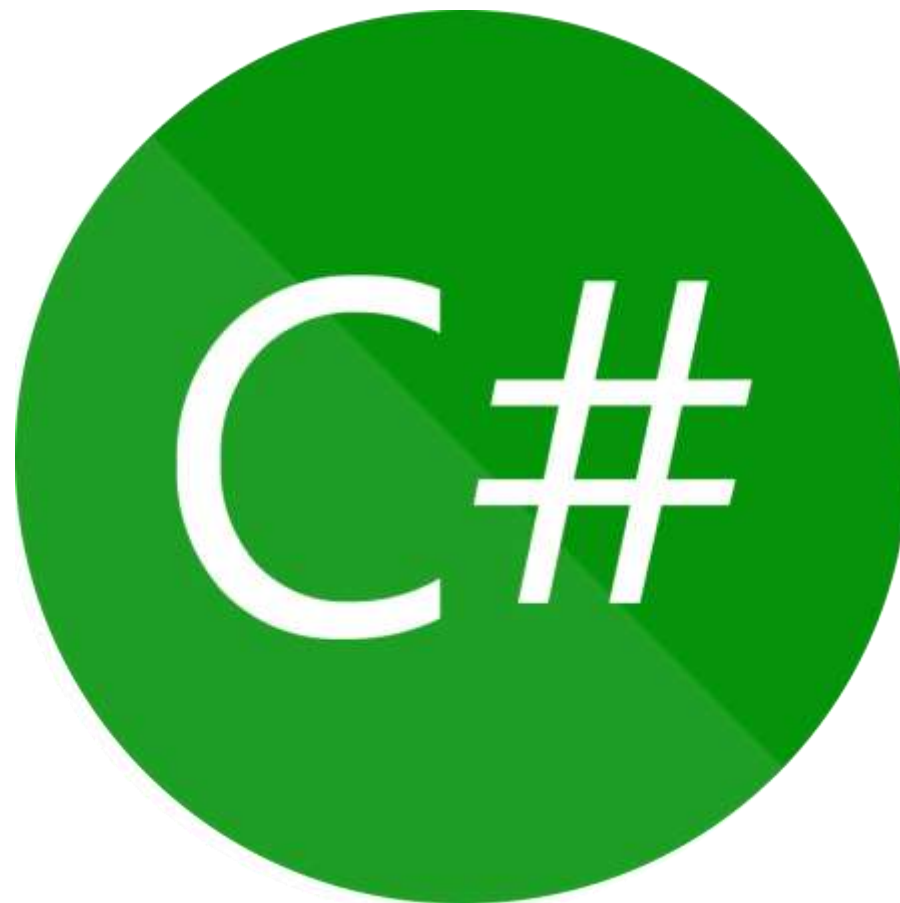
	C# > GETTING STARTED	C# > CLASS CONSOLE	C# > SCOPES & CALL STACK	C# > HANDLING TEXT	C# > HANDLING NUMBERS	C# > CONDITIONS
C# > LOOPS	C# & DEBUGGING	C# > ARRAYS	C# & LINQ METHODS	C# > FUNCTIONS	C# > ENUMERATORS	DRAFT C# > NAMESPACES
C# > CLASSES	C# > TYPES & VARIABLES & MEMORY	C# > FILE ACCESS	C# & SERIALIZING	C# > INHERITANCE	C# > EXCEPTIONS HANDLING	C# > CLASS OBJECT
C# > INTERFACES	C# > POLYMORPHISM	DRAFT C# > STRUCTS	DRAFT C# > ATTRIBUTES	C# > GENERIC	C# > GENERIC COLLECTIONS	C# & WINDOWS FORMS
C# > DELEGATES, ANONUMOUS FUNCTIONS	C# > HANDLING EVENTS	DRAFT C# > HANDLING TIME	C# & UNIT TESTS	C# > ASYNC, TASK, THREAD & CO	C# > HASHTABLES	C# > TUPELS
C# & LINQ EXPRESSIONS	C# > SOLID	C# > PREPROCESSOR DIRECTIVES	DRAFT C# > DEPLOYMENT	C# > ECOSYSTEM	DRAFT C# & APP RESOURCES	DRAFT C# > FACTS



C# > GETTING STARTED

C# > GETTING STARTED > IDEA OF #C# #CSHARP #CS

• WAS IST C#

- eine Programmiersprache
- objektorientierte Sprache
- Weiterentwicklung von C und C++
- aktuelle Version 8.0 (23. Sep. 2019) #checkForUpdates
- preview release 9.0 (20. Mai 2020)

WOZU IST C#

- Microsoft .NET (Windows Desktop)
- UWP – Windows 10 Apps
- .NET Core – Windows/Linux/Mac (aber dann ohne WinForms...)
- XAMARIN – iOS/Android
- ASP.NET – Webserver
- Spiele-Programmierung (Unity)

C# > GETTING STARTED > LINKS

- OFFICIAL HOMEPAGE

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

- OFFICIAL DOCS

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

- OFFICIAL BLOG

- <https://devblogs.microsoft.com/dotnet/>

- OFFICIAL CODE

- f12

C# > GETTING STARTED > PRIMARY TOOLS

- eine IDE, z.B. Visual Studio (oder Xamarin Studio oder Unity)
 - SDK (Software Developer Kit), z.B. .NET
 - ein Compiler, z.B. Roslyn (C#/VB Compiler)

C# > GETTING STARTED > SYNTAX Sprachgrundlagen

- Kommentare

```
int Nummer = 20; //Einzeiliger Kommentar  
/*Mehrzeiliger  
   Kommentar*/
```



ctrl k c - comment
ctrl k u - uncomment

C# > GETTING STARTED > SEMIKOLON

- Befehle / Expressions / Ausdrücke
- Jeden Befehl mit Semikolon abschließen!

```
Console.WriteLine(„Hello World“);  
Console.ReadKey();
```


C# > GETTING STARTED > ESSENTIALS > REGIONS

- weitere Möglichkeit, den Code zu kommentieren
- mit Regions kann man den Code in thematische / funktionale Abschnitte aufteilen
- Regions lassen sich auch verschachteln
- Regions kann man zusammenklappen
- Es gibt keine Tastenkombination in VS für Zusammen-/Aufklappen von Regions
 - Benutze die Erweiterung Collapse Region dafür



region tab tab

```
#region <NameDesCodeabschnitts>  
    // code  
#endregion
```

C# > GETTING STARTED > VERSIONS

Zielframework	Version	C# Standardversion
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET-Standard	2.1	C# 8.0
.NET-Standard	2.0	C# 7.3
.NET-Standard	1.x	C# 7.3
.NET Framework	alle	C# 7.3

C# > GETTING STARTED > VERSIONS

- Versucht man auf die Funktionalitäten einer C#-Version zuzugreifen, die höher ist, als die Standardversion für ein entsprechendes Framework, dann kommt ein Prompt, ob man die C#-Version auf die höhere Version umstellen möchte.
- Version, die höher als Standardversion ist, kann Kompilierzeit- und Runtimefehler hervorrufen
- In Visual Studio gibt es keine Benutzeroberfläche, um die C#-Version manuell anzupassen

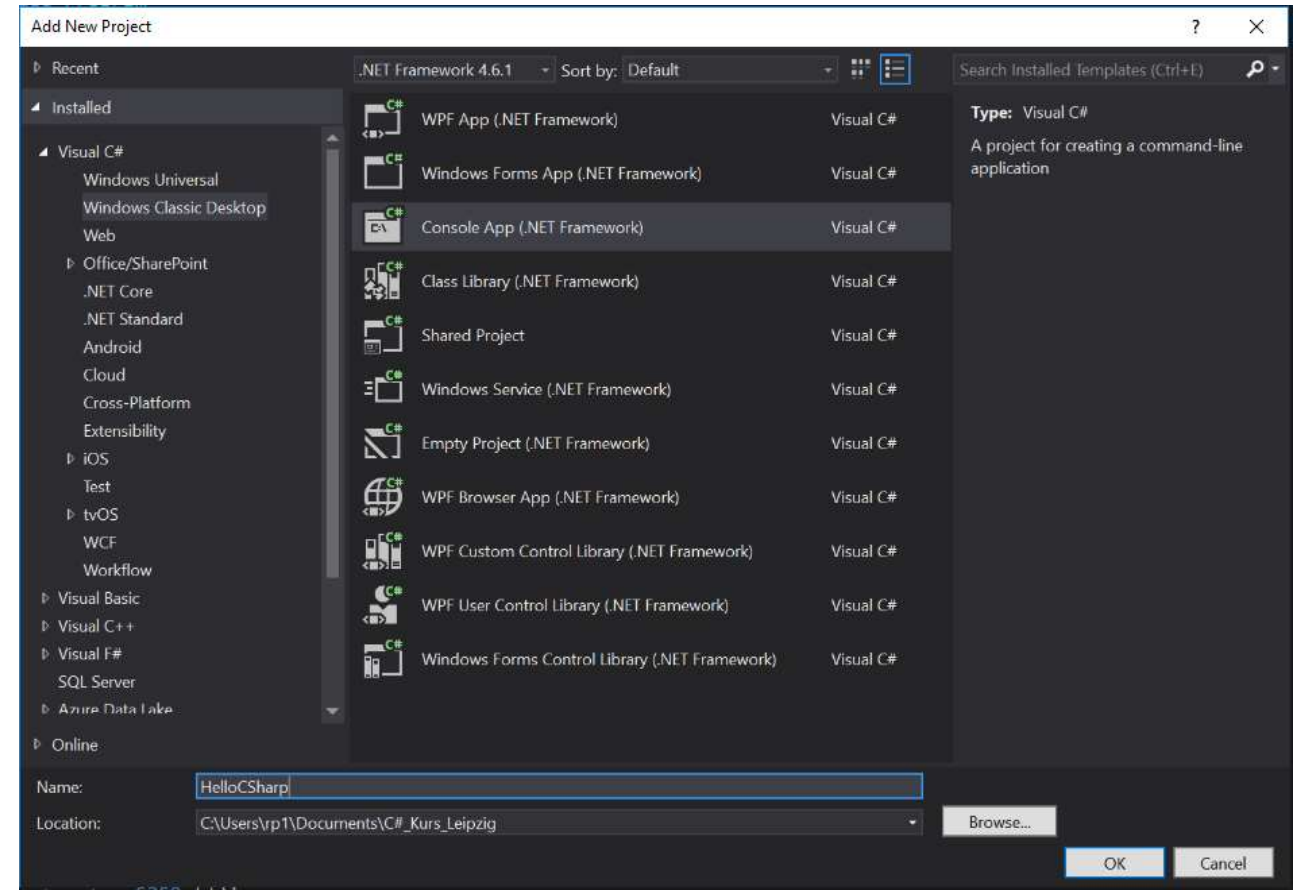
C# > GETTING STARTED > LAB

- allgemeine Syntax
 - Geschweifte Klammern
 - Runde Klammern
 - Eckige Klammern
 - Spitze Klammern
 - Semikolon / Strichpunkt
 - Komma
 - Punkt
 - Schlüsselwörter / Reservierte Wörter
 - Operatoren
 - Kommentare
 - Case-sensitive
 - Rautezeichen

C# > CLASS CONSOLE

C# > CLASS CONSOLE > LAB

- VS_N_VSCODE.pptx > VS GETTING STARTED
- VS_N_VSCODE.pptx > VS SOLUTIONS & PROJECTS
- Hello C# Projekt
- Solutions am besten pro Teilnehmer unterschiedlich benennen, um sie im gemeinsamen Repository zu unterscheiden



Übung

C# > CLASS CONSOLE >

- `int` alterVonMax = 20;
- `Console.WriteLine("Max ist {0} Jahre alt.",
alterVonMax); //Indexplatzhalter`



cw tab tab

C# > CLASS CONSOLE > Konsoleneingaben verarbeiten

- Einlesen von Strings

```
string stringEingabe = Console.ReadLine();  
int intEingabe = int.Parse(Console.ReadLine());  
double doubleEingabe = double.Parse(Console.ReadLine());
```

- Bemerkung: `int.Parse()` ist nur für strings gedacht
- Einlesen von einzelнем Zeichen (Character)

```
Console.WriteLine("Zum Beenden beliebige Taste drücken: ");  
Console.ReadKey();
```


C# > CLASS CONSOLE >

- Konsolenanwendung in Visual Studio
- Variablen deklarieren und Text ausgeben

C# >

SCOPES & CALL STACK

C# > SCOPES & CALL STACK >

- Assemblies? - ja
- Namensräume? - ja
- Klassen? - ja
- Methoden? - ja
- Geschweifte Klammern? - nein
- Schleifen

C# > HANDLING TEXT

C# > HANDLING TEXT

Schlüsselwort	native Typen / .NET Typen (auch für F#, VB genutzt)	Beschreibung
<code>char</code>	System.Char	Einzelnes Textzeichen
<code>string</code>	System.String	Text (max. 2GB)

C# > HANDLING TEXT > Ausgabe von Strings auf der Konsole

- verschiedene Schreibweisen:

```
int alterVonMax = 20;  
Console.WriteLine("Alter von Max: " + alterVonMax); // old school, vermeiden  
Console.WriteLine($"Max ist {alterVonMax} Jahre alt."); // #v.6.0
```

C# > HANDLING TEXT > LAB

- LAB > DEMO
 - Dstring
 - Demo-char

C# > HANDLING NUMBERS

C# > HANDLING NUMBERS

Schlüsselwort	native Typen / .NET Typen (auch für F#, VB genutzt)	Beschreibung
<code>short</code>	System.Short	16-Bit-Integer (0000000000000000)
<code>int</code>	System.Int32	32-Bit-Integer
<code>long</code>	System.Int64	64-Bit-Integer
<code>float</code>	System.Single	16-Bit-Gleitkommazahl
<code>double</code>	System.Double	32-Bit-Gleitkommazahl
<code>decimal</code>	System.Decimal	128-Bit-Gleitkommazahl

Standardgemäß:

- Ohne Nachkommastellen -> `int`
- Mit Nachkommastellen -> `double`

C# > HANDLING NUMBERS > DIGIT SEPARATORS #v7.0

```
public void Literale()  
{  
    int hex = 0x00FF00;  
    byte bin = 0b00001111;  
    byte bin2 = 0b_0000_1111;  
    int hundertdreieundzwanzig = 1____2____3;  
}
```

C# > HANDLING NUMBERS > ANWEISUNGSSCHLÜSSELWÖRTER

- Checked{} & unchecked{}
- bei Überläufen (z.B. int-Variable bekommt höheren Wert, als ihr Maximum)
- addiert man die Zahlen und übergibt der int-Variable - Überlauffehler
- wird mindestens eine von diesen Zahlen in der Summe nicht direkt als Wert, sondern über eine Variable übergeben, dann gibt es keine Überlauffehler (man landet im Minusbereich)
- will man auch beim Addieren von Variablen die Überlauffehler sehen, verwendet man das Schlüsselwort 'checked'

C# > HANDLING NUMBERS > (Casting)

- Konvertierung von Datentypen

- Zahl => String

- Implizit:

- ```
string satz = "Wert der Zahl: " + zahl;
```

- Explizit:

- ```
string satz = zahl.ToString();
```

- String => Zahl

- Nur per Hilfsfunktion möglich:

- ```
int zahl = int.Parse(Console.ReadLine());
```

- ```
double zahl2 = double.Parse(Console.ReadLine());
```

- Zahl => Zahl

- ```
int ganzzahl = 25;
```

- ```
double kommazahl = ganzzahl;
```

- ```
ganzzahl = (int)kommazahl;
```

## C# > HANDLING NUMBERS > CASTING

- Ungenaueres zum Genaueren - implizit
- Gleitkommazahl-Typen zu Ganzzahl-Typen - Nachkommastellen werden abgeschnitten (2,5665 -> 2)
- Genaueres zum Ungenaueren - muss explizit gecastet werden  
`varVomNeuenDTyp = (NeuerDTyp) varVomAltenDTyp;`

## C# > HANDLING NUMBERS >

- Mathematische Operatoren und Funktionen

- $a + b$  Plus
- $a - b$  Minus
- $a * b$  Multiplikation
- $a / b$  Division
- $a \% b$  Modulo (Rest der Division)
- $a++$   $a = a + 1$
- $a--$   $a = a - 1$
- $a += 2$   $a = a + 2$
- $b /= 2$   $b = b / 2$
-

## C# > HANDLING NUMBERS > CLASS MATH

- `Math.Max(a,b)`     Ermittle Maximum von beiden Zahlen
- `Math.Min(a,b)`     Ermittle Minimum von beiden Zahlen
- `Math.Round(2.5)`   Gerundete Zahl, bei .5 runde auf nächste ganze Zahl
- `Math.Floor()`       Runde Zahl ab (2,8 -> 2)
- `Math.Ceiling()`      Runde Zahl auf (2,3 -> 3)

# C# > HANDLING NUMBERS > LAB

- LAB > DEMO
  - Dint
  - Ddouble
- LAB > QUESTIONS (lieber am nächsten Tag)
  - s. nächste Folie
- LAB > EXERCISE
  - Mini-Rechner



## C# > HANDLING NUMBERS > LAB > QUESTIONS

1. Eine Variable wurde mit `int x = 2;` initialisiert, welchen Wert hat sie nach folgenden Operationen:  
`x += 2;`  
`x++;`  
`x = x % 2;`
2. Welcher Datentyp hat das Ergebnis der Division eines Integer-Werts mit einem Double-Wert?
3. Mit welcher Funktion kann man einen String in einen Double konvertieren?

# C# > CONDITIONS

# C# > CONDITIONS >

| Schlüsselwort     | native Typen / .NET Typen (auch für F#, VB genutzt) | Beschreibung    |
|-------------------|-----------------------------------------------------|-----------------|
| <code>bool</code> | System.Boolean                                      | true oder false |

## C# > CONDITIONS > Logische Operatoren

- Jede Operation ergibt entweder true oder false
- $a == b$                       a gleich b
- $a != b$                         a ungleich b
- $a < b$                          a kleiner b
- $a > b$                          a größer b
- $a <= b$                       a kleiner oder gleich b
- $a \&\& b$                       a und b sind wahr
- $a || b$                         a oder b ist wahr
- $a \wedge b$                      entweder a oder b
- $!a$                             true wird zu false, false zu true

# C# > CONDITIONS > Bedingungen

```
if (Bedingung A)
{
 //Anweisungen wenn Bedingung A wahr
}
else if (Bedingung B)
{
 //Anweisungen wenn Bedingung A falsch und B wahr
}
else
{
 //Anweisungen wenn Bedingung A und B falsch sind
}
```



if tab tab  
else tab tab

# C# > CONDITIONS > Switch

```
switch (tag)
{
 case Wochentag.Mo:
 Anweisung A;
 break;
 case Wochentag.Di:
 case Wochentag.Mi:
 Anweisung B
 Anweisung C;
 break;
 default:
 Anweisung D;
 break;
}
```



switch tab tab

# C# > CONDITIONS > LAB

- LAB > DEMO
  - Dbool

# C# > LOOPS



# C# > LOOPS > Schleifen

## Kopfgesteuert

**while** (Bedingung)

{

//wiederholt solange bis Bedingung falsch ist

}

## Fußgesteuert

**do**

{

//führt mindestens 1 Mal aus

} **while** (Bedingung);



while tab tab  
do tab tab

## C# > LOOPS > BREAK & CONTINUE

- Schlüsselwörter innerhalb von Schleifen:
- `break;` //verlasse die Schleife
- `continue;` //springe zum nächsten Schleifendurchlauf (Schleifen-Bedingung wird erneut geprüft)

# C# > LOOPS > Zähl-Schleifen

`for(Initialisierung;Bedingung;Anweisung)`

Hochzählen(Inkrementierung):

```
for (int i = 0; i<10; i++)
{
 //Zähle von 0 bis 9
}
```



for tab tab

Runterzählen(Dekrementierung):

```
for (int i = 10; i>0; i--)
```



forr tab tab

# C# > LOOPS > LAB > Zahlenratespiel

- Zufallszahl ermitteln

```
//Initialisiere Zufallsgenerator
Random generator = new Random();
//Erzeuge Zahl zwischen 1 und 4
int zufallszahl = generator.Next(1, 5);
```

Übung

# C# > LOOPS > LAB > Zahlenratespiel

- Teil 1
  - Erzeuge eine Zufallszahl zwischen 1 und 5
  - Prüfe Nutzereingabe und gebe aus ob seine Zahl größer, gleich oder kleiner der Zufallszahl ist
- Teil 2
  - Der Nutzer soll solange neue Zahlen eingeben können bis er die richtige Zahl erraten hat
  - Am Ende des Spiels die Anzahl der Versuche ausgeben

Übung

# C# & DEBUGGING

# C# > DEBUGGING

- siehe VS > DEBUGGING

# C# > ARRAYS



## C# > ARRAYS >

### Aufzählung von gleichartigen Datentypen

```
int[] zahlen = new int[] { 2, 4, 5 };
string[] buchstaben = new string[] { "A", "B", "C" };
double[] kommazahlen = new double[10];
```

### Zugriff über Index

```
zahlen[2]; //5
```

```
buchstaben[1]; //"B"
```

```
zahlen[2] = 4; //setzt 3. Element im Array auf Wert 4
```

## C# > ARRAYS > Array durchlaufen

```
for(int i=0;i<zahlen.Length;i++)
{
 Console.WriteLine(zahlen[i]);
}
```

Oder

```
foreach(var item in zahlen)
{
 Console.WriteLine(item);
}
```



foreach tab tab

Achtung: foreach kann nur Werte auslesen, nicht verändern

# C# > ARRAYS > Eigenschaften von Arrays

```
int[] numbers = new int[] { 2, 9, 5 };
```

| Funktionsname | Beschreibung                                 | Beispiel            |
|---------------|----------------------------------------------|---------------------|
| <b>Length</b> | Gibt die Anzahl der Elemente im Array zurück | numbers.Length; //3 |

## C# > ARRAYS >

- Mehrdimensionale Arrays
- Mehrdimensionale Arrays:
- `int[,] diagramm = new int[2,5];`
- `diagramm[1,4] = 5;`

# C# > ARRAYS > LAB > QUESTIONS

1. Bool a = true und bool b = false, welchen Wahrheitswert ergeben folgende Ausdrücke:
  1. a == b
  2. a || b
  3. !(a && b)
  4. !b
  5. a ^ b
2. Welche der folgenden Array-Initialisierungen ist syntaktisch falsch?
  1. int[] a = new int[20];
  2. int[] b = new int[] {2, 5, 8}
  3. int[] c = new int[];
3. Wie oft wird eine do-while-Schleife mindestens durchlaufen?
4. Was ist der Unterschied zwischen break und continue in einer Schleife?

# C# & LINQ METHODS

# C# & LINQ METHODS > #INTRO

- WAS IST LINQ
  - Language Integrated Query / Sprachintegrierte Abfrage
  - Eine Bibliothek aus der .NET-Welt
- WOZU IST LINQ
- OFFICIAL DOCS
  - <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/concepts/linq/>
  - <https://docs.microsoft.com/de-de/dotnet/csharp/linq/>

## C# & LINQ METHODS > *Exkurs:* LINQ

- Language Integrated Query
- Abstrahierte Abfragesprache auf Objektmengen
- Direkter Bestandteil von C#
- 101 LINQ Beispiele
  - <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>



# C# & LINQ METHODS > Web API und HTTP

## • Exkurs LINQ

- Language Integrated Query
- Abfragesprachen auf Objektmengen
- Direkter Bestandteil von C# und VB.NET
- Bietet die Möglichkeit, Abfragen abstrakt zu beschreiben, die später durch entsprechende Provider in konkrete Abfragen für die entsprechenden Zielsysteme überführt werden.
- 101 LINQ Beispiele:
  - <https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b>

# C# & LINQ METHODS >

Nützliche Funktionen und Eigenschaften von Arrays

```
int[] numbers = new int[] { 2, 9, 5 };
```

Methoden kommen aus Linq!

| Funktionsname                    | Beschreibung                              | Beispiel                                  |
|----------------------------------|-------------------------------------------|-------------------------------------------|
| <code>Contains(int value)</code> | Prüft ob Array ein Element enthält        | <code>numbers.Contains(5); // true</code> |
| <code>Max()</code>               | Gibt den größten Wert im Array zurück     | <code>numbers.Max(); //9</code>           |
| <code>Min()</code>               | Gibt den niedrigsten Wert im Array zurück | <code>numbers.Min(); //2</code>           |
| <code>First()</code>             | Gibt das erste Element im Array zurück    | <code>numbers.First(); //2</code>         |
| <code>Last()</code>              | Gibt das letzte Element im Array zurück   | <code>Numbers.Last(); //5</code>          |
| <code>Sum()</code>               | Gibt die Summe aller Elemente zurück      | <code>Numbers.Sum(); //16</code>          |

# C# > FUNCTIONS

# C# > METHODS | FUNCTIONS > DECLARE & CALL

- KONVENTIONEN
- Funktionen und Methoden großgeschrieben
- Parameter für die Methoden - kleingeschrieben

```
public static int Addiere(int a, int b)
{
 int c = a + b;
 return c;
}
```

Modifier Rückgabewert Name(Parameterliste)

## Aufruf

```
int summe = Addiere(2, 4);
```

## C# > FUNCTIONS > Lokale Funktionen #v7.0

- Hilfsmethoden, die nur für eine Methode bestimmt sind, werden in der betroffenen Methode „eingeschlossen“

```
public void WertVerdoppeln()
{
 int wert = 123;
 void LokaleFunktion()
 {
 wert *= 2;
 }
 LokaleFunktion();
 Console.WriteLine(wert);
}

public int Fibonacci(int x)
{
 return Fib(x - 1).AktuellerWert;

 (int AktuellerWert, int VorherigerWert) Fib(int i)
 {
 if (i == 0) return (1, 0);
 var (p, pp) = Fib(i - 1);
 return (pp + p, p);
 }
}
```

## C# > FUNCTIONS > Spezielle Parametertypen: out/ref

- Normalerweise werden Werttypen wie int, bool, string oder double in Funktionen als Wert übergeben
- Mittels out/ref können sie als Referenz übergeben werden

```
public void changeAlter(out int wert)
{
 wert = 6;
}
```

```
//Aufruf
int alter;
changeAlter(out alter); //alter ist jetzt 6
```

- Out behandelt die Variable so als hätte sie noch keinen Wert (nicht initialisiert)
- Mit out kann man eine Variable direkt beim Aufruf anlegen
- Bei Ref muss eine bereits initialisierte Variable übergeben werden

# C# > FUNCTIONS > out Variablen #v7.0

```
private void IntCheck()
{
 string eingabe = "12345";
 int ausgabe; // <- deklaration erforderlich !
 if(Int32.TryParse(eingabe,out ausgabe))
 {
 Console.WriteLine(ausgabe);
 }
}
```

```
private void IntCheck()
{
 string eingabe = "12345";
 if(Int32.TryParse(eingabe,out var ausgabe))
 {
 Console.WriteLine(ausgabe);
 }
}
```

## C# > FUNCTIONS > Spezielle Parametertypen: params

- ermöglicht variable Anzahl von Parametern von einem Typ

```
public static int BildeSumme(params int[] summanden)
{
 int summe = 0;
 foreach (var item in summanden)
 {
 summe += item;
 }
 return summe;
}
```

```
BildeSumme(2, 3, 4); //ergibt 9
BildeSumme(); //ergibt 0
BildeSumme(new int[]{3, 3, 4}); //ergibt 10
```



## C# > FUNCTIONS > Spezielle Parametertypen: optional

- Parameter können schon in der Deklaration mit Defaultwerten initialisiert werden, beim Aufruf ist der Parameter dann optional

```
public string StringCombine(string[] strings, char trenner = '/')
{
 string resultString = strings[0];
 for(int i = 1; i < strings.Length; i++)
 {
 resultString += (trenner + strings[i]);
 }
 return resultString;
}

StringCombine(new string[] { "A", "B", "C" }); // "A/B/C"
StringCombine(new string[] { "A", "B", "C" }, '-'); // "A-B-C"
```

- Die optionalen Parameter müssen in der Funktionsdeklaration immer nach den Pflicht-Parametern folgen

## C# > FUNCTIONS > Lambda-Ausdrücke (fortgeschritten)

- Kurzschreibweise für anonyme Methoden

```
EventHandler onClick = delegate(Object sender, EventArgs e) { /*Code*/ };
```



```
EventHandler onClick = (Object x, EventArgs y) => { /*Code*/ };
```



```
EventHandler onClick = (x, y) => { /*Code*/ };
```

## C# > FUNCTIONS > readonly ref #v7.2

- ref: Variable wird als Referenz übergeben
- out: Variable wird als Referenz übergeben und muss einen neuen Wert zugewiesen bekommen
- in: Variable wird als Referenz übergeben, ist aber readonly !

```
public static void InParameter(in int x)
{
 int y = x++;
}
```

[?] (parameter) in int x

Cannot assign to variable 'in int' because it is a readonly variable

## C# > FUNCTIONS > LAB > Fragen

1. Was stimmt nicht an folgenden Funktionsdeklaration?

- `void funktion1(int x, string y = "abc", int z)`
- `void funktion2(params int x)`
- `void funktion3(int x, out y, int x, out y)`

2. Warum sollte man statt fester String-Werte lieber Enumeratoren verwenden, um Zustände des Programms zu verwalten?

# C# > ENUMERATORS

## C# > ENUMERATORS >

- dt.: Aufzählungstyp
- Eigenen Datentyp definieren, der nur ganz bestimmte Werte zulässt
- Achtung: Die Enums lassen sich nicht innerhalb von Methoden deklarieren.
- Elemente in Enums heißen Konstanten

```
enum Wochentag { Mo=1, Di=2, Mi, Do, Fr, Sa, So};
```

- Vorteil: bessere Lesbarkeit des Codes

```
Wochentag tag = Wochentag.Do;
if (tag == Wochentag.Mi) {
 //...
}
```



enum tab tab

# C# > ENUMERATORS > LAB

- Denum

# C# > CLASSES



# C# > CLASSES > IDEA

- WAS SIND KLASSEN

- Bausteine bei objektorientierten Sprachen
- Datenstrukturen - siehe Klassenmitglieder
- Datentypen - z.B. bei Instanzen von einer Klasse

- WOZU SIND KLASSEN

- Code wiederverwendbar - Klassen als Werkzeug beim Programmieren
- Bessere Strukturierung und Lesbarkeit
- Sicherheit (public/private)
- beschreiben Struktur von Objekten - Klassen als Datenbankmodelle

# C# > CLASSES > Objektorientierte Programmierung

- Aus einer Klasse beliebig viele Objekte erzeugbar:

- `Random random1 = new Random();`
- `Random random2 = new Random();`

- Klassen bestehen aus

- Feldern (Membervariablen)
- Eigenschaften (Properties)
- Funktionen (Methoden)
- Konstruktor/Destruktor

# C# > CLASSES > Felder und Methoden

```
public class Person
{
 //Felder
 private string vorname;
 //Methoden
 public void SetVorname(string vorname)
 {
 this.vorname = vorname;
 }
 public string GetVorname()
 {
 return this.vorname;
 }
}
```



class tab tab

## C# > CLASSES > Konstruktoren/Destruktoren

- Legt den Startzustand des Objekts nach Initialisierung fest
- Mehrere Konstruktoren pro Klasse möglich (verschiedene Parameter)

```
public Person(string vorname, string nachname)
{
 Vorname = vorname;
 Nachname = nachname;
}
```

- Destruktor wird bei Zerstörung des Objektes aufgerufen

```
~Person()
{
 Console.WriteLine("Destruktor");
}
```



ctor tab tab  
~ tab tab

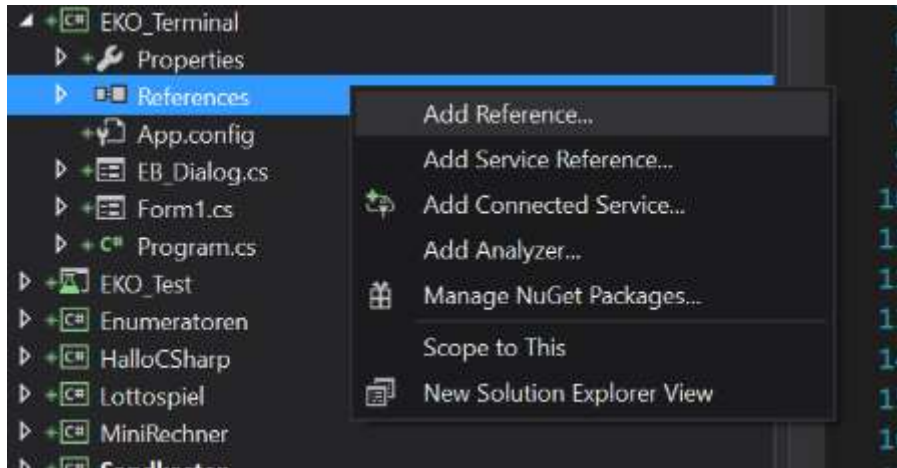
## C# > CLASSES > Statische Member (Methoden/Eigenschaften)

- Als statisch markierte Methoden oder Eigenschaften gelten für die Klasse selbst und nicht für Objekte der Klasse
- Der Zugriff erfolgt über den Klassennamen
- Jede Eigenschaft existiert nur 1 Mal für die jeweilige Klasse
- Innerhalb von statischen Methoden darf nicht auf nicht-statische Methoden/Eigenschaften der Klasse zugegriffen werden
- WriteLine() und ReadLine() sind statische Methoden der Klasse Console, um auf sie zuzugreifen muss zuvor keine Instanz der Klasse Console erzeugt werden

## C# > CLASSES > Wo die Klassen definieren?

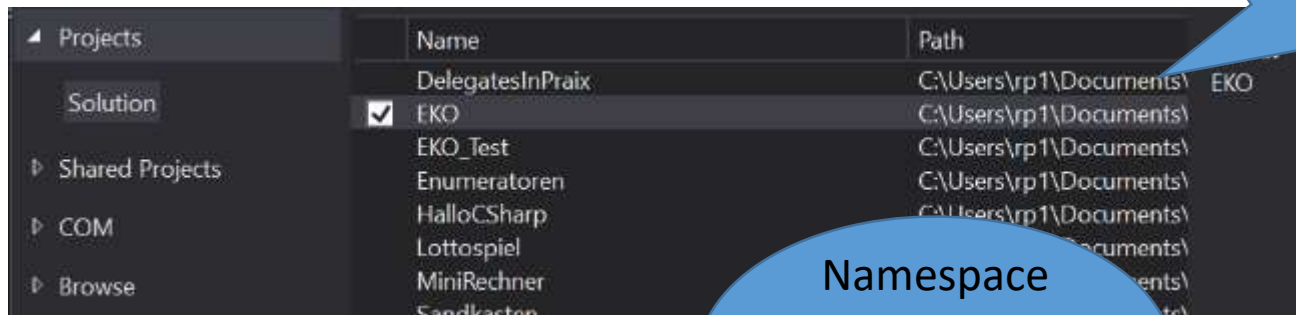
- in der gleichen Datei nacheinander
  - in kleineren Projekten oder beim Lernen
- in einzelnen Dateien
  - empfehlenswert in Production-Build
- innerhalb einer anderen Klasse
  - nested classes
- über mehrere Dateien/Orte aufgeteilt
  - partial classes

# C# > CLASSES > Klassen aus anderen Projekt einbinden



Referenz zum  
Projekt  
hinzufügen

Einzubindende  
Projekte  
auswählen



Namespace  
per using in  
jedem Codefile  
einbinden

```
using EKO;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

## C# > CLASSES > LAB

- QUESTIONS
- DEMO
  - Library anlegen
  - Konsumierende App anlegen
  - in der App Rechtsklick > Add > Reference



# C# > CLASSES > Properties

- Kurzschreibweise um Zugriff auf private Membervariablen über Methoden zu definieren
- KONVENTIONEN
- Eigenschaften / Properties von Klassen - großgeschrieben

```
public class Person
{
 //Properties
 public string Vorname { get; private set; }
 public string Nachname { get; private set; }

 //Konstruktor
 public Person(string vorname, string nachname)
 {
 Vorname = vorname;
 Nachname = nachname;
 }
}
```



prop tab tab  
ctor tab tab

## C# > CLASSES > Varianten von Properties

//öffentliches Lesen und Schreiben erlauben

```
public string Vorname { get; set; }
```

//Eigenschaft kann nur noch intern gesetzt werden

```
public string Nachname { get; private set; }
```

//Eigenschaft kann von Außen nur überschrieben werden

```
public string Geheim { private get; set; }
```

## C# > CLASSES > Zugriff auf Property genau steuern

```
private int alter; //Membervariable
public int Alter //dazugehörige Property
{
 get
 {
 return alter;
 }
 set
 {
 if(value > 0)
 {
 alter = value;
 }
 }
}
```



propfull tab tab

# C# > CLASSES > Expression Bodied Members #v7.0

```
class Person
{
 public Person(string Name) => this.Name = Name;
 ~Person() => Name = null;

 private string name;
 public string Name
 {
 get => name;
 set => name = value;
 }
}
```

## C# > CLASSES > Kopierkonstruktor

Kopie eines Objektes erstellen (Opt 1):

Objekte werden standardmäßig immer per Referenz zugewiesen

```
Person person1 = new Person("Alex");
```

```
Person person2 = person1; //beide zeigen auf das gleiche Objekt im Speicher
```

Kopierkonstruktor definieren:

```
public Person(Person person)
{
 this.Name = person.Name;
}
```

```
Person person2 = new Person(person1);
```

## C# > CLASSES > Modifier

- Können für Klassen, Felder, Eigenschaften und Methoden verwendet werden

| Modifier                        | Zugriff                                                                 |
|---------------------------------|-------------------------------------------------------------------------|
| <code>public</code>             | Von Außerhalb der Klasse                                                |
| <code>private</code>            | Nur innerhalb der Klasse                                                |
| <code>internal</code>           | Zugriff nur innerhalb der aktuellen Assembly/des selben Namespaces      |
| <code>readonly</code>           |                                                                         |
| <code>protected internal</code> | kann in anderen Assemblies nur in abgeleiteten Klassen verwendet werden |

## C# > CLASSES > Fragen

1. Was ist der Unterschied zwischen private und protected?
2. Welche Zugriffsmöglichkeiten innerhalb und außerhalb der Klasse bieten die folgenden Eigenschaften:

1. `public int Property1 { get; set; }`
2. `public int Property2 { get; private set; }`
3. `private int Property3 { get; set; }`
4. `public int Property4 { get; }`

3. Woran erkennt man eine Konstruktor-Deklaration?

# C# >

# TYPES & VARIABLES & MEMORY



# C# > DATA TYPES > IDEA

- WAS SIND DATENTYPEN
  - Attribute für Daten (Metainformationen über Daten)
- WOZU SIND DATENTYPEN
  - Infos für Compiler, wie viel Speicher für die Variablen reserviert werden soll
  - Infos darüber, was hat der Programmierer mit diesen Daten vor
  - in C# kann man über die Datentypen-Schlüsselwörter auch die entsprechende Klasse ansprechen
- BEGRIFFE
  - built-in types / eingebaute Datentypen
  - primitives / primitive Datentypen
  - custom types / benutzerdefinierte (genauer: programmiererdefinierte) Datentypen

## C# > DATA TYPES > VARIABLES

- Variablen sind Behälter/Container für Werte
- Syntax beim Deklarieren und evtl Initialisieren von Variablen:
  - <type> <identifier>;
  - <type> <identifier> = <literal>;
  - <type> <identifier> = <expression>;
- Syntax beim Aufrufen von Variablen:
  - <identifier>

```
int Alter; //Deklarieren / Deklaration
Alter = 20; //Initialisierung: Integer mit Wert 20
int Summe = Alter*2; //Summe bekommt Wert 40
string Stadt = "Berlin"; //Zeichenkette "Berlin"
Console.WriteLine(Stadt); //Berlin in Konsole schreiben
```

## C# > DATA TYPES >

- KONVENTIONEN
- Felder / Variablen in Klassen - kleingeschrieben
- Camel Case bei zusammengesetzten Variablen / Identifiers
- <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/inside-a-program/coding-conventions>
  - Naming Conventions

## C# > DATA TYPES > ZUWEISUNG

- Zuweisungen / Statements

```
int Nummer = 20;
```

```
string Wort = „Wort“;
```

- Zuweisungsoperator „=“ entspricht nicht dem mathematischen =

## C# > DATA TYPES > VALUE & REFERENCE TYPES

Value Types verwalten direkt ihre Werte

Reference Types verwalten Referenzen auf ihre Werte. Das heißt, mehrere Variablen können auf den gleichen Wert verweisen und diesen beeinflussen.

Eingebaute Reference Types sind string und object

- bei Zahlentypen ist es 0
- bei Referenz-Typen ist es null

## C# > DATA TYPES > VAR

- VAR
- steht für variant
- wenn Datentyp unbekannt

```
foreach (var item in myArray) { ... }
```

## C# > DATA TYPES > TYPE CHECK

- Operator is
  - Syntax: `<expression> is <type>`
  - gibt bool zurück
- Methode `myVar.GetType()`

```
alter + stadt is string
```

# C# > FILE ACCESS



## C# > FILE ACCESS > Dateien schreiben/lesen

```
using System.IO;
StreamWriter sw = new StreamWriter("Testdatei.txt");
sw.WriteLine("Testzeile1");
sw.WriteLine("Testzeile2");
sw.Close();

StreamReader sr = new StreamReader("Testdatei.txt");
while (!sr.EndOfStream)
{
 string zeile = sr.ReadLine();
}
sr.Close();
```

## C# > FILE ACCESS > Dateidialog anzeigen

- Dialog zum Datei öffnen bzw. speichern

```
using System.Windows.Forms;
```

```
...
```

```
SaveFileDialog saveDialog = new SaveFileDialog();
saveDialog.FileName = "Personen.pl";
saveDialog.Filter = "Personenliste|*.pl|Textdokument|*.txt";
if(saveDialog.ShowDialog() == DialogResult.OK){...}
```

- FileName: ausgewählte (bzw. vorbelegter) Dateiname
- Filter: welche Dateitypen sollen auswählbar sein
- zum Datei Öffnen analog die Klasse OpenFileDialog verwenden

## C# > FILE ACCESS > Fragen

- Warum muss man beim Deserialisieren das Format, in das deserialisiert werden soll, mit angeben aber beim Serialisieren nicht?
- Wie kann man beim OpenFileDialog die Auswahl auf Text-Dateien einschränken?

# C# & SERIALIZING

## C# & SERIALIZING > Objekte als JSON serialisieren

- Serialisierung: Beliebiges Objekt in einen String konvertieren
- Vorteil: Der Zustand eines Objektes kann als String in eine Datei geschrieben und später wieder ausgelesen und zurückkonvertiert (deserialisiert) werden

```
using Newtonsoft.Json; //vorher per NuGet installieren
var daten = new List<string> { „Wert1“, „Wert2“, „Wert3“ };
//.NET-Objekt in JSON-String konvertieren
string result = JsonConvert.SerializeObject(daten);
sw.Write(result);
//JSON-String in .NET-Objekt umwandeln
objekt = JsonConvert.DeserializeObject<List<string>>(sr.ReadToEnd());
```

# C# > INHERITANCE

## C# > INHERITANCE > Vererbung von Klassen

- Alle Eigenschaften und Methoden von Basisklasse übernehmen

```
class CustomRandom : Random
{
 public int NextInclusive(int min, int max)
 {
 return base.Next(min, max+1);
 }
}
```

- **protected**
- **Innerhalb der Klasse und in allen abgeleiteten Klassen**

# C# > INHERITANCE > Konstruktoren in vererbten Klassen

```
public class Lebewesen
```

```
{
 public int Alter { get; set; }
 public Lebewesen(int alter)
 {
 this.Alter = alter;
 }
}
```

```
public class Mensch : Lebewesen
```

```
{
 public string Wohnort { get; set; }
 public Mensch(int alter, string wohnort) : base(alter)
 {
 this.Wohnort = wohnort;
 }
}
```

Ruft zuerst Basiskonstruktor auf und  
danach Konstruktor der Kindklasse



# C# > INHERITANCE > Virtuelle Methoden

```
public class Fahrzeug
{
 public virtual string BeschreibeMich() {
 return "Ich bin ein Fahrzeug";
 }
}
public class Auto : Fahrzeug
{
 public override string BeschreibeMich()
 {
 return base.BeschreibeMich() + " und habe 4 Räder";
 }
}
```

# C# > INHERITANCE > Abstrakte Methoden und Klassen

```
abstract class Fahrzeug
{
 public abstract bool IstFahrbereit();
}

public class Fahrrad : Fahrzeug
{
 public int Luftdruck { get; set; }
 public override bool IstFahrbereit()
 {
 return (Luftdruck > 0);
 }
}
```

# C# > INHERITANCE > Pattern Matching mit Switch #v7.0

```
private void MusterabgleichMitSwitch(Grafik g)
{
 switch (g)
 {
 case Kreis k:
 Console.WriteLine($"Kreis mit dem Radius {k.Radius}");
 break;
 case Rechteck q when (q.Länge == q.Breite):
 Console.WriteLine($"Quadrat mit der Seitenlänge {q.Länge}");
 break;
 case Rechteck r:
 Console.WriteLine($"Rechteck mit der Länge {r.Länge} und Breite {r.Breite}");
 break;
 default:
 Console.WriteLine("Unbekannte Grafik");
 break;
 }
}
```

# C# > INHERITENCE > LAB

- Fragen

1. Von wie vielen Klassen kann eine Klasse maximal erben?
2. Was ist der Unterschied zwischen abstrakten und virtuellen Methoden?

- DEMO

- DClass zu DLibrary umbenennen
- DUseClass zu DUseLibrary umbenennen

# C# > EXCEPTIONS HANDLING

# C# > EXCEPTIONS > Exception-Handling

- Mittels try/catch Fehlerfälle in Anwendung abfangen
- finally-Block ist optional

```
try
{
 int Eingabe = int.Parse(Console.ReadLine());
}
catch (FormatException e)
{
 Console.WriteLine("Bitte eine Zahl eingeben.");
 return;
}
catch (Exception e)
{
 Console.WriteLine(e.Message);
 return;
}
finally
{
 Console.WriteLine("Danke");
}
```



try tab tab  
tryf tab tab

## C# > EXCEPTIONS > Eigene Exception-Klasse mit Fehlermeldung

```
public class MyException : Exception
{
 public MyException(string message) : base(message)
 {
 }
}
```

- Fehler werfen:

```
throw new MyException("Errormessage");
```

- Fehler abfangen:

```
catch (MyException e)
{
 Console.WriteLine(e.Message);
}
```

## C# > EXCEPTIONS > Fragen

- Wozu benötigt man den Finally-Block in einem try-catch-Block?
- Was passiert wenn man innerhalb eines catch-Blockes erneut eine Exception wirft?



# C# > CLASS OBJECT

## C# > CLASS OBJECT > Kopie eines Objektes erstellen (Opt 2): Clone-Funktion

```
public class Person : ICloneable
{
 public string Name { get; set; }
 public Person Ehepartner { get; set; }
 public object Clone()
 {
 Person newPerson = (Person) this.MemberwiseClone();
 newPerson.Ehepartner =
(Person) this.Ehepartner?.Clone();
 return newPerson;
 }
}
```

# C# > INTERFACES

## C# > INTERFACES >

- Geben Schnittstellen für Methoden und Eigenschaften vor, ohne diese zu implementieren
- Unterschied zu abstrakten Klassen: dort können einzelne Methoden bereits implementiert sein
- Eine Klasse kann beliebig viele Interfaces gleichzeitig implementieren
- KONVENTION: Namen von Interfaces fangen mit 'I' an

```
public interface IBewegbar
{
 int Räderanzahl { get; set; }

 void Bewegen();
}
```



interface tab tab

# C# > INTERFACES > Interface implementieren

```
class Fahrrad : Fahrzeug, IBewegbar
{
 //explizite Schnittstellenimplementierung
 void IBewegbar.Bewegen()
 {
 Console.WriteLine("Bewegen Explizit");
 }
}

class Fahrrad : Fahrzeug, IBewegbar
{
 //normale Implementierung
 public void Bewegen()
 {
 Console.WriteLine("Bewegen Implizit");
 }
}
```

## C# > INTERFACES > Fragen

1. Wie viele Interfaces kann eine Klasse implementieren?
2. Von welcher Klasse erben alle anderen Klassen automatisch mit?
3. Welche der Folgenden Bedingungen ergibt true?

```
class Fahrzeug {...}
class Fahrrad : Fahrzeug{...}
Fahrzeug fz = new Fahrzeug();
Fahrzeug fr = new Fahrrad();
if (fz is Fahrzeug){...}
if (fz is Fahrrad){...}
if (fr is Fahrzeug){...}
if (fr.GetType() == typeof(Fahrzeug)){...}
```

# C# > POLYMORPHYSM

## C# > POLYMORPHYSM >

- Objekte einer bestimmten Klasse können entweder als ihr eigener Typ, als beliebiger Basistyp (Klasse von der sie abgeleitet sind) oder als Typ einer Schnittstelle, die sie implementieren, verwendet werden

```
class Fahrrad : Fahrzeug, IBewegbar { ... }
```

```
Fahrrad fahrrad = new Fahrrad();
```

```
Fahrzeug fahrzeug = fahrrad;
```

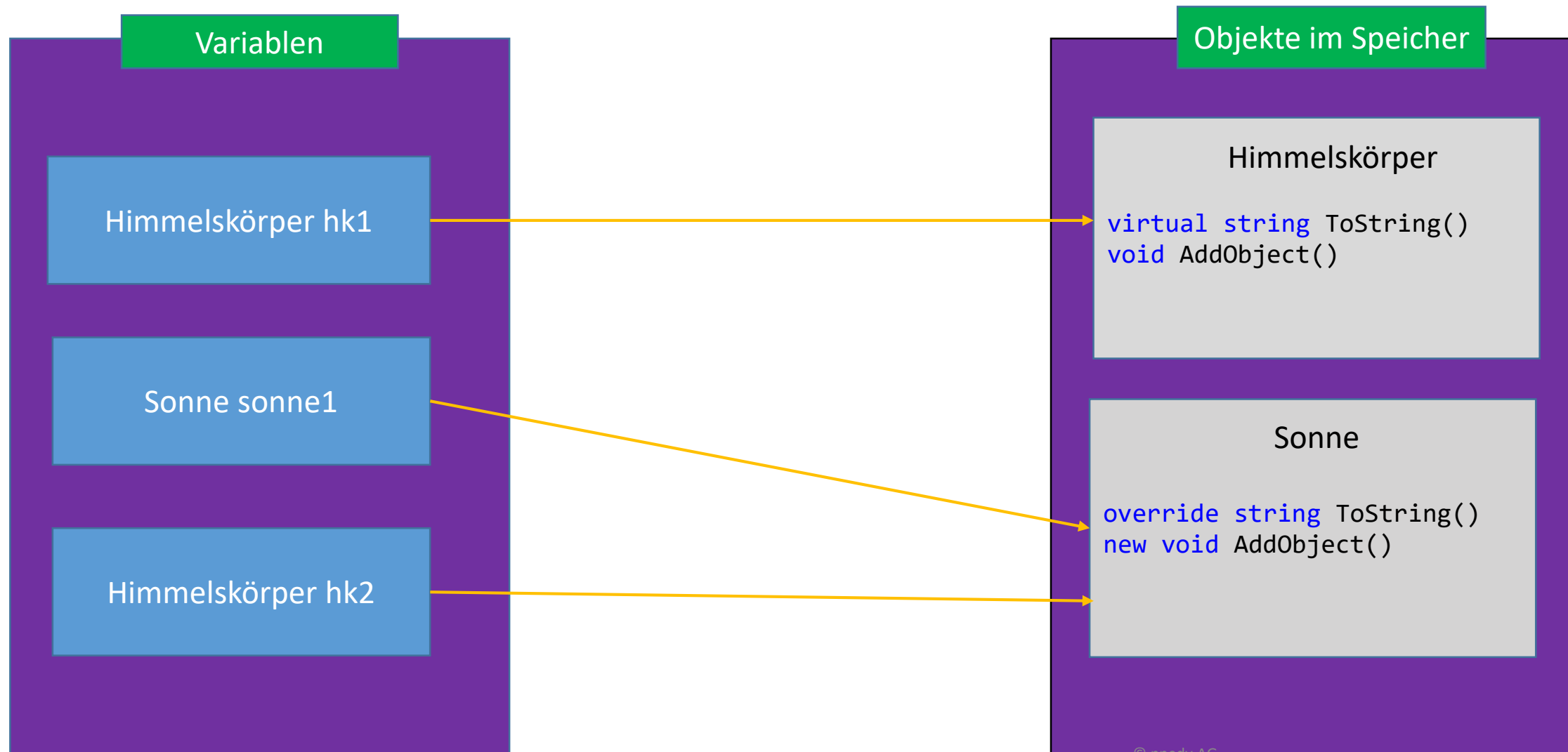
```
IBewegbar bewegbar = fahrrad;
```

```
Variablentyp bezeichner = new Laufzeittyp();
```

```
Static_type identifier = new Run_time_type();
```



# C# > POLYMORPHISM > VIRTUAL & OVERRIDE VS OVERLOAD (HIDING)



## C# > POLYMORPHYSM > Typ eines Objektes prüfen

- GetType() ermittelt den Typ des Objektes, auf den eine Variable referenziert

```
Fahrzeug fahrzeug = new Fahrrad();
if(fahrzeug.GetType() == typeof(Fahrrad)) // true
if(fahrzeug.GetType() == typeof(Fahrzeug)) // false
```

- is prüft ob ein Objekt Typ einer bestimmten Klasse ist, ein Interface implementiert oder Kind einer Basisklasse ist (fahrzeug steht für Fahrzeug-Objekt und fahrrad für Fahrrad-Objekt)

```
fahrzeug is Fahrrad // true
fahrzeug is Fahrzeug // true
fahrrad is IBewegbar // true
fahrrad is Fahrzeug // true
fahrrad is Object // true
```

# C# > GENERIC

## C# > GENERICS > Modulübersicht

- Einsatz und Verwendung von Generics
- generische Klassen
- generische Methoden
- Constraints
- Vererbung von generischen Klassen
- Kovarianz / Kontravarianz / Invarianz

## C# > GENERICS >

- Platzhalter für Datentypen in Klassen und Methoden

```
void Method<T>(T obj)
{
 ...
}
```

```
Method<object>(new object());
Method(new object());
Method("Hello World");
```

- Beispiele im .NET Framework:  
List<T>, Stack<T>, Dictionary<TKey, TValue>, Comparer<T>  
Action<T>, Func<T>, EventHandler<T> ...

## C# > GENERICS > Demo

Vergleich zwischen einem selbst implementierten Stack für den Datentyp „object“, einer typenspezifischen Implementierung und einer generischen Implementierung

## C# > GENERICS > Constraints

- Einschränkungen für die Wahl des Datentypen
- Arten von Constraints:

```
class InterfaceConstraint<T> where T : IDisposable { }
class KlassenConstraint<T> where T : Beispielklasse { }
class WertetypConstraint<T> where T : struct { }
class ReferenztypConstraint<T> where T : class { }
class KonstruktorConstraint<T> where T : new() { }
```

## C# > GENERICS > Vererbung

- Ist die Basisklasse generisch, kann die Subklasse den Typenparameter übernehmen
- Soll die Subklasse nicht generisch sein, kann man einen konkreten Typen einsetzen

```
class Basisklasse<T> { }
class SubklasseOffen<T> : Basisklasse<T> { }
class SubklasseGeschlossen: Basisklasse<int> { }
```

```
class BasisklasseMitEinschränkungen<T> where T : class { }
class SubklasseOffenMitEinschränkungen<T> : Basisklasse<T> where T : class { }
class SubklasseGeschlossenMitEinschränkungen : Basisklasse<string> { }
```



## C# > GENERICS > Kovarianz (out)

- Eine Instanz eines kovarianten generischen Typen ist implizit zu einem generischen Typen konvertierbar, der einen weniger spezifischen generischen Typenparameter besitzt

```
[TypeDependencyAttribute("System.SZArrayHelper")]
public interface IEnumerable<out T> : IEnumerable
```

```
IEnumerable<Tier> tiere = Enumerable.Empty<Tier>();
IEnumerable<Hund> hunde = Enumerable.Empty<Hund>();
```

```
tiere = hunde;
// hunde = tiere; // Typenfehler
```

## C# > GENERICS > Kontravarianz (in)

- Eine Instanz eines kontravarianten generischen Typen ist implizit zu einem generischen Typen konvertierbar, der einen spezifischeren generischen Typenparameter besitzt

```
public interface IComparer<in T>

IComparer<Tier> tierVergleicher = null;
IComparer<Hund> hundVergleicher = null;

hundVergleicher = tierVergleicher;
// tierVergleicher = hundVergleicher; // Typenfehler
```

## C# > GENERICS > Invarianz

- Es besteht keine Beziehung zwischen invarianten Typen, da man sowohl Werte konsumieren als auch produzieren kann

```
[DefaultMember("Item")]
[TypeDependencyAttribute("System.SZArrayHelper")]
public interface IList<T> : ICollection<T>, IEnumerable<T>, IEnumerable
```

```
IList<Hund> hunde = new List<Hund>();
IList<Tier> tiere = new List<Tier>();
// tiere = hunde; // Typenfehler
```

```
// Wenn IList kovariant wäre, könnte man Elemente vom falschen Typ hinzufügen
IList<Tier> zoo = new List<Katze>(); // Wenn das erlaubt wäre...
zoo.Add(new Hund()); // ... wäre das auch erlaubt
```

```
// Wenn IList kontravariant wäre, könnte man Elemente vom falschen Typ zurückbekommen
IList<Hund> zwinger = new List<Tier>(); // Wenn das erlaubt wäre ...
Hund dog = zwinger[0]; // ... wäre das auch erlaubt
```

# C# >

# GENERIC COLLECTIONS

## C# > GENERIC COLLECTIONS >

```
List<string> StringListe = new List<string>();
StringListe.Add("1. Eintrag");
foreach(var item in StringListe)
{
 Console.WriteLine(item);
}
```

Deklaration der Klasse List

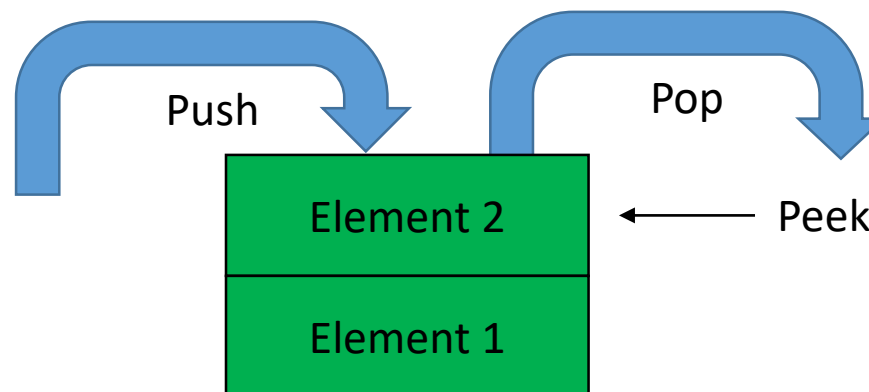
```
public class List<T> {
 public void Add(T item);
}
```

T steht für einen beliebigen Datentypen

## C# > GENERIC COLLECTIONS > Spezielle Arten von Listen: Stack

- LIFO: Last In First Out

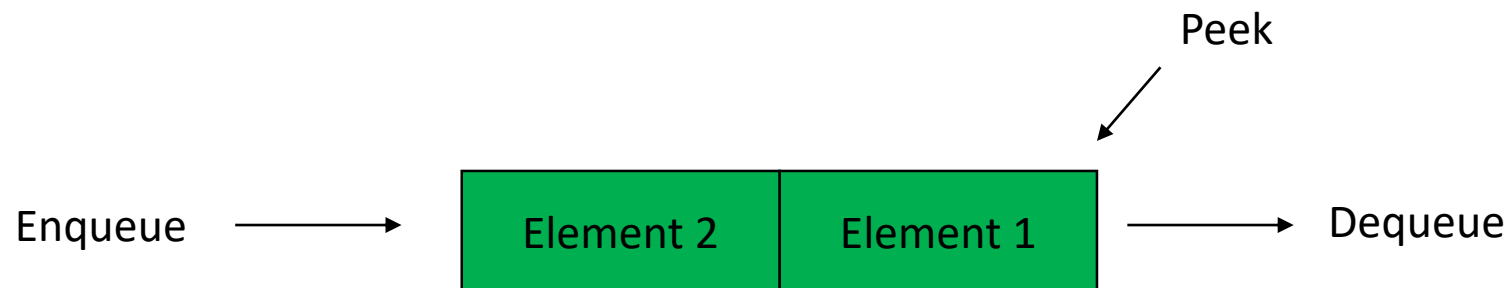
```
Stack<string> stringStack = new Stack<string>();
stringStack.Push("Element 1");
stringStack.Push("Element 2");
stringStack.Peek() //=> "Element 2"
stringStack.Pop() //=> Element 2 herunternehmen
```



## C# > GENERIC COLLECTIONS > Spezielle Arten von Listen: Queue

- FIFO: First In First Out

```
Queue<string> stringQueue = new Queue<string>();
stringQueue.Enqueue("Element 1");
stringQueue.Enqueue("Element 2");
stringQueue.Peek() //=> "Element 1"
stringQueue.Dequeue() //=> Element 1 herausnehmen
```



# C# > GENERIC COLLECTIONS > Spezielle Arten von Listen: Dictionary

- Ordnet einem Wert (**Key**) einen anderen Wert zu (**Value**)

```
Dictionary<int, string> stringDictionary = new Dictionary<int, string>();
stringDictionary.Add(5, "Element 1");
stringDictionary.Add(10, "Element 2");
stringDictionary[10]; // "Element 2"
stringDictionary.ContainsKey(5) //true
stringDictionary.ContainsValue("Element 3") //false
```

| Key vom Typ Integer | Value vom Typ String |
|---------------------|----------------------|
| 5                   | „Element 1“          |
| 10                  | „Element 2“          |



# C# > GENERIC COLLECTIONS > LAB

- QUESTIONS
- DEMO
- EXERCISE

## C# > GENERIC COLLECTIONS > Fragen

1. Welche der folgenden Typen wird als Wert und welche als Referenz übergeben?
  1. int
  2. Random
  3. bool
  4. List<>
  5. decimal
2. Wie kann man innerhalb von Funktionsparametern erzwingen dass ein Wert auf jeden Fall als Referenz und nicht als Wert übergeben wird?
3. Wie definiert man ein Dictionary welches einem Namen eine Liste von Wohnorten zuordnet?

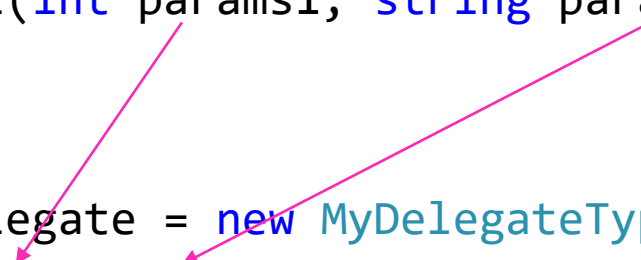
# **C# > DELEGATES, ANONUMOUS FUNCTIONS**

## C# > DELEGATES

- Delegate-Variablen speichern Referenzen auf andere Methoden
- Referenzen können zur Laufzeit hinzugefügt/entfernt werden
- Referenzierte Methoden müssen gleiche Signatur haben wie der Delegate-Typ

```
public delegate int MyDelegateTyp(int param1, string param2);
public int Beispiel(int param1, string param2) {
 return 2;
}

MyDelegateTyp myDelegate = new MyDelegateTyp(Beispiel);
myDelegate.Invoke(5, "Test"); //Rückgabewert: 2
```



# C# > DELEGATES > Vordefinierte Delegate-Typen

| Name                   | Rückgabotyp | Parameter | Beispiel für referenzierbare Methode                                                 |
|------------------------|-------------|-----------|--------------------------------------------------------------------------------------|
| Action<int>            | Void        | 1 Integer | <pre>public void myAction(int p1) {     Console.Write(p1); }</pre>                   |
| Predicate<int>         | Bool        | 1 Integer | <pre>public bool myPredicate(int p1) {     return true; }</pre>                      |
| Func<int, int, string> | String      | 2 Integer | <pre>public string myFunc(int p1, int p2) {     return (p1 + p2).ToString(); }</pre> |
| Func<string>           | String      | Keine     | <pre>public string myFunc() {     return "Test"; }</pre>                             |

## C# > DELEGATES > Anonyme Methoden

- Anonym bedeutet: Methode ohne Namen
- Delegate-Variablen Methoden zuweisen, ohne diese vorher zu deklarieren

```
Func<int, int> myDelegate = new MyDelegates(
 delegate(int p1) {
 return 5;
 }
);
```

- Kurzschreibweise:

```
Func<int, int> myDelegate = delegate(int p1) {
 return 5;
};
```

## C# > DELEGATES > Fragen

- Welche Methode passt zu welchem Delegate-Typen?

```
void methode1(int x)
int methode2(string y)
void methode3(bool x, bool y)
string methode4(List<string> x)
void methode5()
```

```
Func<List<string>, string> delegate1;
Action<bool, bool> delegate2;
Func<string, int> delegate3;
Action delegate4;
Action<int>
```

# C# > HANDLING EVENTS



## C# > HANDLING EVENTS > EventHandler definieren

- Ein Event ist eine Delegate-Variable, an der sich beliebig viele Methoden (EventListener) anmelden können
- unter C# wird für ein Event-Delegate per Konvention der Typ EventHandler benutzt
- Beim Auslösen des Events werden alle referenzierten Methoden aufgerufen

```
event EventHandler onClick +=
 delegate (Object sender, EventArgs args)
 {
 Console.WriteLine("EventBehandlung");
 };

onClick(this, EventArgs.Empty); //Aufruf des Events
```

# C# > HANDLING EVENTS > Events am Beispiel von WindowsForms

## Properties von button1

ChangeUICues  
Click button1\_Click  
ClientSizeChanged  
ContextMenuStripChanged

## Form1.Designer.cs

```
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(244, 144);
this.button1.TabIndex = 1;
this.button1.Text = "Button 1";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
```

## Form1.cs

```
31
32
33 private void button1_Click(object sender, EventArgs e)
34 {
35
36 }
37
38 }
39
```

# C# > HANDLING EVENTS > LAB

- QUESTIONS
- DEMO
- EXERCISE

DRAFT

C# >

HANDLING TIME

## C# > HANDLING TIME >

- System.Forms.Timer
- System.Timers.Timer
- System.Threading.Timer

## C# > HANDLING TIME > Timer verwenden

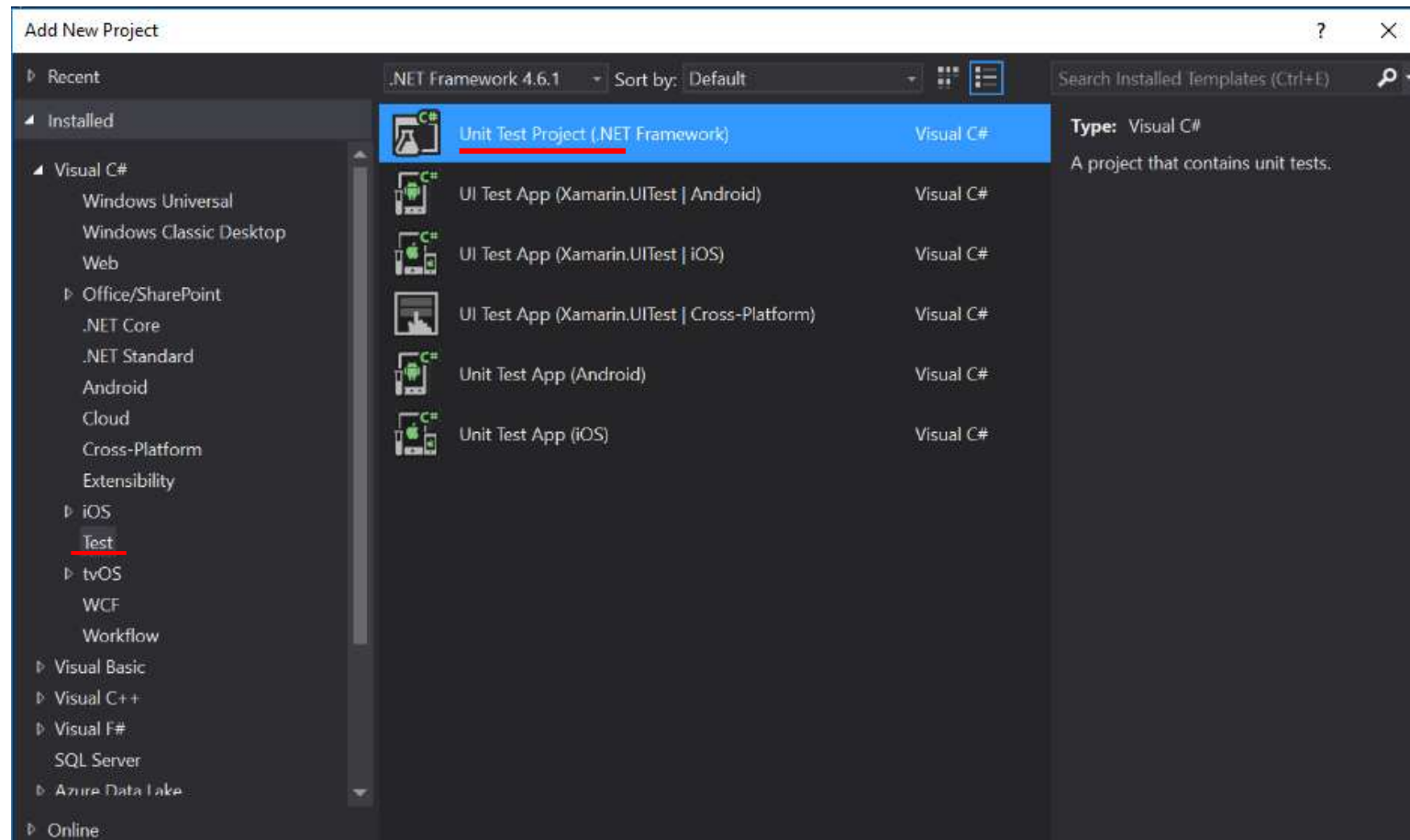
```
using System.Timers;

Timer timer = new Timer();
timer.Elapsed += new ElapsedEventHandler(Countdown);
timer.Interval = 1000;
timer.Start();

void Countdown(Object sender, ElapsedEventArgs args)
{
 countdownNumber--;
 Console.WriteLine(countdownNumber);
 if(countdownNumber <= 0)
 {
 timer.Stop();
 }
}
```

# C# & UNIT TESTS

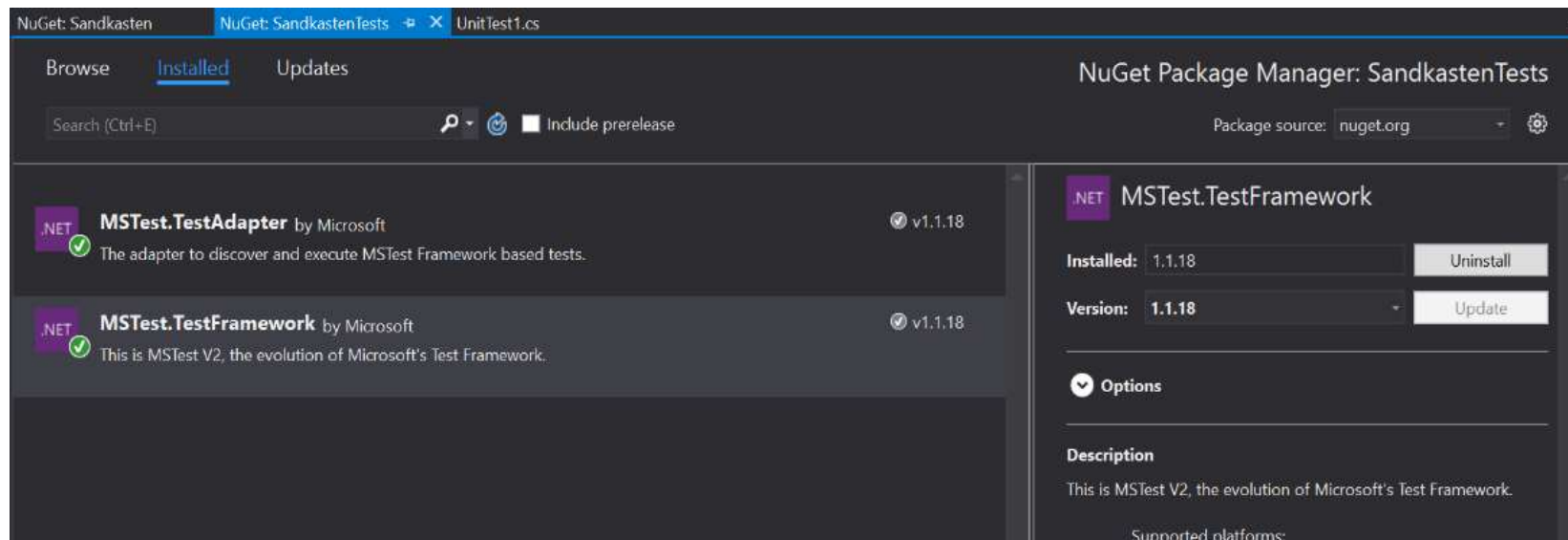
# C# & UNIT TESTS > UnitTests - Testprojekt anlegen





## C# & UNIT TESTS > UnitTests – NuGet Packete installieren

- MSTest.TestFramework und MSTest.TestAdapter über NuGet Package Manager für das Projekt installieren



- Schließlich noch das zu testende Projekt unter References hinzufügen

# C# & UNIT TESTS > UnitTests – Testklasse schreiben

```
[TestClass]
public class UnitTest1
{
 [TestMethod]
 public void TestMethod1()
 {
 KlasseAusProjekt objekt = new KlasseAusProjekt();
 int result = objekt.ZuTestendeMethode(4);
 Assert.IsNotNull(result);
 Assert.AreEqual(2, result);
 }
}
```

- Test laufen lassen über Test->Run->All Tests

# C# & UNIT TESTS > LAB

# **C# > ASYNC, TASK, THREAD & CO**

## C# > ASYNC > Neue Sprachfeatures

- TAP - task asynchronous programming model
  - mehrere gleichzeitig offene, schrittweise (asynchron) vorangeführte Aufgaben innerhalb eines Threads
  - bei mehreren parallel laufenden Aufgaben braucht man aber mehrere Threads. Das ist ein anderes Szenario.
- 
- Viele .NET Klassen haben Async() Operation erhalten
    - HttpClient / WebClient
    - Streams
    - BitmapDecoder / BitmapEncoder
    - ADO.NET (z.b.: SqlCommand)
    - [101 Async Samples](#)

## C# > ASYNC > C# ValueTask<T> #v7.0

- C# 6.0 und darunter erlaubt nur void, Task oder Task<T> für asynchrone Methoden
- ValueTask<T> ist eine alternative für die Referenztypen Task und Task<T> und kann in einigen Situationen die Performance verbessern
- Aus dem Nuget-Package System.Threading.Tasks.Extensions

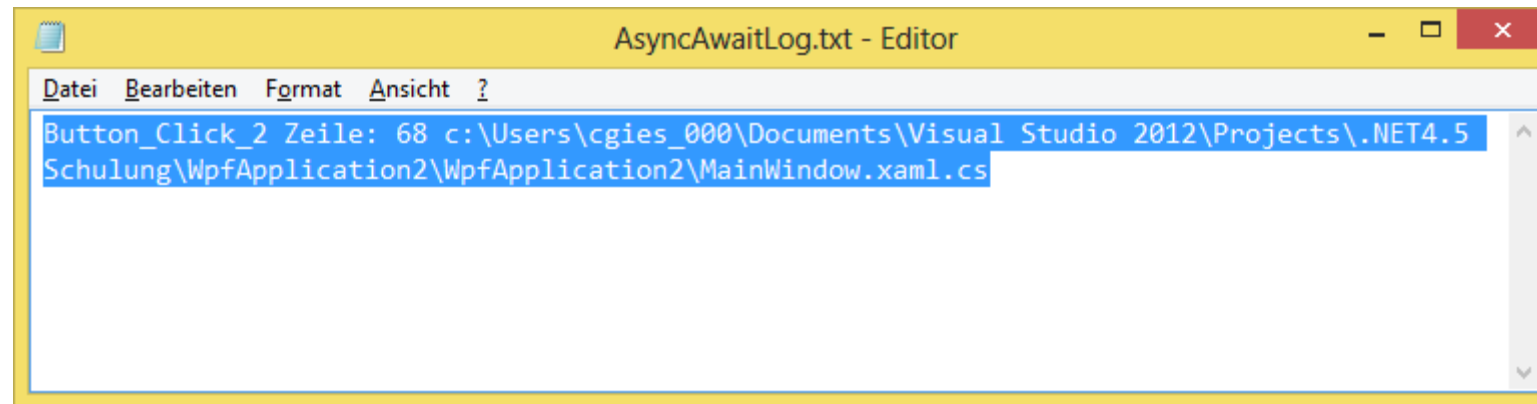
# C# > ASYNC > async Main #v7.1

```
class Program
{
 static async Task Main()
 {
 await Task.Delay(2000);

 Console.WriteLine("Hallo Welt");
 }
}
```

## C# > ASYNC > Neue Sprachfeatures

- Loggen Sie alle Methoden-Aufrufe des vorherigen Beispiels in eine Logdatei „AsyncAwaitLog.txt“



Übung



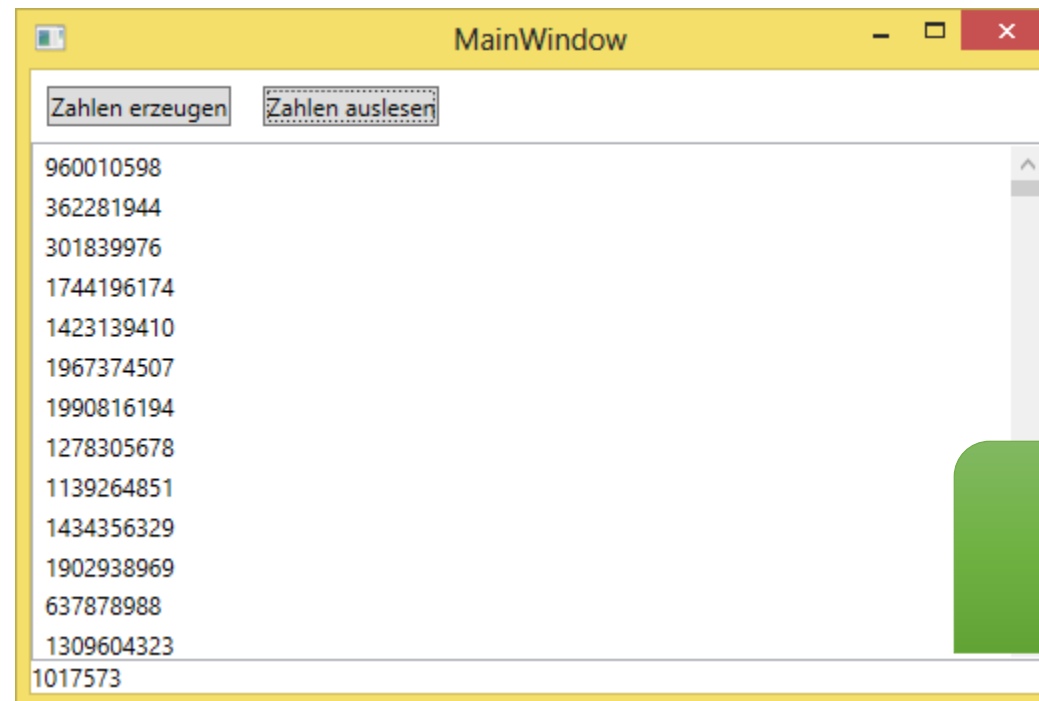
# C# > ASYNC > Asynchrone Streams #v8.0

```
public static async IEnumerable<int> GeneriereZahlen()
{
 for (int i = 0; i < 20; i++)
 {
 await Task.Delay(100);
 yield return i;
 }
}
```

```
public static async void GebeZahlenAus()
{
 await foreach (var zahl in GeneriereZahlen())
 {
 Console.WriteLine(zahl);
 }
}
```

# C# > ASYNC > Async/Await

- Entwickeln Sie eine kleine Anwendung die:
  - Per Button-Click in eine Datei 1 000 000 zufällige Zahlen schreibt
  - Per Button-Click diese wieder ausliest



Übung

# C# > HASHTABLES

## C# > HASHTABLES > Spezielle Arten von Listen: Hashtable

- Ähnlich wie Dictionary, aber Key und Value können einen beliebigen Wertetyp (int, float, char, string) oder Referenztypen haben (List, Stack, eigene Klasse)
- Beim Versuch auf einen nicht-existent Index zuzugreifen, gibt es keine Fehlermeldung

```
Hashtable ht = new Hashtable();
ht.Add("key1", "value1");
ht.Add(20, 100);
ht.Add(10.5, new DateTime(2012, 1, 1));

if(ht[10.5]?.GetType() == typeof(DateTime)) {
 Console.WriteLine("Jahr: " + ((DateTime)ht[10.5]).Year);
}
```

# C# > TUPELS

# C# & LINQ EXPRESSIONS

# C# & LINQ EXPRESSIONS > LINQ Basics

- Typisierte Daten Aggregationen

- Mehrzeilig
- Einzeilig -> auch mehrere Zeilen

- Where

- Find Key

- Group by

- Sort

- Select

- New oder Anonym

- Join

```
var queryLondonCustomers = from cust in customers
 where cust.City == "London"
 select cust;
```

```
var studentNames = studentList.Where(s => s.Age > 18)
 .Select(s => s)
 .Where(st => st.StandardID > 0)
 .Select(s => s.StudentName);
```

# C# & LINQ EXPRESSIONS > ASP.NET CORE & LINQ > INTRO

```
// using System.Linq;
var movies = from m in _context.Movie
 select m;
```



# C# & LINQ EXPRESSIONS > Linq (Language Integrated Query)

- Sammlung von Erweiterungsmethoden zur Abfrage und Sortierung von Elementen innerhalb von Aufzählungstypen (Lists, Arrays, Stacks, Queues ...)
- An SQL-Syntax angelehnt

```
using System.Linq;
string [] Städte = new string[] { "Leipzig", "Hamburg", "Hannover" };
var mitH = from stadt in Städte
 where stadt.StartsWith("H")
 select stadt;
```

Ergebnis: String-Liste mit Hamburg und Hannover

```
//Linq-Syntax entspricht dieser Schreibweise
var mitH = Städte.Where(stadt => stadt.StartsWith("H"));
```

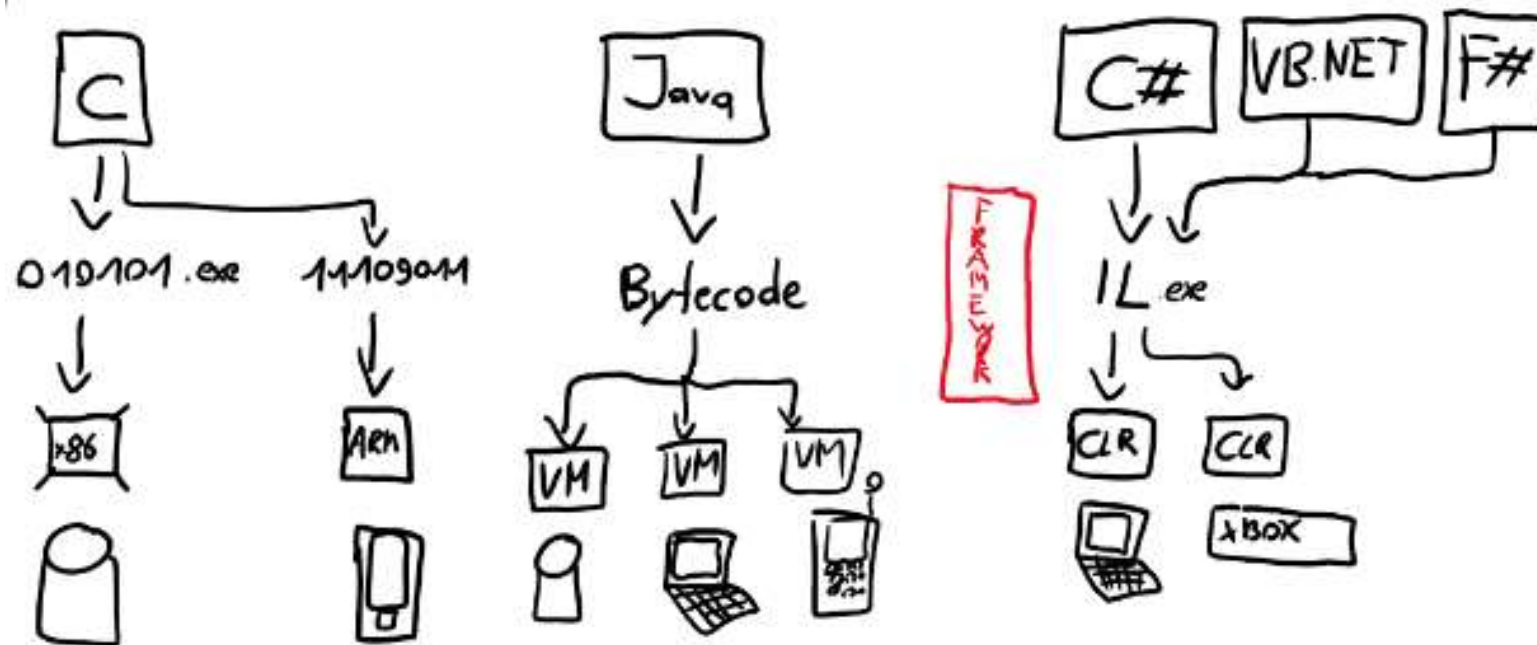
# C# & LINQ EXPRESSIONS > Wichtige Linq-Befehle

| Befehl  | Funktion                      | Beispiel                                                           |
|---------|-------------------------------|--------------------------------------------------------------------|
| From    | Datenquelle festlegen         | <code>from item in StringListe</code>                              |
| Select  | Form des Ergebnisses          | <code>from item in StringListe<br/>select "new " + item</code>     |
| Where   | Ergebnisse filtern            | <code>from item in StringListe<br/>where item.EndsWith("a")</code> |
| OrderBy | Elemente absteigend sortieren | <code>from item in StringListe<br/>orderby item.Length</code>      |

# C# > ECOSYSTEM

# C# > ECOSYSTEM > CLR

- CLR - Common Language Runtime



## C# > ECOSYSTEM > ROSLYN

- <https://github.com/dotnet/roslyn>
- <https://github.com/dotnet/roslyn/blob/master/docs/Language%20Feature%20Status.md>
- <https://docs.microsoft.com/de-de/dotnet/csharp/roslyn-sdk/>