with-wpf?view=vs-2019>

## MVVM – Mehrschichtenmodel



### Model

Alles was mit den Daten zu tun hat, wird als „Model“ bezeichnet.

Controller

* Logik für den Zugriff auf die Daten bzw. dessen Erstellung

Geschäftslogik

* Enthält Daten die von der Benutzeroberfläche dargestellt werden

### View

Alles was mit der Benutzeroberfläche zu tun hat wird als “View” bezeichnet.

### View Model

Das Model und die Views werden von „ViewModel“ Klassen (Objekten) kontrolliert und gesteuert.

# Xaml

## Databinding

### Peer Control’s Properties

<Slider Name="myName" />

<TextBox Text="{Binding ElementName=myName, Path=Value}" />

<!-- SAME AS -->

<TextBox Text="{Binding Value, ElementName=myName}" />

<!-- EXAMPLE: -->

<TextBlock Text="{Binding SelectedItem.Header, ElementName=treeView}" />

### Binding Modes

<TextBox Text="{Binding Path=Text, ElementName=txtUsername Mode=TwoWay,

UpdateSourceTrigger=LostFocus}" />

Mode=OneTime

Mode=OneWay

Mode=OneWayToSource

Mode=TwoWay

## CLR Objects

### Properties

<!-- Binding to Local Properties -->

<ComboBox ItemsSource="{Binding Path=ClientsList, ElementName=myClass, Mode=Default}" />

<TextBlock Text="{Binding Source={StaticResource CurrentPerson}, ElementName=myClass,

Path=Age}" />

### Native Objects

<!-- Namespace Declaration for System namespace -->

xmlns:clr="clr-namespace:System;assembly=mscorlib"

<!-- Regular CLR Object Instantiation -->

<clr:Decimal x:Key="Current">352</clr:Decimal>

<!-- Complex Classes Instantiation -->

<src:Client x:Key="CurrentClient" ClientID="32" Active="True" />

<!-- Binding to CLR Objects -->

<TextBlock Text="{Binding Source={x:Static clr:DateTime.Now}}" />

### Methods

<ObjectDataProvider x:Key="InitializedData" ObjectType="{x:Type src:Person}" MethodName="GetUserByID">

<ObjectDataProvider.ConstructorParameters>

<clr:Int32>152</clr:Int32>

</ObjectDataProvider.ConstructorParameters>

</ObjectDataProvider>

## Converters

<!-- Converters must impliment IValueConverter -->

<BooleanToVisibilityConverter x:Key="VisConverter" />

<CheckBox Name="chkVisible" Content="Show Text" IsChecked="True" />

<TextBlock Text="Hidden Text" Visibility="{Binding IsChecked, ElementName=chkVisible,

ConvertParameter=n, Converter={StaticResurce VisConverter}}" />

## Data Template

<!-- Data Template -->

<DataTemplate DataType="{x:Type src:Client}">

<TextBlock Text="{Binding CleintName}" />

</DataTemplate>

<!-- The following will display using the data template as it's Listbox.ItemTemplate -->

<ListBox ItemsSource="{Binding Source={StaticResource CurrentClients}}" />

<!-- OR -->

<ListBox ItemsSource="{Binding Source={StaticResource CurrentClients}}" >

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding CleintName}" />

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<!-- Hierarchical Data Templates for a TreeView -->

<HierarchicalDataTemplate DataType="{x:Type src:Client}"

ItemsSource="{Binding Path=Projects}">

<TextBlock Text="{Binding Path=Name}" />

</HierarchicalDataTemplate>

<HierarchicalDataTemplate DataType="{x:Type src:Project}"

ItemsSource="{Binding Path=Tasks}">

<TextBlock Text="{Binding Path=ProjectName}" />

</HierarchicalDataTemplate>

<DataTemplate DataType="{x:Type src:Task}">

<TextBlock Text="{Binding Path=TaskName}" ToolTip="{Binding Path=TaskName}" />

</DataTemplate>

## XPath

<!-- Binding to XML -->

<XmlDataProvider x:Key="xmlData" XPath="/" Source="Employees.xml" />

<ListBox ItemsSource="{Binding Source={StaticResource xmlData},XPath=Employees/IT}" />

## Relative Source

<TextBlock Text="{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type TextBlock}}}" />

<!-- FindAncestor Syntax -->

RelativeSource FindAncestor, AncestorLevel=n, AncestorType={x:Type desiredType}}

<!-- To bind to the previous data item in a data-bound collection -->

Text="{Binding RelativeSource={Binding RelativeSource={RelativeSource PreviousData}}}"

<!-- Useful for Templates -->

Text="{Binding RelativeSource={Binding RelativeSource={RelativeSource TemplatedParent}}}"

<!-- Binding to self -->

Text="{Binding RelativeSource={Binding RelativeSource={RelativeSource Self}}}"

<!-- EXAMPLE: -->

<Slider ToolTip="{Binding RelativeSource={RelativeSource Self}, Path=Value}" />

## Static Resources

<!-- Static Resources set a property with a value only once. -->

<!-- They are resources instantiated when the form loads. -->

<!-- They are very fast and efficent to apply, but you do have -->

<!-- the cost of instantiating the object up front even if you -->

<!-- never use the object. -->

<Window.Resources>

<Image x:Key="expandNode" Source="expand.png" />

</Window.Resources>

<Button Content="{StaticResource expandNode}" />

## Dynamic Resources

<!-- Dynamic resources allow the property to be updated when the resource changes -->

<!-- Also, Dynamic resources are not loaded until needed unlike StaticResources that-->

<!-- Load as soon as the XAML is loaded. However, DynamicResources are more expensive.-->

<!-- Perfect for themes because they change when a user changes colors in the CP -->

<Button Background="{DynamicResource {x:Static System.WindowBrush}}" />

## Style

<Application.Resources>

<!-- Inherit a style by useing BasedOn="{StaticResource GeneralStyle}" -->

<Style x:Key="TopButton" TargetType="{x:Type Button}" >

<Setter Property="Background" Value="#FFDDDDDD" />

</Style>

</Application.Resources>

…

<Button Style="{StaticResource TopButton}" />

## Triggers

<Style TargetType="{x:Type Button}">

<Style.Triggers>

<!-- Property Trigger -->

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Foreground" Value="Blue" />

</Trigger>

<!-- Multi Trigger -->

<MultiTrigger>

<MultiTrigger.Conditions>

<Condition Property="IsMouseOver" Value="True" />

<Condition Property="IsFocused" Value="False" />

</MultiTrigger.Conditions>

<MultiTrigger.Setters>

<Setter Property="Foreground" Value="Blue" />

</MultiTrigger.Setters>

</MultiTrigger>

<!-- Data Trigger -->

<DataTrigger Binding="{Binding RelativeSource={RelativeSource Self},

Path=Text}" Value="Blue">

<Setter Property="Forecolor" Value="Blue" />

</DataTrigger>

<!-- Event Trigger Example using a Storyboard Animation-->

<EventTrigger RoutedEvent="Loaded">

<EventTrigger.Actions>

<BeginStoryboard>

<!-- Insert Animations here-->

</BeginStoryboard>

</EventTrigger.Actions>

</EventTrigger>

</Style.Triggers>

</Style>

## Control Template

<!-- Control Template -->

<Style x:Key="RoundButton">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate>

<Border Width="22" Height="22" CornerRadius="15">

<Border.Background>

<RadialGradientBrush GradientOrigin=".3, .3">

<GradientStop Color="White" Offset=".15" />

<GradientStop Color="#888" Offset="1" />

</RadialGradientBrush>

</Border.Background>

<TextBlock Foreground="#333" Text="{Binding Text}" />

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

<!-- Control Template for Content Control -->

<Style x:Key="GridListbox" TargetType="ListBox">

<Setter Property="Template">

<Setter.Value>

<ControlTemplate>

<Grid>

<ScrollViewer VerticalScrollBarVisibility="Auto">

<ItemsPresenter />

</ScrollViewer>

</Grid>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

## Transforms

<!-- Rotate Transform -->

<Button RenderTransformOrigin="1,0" Content="Exit">

<Button.RenderTransform>

<RotateTransform Angle="90" />

</Button.RenderTransform>

</Button>

<!-- Other Transforms -->

<TranslateTransform <!-- Moves an object from its origin -->

<ScaleTransform <!-- Shrinks or Expands an object -->

<SkewTransform <!-- Skews an object -->

<MatrixTransform <!-- Custom Transform an object -->

# MySql

## LINQ

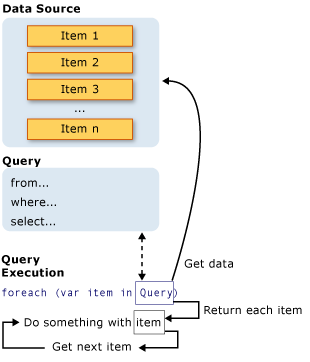
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/walkthrough-writing-queries-linq

Eine Abfrage ist ein Ausdruck, der Daten von einer Datenquelle abruft.

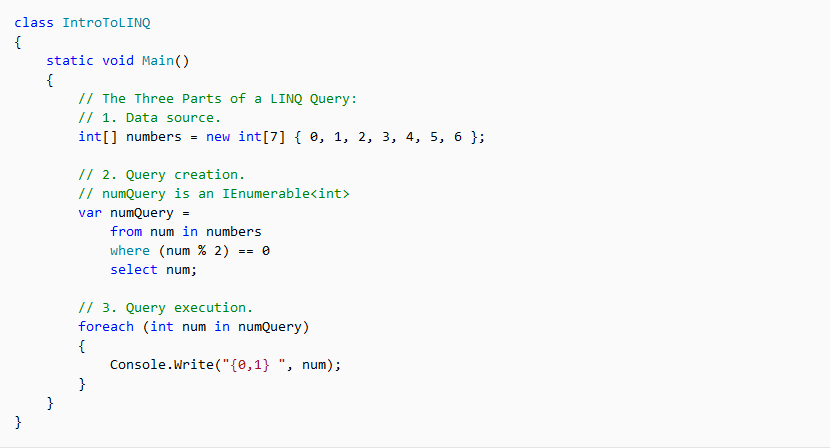
Alle LINQ-Abfrageoperationen bestehen aus drei unterschiedlichen Aktionen:

* Abrufen der Datenquelle
* Erstellen der Abfrage
* Ausführen der Abfrage



### Code – LINQ

Im folgenden Beispiel wird gezeigt, wie die drei Teile einer Abfrageoperation in Quellcode ausgedrückt werden. Das Beispiel verwendet aus praktischen Gründen ein Array von Ganzzahlen als Datenquelle. Dieselben Konzepte gelten jedoch auch für andere Datenquellen. Auf dieses Beispiel wird im Rest dieses Themas Bezug genommen.



# Präprozessoranweisung

Eine Präprozessordirektive muss die einzige Anweisung in einer Zeile sein.

#if:

#if DEBUG

Console.WriteLine("Debug version");

#endif

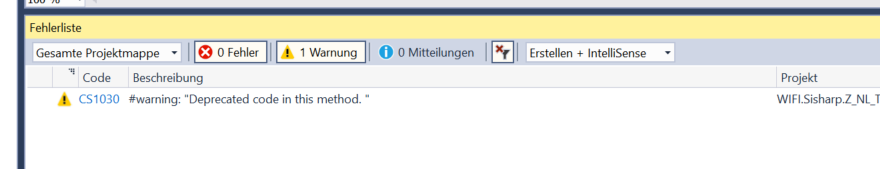
#else

#elif

#endif

#define

#undef

#warning

#error

#line

#region

#endregion

#pragma

#pragma warning

#pragma checksum

# Verwendung von “override” und “new”

Versionsverwaltung mit den Schlüsselwörtern "override" und "new" (C#-Programmierhandbuch)

Die C#-Sprache wurde entwickelt, damit die Versionierung von [base](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/base)- (Basis-) und abgeleiteten Klassen in unterschiedlichen Bibliotheken weiterentwickelt und die Abwärtskompatibilität aufrechterhalten werden kann. Das bedeutet z.B., dass die Einführung eines neuen Members in einer [Basisklasse](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/class) mit demselben Name wie ein Member in einer abgeleiteten Klasse von C# vollständig unterstützt wird und nicht zu unerwartetem Verhalten führt. Das bedeutet auch, dass eine Klasse explizit angeben muss, ob eine Methode für das außer Kraft setzen einer geerbten Methode vorgesehen ist, oder ob eine Methode eine neue Methode ist, die eine Methode mit ähnlichem Namen verbirgt.

In C# können abgeleitete Klassen Methoden mit dem gleichen Namen wie Basisklassen-Methoden enthalten.

* Die Basisklasse muss als [virtual](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/virtual) definiert werden.
* Wenn der Methode in der abgeleiteten Klasse nicht die Schlüsselwörter [new](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/new) oder [override](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/override) vorangestellt sind, gibt der Compiler eine Warnung aus, und die Methode verhält sich, als ob das Schlüsselwort new vorhanden wäre.
* Wenn der Methode in der abgeleiteten Klasse das Schlüsselwort new vorangestellt ist, wird die Methode als unabhängig von der Methode in der Basisklasse definiert.
* Wenn der Methode in der abgeleiteten Klasse das Schlüsselwort override vorangestellt ist, rufen Objekte der abgeleiteten Klasse diese Methode anstatt der Methode der Basisklasse auf.
* Die Methode der Basisklasse kann mithilfe des Schlüsselworts base aus der Basisklasse heraus aufgerufen werden.
* Die Schlüsselwörter override, virtual und new können auch auf Eigenschaften, Indexer und Ereignisse angewendet werden.

# ESCAPE – SEQUENZEN

Zeichenkombinationen, die aus einem umgekehrten Schrägstrich (**\**), gefolgt von einem Buchstaben oder einer Zahlenkombination, bestehen, werden „Escapesequenzen“ genannt. Um ein Zeilenumbruchzeichen, ein einfaches Anführungszeichen oder bestimmte andere Zeichen in einer Zeichenkonstante darzustellen, müssen Sie Escapesequenzen verwenden. Eine Escapesequenz wird als ein einzelnes Zeichen betrachtet und ist daher als Zeichenkonstante gültig.

| **Escapesequenz** | **Repräsentiert** |
| --- | --- |
| **\a** | Glocke (Warnung) |
| **\b** | Rückschritt |
| **\f** | Seitenvorschub |
| **\n** | Zeilenwechsel |
| **\r** | Wagenrücklauf |
| **\t** | Horizontaler Tabulator |
| **\v** | Vertikaler Tabulator |
| **\'** | Einfaches Anführungszeichen |
| **\"** | Doppeltes Anführungszeichen |
| **\\** | Umgekehrter Schrägstrich |
| **\?** | Literales Fragezeichen |
| **\** *ooo* | ASCII-Zeichen in der Oktalnotation |
| **\x** *hh* | ASCII-Zeichen in der Hexadezimalnotation |
| **\x** *hhhh* | Unicode-Zeichen in der Hexadezimalnotation, wenn diese Escapesequenz in einer Konstante mit Breitzeichen oder in einem Unicode-Zeichenfolgenliteral verwendet wird. |

# Warum ist System.Random static

Weil der Zufall normalverteilt werden sein soll, wenn man es jedoch als Objektmethode implimentiert so kann das nicht sein. Bzw. Laut Meisi: Ein System.Random Objekt DARFS NUR GENAU EINMAL in der gesamten Anwendung (wegen der Gleichverteilung)

# Klassenbibliothek

Ist eine Assembly die Klassen enthält die in dem gesamten Projekt (Projektmappe) verwendet werden. Besitzt keine Main Methode

# Assemblyversion

In der Assemblyinformation in den Eigenschaften änderbar.

* Hauptnummer Diese ändert sich, wenn eine andere Technologie eingesetzt wird, z. B. von Windows Forms auf WPF
* Nebennummer Diese ändert sich, wenn neue Klassen ergänzt werden.
* Nummer Diese ändert sich, wenn Eigenschaften, Methoden, … verändert wurden.
* Revision Ändert sich, wenn Fehler ausgebessert wurden.

# Zeichenfolgeninterpolation

Das Sonderzeichen $ kennzeichnet ein Zeichenfolgenliteral als interpolierte Zeichenfolge. Eine interpolierte Zeichenfolge ist ein Zeichenfolgenliteral, das möglicherweise Interpolationsausdrücke enthält. Wenn eine interpolierte Zeichenfolge in eine Ergebniszeichenfolge aufgelöst wird, werden Elemente mit Interpolationsausdrücken durch die Zeichenfolgendarstellungen der Ausdrucksergebnisse ersetzt. Dieses Feature ist in C# 6 und höher verfügbar.

Die Zeichenfolgeninterpolation bietet eine Syntax, die besser lesbar und praktischer beim Erstellen formatierter Zeichenfolgen ist als ein Feature für die [kombinierte Formatierung von Zeichenfolgen](https://docs.microsoft.com/de-de/dotnet/standard/base-types/composite-formatting). Im folgenden Beispiel wird mit beiden Features die gleiche Ausgabe erzeugt:

C#

string name = "Mark";

var date = DateTime.Now;

// Composite formatting:

Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.", name, date.DayOfWeek, date);

// String interpolation:

Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, it's {date:HH:mm} now.");

// Both calls produce the same output that is similar to:

// Hello, Mark! Today is Wednesday, it's 19:40 now.

# Implizit typisierte lokale Variablen

Verwenden Sie die [implizite Typisierung](https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/classes-and-structs/implicitly-typed-local-variables) für lokale Variablen, wenn der Typ der Variablen auf der rechten Seite der Zuweisung offensichtlich ist oder wenn der genaue Typ nicht von Bedeutung ist.

C#

 // When the type of a variable is clear from the context, use var

// in the declaration.

var var1 = "This is clearly a string.";

var var2 = 27;

var var3 = Convert.ToInt32(Console.ReadLine());

 Verwenden Sie nicht [var](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/var), wenn der Typ nicht von der rechten Seite der Zuweisung offensichtlich ist.

C#

 // When the type of a variable is not clear from the context, use an

// explicit type.

int var4 = ExampleClass.ResultSoFar();

 Verlassen Sie sich nicht auf den Variablennamen, um den Typ der Variable anzugeben. Er ist unter Umständen nicht korrekt.

C#

 // Naming the following variable inputInt is misleading.

// It is a string.

var inputInt = Console.ReadLine();

Console.WriteLine(inputInt);

 Vermeiden Sie den Einsatz von var anstelle von [dynamic](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/dynamic).

 Verwenden Sie die implizite Typisierung, um den Typ der Schleifenvariablen in [for](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/for)- und [foreach](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/foreach-in)-Schleifen zu bestimmen.

Im folgenden Beispiel wird die implizite Typisierung in einer for-Anweisung verwendet.

C#

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

}

//Console.WriteLine("tra" + manyPhrases);

Im folgenden Beispiel wird die implizite Typisierung in einer foreach-Anweisung verwendet.

C#

foreach (var ch in laugh)

{

if (ch == 'h')

Console.Write("H");

else

Console.Write(ch);

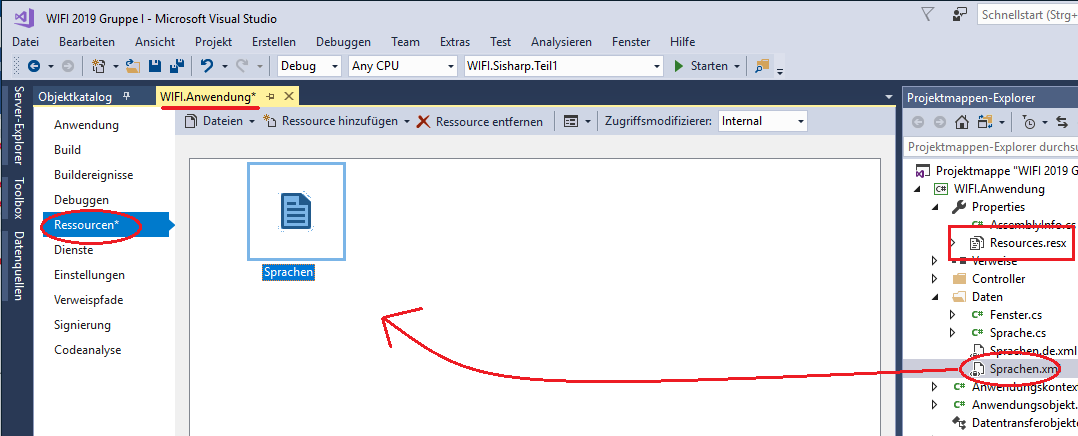
}

Console.WriteLine();

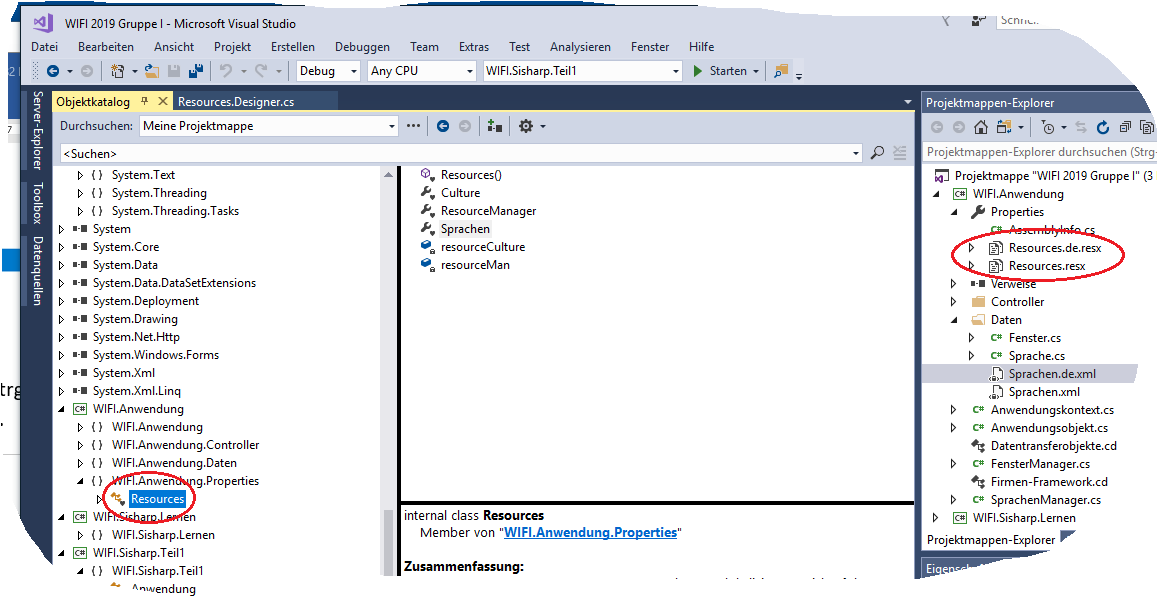
# Zusatzinformation für eine Anwendung

Wo kann Zusatzinformation für eine Anwendung, außer direkt als Dateien im Anwendungsverzeichnis, hinterlegt werden?

* In den Anwendungsressourcen (Projekteigenschaften)

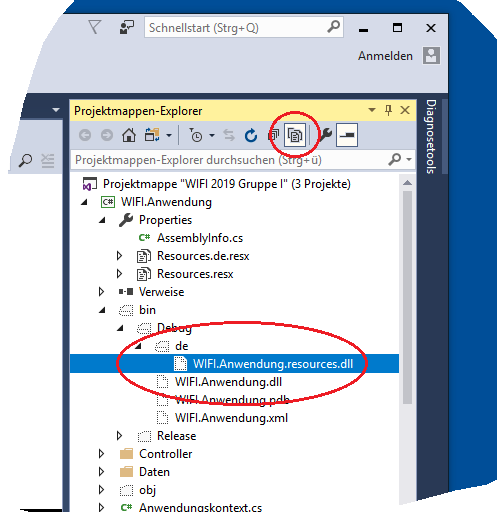


Die Ressourcen werden vom Studio über eine automatisch erstellte Klasse …



… im Properties Namespace bereitgestellt.

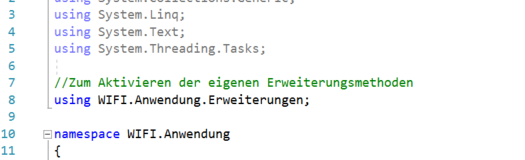
Beim Kompilieren …

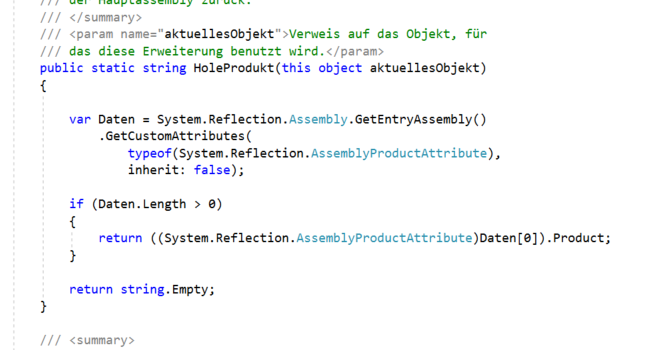


… werden lokalisierte Ressourcen in „Satelliten Assemblies“ hinterlegt.

# Erweiterungsmethoden

Mit Erweiterungsmethoden können Sie vorhandenen Typen Methoden hinzufügen, ohne einen neuen abgeleiteten Typ zu erstellen und ohne den ursprünglichen Typ neu kompilieren oder auf andere Weise bearbeiten zu müssen. Erweiterungsmethoden sind eine besondere Art von statischen Methoden, die Sie jedoch wie Instanzmethoden für den erweiterten Typ aufrufen können. Für in C#, F# und Visual Basic geschriebenen Clientcode gibt es keinen sichtbaren Unterschied zwischen dem Aufrufen einer Erweiterungsmethode und den Methoden, die in einem Typ tatsächlich definiert sind.

Erweiterungsmethoden werden als statische Methoden definiert, jedoch mithilfe einer Instanzmethodensyntax aufgerufen. Der erste Parameter bestimmt, für welchen Typ die Methode gilt, und vor dem Parameter steht der [this](https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/this)-Modifizierer. Erweiterungsmethoden befinden sich nur dann im Bereich, wenn Sie den Namespace explizit mit einer using-Direktive in Ihren Quellcode importieren. 



# Einfügen nicht verwalteter Code

Der Plattformaufruf ist ein Dienst, der es verwaltetem Code ermöglicht, nicht verwaltete Funktionen aufzurufen, die in DLLs (Dynamic Link Library) implementiert sind, z.B. die in der Windows-API enthaltenen Funktionen. Es sucht eine exportierte Funktion, ruft diese auf und marshallt ihre Argumente (ganze Zahlen, Zeichenfolgen, Arrays, Strukturen usw.) bei Bedarf über die Grenzen des dialogfähigen Betriebs hinaus.

Verwaltete Definitionen für nicht verwaltete Funktionen hängen von der Sprache ab, wie Sie in den folgenden Beispielen erkennen können. Ausführlichere Codebeispiele finden Sie unter [Beispiele für Plattformaufrufe](https://docs.microsoft.com/de-de/dotnet/framework/interop/platform-invoke-examples).

VB

Friend Class NativeMethods

Friend Declare Auto Function MessageBox Lib "user32.dll" (

ByVal hWnd As IntPtr,

ByVal lpText As String,

ByVal lpCaption As String,

ByVal uType As UInteger) As Integer

End Class

Um die Felder [BestFitMapping](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.bestfitmapping), [CallingConvention](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.callingconvention), [ExactSpelling](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.exactspelling), [PreserveSig](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.preservesig), [SetLastError](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.setlasterror) oder [ThrowOnUnmappableChar](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute.throwonunmappablechar) auf eine Visual Basic-Deklaration anzuwenden, müssen Sie das [DllImportAttribute](https://docs.microsoft.com/de-de/dotnet/api/system.runtime.interopservices.dllimportattribute)-Attribut anstelle der Declare-Anweisung verwenden.

C#

using System;

using System.Runtime.InteropServices;

internal static class NativeMethods

{

[DllImport("user32.dll")]

internal static extern int MessageBox(

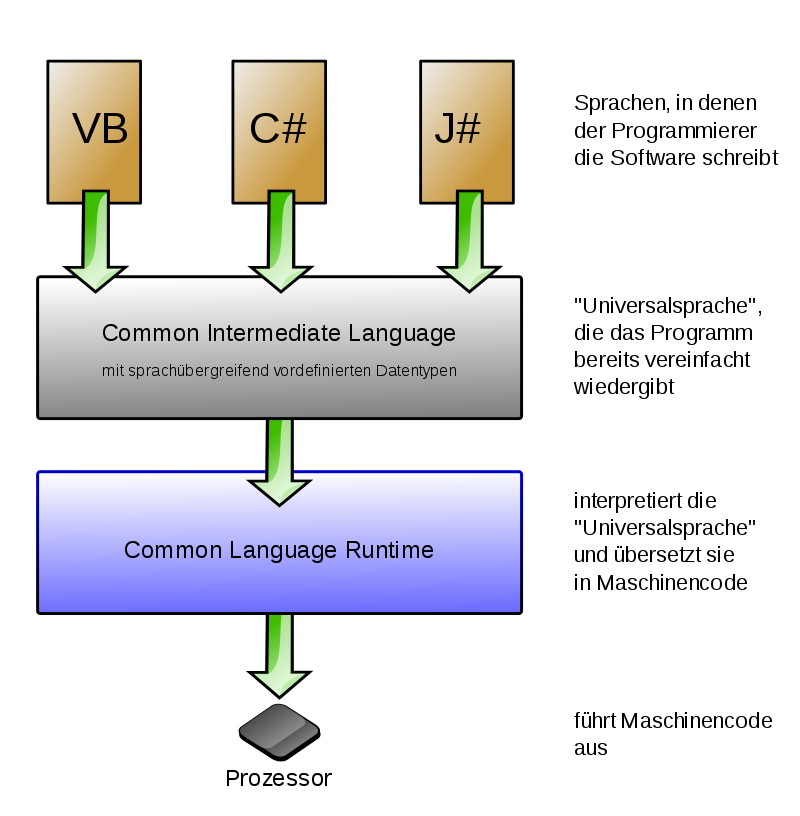
IntPtr hWnd, string lpText, string lpCaption, uint uType);

}

Die Dll wird mit einem Import Attribut eingebunden.

# Common language runtime

Common Language Runtime, kurz CLR, ist der Name für eine Laufzeitumgebung, in der .NET-Anwendungen ausgeführt werden. Sie ist ein zentraler Bestandteil des .NET Frameworks. Dem Namen entsprechend, handelt es sich um eine Laufzeitumgebung, die von Programmcode benutzt werden kann, der in unterschiedlichen Programmiersprachen geschrieben wurde.



Der Programmcode wird üblicherweise mit einem sprachspezifischen Compiler in den sog. CIL-Code übersetzt. Dabei handelt es sich um einen Zwischencode, der von der CLR verwaltet wird und der deshalb auch als managed code bezeichnet wird, als verwalteter Code – in Abgrenzung zu unmanaged code bzw. nativem Code. Die CLR enthält einen Just-in-Time-Compiler, der den CIL-Code in solchen nativen Code bzw. Maschinencode übersetzt, d. h. in Code, der von der CPU ausgeführt werden kann. Ergebnis der Übersetzung von CIL-Code in Maschinencode ist eine sog. Assembly, d. h. eine Datei mit der Dateinamenserweiterung exe oder dll.

# Was ist ein Dotfuscator

Assembler code in der CIL kann einfach dekompiliert werden und somit ist das geistige Eigentum nicht mehr geschützt. Mit dem Dotfuscator bringt man jedoch eine sochle Unordnung in den Code dass dieser für den Dekompilierer keinen Sinn mehr ergiebt.

# Hauptunterschied zwischen WPF und Forms

Windows Forms nutzt die grafische Darstellung vom alten GDI (Graphical Device Interface)

WPF benutzt die Grafikhardware (DirectX) direkt.

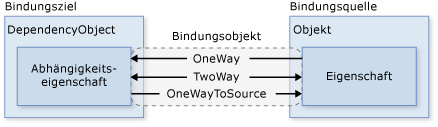
 WPF hat nur mehr „System.Windows“

 Forms hat ja „System.Windows.Forms“ benutzt

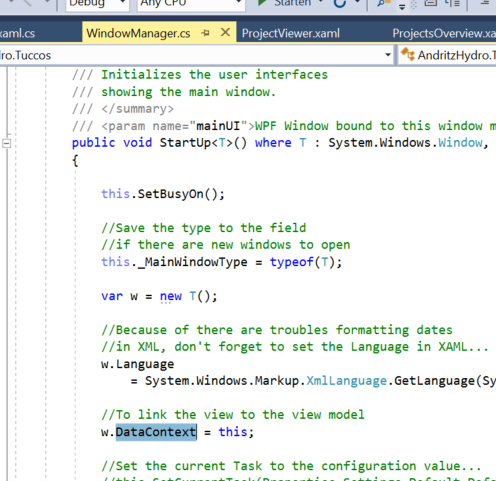
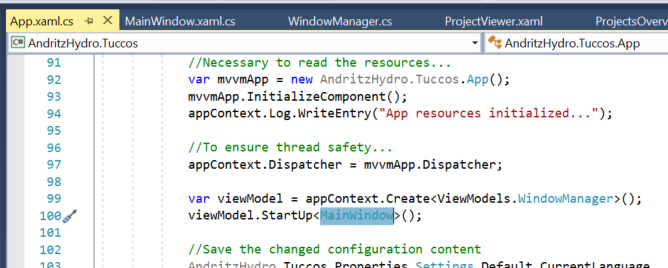
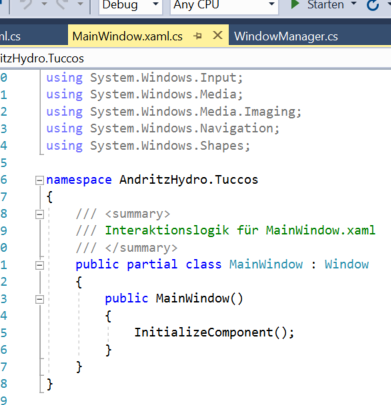
# Datenbindung und Datenfluss

Dazu kennt WPF eine mächtige Datenbindung. Die Eigenschaft, wo diese Datenbindung das ViewModel sucht, heißt…

 DataContext

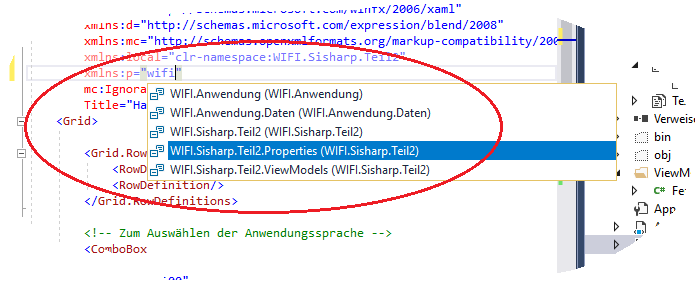


Datenbindung kann nur durch eine ÖFFENTLICHE EIGENSCHAFT erfolgen!

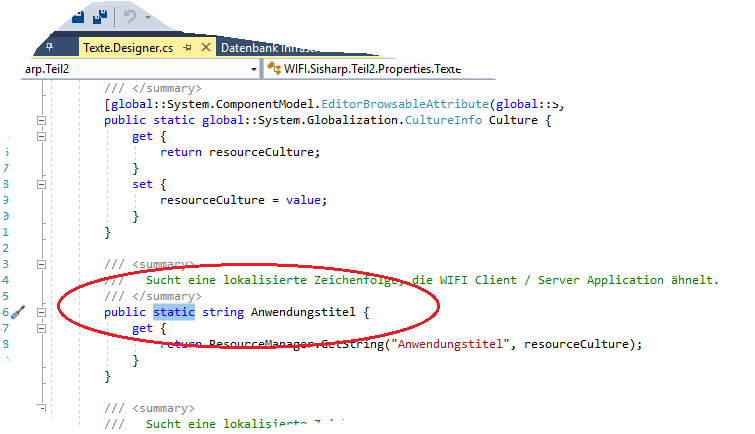
# Binden an eine statische Ressource

* Texte-Ressourcen liegen

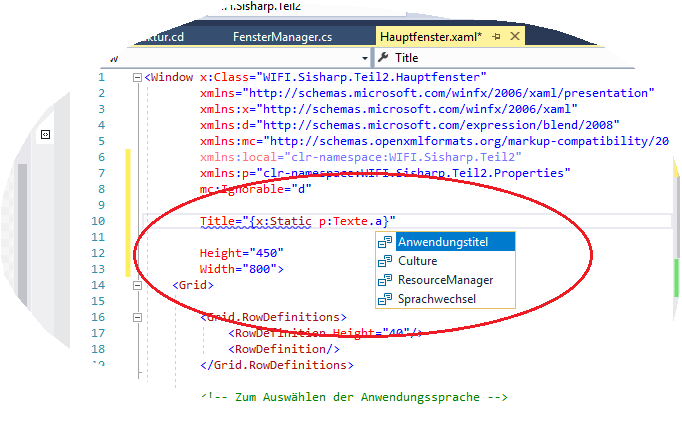


* Deshalb einen neuen Namespace mit Alias „p“ wie Properties auf WIFI.Sisharp.Teil2.Properties zeigen lassen

Zum Binden an Texte.Anwendungstitel …



… muss der Datenbindung mitgeteilt werden, dass es sich um eine statische Bindung handelt.



* Dazu aus dem x – Namespace „Static“ und aus dem p – Namespace Texte.Anwendungstitel benutzen

Wichtig:

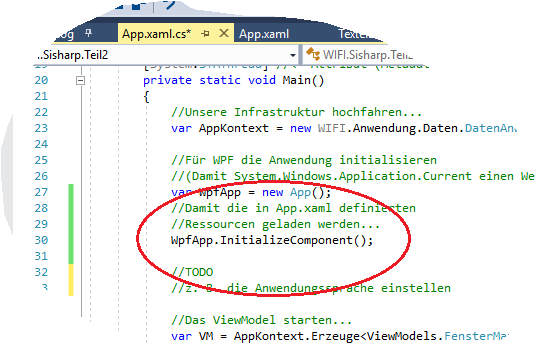
* Wird die statische Datenbindung innerhalb derselben Assembly benutzt, die Assembly vorher neu kompilieren.

# Wie und wo können Einstellungen/Ressourcen hinterlegt werden die man im WPF überall braucht

Dazu ist das App.xaml gedacht

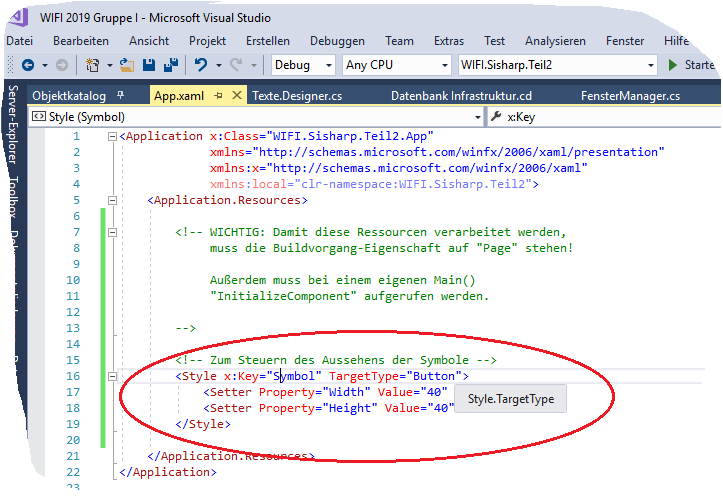
… mit den Formatvorlagen („Stile“) können WPF Vorlagen definiert werden,  
die überall benutzt werden können, wenn sie in App.xaml definiert sind

Damit diese Ressourcen geladen werden, …



… bei einem eigenen Main „InitializeComponent“ aufrufen!

In einem Stil …



… kann mit dem Setter eine Eigenschaft eingestellt werden.

Die Frage, …

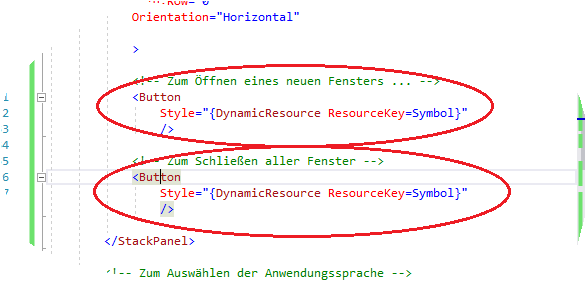
* Wie wird an so einen Stil gebunden?

Bisher:

* {x:Static …} an eine „normale“ Ressource
* {Binding …} an den DataContext

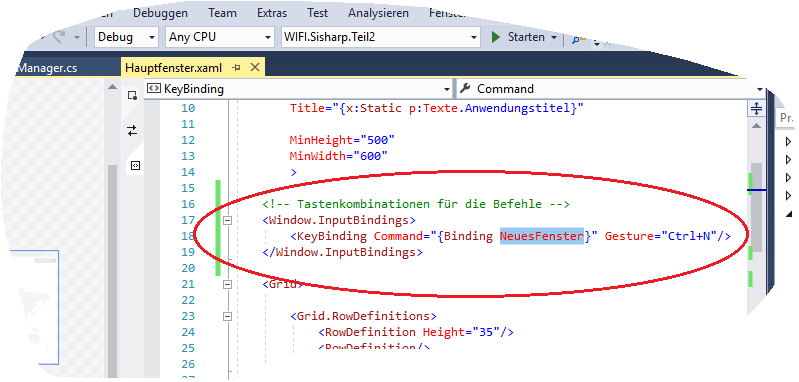
Jetzt das Dritte:

* An die Ressourcen in App.xaml wird „dynamisch“ gebunden



# Wie wird an eine Tastenkombination gebunden

Wie wird ein Befehl mit einer Tastenkombination verknüpft?



* Über die InputBindings Auflistung des Fenster.

# Was sind Normalformen beim Datenbankmodell, und was sind Surrogaten

Vorschriften damit man Datenbanktabellen sinnvoll erstenllen kann. Es gibt 15 solcher Vorschriften.

In der Realität zumindest die ersten vier Normalformen einhalten!!!! Siehe Mitschrift!

Ein Surrogat ist ein künstlicher Schlüssel der keinen wirklichen Realitätsbezug hat (wie z.B. Name, Adresse, etc), ein typischer Surrogat ist Kundennummer.

# Wie kann man eine Betriebssystem Funktion nutzen

Durch Interoperabilität, es handelt sich dabei nicht um .Net Komponenten

# Wann verwendt man using und wann Verweis

Using kann nur den Typen abkürzen, der Verweis muss immer gesetzt werden um etwas zu inkludieren

# Was versteht man unter einem Attribut

Grundsätzlich können Attribute Metadaten zu Code hinzufügen, in den Assembly.Properties sind die Attributes vorhanden

# Warum kann man ein Feld von einem Interface machen?

# Welche Dinge gehören nicht dem Objekt sondern der Klasse

Statische

# Was sind schreibgeschützte Eigenschaften

Eigenschaften die keinen setter haben

# Was sind Aggregatfunktionen im SQL