Inhaltsverzeichnis

1. (Primitive) Datentypen 1
2. Typumwandlungen 1
2.1. Implizite Typumwandlung 2
2.2. Explizite Typumwandlung (Casting) 3
3. Typumwandlung von komplexen Datentypen (Klassen) 3

1. (Primitive) Datentypen

Übersicht über die Datentypen in Java:

Typname	Größe[1]	Wrapper-Klasse	Wertebereich	Beschreibung
boolean	undefiniert[2]	java.lang.Boolean	true / false	Boolescher Wahrheitswert, Boolescher Typ[3]
char	16 bit	java.lang.Character	0 ... 65.535 (z. B. 'A')	Unicode-Zeichen (UTF-16)
byte	8 bit	java.lang.Byte	-128 ... 127	Zweierkomplement-Wert
short	16 bit	java.lang.Short	-32.768 ... 32.767	Zweierkomplement-Wert
int	32 bit	java.lang.Integer	-2.147.483.648 ... 2.147.483.647	Zweierkomplement-Wert
long	64 bit	java.lang.Long	-2^{63} bis $2^{63}-1$, ab Java 8 auch 0 bis $2^{64}-1$[4]	Zweierkomplement-Wert
float	32 bit	java.lang.Float	+/-1,4E-45 ... +/-3,4E+38	32-bit IEEE 754, es wird empfohlen, diesen Wert nicht für Programme zu verwenden, die sehr genau rechnen müssen.
double	64 bit	java.lang.Double	+/-4,9E-324 ... +/-1,7E+308	64-bit IEEE 754, doppelte Genauigkeit

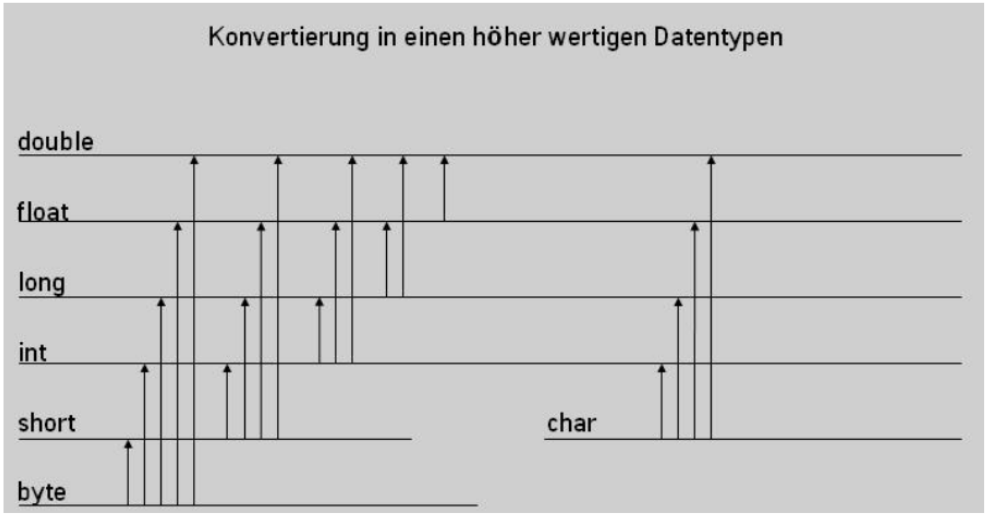
Quelle: → Wikibooks: Datatypes

Quelle: → Oracle: Datatypes

2. Typumwandlungen

Type-Casting mit primitiven Datentypen

Man unterscheidet zwischen einer **expliziten** und einer **impliziten** Typumwandlung.



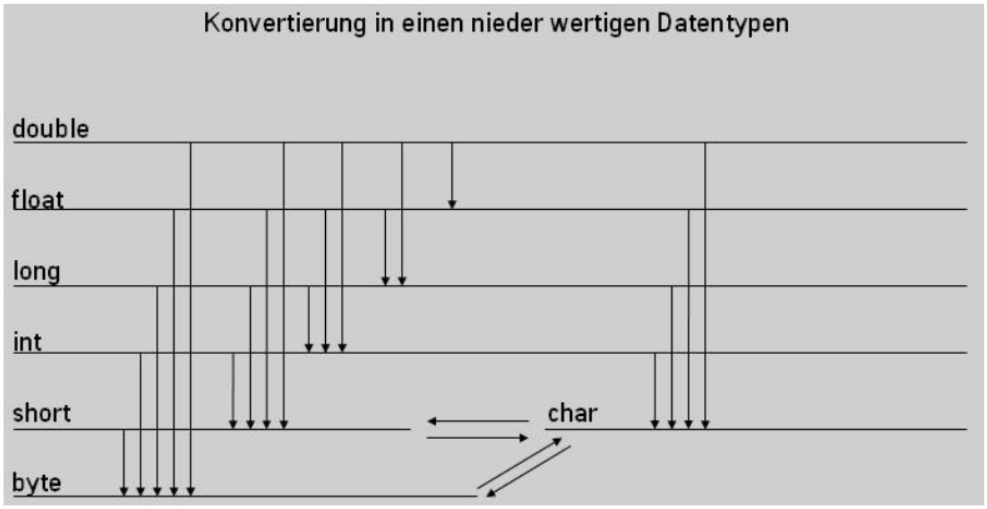


Figure 1. Widening & Narrowing

Bei der Umwandlung in "kleinere" Datentypen können Fehler auftreten (Informationsverlust), es findet das sogenannte "Narrowing" automatisch statt. Es entsteht zwar kein Compiler-Fehler, aber z.B. die Zahl wird verfälscht (→ Narrowing)

2.1. Implizite Typumwandlung

Die implizite Typumwandlung findet automatisch bei der Zuweisung statt. Dies geht jedoch nur, wenn ein niederwertiger Datentyp in einen höher wertigeren Datentypen umgewandelt wird, also z.B. vom Datentyp `int` in den Datentyp `long`:

```
int wert1 = 10;
long wert2 = 30;
wert2 = wert1; // automatische Umwandlung
```

2.2. Explizite Typumwandlung (Casting)

Die explizite Umwandlung erfolgt durch den sogenannten `cast`-Operator mit runden Klammern. Hier wird von einem höher wertigeren Datentyp in einen nieder wertigen Datentypen umgewandelt. In welchen Datentyp umgewandelt werden soll, muss bei dem cast Operator explizit angegeben werden.

```
int wert1 = 10;
float wert2 = 30.5f;
wert1 = (int) wert2; // Umwandlung per 'cast'
```

3. Typumwandlung von komplexen Datentypen (Klassen)

Auch **komplexe Datentypen** - also Instanzen von Java **Klassen** - können den Typen wechseln. Das ist natürlich ebenso klaren Regeln unterworfen.

So z.B. kann eine Klasse `Regionalzug` den Datentypen `Zug` bekommen, wenn die Klassen in einer Vererbungshierarchie stehen, ähnlich bei `Interfaces`. Das haben wir insb. schon im vorigen Modul `module-classes` gesehen und genutzt.

Das ist also insbesondere im Kontext der Vererbung interessant bzw. von Nutzen:

Beispiel:

```
public abstract class HumanBeing {
 public String name;
}

public class German extends HumanBeing {
  public String country;
}

HumanBeing human = new German();
human.name = "Johnny Walker"; ①

German german = (German)human; ②
german.country = "Germany";
```

① Zugriff auf das Feld ´name´ der abstrakten Klasse `Human Being`

② Zugriff auf das Feld `country` der konkreten Klasse `German` erst NACH dem **Down-Casting** möglich.

Demo/Beispiele:

```
→ src/test/java/de/dhbw/demo/DatatypesDemoTest.java
```

Übungen:

Übung 1 - Typumwandlung/Casting

```
→ src/test/java/de/dhbw/exercise/DatatypesExerciseTest.java
```

Schreibe je einen Test für

1. Gegeben `char c = '1'` → Umwandlung in `int`
2. Gegeben `int i = 127` → Umwandlung in `byte`

und prüfe das Ergebnis, also den erwarteten Wert, jeweils mithilfe der Assertions-Methode `assertEquals(<expected>, <actual>)`.