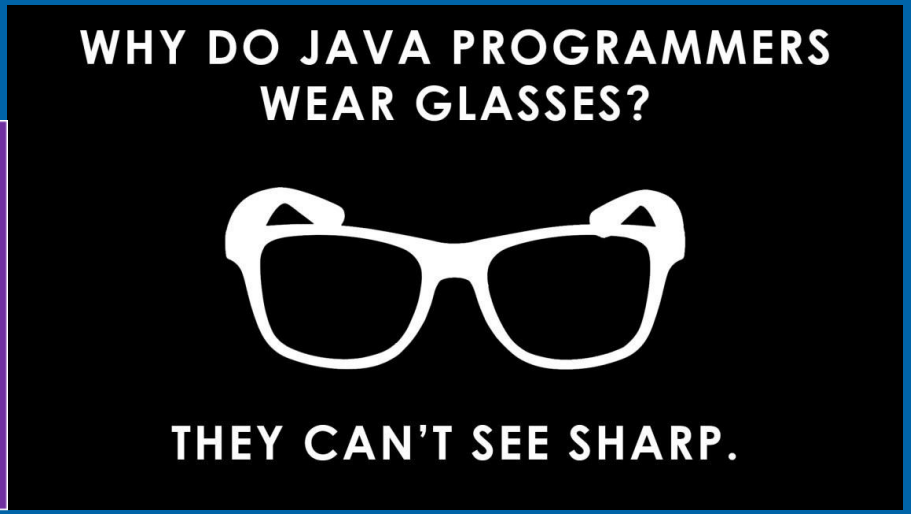
TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

C#
Einführung

C# 7.0



Welche Konzepte verfolgt C#?
Hello C# 7.0
einfache Datentypen und Ausrücke, Kontrollstrukturen

C# Entwicklung

- Version 1.0 im Jahre 2002
- Aktuell: C# 7.2 (16.08.2017)
- multiparadigmatische Allzweck-Programmiersprache
- typsicher
- sehr gute Eignung zum Schreiben von großen und komplexen Softwaresystemen

Was ist .NET?

- ursprünglich Microsoft-eigenes Framework voller Bibliotheken, Tools, Programmen und anderer Konstrukte
- Offenlegung von großen Teilen des Quelltextes im Jahre 2016 unter dem Namen ***.NET Core***
- zusätzlich freie quelloffene Implementierung ***Mono***
- unweigerlich ist C# mit dem .NET Framework eng verbunden
- historisch nur für Windows entwickelt. Heute praktisch plattformunabhängig durch .NET Core und Mono

Die .NET Laufzeitumgebung

Common Language Runtime - CLR	Die Laufzeitumgebung, in der .NET Programme ausgeführt werden -> virtuelle Maschine mit eigenem Befehlssatz
Common Intermediate Language - CIL	Befehlssatz der virtuellen Maschine. Alle .NET Sprachen werden hierhin übersetzt.
Common Type System - CTS	Genaue Definition von allen Datentypen von allen möglichen .NET Sprachen -> Erlaubt schnelle und sehr gute Interoperabilität.
Just-In-Time-Compilation - JIT	Kurz vor der eigentlichen Programmausführung wird der CIL Code in Maschinencode übersetzt.

C# Kompilieren

-> Tafel

Intermediate Language

- praktisch der Assembler der virtuellen Maschine
- kann mittels `ildasm.exe` (.NET Framework) oder `monodis` (Mono Framework) angezeigt werden (disassemblierer)
- wird mittels JIT in nativen Maschinencode übersetzt
- Kann auch direkt programmiert werden:

```
.assembly HalloWelt { }
.assembly extern mscorlib { }
.method public static void Main() cil managed
{
    .entrypoint
    .maxstack 1
    ldstr "Hallo Welt!"
    call void [mscorlib]System.Console::WriteLine(string)
    ret
}
```

Assemblies (modulare Sprache)

- .NET unterstützt komponentenorientierte Softwareentwicklung
- Komponenten heißen Assemblies und sind die kleinsten Bausteine
- bestehen aus Klassen und anderen Ressourcen
- beinhalten neben dem CIL Code auch das **Manifest** und **Metadaten**
- Assemblies sind selbstbeschreibend
- erlaubt *side-by-side execution*

Aufbau von Assemblies

-> Tafel

Hello C# World

Ein einfaches Programm: *helloworld.cs*:

```
class HelloWorld
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Hello World!");
    }
}
```

```
csc helloworld.cs
```

Ausführung in Windows nativ. Unter Linux/Mac mittels `mono`

Alternative (Standard für neue Projekte bei VS 2017)

```
using System;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- Namensräume – können betrachtet werden wie eine Telefonvorwahl
- `using` Direktive „wählt“ in einen Namensraum ein

Gliederung von Programmen

-> Tafel

Nutzung von verteilten Programmbausteinen

```
public class Calculator
{
    public static int Add(int x, int y)
    {
        return x + y;
    }
}
```

Calculator.cs

```
using System;

public class Program
{
    static void Main(string[] args)
    {
        int a = 1;
        int b = 2;
        int c;
        c = Calculator.Add(a, b);
        Console.WriteLine("Ergebnis = " + c);
    }
}
```

Program.cs

Entweder gemeinsam übersetzen:

```
csc Program.cs Calculator.cs
```

oder als Bibliothek getrennt:

```
csc /target:library Calculator.cs
csc /reference:Calculator.dll Program.cs
```

C# Syntax

- 77 Schlüsselwörter
- Groß- und Kleinschreibung werden unterschieden
- Kommentare wie in C:

```
// Einzeiliges Kommentar
```

```
/* Mehrzeiliges Kommentar
hier geht es weiter */
```

```
/* So sieht es aber wesentlich besser aus
 * mit so einem Stern dazwischen.
 * Das macht VS2017 automatisch.
 * Cool, oder? */
```

einfache Datentypen in C#

C# beinhaltet von Haus aus:

- vorzeichenbehaftete ganzzahlige Typen
- vorzeichenlose ganzzahlige Typen
- Fließkommatypen
- bool
- char

Literale

- verschiedene Zahlensysteme möglich

```
int dezimal = 156;
int hexadezimal = 0x7F;
int binär = 0b110010;
```

- gute Lesbarkeit durch *digit separators*

```
int sehr = 1_000_000;
int gute = 0x7F_23_11;
int lesbarkeit = 0b1100_1001;
```

- Nutzung der Exponentenschreibweise

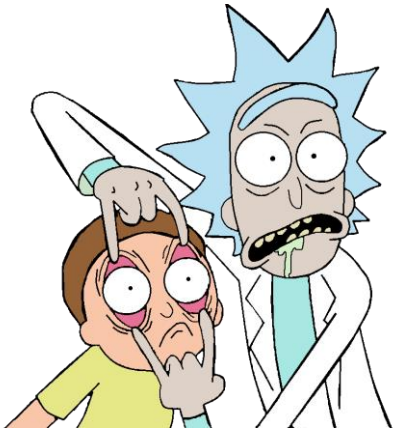
```
double milliarde = 1.0E9;
```

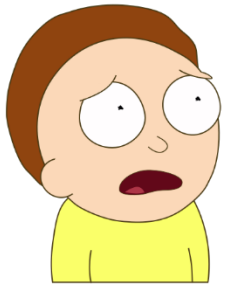
Arithmetik

- Arithmetische Ausdrücke:
  + - * / %
- Inkrement und Dekrement
  ++ --

```
int x = 1, y = 2;
int z = x++ * ++y; // z = 1*3
```

Achtung!
Es gibt keine automatische Überlaufprüfung!



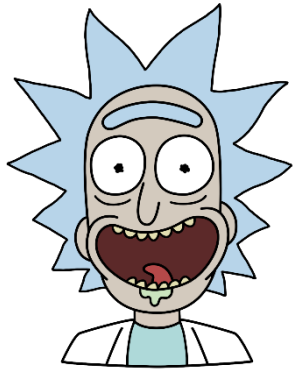


Was passiert, wenn unterschiedliche Typen miteinander verrechnet werden?

Der Ergebnistyp ist immer der kleinste arithmetische Typ, der beide Operanden einschließt. Mindestens aber `int`.

Bei vorzeichenlosen Werten wird der Typ entsprechend erweitert.

Eine automatische Umwandlung nach `decimal` gibt es aber nicht.



Typumwandlung einfacher Ausdrücke

Bei Typumwandlung wird unterschieden in:

- impliziter Cast (Konvertierung)

```
int num = 2147483647;
long bigNum = num;
```

```
float x = 20.5F;
double y = x;
```

- expliziter Cast (Umwandlung)

```
double x = 1234.7;
int a;
// Cast double to int.
a = (int)x;
```

```
uint a = 12;
int b = (int)a;
```

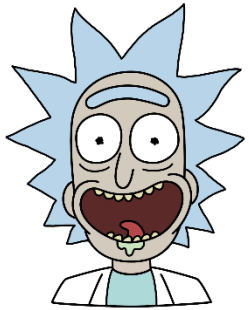
Vergleiche

Operatoren:

< > <= >= == !=

Bei ungleichen Operanden wird der kleinere Typ vor dem Vergleich in den größeren Typ umgewandelt.

Auch hier gibt es keine automatische Konvertierung nach *decimal*.



```
float x = 20.5F;
decimal y = 12.4M;
bool z = (decimal)x > y;
```

Logische Operatoren

- & bitweises Und
- | bitweises Oder
- ^ bitweises exklusives Oder (XOR)
- ~ bitweise Negation (Einerkomplement)
- << shift nach links
- >> shift nach rechts

boolsche Ausdrücke

Operator !:

- Negation

Operator &&:

- boolsches und
- a && b wird in fester Reihenfolge ausgewertet.

Operator ||:

- boolsches oder
- wird ebenfalls mit fester Reihenfolge ausgewertet

Kontrollstrukturen

die Kontrollstrukturen in C# sind:

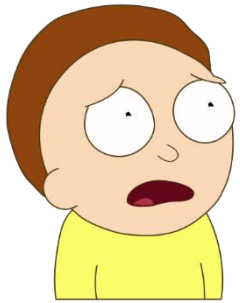
- if (bedingte Anweisung, Alternative)
- switch (Mehrfachverzweigung)
- while (Durchlaufschleife)
- do-while (Abweisschleife)
- for (Zählschleife)
- break und continue
- goto (Sprung)
- return (Rückgabe)

- foreach

if

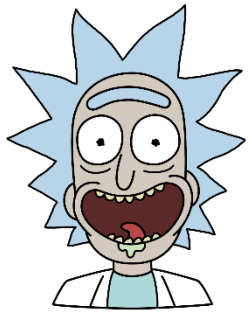
- bedingte Ausführung von Code
- viele Schreibweisen

```
if (x > y) max = x;
if (y > x) max = y;

if (x > y) max = x;
else max = y;

max = x > y ? x : y;
```



Kann ich auch den Wert 0 oder null als false interpretieren?

Nein. Anders als zum Beispiel C oder C++ unterstützt C# ausschließlich den Typ bool in der Bedingung.



switch

- ohne break automatisches Rutschen in die Nächste Marke nach Abschluss der eigenen
- default, wenn keine Marke passt
- wenn default fehlt und keine Marke passt, dann Sprung an das Ende der Anweisung
- statt break sind auch andere Sprünge möglich

```
switch (x)
{
    case 1:
        y = 13;
        break;
    case 2: case 3: case 4:
        y = 25;
        break;
    case 0:
        y = 5;
        break;
    default:
        y = 0;
        break;
}
```

while Schleife

- Prüfung der Schleifenbedingung vor jedem Durchlauf des Schleifenrumpfes

```
while (i > 3)
{
    sum = sum + i;
    i--;
}
while (i > 3)
    sum += i--;
```

- Anders herum mittels do-while möglich: Das wird die Schleife immer mindestens einmal durchlaufen.

```
do
 {
     sum += i;
     i++;
 }
 while (i < 5);
```

Zählschleife mit for

- sehr flexible Schleife
- Initialisierung und Inkrementierung kann aus mehreren Anweisungen bestehen.
- Bedingung muss mittels boolscher Ausdrücke angepasst werden.

```
for (int i = 0; i < 5; i++)
    sum += i;

for (int i = 0, j = 0; i < 4 && j > -8; i++, j = j - 2)
{
    sum += i + j;
}
```

break und continue

- break verlässt die aktuelle Schleife
- nur innerste Schleife wird bei Verschachtelung verlassen

```
for(; ;)
 {
     x = x + 1;
     if (x > 20)
         break;
     else
         sum += x;
 }
```

- continue überspringt den restlichen Schleifenrumpf

```
int t = 5;
while (true)
{
    t++;
    if (x < 12) continue;
    sum += t;
    if (t > 15) break;
}
```

goto

- Aufgrund moderner Schleifen nur noch wenig sinnvolle Einsatzgebiete

- Ein Einsatzgebiet: Sprung aus mehreren Schleifen heraus.

```
top:
sum += i;
i++;
if (i <= max) goto top;
```

```
for (int i=0; i < 10; i++)
{
    for(int j=0; j < 20; j++)
    {
        if (i * j > 20) goto end;
    }
}
end:
```

return

- vorzeitiges Beenden von Methoden ohne Rückgabewert

```
void fun(int x)
{
    if (x < 0) return;
    ...
}
```

- Rückgabe von Funktionswerten

```
int Max(int a, int b)
{
    if (a > b) return a; else return b;
}
```

Zusammenfassung

- Einführung in die .NET Umgebung
- Hello World in C#
- Ausdrücke und Literale
- Kontrollstrukturen