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)`**
- **Konstruktoren:**
  - **`Integer (int value)`**,
  - **`Integer (String s)`**
- **Statische Methoden:**
  - **`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 = new Integer (123);
Integer y = new Integer ("123");
int z = Integer.parseInt("123");

String s1 = Integer.toString (123);
String s2 = x.toString ();

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

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

123
123
123
123
123

denn x und y zeigen nicht auf gleiche Objekt

false,

true

true

Autoboxing und Unboxing

automatisch von int nach Integer

automatisch von Integer nach int

```
Integer x = 123;

int i = x;

Integer y = x + 2;

Double z = Math.sqrt(y);

Double d = 4;

Double d = new Double(4);

Double e = 4.0;
```

```
Integer x = new Integer(123);

int i = x.intValue();

Integer y = new Integer(x.intValue() + 2);

Double z = new Double(Math.sqrt(y.intValue()));

Double d = new Integer(4);

Double e = new Double(4.0);
```

Argument: int, eigentlich double

Resultat: double

hat Typ int. Kann nach Integer überführt werden, aber keine Konversion von Integer nach Double

Typfehler!

automat. Konversion von int nach double

Typ double, kann nach Double überführt werden

Autoboxing und überladene Methoden

```
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;}
```

vararg Methoden möglichst nicht überladen!

Autoboxing + Unboxing erst dann, wenn sonst keine Methode passt. Wenn auch das nicht klappt, dann varargs ...

```
f (new Integer(1))  = 2
f (1)               = 3
```

Fehler!

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));
System.out.println ("s == u: " + (s == u));
System.out.println ("s.equals(u): " + s.equals(u));
System.out.println ("u == v: " + (u == v));
System.out.println ("u.equals(v): " + u.equals(v));

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

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

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

← Kurzschreibweise zur Erzeugung von String-Objekten

← true

← false

← true

← false

← true

← r

← 4

← r