

# Vergleich zu C#

## Entwicklung der Programmiersprache Java

Florian Gehring

18.06.2020

Dozent: Prof. Plümicke

<https://github.com/flo-gehring/seminar-2020-entwicklung-java/>

# 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 }
```

- 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

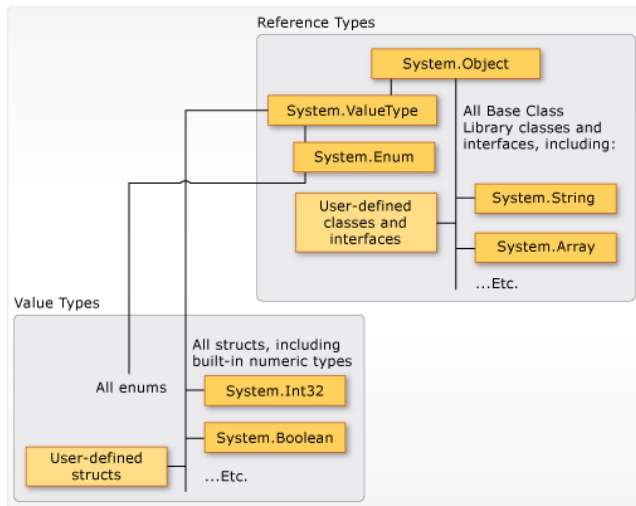


Abbildung: Common Type Systems (CTS) [1]

```
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
- Oft automatische Konvertierung, aber int ist kein Objekt

```
8  int i = 2;
9  // i.getClass();
10 Integer o = (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));
```

- Enthalten: `properties`, `indexers`, `events`, `operators`
- In `C#` kann nur von einer Elternklasse geerbt werden
  - Composition vs. Inheritance
- Virtuelle Methoden bei Vererbung
  - Java
    - (Fast) Alle Methoden sind virtuell
    - Override wird mit `final` verhindert.
    - `@Override` Dekorator
  - `C#`
    - Nicht alle Methoden virtuell
    - Können mit `virtual` Modifier virtuell werden
    - `new` und `override` Keyword



```
2  class GenericClass<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         Boolean comp<C>(C c) where C: IComparable<T> {
12             return c.CompareTo(t) < 0;
13         }
14     }
```

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

# Generics - Subtyping

```
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

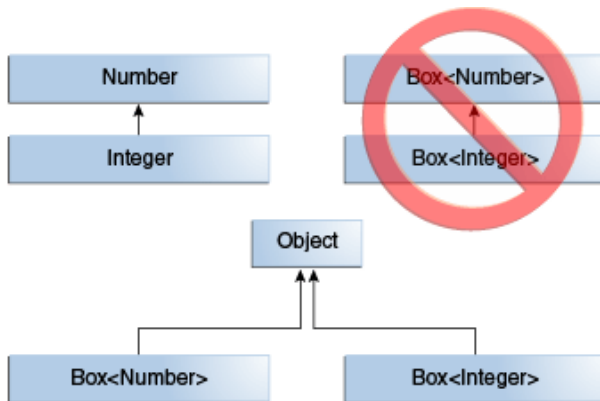


Abbildung: Subtype Relationship [9]

- „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.

```
23 IEnumerable<Derived> d = new List<Derived>();  
24 IEnumerable<Base> b = d;
```

- Contravarianz:

```
28 IComparer<Base> b = new BaseComparer(); // Implements ICompare  
29 IComparer<Derived> d = b;
```

# Contravarianz (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 : 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]

# Contravarianz (2)

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

```
17 class ShapeAreaComparer : 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         // SortedSet<Circle> erwartet IComparer<Circle>. IComparer ist Contravariant
31         // IComparer<Shape> zu übergeben ist also legal
32         SortedSet<Circle> circlesByArea =
33             new SortedSet<Circle>{
34                 // IComparer<Shape>
35                 new ShapeAreaComparer()
36                 // Beispielwerte
37                 { new Circle(7.2), new Circle(100), null, new Circle(.01) };
38             }
39 }
```

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# - Invariante Klassen

Erinnerung:

```
4 interface IVariant<out R, in A>
```

Variablen die eine Klasse als Typ haben sind invariant:

```
5 // Das Interface ist Variant.  
6 IVariant<Derived, Base> variant = new Variant<Derived, Base>();  
7 IVariant<Base, Derived> co_contra_variant = variant;  
8  
9 // Klassenvariablen sind Invariant  
10 Variant<Derived, Base> invariant = new Variant<Derived, Base>();  
11 // Klassen sind invariant:  
12 // Variant<Base, Derived> invariant2 = invariant
```

# Varianz in Java: Wildcards

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

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

## Contravariant („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

- Delegate 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);
```

# Delegate

## 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 Delegaten ist möglich
- Methoden TimesTwo (d1) und TimesThree werden nacheinander mit 2 aufgerufen
  - Delegatenaufruf gibt jetzt void zurück

```
27 DelegMult d1 = TimesTwo;  
28 d1 += TimesThree; // d1 is now a Multicast Delegate  
29 d1(2); // Call Both Methods. Returns void  
30 d1 -= TimesThree; // Remove TimesThree from Multicast Delegate
```

# Java - Functional Interfaces

```
1  @FunctionalInterface
2  public interface DelegMult {
3      public int mult(float f);
4  }
```

```
3  static int TimesTwo(float f) {
4      return (int) (2 * f);
5  }
6  public static void test() {
7      DelegMult timesTwo = UseFunction::TimesTwo;
8      DelegMult timesThree = (f) -> (int) (3 * f);
9      // timesTwo(2);
10     timesTwo.mult(2.0f);
11     timesThree.mult(0.12f);
12 }
```

# Generische Delegate und Varianz

- Ein Deleгат-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

```
1 class GenericDelegates
2 {
3     public delegate R Function<in T, out R>(T arg);
4
5     static bool largeShape(Shape s) {
6         return s.Area > 100;
7     }
8     public static void test()
9     {
10         Function<double, Shape> shapeWithArea;
11         shapeWithArea = (double d) => {return new Circle(d);};
12         Shape shape = shapeWithArea(2);
13
14         Function<Circle, bool> circleComparer;
15         circleComparer = GenericDelegates.largeShape;
16     }
17 }
```

# Functional Interfaces und Varianz

Co- und Contravarianz in Functional Interfaces lässt sich ebenfalls über Wildcards realisieren.

```
1  import java.util.function.Function;
2
3  public class GenericFunctionalInterfaces {
4
5      static Boolean compare(Number n) {
6          return (Boolean) ((int) (n.doubleValue()) == 2 );
7      }
8
9      public static void test() {
10         Function<Integer,
11             ? extends Number> getNumber;
12         getNumber = (Integer i) -> {return (Double) i.doubleValue(); };
13         Number n = getNumber.apply(100);
14
15         Function<? super Integer, Boolean> numberComparer;
16         numberComparer = GenericFunctionalInterfaces::compare;
17         Boolean b = numberComparer.apply(1);
18     }
19 }
```



- 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
- Man sieht nicht am Datentyp, dass es sich um eine Funktion handelt
- Vgl. Vortrag letztes Mal: „Real Function Types“

- [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