universität innsbruck

Wrapper-Klassen

Programmiermethodik

Lukas Kaltenbrunner, Simon Priller
Universität Innsbruck

Motivation

- Kapselt die primitive Variable in einer objektorientierten Hülle.
  - Methoden für den Zugriff und die Umwandlung.
  - Java Datencontainer (Listen, Mengen, etc.) nehmen nur Referenzen auf.
  - Primitve Datentypen können nicht bei Generics verwendet werden (später mehr dazu).
- Zu jedem primitiven Datentyp in Java gibt es eine sogenannte **Wrapper-Klasse**.

Klasse	Primitiver Datentyp
`Byte`	`byte`
`Short`	`short`
`Integer`	`int`
`Long`	`long`
`Float`	`float`
`Double`	`double`
`Boolean`	`boolean`
`Character`	`char`

Handling Wrapper-Klassen (1)

- Erzeugung
  - Konstruktor mit primitivem Typ (deprecated)

```
Integer i = new Integer(10);
```

  - Konstruktor mit String (deprecated)

```
Integer i = new Integer("300");
```

  - Statische Methode (primitiver Typ → Cache für mindestens -128 bis 127)

```
Integer i = Integer.valueOf(10);
```

  - Autoboxing (verhält sich wie `valueOf()`)

```
Integer i = 3;
```

Handling Wrapper-Klassen (2)

- Rückgabemethoden
  - Rückgabe als primitiver Typ

```
Integer i = 3;
int j = i.intValue();
```

  - Rückgabe als String
- Alle Wrapper-Klassen sind unveränderlich (immutable).
  - Die Wrapper-Klassen sind nur als Ummantelung und nicht als vollständiger Datentyp gedacht.
  - "Werte-Objekte"
  - Änderung des Wertes führt zu neuem bzw. anderem Objekt.
- Alle numerischen Wrapper-Klassen erweitern die abstrakte Klasse `Number`.

Hilfsmethoden

- Wrapper-Klassen bieten nützliche (statische) Hilfsmethoden an.
- Beispiele:
  - Parsen von Strings (z.B. `Integer`)

    ```
    public static int parseInt(String s)
    ```

    - Versucht einen String in eine Ganzzahl umzuwandeln.
    - Wenn der übergebene String nicht einer Ganzzahl entspricht, dann wird das Programm abgebrochen (Ausnahme/Exception).
  - Berechnen des Hash-Werts (z.B. `Integer`)

    ```
    public static int hashCode(int value)
    ```
  - Vergleiche mit `compare` (z.B. `Double`)

    ```
    public static int compare(double d1, double d2)
    ```

```
double d1 = -0.0;
double d2 = 0.0;
System.out.println(d1 == d2);
System.out.println(Double.compare(d1, d2) == 0);
```

```
Ausgabe:
true
false
```

Konstanten

- `Byte`, `Short`, `Integer`, `Long` und `Character`:
  - `MIN_VALUE`
  - `MAX_VALUE`
  - …
- `Float` und `Double`:
  - `MIN_VALUE`
  - `MAX_VALUE`
  - `NEGATIVE_INFINITY`
  - `POSITIVE_INFINITY`
  - `NaN`
  - …

Autoboxing/Autounboxing (1)

- Automatische Konvertierung zwischen primitiven Datentypen und Wrapper-Klassen.
- Compiler fügt notwendigen Code automatisch ein.
- Zwei Richtungen
  - Umwandlung primitiver Wert → Wrapper-Objekt (Boxing)
  - Umwandlung Wrapper-Objekt → primitiver Wert (Unboxing)

```
Integer i = Integer.valueOf(5); // Boxing
int j = i.intValue();           // Unboxing
```

```
Integer i = 5; // Autoboxing
int j = i;     // Autounboxing
```

Autoboxing/Autounboxing (2)

- Primitive Typen können damit wie Referenztypen verwendet werden.
- Durch das Autounboxing können arithmetische und bitweise Operatoren auf Objekten von Wrapper-Klassen angewendet werden.
- Funktioniert nicht bei Arrays, nur bei primitiven Typen.
- Bei Autoboxing kann ein Wert eines primitiven Datentyps nur in die entsprechende Wrapper-Klasse umgewandelt werden.

```
Integer[] array1 = {1, 2, 3};
++array1[0];
int[] array2 = array1; // compiler error due to type missmatch

float fa = 123;
Float fb = 123;  // compiler error due to type missmatch
Float fc = 123f;
```

Autoboxing/Autounboxing (3)

- Autoboxing verschleiert Objektgenerierung.
  - z. B. wenn zwei Wrapper-Typen verglichen werden, wird ein Referenzvergleich durchgeführt (kein Autounboxing).
- Ganzzahl-Caching bei Autoboxing bzw. `valueOf`-Methode:
  - Alle Objekte im Bereich -128 bis +127 werden garantiert gecached.
  - In diesem Bereich erhält man immer die gleiche Referenz!

```
Integer i1 = 2;
Integer i2 = 2;
System.out.println(i1 == i2); // true

Integer j1 = 128;
Integer j2 = 128;
System.out.println(j1 == j2); // false (if cache size was not increased)

Integer k1 = new Integer(10);     // new object
Integer k2 = Integer.valueOf(10); // Boxing => Cache
Integer k3 = 10;                  // Autoboxing => Cache
System.out.println(k1 == k2); // false
System.out.println(k1 == k3); // false
System.out.println(k2 == k3); // true
```

Autoboxing/Autounboxing (4)

- Wird ein primitiver Typ mit einem Wrapper-Typ verglichen, findet immer Autounboxing statt.
- Bei Operationen, welche nicht für Referenztypen definiert sind, findet Autounboxing statt.

```
Integer k1 = new Integer(10);
Integer k2 = 10;
System.out.println(k1 == 10); // true
System.out.println(k2 == 10); // true

System.out.println(k1 >= k2); // true
System.out.println(k1 <= k2); // true
System.out.println(k1 == k2); // false
```

Quellen

- Christian Ullenboom: **Java ist auch eine Insel: Einführung, Ausbildung, Praxis**, Rheinwerk Verlag, 16. Auflage, 2022 (Java 17)
- Michael Inden: **Der Weg zum Java-Profi: Konzepte und Techniken für die professionelle Java-Entwicklung**, dpunkt.verlag, 5. Auflage, 2021