II.4.5 Generische Datentypen

Freitag, 8. Dezember 2017 09:00

Nachteile der allgemeinen Liste mit Werten v. Typ Object:

- Man kann keine Anforderungen an die Klassen stellen, aus denen die Werte kommen.
  Abhilfe: Verwende abstrakte Klassen / Interfaces. Statt Object kann man VergleichBar benutzen, mit abstrakter Methode "gleich".
- Listen können bel. Objekte durcheinander enthalten.
- Wenn man Werte aus der Liste holt, haben sie den Typ Object. ⇒ Um sie weiter zu verarbeiten, benötigt man oft eine explizite Typumwandlung zur Unterklasse.

Abhilfe für die letzten 2 Nachteile: Generische Typen

Idee: Eine Klassendeklaration definiert viele Typen, indem die Typvariable T durch bel. Typen ersetzt wird.

Bsp: Hier definieren wir die Typen
Element<Bruch>, Liste<Bruch>
Element<String>, Liste<String>, ...

Parametrischer Polymorphismus

Eine Implementierung für verschiedene Instantiierungen des Typparameters. (Typisch für funktionale Prog.spr., gibt es aber inzwischen auch in Java.)

Ad-Hoc Polymorphismus

Verschiedene Implementierungen für Argumente/Objekte versch.

Typen (Typisch für OO-Sprachen)

- Beim Aufruf eines Konstruktors eines generischen Typs darf man <> schreiben, wenn die Instantiierung des Typparameters aus dem Typ der Var. folgt.
- Durch Typ-Checking zur Compile-Zeit wird jetzt sichergestellt, dass eine Liste vom Typ Liste<Bruch> nur Werte v. Typ Bruch enthält.
- Auch Interfaces können generisch sein (d.h. sie können Typvariablen haben).
- Damit alter Java-Bytecode weiterhin läuft, enthält JBC keine generischen Typen.
  ⇒ Gener. Typen werden bei der Compilierung überprüft und anschließend in den entsprechenden "raw type" ohne Typparameter übersetzt. ← instanceof Liste<Bruch> nicht möglich
  instanceof Liste möglich
- Statische Methoden in gener. Klassen:
  - Methode hängt nicht von konkretem Objekt ab.
    ⇒ Typ-Parameter können nicht "durch Objekt" instantiiert werden
  - Methode selbst kann aber als generisch vereinbart werden. Instantiierung / Name des Typ-Parameters der stat. Methode ist unabh. v. Typ-Param. der Klasse.
- Konzepte Typhierarchie + param. Polymorphismus können kombiniert werden:
  Man kann fordern, dass Typvar. T nur durch Typen instantiiert werden darf, die das Interface Vergleichbar implementieren.

<T extends Vergleichbar>

↑ T darf nur mit "Vergleichbar" oder seinen Untertypen instantiiert werden.

- Weitere Möglichkeiten:

  <?> Wildcard, steht für bel. Typ

  Wildcards können durch "extends" oder "super" in der Typhierarchie nach oben oder unten eingeschränkt werden.

...

Ober- und Unterklassen bei generischen Typen

- B ist U-Klasse von A, C ist U-Klasse von A

B[] ist U-Klasse von A[]

aArray = bArray erlaubt

Typ A[] Typ B[]

aArray[0] = new C() erlaubt

Typ A Typ C

Design-Fehler von Java. Typ-Fehler sollten zur Compile-Zeit gefunden werden, nicht erst zur Laufzeit.

Folge: bArray[0] sollte den Typ B haben, hat aber Typ C.

Bei gener. Typen hat man den Fehler nicht wiederholt.

Wenn B U-Klasse v. A ist,

folgt daraus nicht, dass Liste<B> U-Klasse v. Liste<A> ist.