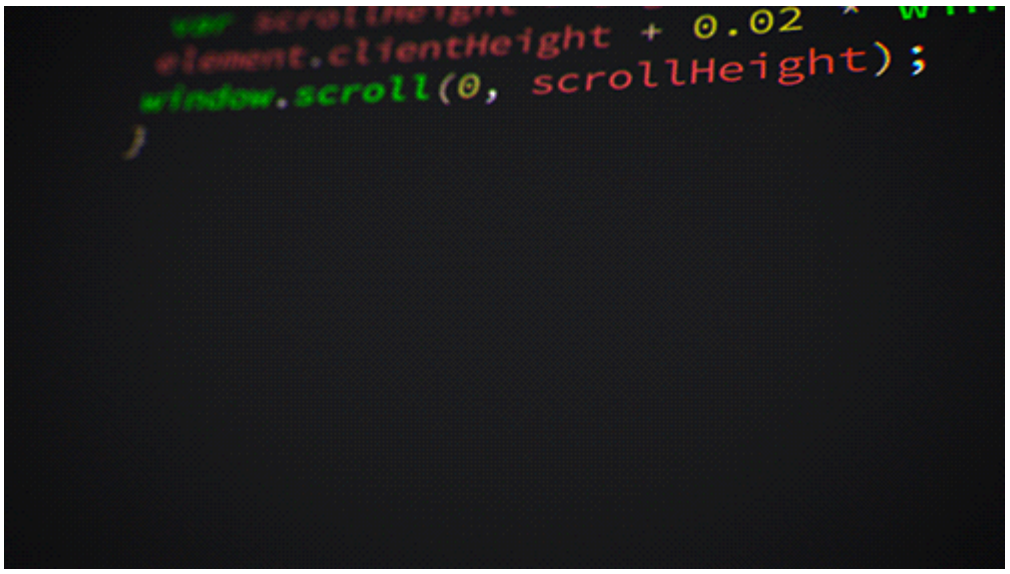
C# Grundlagen II

Parameter	Kursinformationen
Veranstaltung:	`Vorlesung Softwareentwicklung`
Teil:	`4/27`
Semester	`Sommersemester 2025`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	Werte- und Referenzdatentypen, Array, String, implizite Variablendefinition und Nullables
Link auf den GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_Softwareentwicklung/blob/master/04_CsharpGrundlagenII.md
Autoren	Sebastian Zug, Galina Rudolf, André Dietrich, `Lina` & `Florian2501`



Einschub Explizite Projektdefinitionen

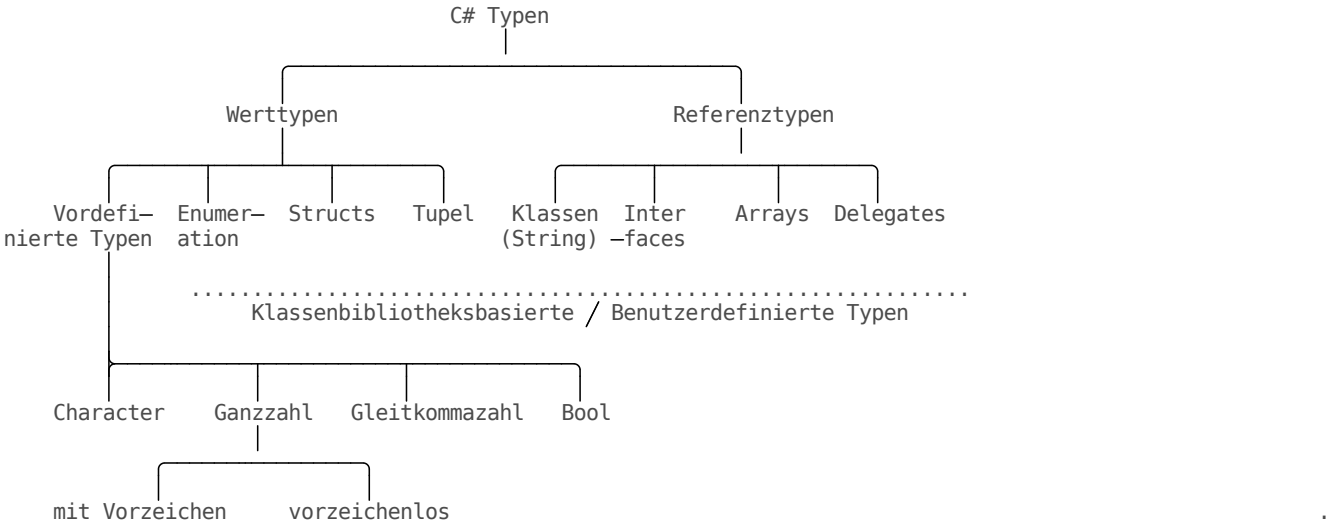
In der letzten Veranstaltung haben wir uns mit der Erstellung von C#-Programmen beschäftigt und die Notwendigkeit

- der expliziten Beschreibung von Abhängigkeiten
- der Angabe von Versionsinformationen
- der Autorenschaft etc.

diskutiert.

> Natürlich ist das kein ausgemachtes Problem von C# ... Codebeispiel in Python

Referenzdatentypen

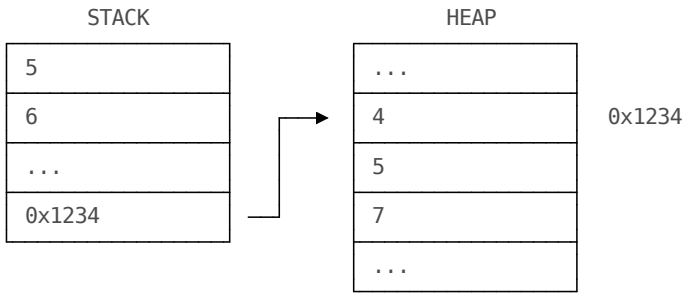


In der vergangenen Veranstaltung haben wir bereits über die Trennlinie zwischen Werttypen und Referenztypen gesprochen. Was bedeutet die Idee aber grundsätzlich?

Aspekt	Stack	Heap
Format	Es ist ein Array des Speichers. Es ist eine LIFO (Last In First Out) Datenstruktur. In ihr können Daten nur von oben hinzugefügt und gelöscht werden.	Es ist ein Speicherbereich, in dem Chunks zum Speichern bestimmter Arten von Datenobjekten zugewiesen werden. In ihm können Daten in beliebiger Reihenfolge gespeichert und entfernt werden.
Was wird abgelegt?	Wertedatentypen	Referenzdatentypen
Was wird auf dem Stack gespeichert?	Wert	Referenz
Kann die Größe variiert werden?	nein	ja
Zugriffsgeschwindigkeit	hoch	gering
Freigabe	vom Compiler organisiert	vom Garbage Collector realisiert

Wie werden Objekte auf dem Stack/Heap angelegt?

```
int x = 5;
int y = 6;
int[] array = new int[] { 4, 5, 7};
```



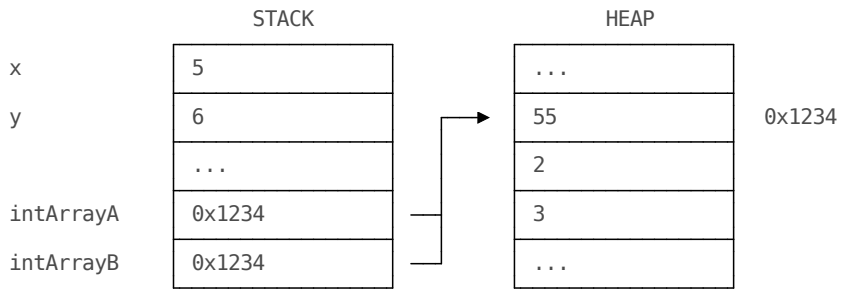
Und was bedeutet dieser Unterschied?

Ein zentrales Element ist die unterschiedliche Wirkung des Zuweisungsoperators `=`. Analoges gilt für den Vergleichsoperator `==` den wir bereits betrachtet haben.

ExampleArrays

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      // Zuweisung für Wertetypen
8      int x = 5;
9      int y = 6;
10     y = x;
11     Console.WriteLine("{0}, {1}", x, y);
12     // Zuweisung für Referenztypen
13     int [] intArrayA = new int[]{1,2,3};
14     int [] intArrayB = intArrayA;
15     Console.WriteLine("Alter Status {0}",intArrayB[0]);
16     intArrayA[0] = 55;
17     Console.WriteLine("Neuer Status {0}",intArrayA[0]);
18     Console.WriteLine("Neuer Status {0}",intArrayB[0]);
19     // Und wenn wir beides vermischen?
20     intArrayA[1] = x;
21     Console.WriteLine("Neuer Status {0}",intArrayA[1]);
22   }
23 }
```

```
5, 5
Alter Status 1
Neuer Status 55
Neuer Status 55
Neuer Status 5
5, 5
Alter Status 1
Neuer Status 55
Neuer Status 55
Neuer Status 5
```



Muss die Referenz immer auf ein Objekt auf dem Heap zeigen?

NullReference.cs

```cs
using System;

public class Program
{
  static void Main(string[] args)
  {
    int [] intArrayA = new int[]{1,2,3};
    int [] intArrayB; //= null;
    if (intArrayB is not null) {
      Console.WriteLine("Alles ok, mit intArrayB");
    } else {
      Console.WriteLine("intArrayB ist null");
    }
  }
}
```

myproject.csproj

```xml
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
  </PropertyGroup>
</Project>
```

```
/tmp/tmppohaj8or/Program.cs(9,9): error CS0165: Use of unassigned local
variable 'intArrayB' [/tmp/tmppohaj8or/project.csproj]

The build failed. Fix the build errors and run again.
/tmp/tmpv5bm3o9a/Program.cs(9,9): error CS0165: Use of unassigned local
variable 'intArrayB' [/tmp/tmpv5bm3o9a/project.csproj]

The build failed. Fix the build errors and run again.
```

Mit `null` kann angezeigt werden, dass diese Referenz noch nicht zugeordnet wurde.

Array Datentyp

Arrays sind potentiell multidimensionale Container beliebiger Daten, also auch von Arrays und haben folgende Eigenschaften:

- Ein Array kann eindimensional, mehrdimensional oder verzweigt sein.
- Die Größe innerhalb der Dimensionen eines Arrays wird festgelegt, wenn die Arrayinstanz erstellt wird. Eine Anpassung zur Lebensdauer ist nicht vorgesehen.
- Arrays sind nullbasiert: Der Index eines Arrays mit n Elementen beginnt bei 0 und endet bei $n - 1$.
- Arraytypen sind Referenztypen.
- Arrays können mit `foreach` iteriert werden.

> **Merke:** In C# sind Arrays tatsächlich Objekte und nicht nur adressierbare Regionen zusammenhängender Speicher wie in C und C++.

Eindimensionale Arrays

Eindimensionale Arrays werden über das Format

```
<typ>[] name = new <typ>[<anzahl>];
```

deklariert.

Die spezifische Größenangabe kann entfallen, wenn mit der Deklaration auch die Initialisierung erfolgt.

```
<typ>[] name = new <typ>[] {<eintrag_0>, <eintrag_1>, <eintrag_2>};
```

ExampleArrays

```
using System;

public class Program
{
  static void Main(string[] args)
  {
    int [] intArray = new int [5];
    short [] shortArray = new short[] { 1, 3, 5, 7, 9 };
    for (int i = 0; i < 3; i++){
      Console.Write("{0, 3}", intArray[i]);
    }
    Console.WriteLine("");
    string sentence = "Das ist eine Sammlung von Worten";
    string [] stringArray = sentence.Split();
    foreach(string i in stringArray){
      Console.Write("{0, -9}", i);
    }
  }
}
```

```
  0  0  0
Das      ist      eine     Sammlung von      Worten     0  0  0
Das      ist      eine     Sammlung von      Worten
```

Recherche: Wie kann ich nach mehreren Zeichen splitten?

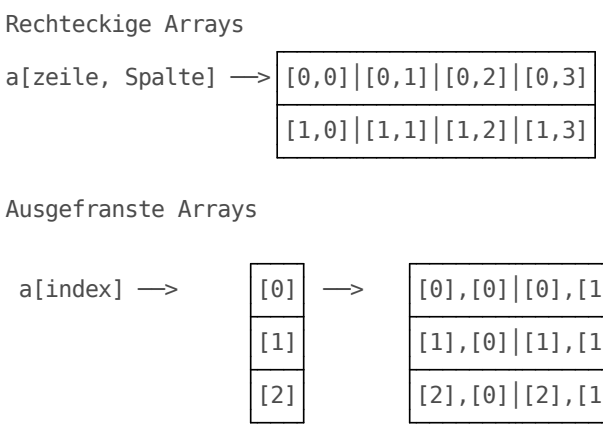
ExampleArrays

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      int [] intArray = new int [5];
8      short [] shortArray = new short[] { 1, 3, 5, 7, 9 };
9      for (int i = 0; i < 3; i++){
10       Console.Write("{0, 3}", intArray[i]);
11     }
12     Console.WriteLine("");
13     string sentence = "Das ist eine Sammlung von Worten";
14     string [] stringArray = sentence.Split();
15     foreach(string i in stringArray){
16       Console.Write("{0, -9}", i);
17     }
18   }
19 }
```

```
  0  0  0
Das      ist      eine     Sammlung von      Worten     0  0  0
Das      ist      eine     Sammlung von      Worten
```

Mehrdimensionale Arrays

C# unterscheidet zwei Typen mehrdimensionaler Arrays, die sich bei der Initalisierung und Indizierung unterschiedlich verhalten.




```
int[,]  rectangularMatrix = //entspricht int[3,3]
{
  {1,2,3},
  {0,1,2},
  {0,0,1}
};

int [][] jaggedMatrix ={ //entspricht int[3][]
    new int[] {1,2,3},
    new int[] {0,1,2},
    new int[] {0,0,1}
};
```

String Datentyp

Als Referenztyp verweisen `string` Instanzen auf Folgen von Unicodezeichen, die durch ein Null `\0` abgeschlossen sind. Bei der Interpretation der Steuerzeichen muss hinterfragt werden, ob eine Ausgabe des Zeichens oder eine Realisierung der Steuerzeichenbedeutung gewünscht ist.

StringVerbatim.cs

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      string text1 = "Das ist ein \n Test der \t über mehrere Zeilen ge
8      string text2 = @"Das ist ein
9      Test der
10     über mehrere Zeilen geht!";
11     Console.WriteLine(text1);
12     Console.WriteLine(text2);
13   }
14 }
```

```
Das ist ein
 Test der        über mehrere Zeilen geht!
Das ist ein
    Test der
    über mehrere Zeilen geht!
Das ist ein
 Test der        über mehrere Zeilen geht!
Das ist ein
    Test der
    über mehrere Zeilen geht!
```

Der Additionsoperator steht für 2 `string` Variablen bzw. 1 `string` und eine andere Variable als Verknüpfungsoperator (sofern für den zweiten Operanden die Methode `toString()` implementiert ist) bereit.

ToString.cs

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      Console.WriteLine("String + String = " + "StringString" );
8      Console.WriteLine("String + Zahl 5 = " + 5); // Implizites .ToStr
9    }
10 }
11
```

```
String + String = StringString
String + Zahl 5 = 5
String + String = StringString
String + Zahl 5 = 5
```

Der Gebrauch des `+` Operators im Zusammenhang mit `string` Daten ist nicht effektiv. eine bessere Performanz bietet `System.Text.StringBuilder`.

In der nächsten Vorlesung werden wir uns explizit mit den Konzepten der Ausgabe und entsprechend den Methoden der String Generierung beschäftigen.

Konstante Werte

Konstanten sind unveränderliche Werte, die zur Compilezeit bekannt sind und sich während der Lebensdauer des Programms nicht ändern. Der Versuch einer Änderung wird durch den Compiler überwacht.

ToString.cs

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      const double pi = 3.14;
8      pi = 5; //erzeugt Fehlermeldung, da pi konstant ist
9      Console.WriteLine(pi);
10   }
11 }
```

```
Compilation failed: 1 error(s), 0 warnings
main.cs(8,6): error CS0131: The left-hand side of an assignment must be
a variable, a property or an indexer
Compilation failed: 1 error(s), 0 warnings
main.cs(8,6): error CS0131: The left-hand side of an assignment must be
a variable, a property or an indexer
```

In C# gibt es auch das Schlüsselwort `readonly`, das eine Variable als konstant kennzeichnet, aber erst zur Laufzeit initialisiert wird.

`const`	`readonly`
Muss zur Compilezeit definiert werden	Kann zur Kompilierzeit oder zur Laufzeit definiert werden
Implizit statisch	Instanz-Ebene oder statisch
Assembler-übergreifend kopiert	Assembler-übergreifend gemeinsam genutzt
Speicher nicht zuweisen	Speicher zuweisen

Implizit typisierte Variablen

C# erlaubt bei den lokalen Variablen eine Definition ohne der expliziten Angabe des Datentyps. Die Variablen werden in diesem Fall mit dem Schlüsselwort `var` definiert, der Typ ergibt sich infolge der Auswertung des Ausdrucks auf der rechten Seite der Initialisierungsanweisung zur Compilierzeit.

```
var i = 10; // i compiled as an int
var s = "untypisch"; // s is compiled as a string
var a = new[] {0, 1, 2}; // a is compiled as int[]
```

`var`-Variablen sind trotzdem typisierte Variablen, nur der Typ wird vom Compiler zugewissen.

Vielfach werden `var`-Variablen im Initialisierungsteil von `for`- und `foreach`- Anweisungen bzw. in der `using`-Anweisung verwendet. Eine wesentliche Rolle spielen sie bei der Verwendung von anonymen Typen.

UsageVar.cs

```
1  using System;
2  using System.Collections.Generic;
3
4  public class Program
5  {
6    static void Main(string[] args)
7    {
8      //int num = 123;
9      //string str = "asdf";
10     //Dictionary<int, string> dict = new Dictionary<int, string>();
11     var num = 123;
12     var str = "asdf";
13     var dict = new Dictionary<int, string>();
14     Console.WriteLine("{0}, {1}, {2}", num.GetType(), str.GetType(),
         .GetType());
15   }
16 }
```

```
System.Int32, System.String,
System.Collections.Generic.Dictionary`2[System.Int32,System.String]
```

Weitere Infos https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/classes-and-structs/implicitly-typed-local-variables

Nullable - Leere Variablen

Ein "leer-lassen" ist nur für Referenzdatentypen möglich, Wertedatentypen können nicht uninitialisiert bleiben (Compilerfehler)

Initialisation.cs

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args){
6      string text = null;   // Die Referenz zeigt auf kein Objekt im He
7      //int i = null;
8      if (text == null) Console.WriteLine("Die Variable hat keinen Wert
9      else Console.WriteLine("Der Wert der Variablen ist {0}", text);
10   }
11 }
```

```
Die Variable hat keinen Wert!
Die Variable hat keinen Wert!
```

Aus der Definition heraus kann zum Beispiel eine `int` Variable nur einen Wert zwischen int.MinValue und int.MaxValue annehmen. Eine `null` ist nicht vorgesehen und eine `0` gehört zum "normalen" Wertebereich.

Um gleichermaßen "nicht-besetzte" Werte-Variablen zu ermöglichen integriert C# das Konzept der sogenannte null-fähigen Typen (*nullable types*) ein. Dazu wird dem Typnamen ein Fragezeichen angehängt. Damit ist es möglich diesen auch den Wert `null` zuzuweisen bzw. der Compiler realisiert dies.

Iniitalisation

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args){
6      int? i = null;
7      if (i == null) Console.WriteLine("Die Variable hat keinen Wert!")
8      else Console.WriteLine("Der Wert der Variablen ist {0}", i);
9    }
10 }
```

```
Die Variable hat keinen Wert!
Die Variable hat keinen Wert!
```

Wie wird das Ganze umgesetzt? Jeder `Typ?` wird vom Compiler dazu in einen generischen Typ `Nullable<Typ>` transformiert, der folgende Methoden implementiert:

```
public struct Nullable <T>{
  private bool defined;
```

```
  public bool HasValue {get;}
  ...
  private T value;
  public T Value {get;}
  ...
  public T GetValueOrDefault()    // value oder default Value entsprechend d
                                  // der Liste unter dem untenstehenden Link
  ...
}
```

https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/default-values-table

Aufgaben

- [ ] Experimentieren Sie mit Arrays und Enumerates. Schreiben Sie Programme, die Arrays nach bestimmten Einträgen durchsuchen. Erstellen Sie Arrays aus Enum Einträgen und zählen Sie die Häufigkeit des Vorkommens.

- [ ] Studieren Sie C# Codebeispiele. Einen guten Startpunkt bieten zum Beispiel die ''1000 C# Examples'' unter https://www.sanfoundry.com/csharp-programming-examples/

Quizze

Als was kann ein String (z.B. "Hello World") auch gesehen werden?

Auswahl

Welche Funktion realisiert das folgende Codebeispiel?

PrimeNumbers.cs

```
1  using System;
2
3  public class Program
4  {
5    static void Main(string[] args)
6    {
7      for (int number = 0; number < 20; number ++)
8      {
9        bool prime = true;
10       for (int i = 2; i <= number / 2; i++)
11       {
12         if(number % i == 0)
13         {
14           prime = false;
15           break;
16         }
17       }
18       if (prime == true) Console.Write("{0}, ", number);
19     }
20   }
21 }
```

```
0, 1, 2, 3, 5, 7, 11, 13, 17, 19, 0, 1, 2, 3, 5, 7, 11, 13, 17, 19,
```