
II.2. Objekte, Klassen und Methoden

- 1. Grundzüge der Objektorientierung
- 2. Methoden, Unterprogramme und Parameter
- 3. Datenabstraktion
- 4. Konstruktoren
- 5. Vordefinierte Klassen

Hüllklassen

- **Primitive Typen** (`boolean`, `char`, `int`, `double`, ...) **passen nicht ins Konzept von Klassen und Objekten.**
- **Nachteil:**
 - unsystematisch
 - keine Referenzparameter für Objekte primitiver Typen
 - manche Methoden verlangen Klassentypen als Parameter
- **Daher existieren für alle primitive Datentypen sogenannte Hüllklassen:**
 - `Boolean`
 - `Character`
 - `Byte`, `Short`
 - `Integer`, `Long`
 - `Float`
 - `Double`

Attribute und Methoden von Integer

- **Objekt-Attribut (nicht public):** der eingehüllte `int`-Wert
- **Klassen-Attribute (statisch):**
 - `MIN_VALUE` : kleinster Wert vom Typ `int` (`-2.147.483.648`)
 - `MAX_VALUE` : größter Wert vom Typ `int` (`2.147.483.647`)
- **Statische Methoden:**
 - `static Integer valueOf (int i)`
 - `static Integer valueOf (String s)`
 - `static int parseInt (String s)`
 - `static String toString (int i)`
- **Methoden:**
 - `String toString ()`
 - `boolean equals (Integer i)`
 - `byte byteValue ()` , `int intValue ()` , `float floatValue ()` , ...

Beispiel zur Verwendung von Integer

```
Integer x = Integer.valueOf(321);  
Integer y = Integer.valueOf("321");  
int z = Integer.parseInt("321");
```

321

321

321

```
String s1 = Integer.toString(321);  
String s2 = x.toString();
```

"321"

"321"

```
System.out.println("x: " + x);  
System.out.println("y: " + y);  
System.out.println("z: " + z);  
System.out.println("s1: " + s1);  
System.out.println("s2: " + s2);
```

```
System.out.println("x == y: " + (x == y));
```

false

```
System.out.println("x.equals(y) : " + x.equals(y));
```

true

```
System.out.println("x.intValue() == z: " +  
                    (x.intValue() == z));
```

true

Autoboxing und Unboxing

Automatische Konversion von prim. Typ zu Hüllklassentyp u. zurück

```
Integer x = 321;
```

```
int i = x;
```

```
Integer y = x + 2;
```

```
Double z = Math.sqrt(y);
```

```
Double d = 4;
```

```
Double d = Double.valueOf(4);
```

```
Double e = 4.0;
```

```
Integer x = Integer.valueOf(321);
```

Autoboxing

```
int i = x.intValue();
```

Unboxing

```
Integer y = Integer.valueOf(x.intValue() + 2);
```

Autoboxing unboxing

```
Double z = Double.valueOf(Math.sqrt(y.intValue()));
```

Danach wird der int-Wert in double ^{unboxing} konvertiert.

```
Double d = Integer.valueOf(4); Typfehler!
```

*Autoboxing u. Unboxing immer nur zwischen
Prim. Typ u. dazugehörigen Hüllklassentyp.
Keine Konversion v. Integer nach Double*

```
Double e = Double.valueOf(4.0);
```

Autoboxing und überladene Methoden

• Autoboxing/
Unboxing
beim Über-
laden:
Erst dann,
wenn sonst
keine Methode
passt.

```
static int f (int i)           {return 1;}
static int f (Integer x)       {return 2;}
static int f (double d)        {return 3;}
static int f (int... a)         {return 4;}
static int f (Integer... b)    {return 5;}
```

• Var args
erst dann,
selbst bei
Auto-/
Unboxing
sonst keine
Methode
passt

```
f (Integer.valueOf(1)) = 2
f (1)                  = 2
```

Fehler!

vararg Methoden möglichst
nicht überladen!

Beispiel zur Verwendung von String

```
String s = "Wort";  
String t = "Wort";  
String u = new String("Wort");  
String v = new String("Wort");
```

```
System.out.println ("s == t: " + (s == t)); true  
System.out.println ("s == u: " + (s == u)); false  
System.out.println ("s.equals(u): " + s.equals(u)); true  
System.out.println ("u == v: " + (u == v)); false  
System.out.println ("u.equals(v): " + u.equals(v)); true
```

```
System.out.println ("Zeichen in " + u +  
                    " an Index 2: " + u.charAt(2)); r
```

```
System.out.println ("Laenge von " + u + ": " + u.length()); 4
```

```
System.out.println ("Zeichen in " + u +  
                    " an Index 2: " + u.toCharArray() [2]); r
```