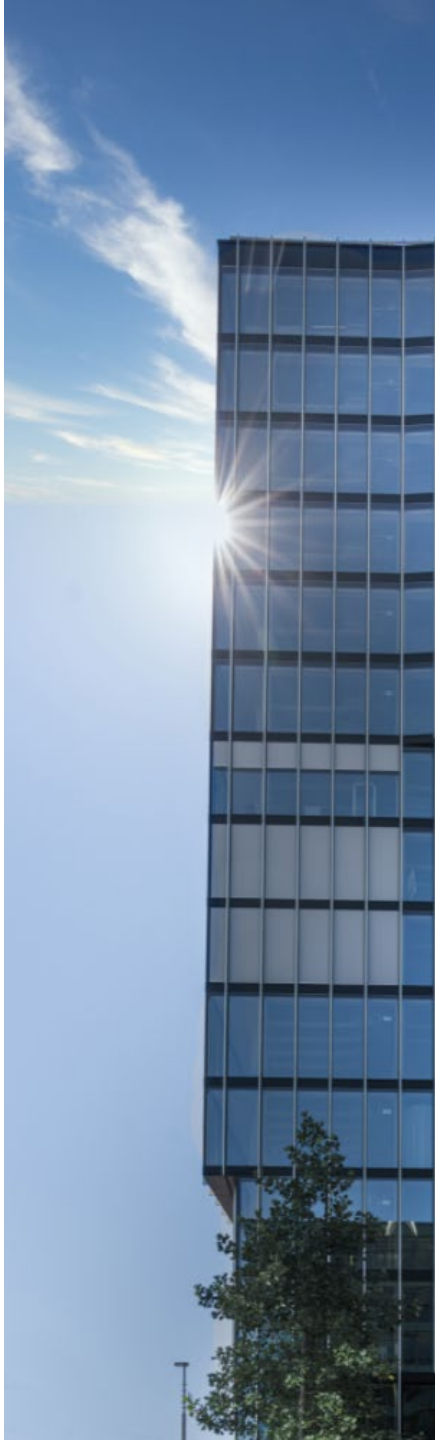
HSLU Hochschule Luzern

Objektorientierte Programmierung

Datentypen, Operatoren und Typumwandlungen

Roland Gisler

FH Zentralschweiz



Inhalt

- Was sind elementare Datentypen
- Übersicht: Acht elementare Datentypen in Java
- Wertebereich versus Genauigkeit
- Einfache Operatoren
- Implizite und explizite Typumwandlungen (Casting)

Lernziele

- Die elementaren Datentypen
  `boolean`, **`byte`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`char`**
  kennen.
- Sich dem Unterschied zwischen Wertebereich und Genauigkeit bewusst sein.
- Von Datentypen abhängige Wirkung von Operatoren kennen.
- Sie beherrschen einfache Typumwandlungen zwischen den Datentypen.

Was sind elementare Datentypen?

Elementare / Primitive Datentypen

Quelle: Ullenboom, "Java ist auch eine Insel"

- Ein elementarer Datentyp legt den Wertebereich, die Genauigkeit und die möglichen Operationen mit einer Variablen fest.
- Variablen eines elementaren Datentyps enthalten jeweils nur genau **einen einzigen Wert**, sie sind also **keine Objekte**!

Elementare Datentypen

- Der Computer speichert alle Werte (Zahlen, Zeichen etc.) als „Bitmuster" (Folge von **`1`** und **`0`**) ab.
- Mit einem Datentyp wird beschrieben, wie dieses Muster interpretiert wird und welcher **Wertebereich** damit möglich ist.
- Diese Darstellungen sind relativ speichereffizient, das heisst sie brauchen wenig Speicher.
- Jeder Datentyp legt implizit fest, welche Operationen möglich sind, bzw. wie diese ausgeführt werden. Beispiel für "**+**"-Operator:
  - Bei Zahlentypen werden die Zahlen addiert.
  - Bei einer Zeichenketten werden diese konkateniert.
- **Wichtig:** Datentypen legen implizit auch die **Genauigkeit** fest!

Verwendung von elementaren Datentypen

- Elementare Datentypen können an folgenden Stellen eingesetzt werden:
  - Lokale Variablen.
  - Parameter von Methoden und Konstruktoren.
  - Rückgabewerte von Methoden.
  - Attribute von Klassen.
- Vereinfacht: Überall dort, wo ein «Typ» (Klasse) verlangt ist, kann meist auch ein elementarer Datentyp verwendet werden.
- Warum ist nicht alles konsequent ein Objekt bzw. eine Klasse? Viele OO- Programmiersprachen brechen mit ihren elementaren Datentypen tatsächlich die Grundprinzipen der Objektorientierung zugunsten von Einfachheit und Effizienz.

Übersicht: Primitive Datentypen in Java

Übersicht: Primitive Datentypen für ganze Zahlen

Typ	Speicherbedarf	Wertebereich (inklusive)
`byte`	1 byte = 8 bits	**-128** bis **127**
`short`	2 bytes = 16 bits	**-32‘768** bis **32‘767**
`int`	4 bytes = 32 bits	**-2‘147‘483‘648** bis **2‘147‘483‘647**
`long`	8 bytes = 64 bits	**-9‘223‘372‘036‘854‘775‘808** bis **9‘223‘372‘036‘854‘775‘807**

Primitive Datentypen für ganze Zahlen

- Ohne explizite Angabe werden ganze Zahlen (z.B. `3671`) in Java per Default als `int` betrachtet.
  - Vorsicht bei Berechnungen mit Zwischenresultaten!
- Die Datentypen `byte` und `short` werden tendenziell eher selten verwendet, können aber bei grossen Datenmengen (Big Data) viel Platz sparen!
- Auch wenn wir heute meist über viel Speicher verfügen, lohnt sich dennoch eine seriöse Auswahl des geeigneten Datentyps bzw. Wertebereiches!
- Der gewählter Datentyp hat Einfluss auf: Ressourcenbedarf, Geschwindigkeit und Genauigkeit!

Primitive Datentypen für Gleitkommazahlen

Typ	Speicherbedarf	Wertebereich (inklusive)
`float`	4 bytes = 32 bits	± **3.40282347E+38**
`double`	8 bytes = 64 bits	± **1.79769313486231570E+308**

- Ohne explizite Angabe werden Gleitkommazahlen (z.B. `2.781`) in Java per Default als `double` betrachtet!
- Daumenregel für die **Genauigkeit** (relevante Stellen):
  - `float`: **~7** relevante Stellen
  - `double`: **~14** relevante Stellen

- **Vorsicht**: Sich nicht vom grossen Wertebereich täuschen lassen!

Restliche primitive Datentypen

Typ	Speicherbedarf	Wertebereich (inklusive)
`boolean`	1 byte = 8 bits (!)	`true`, `false`
`char`	2 bytes = 16 bits	UTF-16 Unicode Zeichen

- Beim Datentyp `boolean` wird zugunsten der Geschwindigkeit relativ viel Speicherplatz «verschenkt».
- Ein `char` enthält genau **ein** Zeichen und wird mit **einfachen** Anführungszeichen (`'`) eingefasst.
  - Beispiel: `char c = 'A';`
  - Nicht verwechseln mit `String`: Das ist eine Zeichen**kette.**

Wertebereich versus Genauigkeit

- Bei den Ganzzahltypen (z.B. `long`) kann jeder mögliche Wert des jeweiligen Wertebereiches präzise abgebildet werden.
- Bei den Gleitkommatypen ist das **nicht** der Fall! Hier ist mit dem Datentyp nicht nur ein Wertebereich, sondern auch eine Anzahl relevanter Stellen vorgegeben!
- Beispiel:
  Die Zahl `2'000'000.05` ist zwar vollständig im Wertebereich von `float`, kann aber **nicht** präzise abgebildet werden!
- Kommt es also auf Genauigkeit an (z.B. bei Geldbeträgen!), muss man bei der Auswahl des Datentyps **sehr sorgfältig** vorgehen!

Einfache Operatoren

Übersicht: Einfache Operatoren

- `+` Addition oder optionales Vorzeichen
- `-` Subtraktion oder Vorzeichen
- `*` Multiplikation
- `/` Division
- `=` Zuweisung* (erfolgt immer von rechts nach links ←)

*Hinweis: Wenn Sie einen Test auf «Gleichheit» machen wollen, müssen wir in Java ein Doppelgleich ("`==`") schreiben (mehr dazu in OOP04 und OOP05).

Anwendung von einfachen Operatoren

- Mit Hilfe von Operatoren kann man beliebige Ausdrücke formulieren und Werte zuweisen.
- Beispiel: Addition ganzer Zahlen

```
int summe = 128 + 132;
```

- Beispiel: Aneinanderhängen von Strings (Konkatenation)

```
String word = "Weihnacht" + "s" + "mann";
```

- Hinweis: `String` ist in Java **kein** elementarer Datentyp, sondern eine Klasse, von welcher Objekte erzeugt werden!
  - Leicht erkennbares Merkmal: Klassennamen beginnen **immer** mit einem Grossbuchstaben!

Operatoren sind Polymorph

- Ein Operator hat je nach vorhandenen Datentypen eine unterschiedliche Bedeutung/Wirkung (Polymorph):
- Beispiel: "**+**"-Operator
  - als Vorzeichen für positive Werte: `+100`
  - als Additionsoperator: `100 + 200`
  - zur Konkatenation von Strings: `"abc" + "def"`
- Die Interpretation hängt also von der Position im Ausdruck und vom Datentyp ab auf den er angewendet wird ab:
  - vor einer Zahl: Vorzeichen
  - zwischen zwei Zahlen: Mathematische Addition
  - zwischen zwei Zeichenketten: Konkatenation (Verbindung)

Typumwandlungen

Implizite und explizite Typumwandlungen

- Typumwandlung: Konvertierung eines Wertes von einem Typ zu einem Anderen, auch als «casting» bezeichnet.
- Beispiele:
  - **`short`**-Wert (z.B. **`100`**) in **`long`**-Wert (auch **`100`**) umwandeln.
  - **`float`**-Wert (z.B. **`2.178f`**) in **`double`**-Wert umwandeln.
  - **`boolean`**-Wert (z.B. **`true`**) in String umwandeln: **`"true"`**.
- Implizite Typumwandlung:
  Java konvertiert den Wert direkt und automatisch bei einer normalen Zuweisung, z.B. **`long wert = 100;`**
- Explizite Typumwandlung (cast):
  Wir geben Java explizit den Befehl, in welchen Typ etwas konvertiert werden soll, z.B. **`long wert = (long) 100;`**

Regeln für implizites (automatisches) Casting

Die folgenden Castings (von ➔ nach) sind implizit möglich:

von	nach
`byte`	`short`, `int`, `long`, `float`, `double`
`char`, `short`	`int`, `long`, `float`, `double`
`int`	`long`, `float`, `double`
`long`	`float`, `double`
`float`	`double`
„alle“ (x)	nach **`String`** bei Konkatenation mit einem String Beispiel: `String s = s + x;`

Achtung: Einzelne Castings können problematisch sein, da es dabei zu Genauigkeitsverlusten kommen kann!

Beispiel: Division ganzer Zahlen

- Division von zwei ganzen Zahlen:

```
int i1 = 5;
int i2 = 2;
float f = i1 / i2;
```

- Das korrekte Ergebnis wäre `2.5` (und somit eine Gleitkommazahl), weshalb als Typ für das Resultat auch **`float`** verwendet wird.
- **Aber**: Das Resultat dieser Berechnung ist `2.0f`! **Warum?**

Beispiel: Division ganzer Zahlen - Erklärung

```
int i1 = 5;
int i2 = 2;
float f = i1 / i2;
```

- Zuerst wird die Division ausgeführt.
  - Ganzzahldivision (weil zwei **`int`**) ➔ Resultat = **`2`**
- Dann wird für die Zuweisung ein impliziter Cast auf **`float`** durchgeführt.
  - Cast auf **`float`** ➔ Resultat **`2.0f`**
- Korrekte Implementation (Varianten):

```
float f = (float) i1 / i2
```

oder:

```
float f = i1 / (float) i2;
```

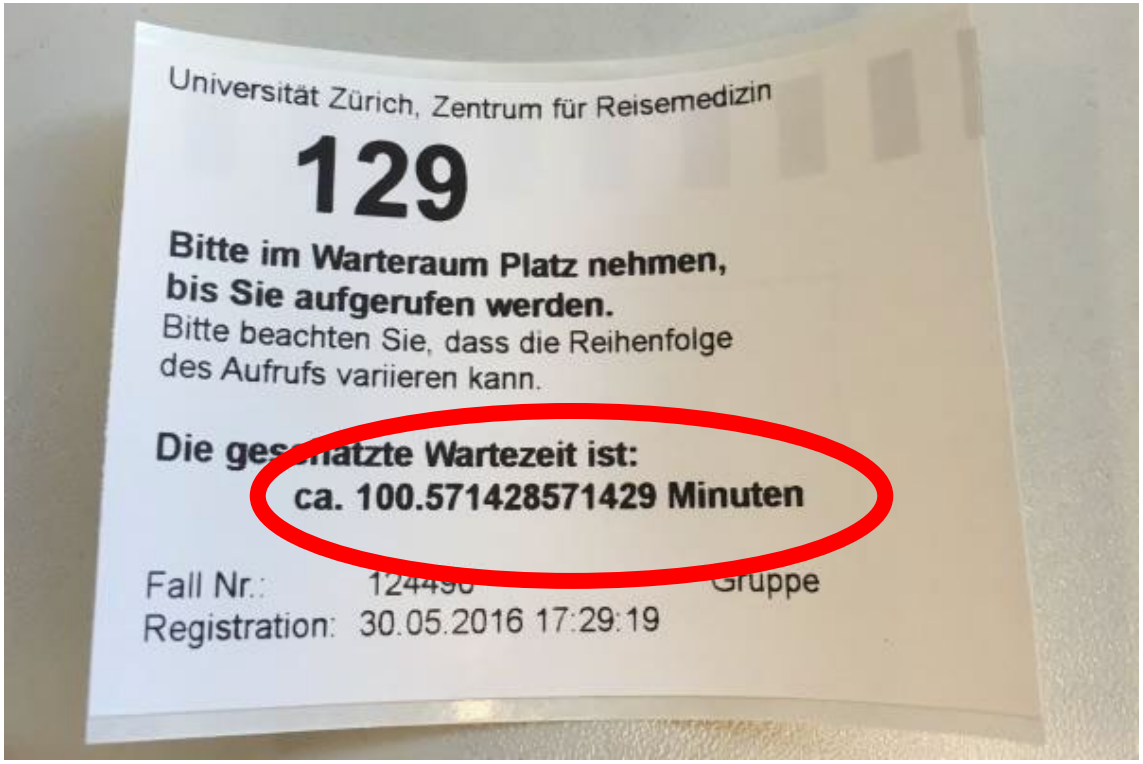
Empfehlungen zu Datentypen und Castings

- Relevanz der Typenwahl und Typumwandlung (speziell bei mathematischen Berechnungen) wird häufig unterschätzt!
- Bei impliziten Castings:
  Vorsicht vor unerwarteten Genauigkeitsverlusten!
- Bei expliziten Castings: Vorsicht vor unerwarteten Resultaten, Java macht was man ihm befiehlt, und das ist nicht unbedingt das, was man wirklich will!
  - Fliesskomma nach Ganzzahl: Dezimalstellen werden abgeschnitten, **keine** Rundung!
  - Zu grosser Wert in zu kleine Datentypen: Bits werden einfach abgeschnitten, häufig kein sinnvolles Resultat mehr!

➔Berechnungen **immer** gut mit sinnvollen Wertebereichen testen!

Datentypen – ein Praxisfall!

- Was für ein Datentyp wurde wohl hier verwendet?



Quelle: http://www.20min.ch/schweiz/zuerich/story/22403547

Zusammenfassung

- Elementare / Primitive Datentypen: Für «reine» Werte.
  - Keine Objekte, keine Methoden ➔ eigentlich nicht OO!
- Effiziente, platzsparende Speicherung, auch für grosse Datenmengen.
- Datentypen immer bewusst auswählen:
  Wertebereiche **und** Genauigkeit beachten.
- Wirkung von Operatoren ist abhängig vom Position und Datentyp.
- Typumwandlungen sind explizit und implizit möglich.
  - Gefahr: Unerwarteter Genauigkeitsverlust.
- Berechnungen immer gut mit sinnvollen Wertebereichen testen.

Fragen?