Vergleich zu C#

Florian Gehring

18.06.2020

Hello, World!

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

Dieses und viele folgende Beispiele: Tour of C#

Geschichte

- Von Microsoft Entwickelt
- „Direkter Konkurent“ zu Java

Typsystem Allgemein

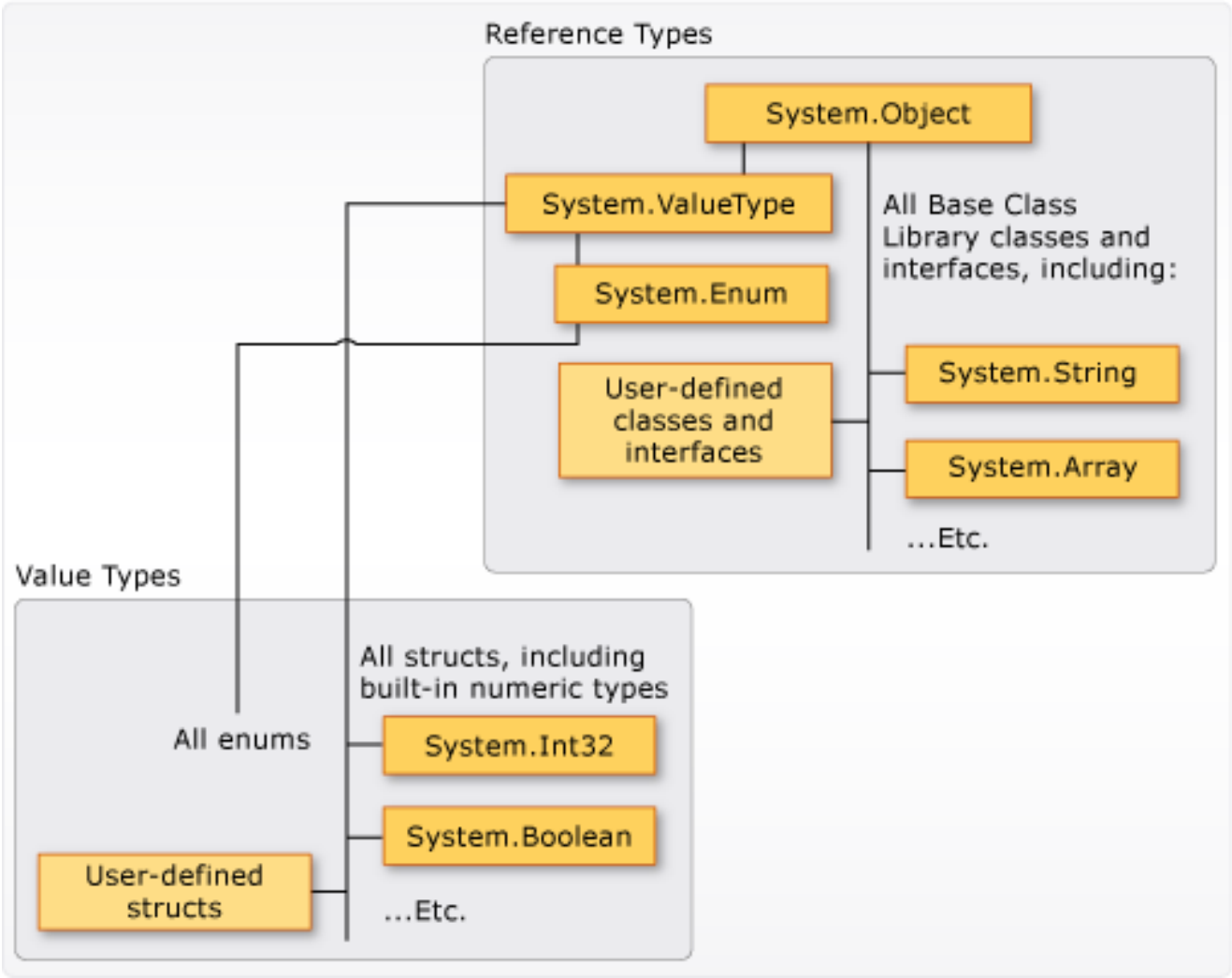


Abbildung: Type System in C# [1]

Allgemeines Typsystem

- 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

Werttypen

```
6  int i = 2;
7  // Get the Type
8          Type t = i.GetType(); // Methodenaufruf auf int!
9          Console.WriteLine(t);
10         // Which Methods can be called on this object?
11         foreach(var method in t.GetMethods()) {
12             Console.Write(method + " ");
13         }
14         Console.WriteLine();
15         // Print Parent classes.
16         Type b = t;
17         while(b != null) {
18             Console.Write(b + " -> ");
19             b = b.BaseType;
20         }
```

System.Int32
Int32 CompareTo(System.Object) Int32 CompareTo(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));
15
```

Zusammenfassung Typen

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

Klassen

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

Indexer, Properties, Discards

- 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 }
```

Vererbung

```
1  public static void modifierBehavior() {
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
0      trueBase.ex2();
1      actuallyDerived.ex2();
2      trueDerived.ex2();
3
4      trueBase.ex3();
5      actuallyDerived.ex3();
6      trueDerived.ex3();
7
8  }
```

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
```

Vererbung - Vergleich Java

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

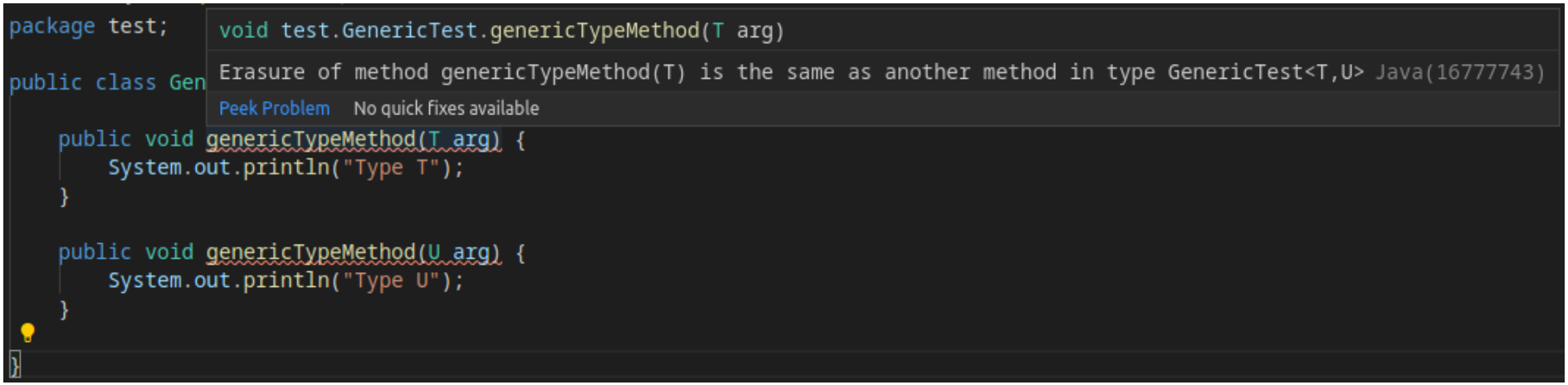
Generics

- Syntax vorstellen

Java Konzept: Type Erasure

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

Type Erasure: Methoden




```
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

```java
public static void genericListType(Object o) {
    // if (o instanceof List<String>); Nicht möglich
    if (o instanceof List) {
        Object member = ((List) o).get(0);
        System.out.println(member instanceof String);
    }
}
```

C#

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

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  }
```

Generics - Reflection

- 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

- `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 }
```

Delegates

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 (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 }
```

Delegates - Vergleich Java

- 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“

Delegates, Lambdas and Expression Trees

- „An expression tree is a data structure that represents some code “
- Baumstruktur die einen Ausdruck Repärsentiert.
- Methoden, Delegate oder Lambda Ausdrücke als Knoten
- Verbinde Knoten zu Baum

$\rightarrow$ Kann dynamisch zur Laufzeit erstellt werden

- Ganzer Baum kann in ausführbaren Code übersetzt werden

Events

events

Quellen

[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.jprl.com/Blog/archive/development/2007/Aug-31.html, 09.06.2020

[8] 'SO: Get Actual Types', https://stackoverflow.com/a/5684761,09.06.2020