II.4.5 Generische Datentypen

Mittwoch, 28. November 2018 09:30

Nachteile bei Listen mit Werten vom Typ Object:

1. Man kann keine Anforderungen an Klassen formulieren. Werte sollten von solchen Typen sein, bei denen es eine "gleich"-Methode gibt.

Abhilfe: Verwende abstr. Klasse/Interface "Vergleichbar" mit abstr. Methode "gleich".

2. Listen können versch. Objekte durcheinander enthalten (z.B. Brüche und Worte).

3. Wenn man Werte aus einer Liste weiter bearbeiten will, muss man mit expliziter Typumwandlung casten.

Abhilfe: Verwende gene-

viele Typen

Eine Klasse Element<T> definiert viele Typen:

Element<Bruch>
Element<String> ...

Typvariable wird durch beliebige konkrete Typen instantiiert.

Parametrischer Polymorphismus:

die selbe Implementierung einer Methode wird für Argumente verschiedener Typen T ausgeführt.

(typisch für fkt. Programmiersprachen, aber auch in Java)

Ad hoc - Polymorphismus:

Je nach Typ wird unterschiedliche Implementierung der Methode ausgeführt

(Überschreiben v. Methoden)

(typisch für OO-Sprachen, aber auch in Haskell)

- "Diamond"-Schreibweise
  neue Liste <> ( )
  ist möglich, weil aufgrund des Typs der Variable l klar ist, dass der Typparameter T mit Bound instantiiert werden muss.
- Zur Compilezeit wird (statisch) sichergestellt, dass nur Werte des gleichen Typs in der Liste sind.
- Klassen können mehrere Typvar. haben
- Auch Interfaces können

generisch sein (d.h. Typ-
variablen besitzen)

- Generische Typen existie-
ren nur zur Compilezeit.
Danach werden alle
generischen Typen wie
Liste<T> durch ihren
RawType Liste
ersetzt (mit Object statt
T).
⇒ instanceof Liste<Bruch>
nicht möglich, aber
instanceof Liste
ist möglich.

- Die Typvar. einer gene-
rischen Klasse kann nicht
in statischen Methoden
benutzt werden.

Stat. Methode existiert "nur einmal" in der Klasse.

Abhilfe: Statische Methoden können eigene Typvar. haben und damit generisch sein. Instant. der Typvar. der statischen Methode ist unabhängig v. Instant. der Typvar. der Klasse.

- Bisher: Typvar. können mit beliebigen Typen instantiiert werden.

Jetzt: Kombiniere param. Polymorphismus (Typvariable) mit Typhierarchie

- <T extends Vergleichbar> bedeutet: T darf nur mit Vergleichbar oder Untertypen

davon instantiiert werden.

"Typebounds"

Es ex. auch Typebounds "nach oben".

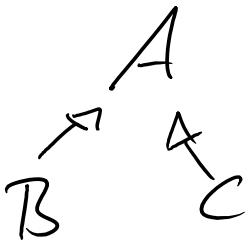
Es ex. auch Wildcards als Typparameter:

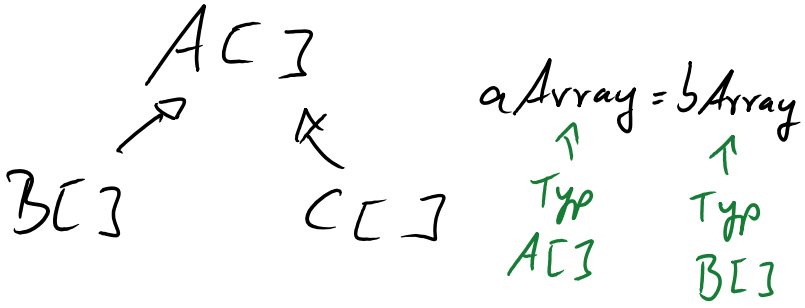
Liste <?>

Liste <? extends Vergleichbar>

⋮

Ober- und Unterklassenverhältnis bei generischen Typen





```
aArray[0] = new C();
```
↑ Typ A ↑ Typ C

bArray[0] ist vom Typ C, obwohl bArray vom Typ B[] ist.

Fehler beim Design von Java:
Statisches Typ-Checking zur Compile-Zeit sollte sicherstellen, dass zur Laufzeit keine Typ-fehler auftreten können.
Dies ließ sich nicht mehr korrigieren, da alter Java-Code weiterhin lauffähig sein soll.

Bei Einführung der Generics wurde der Fehler nicht wiederholt:

Liste<B> ist keine Unterklasse

Liste<C>' von Liste<A>

Aufruf statischer Methoden über den Raw Type:

Liste.suche(...), nicht

Liste<Bruch>.suche(...)

JBC bestellt aus class-Files. Dat existieren zwar keine generischen Klassen mehr (nur Raw Types), aber als Meta-Information sind die Typen in den Klassen- u. Methodenköpfen noch gespeichert. ⇒ nötig für Typ-Checking bei der Compilierung