

Vergleich zu C#

Florian Gehring

18.06.2020

Hello, World!

```
1  using System;
2  namespace Beispielcode
3  {
4      class Hello
5      {
6          public static void main(String[] args)
7          {
8              Console.WriteLine("Hello, World");
9          }
10     }
11 }
```

Dieses und viele folgende Beispiele: Tour of C#

- Entwicklung seit 2000 von Microsoft
- C#-Version History von Microsoft:[12]
 - „When you go back and look, C# version 1.0, released with Visual Studio .NET 2002, looked a lot like Java.“
 - „C# version 1.0 was a viable alternative to Java on the Windows platform.“
 - „And yet, C# continued to play a bit of catch-up with Java. Java had already released versions that included generics and iterators.“ (Version 2.0)
 - „With version 3.0, C# had moved the language firmly out from the shadow of Java and into prominence.“ (Ende 2007)

- Virtuelle Maschine, wird zu Zwischencode kompiliert
 - Spezifiziert durch Common Language Infrastructure Standard
 - Implementierung durch .NET, .NET Core und Mono
- CLR (Common Language Runtime)

Java

- Java Virtual Machine (JVM)
- Bytecode

C#

- Common Language Runtime (CLR)
- Common Intermediate Language (CIL)

Typsystem Allgemein

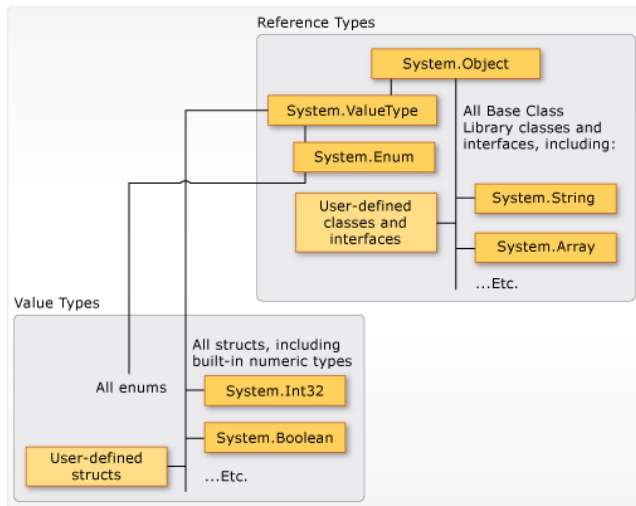


Abbildung: Type System in C# [1]

- Stark Typisiert
 - jede Konstante, Variable und jeder Ausdruck hat einen Typ
- CTS (Common Type System)
 - **Jeder** Typ ist von `System.Object` (`object`) abgeleitet
- Integrierte Typen
 - „Zahlen“, `boolean`, `char`, `object`, `string`
- Referenztypen / Werttypen
 - `System.ValueType`
- Benutzerdefinierte Typen
 - `class`, `enum`, `struct`

Variablen Deklarationen

```
5 // Declaration only:
6 float temperature;
7 string name;
8 Beispielcode.Hello hello;
9
10
11 // Declaration with initializers (four examples):
12 char firstLetter = 'C';
13 var limit = 3;
14 int[] source = { 0, 1, 2, 3, 4, 5 };
15 var query = from item in source
16             where item <= limit
```

- var Keyword [4]
- LINQ

```
6  int i = 2;
7  // Get the Type
8  Type t = i.GetType(); // Methodenaufruf auf int!
9  Console.WriteLine(t);
10 // Print Parent classes.
11 Type b = t;
12 while(b != null) {
13     Console.Write(b + " -> ");
14     b = b.BaseType;
15 }
```

System.Int32

System.Int32 -> System.ValueType -> System.Object ->

Werttypen - Vergleich Java

- In Java: Integer \neq int
- Zwar automatische Konvertierung, aber int ist kein Objekt
- Auskommentierte Zeilen führen zu Fehlern

```
9 // i.getClass();  
10 Object o = i; // <==> Object o = new Integer(i);  
11 System.out.println(o.getClass());  
12 // System.out.println((i instanceof Integer));  
13 System.out.println((o instanceof Integer));  
14 // System.out.println((o instanceof int));
```

Java

- Primitive Typen nicht von Object abgeleitet
- Call-by-Value, Call-By-Reference
- Wrapper-Klasse Integer für int

C#

- Alles (auch int) von object abgeleitet
- Zahlen, boolean sind „Werttype“
- int kann mit int? Nullable gemacht werden

- Erben implizit von `object`
- Enthalten: `constructors`, `properties`, `indexers`, `events`, `operators` and `destructors`
- `sealed`-Modifizier: Für die gesamte Klasse oder einzelne Methoden

- Indexer
 - Array Ähnlicher Zugriff
- Properties
 - Zugriff wie Felder, aber es wird Code ausgeführt.
 - Rückgabewert könnte berechnet werden
 - Unterschiedliche Sichtbarkeit für „get“ und „set“
- Discards
 - Kann einzelne Werte in einem zurückgegebenen Tupel verwerfen.

Vererbung

Klassen Base und Derived mit den Methoden ex1, ex2, ex3.

```
1 using System;
2 class Base {
3     virtual public void ex1() {
4         Console.WriteLine("Base, example 1");
5     }
6     virtual public void ex2() {
7         Console.WriteLine("Base, example 2");
8     }
9     public void ex3() {
10        Console.WriteLine("Base, example 3");
11    }
12 }
13
14 class Derived : Base{
15
16     new public void ex1() {
17         Console.WriteLine("Derived, example 1");
18     }
19     // sealed: Child classes of Derived can't override ex2
20     sealed override public void ex2() {
21         Console.WriteLine("Derived, example 2, ");
22     }
23
24     // Forbidden: override public void ex3() ...
25     new public void ex3() {
26         Console.WriteLine("Derived, example 3");
27     }
28 }
```

```
2 Base trueBase = new Base();
3 Base actuallyDerived = new Derived();
4 Derived trueDerived = new Derived();
5
6 trueBase.ex1();
7 actuallyDerived.ex1();
8 trueDerived.ex1();
9
10 trueBase.ex2();
11 actuallyDerived.ex2();
12 trueDerived.ex2();
13
14 trueBase.ex3();
15 actuallyDerived.ex3();
16 trueDerived.ex3();
17
```

Ausgabe:

Base, example 1
Base, example 1
Derived, example 1

Base, example 2
Derived, example 2
Derived, example 2

Base, example 3
Base, example 3
Derived, example 3

- Java: Alle Methoden sind virtuell
 - Override wird mit `final` verhindert.
 - Kein Äquivalent zu C# `new`
- Java: `@Override` Dekorator soll Code lesbarer machen
- In C# Insgesamt expliziter als in Java
 - Fokus auf Versionierung und Kompatibilität von Code
 - Für interessierte: Interview mit C# Chefdesigner [5]

Generics - C#

Deklarierung der Generischen Klasse:

```
2 class GenericTest<T, U> where T: new() {
```

Instanziierung von Variablen:

```
1 class GenericDemo<A, B> {  
2     public static void test() {  
3         GenericTest<int, int> test1;  
4         var t2 = new GenericTest<int, float>();  
5         // string doesn't have a parameterless constructor:  
6         // GenericTest<string, int> t3 =  
7         //     new GenericTest<string, int>();  
8         GenericTest<int, B> t4 = new GenericTest<int, B>();  
9     }  
10 }
```


Generics - C# - Einschränkungen Typparameter

```
2 class DemoClass1<T, U>
3     // T kann verglichen werden,
4     // ist von U abgeleitet
5     // und hat einen parameterlosen Konstruktor
6     where T : IComparable, U, new()
7     // U ist eine Klasse (kein Struct o.Ä.)
8     where U : class {
9         T tMember;
10        U uMember;
11    }
```

```
13 public static void printGreaterIf<T> (T toPrint, T check)
14                                     where T: IComparable {
15     if (toPrint.CompareTo(check) > 0){
16         System.Console.Write(toPrint);
17     }
18 }
```

```
1 // T unbounded
2 // U Number oder Unterklasse und implementiert Comparable
3 public class GenericDemo<T, U extends Number & Comparable<U>>{
4
5     U uMember;
6     // Generische Methode
7     // C implementiert vergleich mit U.
8     <C extends Comparable<U>> boolean comp(C c){
9         return c.compareTo(uMember) < 0;
10    }
11
12 }
```

Generics - Subtyping - Java

```
1 public class Box<T> {  
2     T tMember;  
3     public void print() {  
4         System.out.println(tMember);  
5     }  
6 }
```

```
6 Box<Number> numberBox;  
7 // Fehler: Box<Integer> ist kein  
8 // Subtype von Box<Number>  
9 // numberBox = new Box<Integer>();
```

Generics - Subtyping - Java

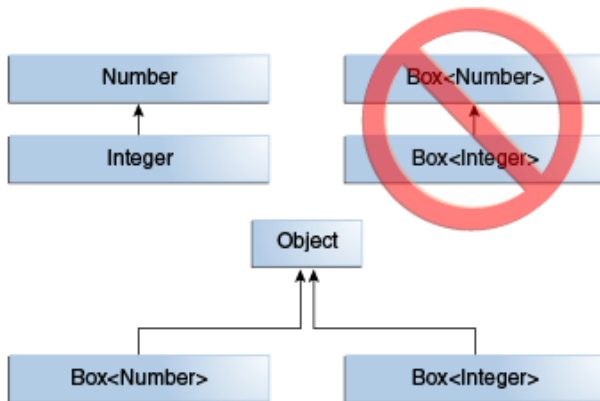


Abbildung: Subtype Relationship [9]

Covariance and Contravariance

- „Covariance and contravariance are terms that refer to the ability to use a more derived type (more specific) or a less derived type (less specific) than originally specified.“ [10]
- Covariante Typparameter sehen wie „gewöhnlicher“ Polymorphismus aus.

```
21 IEnumerable<Derived> d = new List<Derived>();  
22 IEnumerable<Base> b = d;
```

- Contravariant nicht allgemein möglich

```
26 IEnumerable<Base> b = new List<Base>();  
27 IEnumerable<Derived> d = b;
```

Contravariance (1)

Circle implementiert Shape. IComparer<T> ist als Contravariant markiert.

```
1  using System;
2  using System.Collections.Generic;
3
4  abstract class Shape
5  {
6      public virtual double Area { get { return 0; }}
7  }
8
9  class Circle : Shape
10 {
11     private double r;
12     public Circle(double radius) { r = radius; }
13     public double Radius { get { return r; }}
14     public override double Area { get { return Math.PI * r * r; }}
15 }
16
17 class ShapeAreaComparer : System.Collections.Generic.IComparer<Shape>
18 {
19     int IComparer<Shape>.Compare(Shape a, Shape b)
20     {
21         if (a == null) return b == null ? 0 : -1;
22         return b == null ? 1 : a.Area.CompareTo(b.Area);
23     }
24 }
```

Beispiel von [11]

Contravariance (2)

Circle implementiert Shape. IComparer<T> ist als Contravariant markiert.

```
17 class ShapeAreaComparer : System.Collections.Generic.IComparer<Shape>
18 {
19     int IComparer<Shape>.Compare(Shape a, Shape b)
20     {
21         if (a == null) return b == null ? 0 : -1;
22         return b == null ? 1 : a.Area.CompareTo(b.Area);
23     }
24 }
25
26 class Program
27 {
28     static void Main()
29     {
30         // You can pass ShapeAreaComparer, which implements IComparer<Shape>,
31         // even though the constructor for SortedSet<Circle> expects
32         // IComparer<Circle>, because type parameter T of IComparer<T> is
33         // contravariant.
34         SortedSet<Circle> circlesByArea =
35             new SortedSet<Circle>(
36                 // IComparer<Shape>
37                 new ShapeAreaComparer())
38                 // Beispielwerte
39                 { new Circle(7.2), new Circle(100), null, new Circle(.01) };
40     }
41 }
```

Beispiel von [11]

C# - Variante Interfaces

```
1 // From: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/covariance-contravariance/
2 // out R: R ist Covariant
3 // in A: A ist Contravariant
4 interface IVariant<out R, in A>
5 {
6     R GetSomething();
7     void SetSomething(A sampleArg);
8     R GetSetSomethings(A sampleArg);
9
10    // Fehler:
11    // A GetSomethingDifferent();
12    // void SetSomethingDifferent(R sampleArg);
13 }
14
15 class Variant<R, A> : IVariant<R, A>{
16     R rMember;
17     A aMember;
18
19     public R GetSomething() { return rMember;}
20     public void SetSomething(A a) {aMember = a;}
21     public R GetSetSomethings(A a)
22     {
23         aMember = a;
24         return rMember;
25     }
26 }
```

C# - Invariant Classes

```
5  // The interface is variant.
6  IVariant<Derived, Base> variant = new Variant<Derived, Base>();
7  IVariant<Base, Derived> co_contra_variant = variant;
8
9  // The class is invariant.
10 Variant<Derived, Base> invariant = new Variant<Derived, Base>();
11 // The following statement generates a compiler error
12 // because classes are invariant.
13 // Variant<Base, Derived> invariant2 =
14 //     new Variant<Derived, Base>();
```

Generics - Wildcards - Java

Wildcards mit upper (extends) oder lower (super) Bound. extends gilt auch für implementierte Interfaces.

```
11 Box<? extends Number> wildcardBox = new Box<Integer>();  
12 Box<? super Integer> nBox = new Box<Number>();
```

Anwendungsbeispiel:

```
15 public static double sumOfList(List<? extends Number> list) {  
16     double s = 0.0;  
17     for (Number n : list)  
18         s += n.doubleValue();  
19     return s;  
20 }  
21 // https://docs.oracle.com/javase/tutorial/java/generics/upperBound.html
```

```
53 Object no;  
54 Number n;  
55 Integer ni;
```

Contravaiaint („in“) Write-Only

```
57 ArrayList<? super Number> nl = new ArrayList<Object>();  
58 nl.add((Number) 1);  
59 nl.add((Integer)1);  
60 // nl.add(new Object());  
61 no = nl.get(0);  
62 // n = nl.get(0);  
63 // ni = (Integer) nl.get(0);
```

Covariant („out“) Read-Only

```
65 ArrayList<? extends Number> il = new ArrayList<Integer>();  
66 // il.add((Integer) 1);  
67 // il.add((Number) 1);  
68 // il.add(new Object());  
69 no = il.get(0);  
70 n = il.get(0);  
71 // ni = il.get(0);
```

Java Konzept: Type Erasure

- Java unterstützt auf Bytecode Ebene keine Generics
 - Keine generischen Typinformationen zur Laufzeit
- Generisches Argument wird durch möglichst spezifischen Typ ersetzt

```
1 public class Node<N extends Number,  
2         C extends Comparable<C>,  
3         T> {  
4     N nMember;  
5     C cMember;  
6     T tMember;  
7 }  
8  
9 public class Node {  
10     Number nMember;  
11     Comparable cMember;  
12     Object tMember;  
13  
14 }
```

Type Erasure: Methoden

```
package test; void test.GenericTest.genericTypeMethod(T arg)
public class Gen Erasure of method genericTypeMethod(T) is the same as another method in type GenericTest<T,U> Java(16777743)
    Peek Problem No quick fixes available
    public void genericTypeMethod(T arg) {
        System.out.println("Type T");
    }
    public void genericTypeMethod(U arg) {
        System.out.println("Type U");
    }
}
```

```
2 class GenericTest<T, U> where T: new() {
3     public void genericTypeMethod(T arg) {
4         System.Console.WriteLine("Type T");
5     }
6     public void genericTypeMethod(U arg) {
7         System.Console.WriteLine("Type U");
8     }
}
```

Type Erasure: Typen Generischer Klassen

Java

```
12 public static void genericListType(Object o) {  
13     // if (o instanceof List<String>); Nicht möglich  
14     if (o instanceof List) {  
15         Object member = ((List) o).get(0);  
16         System.out.println(member instanceof String);  
17     }  
18 }
```

C#

```
10 public static void genericListType(object o) {  
11     if (o.GetType() == typeof(List<string>)){  
12         System.Console.WriteLine(  
13             "Generic Typeinformation is available at Runtime");  
14     }  
15 }
```

Type Erasure: Objekt Instanziierung

Java

```
27 public T createObject(Class<T> clazz)
28     throws InstantiationException, IllegalAccessException, Ill
29     NoSuchMethodException, SecurityException {
30     // Falls bekannt, dass tMember nicht null ist:
31     // tMember.getClass()
32     Constructor<T> c = clazz.getConstructor();
33     // Konstruktor vorhanden?
34     return c.newInstance();
35 }
```

C#

```
20 public T createObject() { // new constraint
21     return new T();
22 }
```


Type Erasure: Statische Methoden

```
|  
Cannot make a static reference to the non-static type T Java(536871434)  
Peek Problem No quick fixes available  
public static void statMethod(T param) {  
}
```

C#

```
25 public static int typeCounter = 0;  
26 public static void statMethod(T param) {  
27     typeCounter += 1;  
28     System.Console.WriteLine("This Works! " + param.ToString());  
29 }  
30 public static void staticMethodDemo() {  
31     GenericTest<int, string>.statMethod(1);  
32     GenericTest<int, string>.statMethod(2);  
33  
34     GenericTest<int, bool>.statMethod(3);  
35  
36     System.Console.WriteLine("<int, string>    typeCounter: " +  
37         GenericTest<int, string>.typeCounter);  
38     System.Console.WriteLine("<int, bool>      typeCounter: " +  
39         GenericTest<int, bool>.typeCounter);  
40 }
```

- Java
 - Es ist trotz Type Erasure teilweise möglich mithilfe von `java.lang.reflection` während der Laufzeit auf die Parametrisierten Typen zuzugreifen.
 - „Umweg“ über Superclass
 - Andere Möglichkeit: Bei Konstruktion immer Klasse mit übergeben.
- C#
 - Besserer Unterstützung für generische Reflektion

- Delegates sind Referenz**typen** (Objekte)
- Sie kapseln die Funktionalität einer (anonymen) Methode
- Die Methode wird mittels des Delegats aufgerufen

```
3 // Deklaration eines neuen Datentyps!  
4 public delegate int DelegMult(float f);
```

Delegates

Erstes Beispiel:

```
4 public delegate int DelegMult(float f);
5
6 public static int TimesTwo(float f) {
7     Console.WriteLine(f + " * 2 as Delegate");
8     return (int)f * 2;
9 }
10 public static int TimesThree(float f) {
11     Console.WriteLine(f + " * 3 as Delegate");
12     return (int)f * 3;
13 }
14
15 public static void demonstrate_delegates() {
16     DelegMult deleg = TimesTwo;
17     Console.WriteLine(deleg(2));
18 }
```

Initialisierung Möglich als: Methode mit Namen, Anonyme Methode und Lambda Funktion

```
35 public static void different_initializers(DelegMult param){  
36     DelegMult as_param = param;  
37     DelegMult named_function = new DelegMult(TimesTwo);  
38     DelegMult anonymous_function = delegate(float f)  
39         { return 2 * (int) f; };  
40     DelegMult lambda = (f) => { return 2 * (int)f;};  
41 }
```

Delegates - Invocation List

- Eine Kombination von Delegates ist möglich
- Methoden TimesTwo (d1) und TimesThree (d2) werden nacheinander mit 2 aufgerufen
 - Delegatenaufruf gibt jetzt void zurück

```
20 public static void as_parameters() {
21     DelegMult d1 = TimesTwo;
22     DelegMult d2 = TimesThree;
23     multicast_delegates(d1, d2);
24 }
25 public static void multicast_delegates(DelegMult d1,
26     DelegMult d2){
27     Console.WriteLine("Length of Invocation List: "
28         + d1.GetInvocationList().Length); // 1
29     d1 += d2; // d1 is now a MultiCast Delegate
30     Console.WriteLine("Length of Invocation List: "
31         + d1.GetInvocationList().Length); // 2
32     d1(2);
33 }
```



- Ein Delegat-Typ akzeptiert auch Delegate mit:
 - „More Derived Types“ als Rückgabewert (Covarianz)
 - „Less Derived Types“ als Parameter (Contravarianz)
 - Implizite Konvertierung, falls `in` bzw. `out` Keyword vorhanden

```
public delegate R DVariant<in A, out R>(A a);
```

- Java: Functional Interfaces
 - bestehend aus einer Funktion
 - Automatisches Casten von Lambda-Ausdrücken
 - „Lambda expressions let you express instances of single-method classes more compactly.“
 - `@FunctionalInterface` Annotation
- Wieder weniger explizit
- Man sieht nicht am Datentyp, dass es sich um eine Funktion handelt
- Vgl. Vortrag letztes Mal: „Real Function Types“

- „An expression tree is a data structure that represents some code “
 - Baumstruktur die einen Ausdruck Repäsentiert.
 - Methoden, Delegate oder Lambda Ausdrücke als Knoten
 - Verbinde Knoten zu Baum
- Kann dynamisch zur Laufzeit erstellt werden
- Ganzer Baum kann in ausführbaren Code übersetzt werden

- [1] Programming Guide C#, <https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide/>, 05.06.2020
- [2] C# Language Specification, <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/language-specification/introduction>, 05.06.2020
- [3] Introduction to C#, <https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/language-specification/introduction>, 06.06.2020
- [4] '"What is the equivalent of the C# 'var' keyword in Java?"', <https://stackoverflow.com/a/49598148>, 05.06.2020
- [5] 'Interview with C# Designer', <https://www.artima.com/intv/nonvirtual.html>, 06.06.2020
- [6] 'Java Generics' <https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html>, 08.06.2020
- [7] 'Blogpost C# / Java Generics', <http://www.jpri.com/Blog/archive/development/2007/Aug-31.html>, 09.06.2020
- [8] 'SO: Get Actual Types', <https://stackoverflow.com/a/5684761>, 09.06.2020
- [9] 'Java Generics: Inheritance', <https://docs.oracle.com/javase/tutorial/java/generics/inheritance.html>
- [10] 'Covariance and Contravariance', <https://docs.microsoft.com/en-us/dotnet/standard/generics/covariance-and-contravariance>, 13.06.2020
- [11] 'Beispiel Contravariance', <https://docs.microsoft.com/en-us/dotnet/standard/generics/covariance-and-contravariance>, 13.06.2020
- [12] 'C# Geschichte', <https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history>, 15.06.2020