1. Alumni–Treffen
der Berufsakademie Stuttgart, Außenstelle Horb

Type–Casts ade, Scheiden tut nicht weh!

Generische Klassen in Java 1.5

Martin Plümicke

23. Juli 2004

Überblick

1. Grundsätzliche Probleme der Software–Entwicklung
2. Boxed/Unboxed–Types
3. Generische Typen in Java
4. Typberechnungen
5. Zusammenfassung, Fazit und Ausblick

1. Grundsätzliche Probleme der Software–Entwicklung

1. Kommunikation **Auftraggeber** vs. **Software–Entwickler**
   Ist die zu erstellende Software richtig spezifiziert?
2. **Software–Entwickler** macht **Implementierungsfehler**
   Entspricht die entwickelte Software der Spezifikation?

Interessante Fragestellungen des Software–Engineerings:

Ansätze:

- Modellierungstechniken
- Vorgehensmethoden
- Programmiersprachen–Design

2. Boxed/Unboxed–Types

Umwandlung von Basis–Typen in Objekte: `int` nach `Integer`

```
int i = 16;
Integer I = new Integer(i)
I.equals(new Integer(17));
```

Neu: ohne explizite Umwandlung

```
int i = 16;
Integer I = new Integer(i)
I.equals(17);
```

Umwandlung von Objekten in Basis–Typen: `Integer` nach `int`

```
Integer I = new Integer(22);
int erg = I.intValue() + 21;
```

Neu: ohne explizite Umwandlung

```
Integer I = new Integer(22);
int erg = I + 21;
```

3. Generische Typen in Java

Erweitertes Typsystem von Java 1.5

- Typvariablen
- parametrisierte Typen

Bsp.:

```
Seq<A>
Vector<A>
Pair<A,B>
```

Beispiel: parametrisierter `Vector`

```
//Java 1.5                          //Java

class Vector<A> {                   //class Vector {
                                    //
    void add (A elem) { ... }       //    void add (Object elem) { ... }
    A get () { ... }                //    Object get () { ... }
}                                   //}
```

Deklaration eines `Vectors`:

```
Vector<Integer> v = new Vector<Integer>();
```

Bedeutung: Der Vector `v` enthält nur Objekte des Typs `Integer`.

Verbesserung

1. **Vermeidung von Type–Casts:**

   Statt in Java

   ```
   Integer i = (Integer)v.get(6);
   ```

   genügt in Java 1.5

   ```
   Integer i = v.get(6);
   ```

2. **Laufzeitfehler werden bereits zur Compilationszeit festgestellt:**

```
//Java                                  //Java 1.5
Vector                                  //Vector<Integer>
    v = new Vector();                   //    v = new Vector<Integer>();
v.add(new Integer(27));                 //v.add(new Integer(27));

((String)v.get(0)) ++ "Hallo";          //((String)v.get(0)) ++ "Hallo";
```

In Java hat `v.get(0)` zur Compilationszeit den Typ `Object`
und zur Laufzeit den Typ `Integer`
⇒ Type–Cast `((String)v.get(0))` führt zu Laufzeitfehler

In Java 1.5 hat `v.get(0)` bereits zur Compilationszeit den Typ `Integer`
⇒ Type–Cast `((String)v.get(0))` führt zu Compilationsfehler.

4. Anwendungsbeispiel Bevölkerungsentwicklung in (Old) Java

```
class Pair {
    Object x;
    Object y;

    Pair(Object x, Object y) {
        this.x = x;
        this.y = y;
    }

    Object fst() {
        return x;
    }

    Object snd() {
        return y;
    }
}
```

```
class Bevoelkerungsentwicklung {
  Vector Jahr_Stadtteil_Kinder_Erw;

  Bevoelkerungsentwicklung() {
    Jahr_Stadtteil_Kinder_Erw = new Vector();
    for (int i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
      Vector Stadtteil_Kinder_Erw = new Vector();
      for (int j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
        long Kinder = select from Database
        long Erw = select from Database
        Pair Kinder_Erw = new Pair(new Long(Kinder), new Long(Erw));
        Stadtteil_Kinder_Erw.addElement(Kinder_Erw); }
      Jahr_Stadtteil_Kinder_Erw.addElement(Stadtteil_Kinder_Erw); } }

public static void main(String[] args) {
  Bevoelkerungsentwicklung be = new Bevoelkerungsentwicklung();
  for (int i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
    for (int j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
      System.out.print("Jahr " + i + ": Stadtteil " + j + ": ");
      System.out.print("Kinder: " +
        ((Pair)((Vector)be.Jahr_Stadtteil_Kinder_Erw.get(i)).get(j)).fst() + " ");
      System.out.println("Erwachsene: " +
        ((Pair)((Vector)be.Jahr_Stadtteil_Kinder_Erw.get(i)).get(j)).snd());
      } } } }
```

Bevölkerungsentwicklung in New Java

```
class Pair<a,b> {
    a x;
    b y;

    Pair(a x, b y) {
        this.x = x;
        this.y = y;
    }

    a fst() {
        return x;
    }

    b snd() {
        return y;
    }
}
```

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Bevoelkerungsentwicklung() {
    Jahr_Stadtteil_Kinder_Erw = new Vector<Vector<Pair<Long,Long>>>();
    for (int i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
      Vector<Pair<Long,Long>> Stadtteil_Kinder_Erw = new Vector<Pair<Long,Long>>();
      for (int j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
        long Kinder = select from Database
        long Erw = select from Database
        Pair<Long,Long> Kinder_Erw = new Pair<Long,Long>(Kinder, Erw);
        Stadtteil_Kinder_Erw.addElement(Kinder_Erw); }
    Jahr_Stadtteil_Kinder_Erw.addElement(Stadtteil_Kinder_Erw); }}

public static void main(String[] args) {
  Bevoelkerungsentwicklung be = new Bevoelkerungsentwicklung();
  for (int i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
    for (int j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
      System.out.print("Jahr " + i + ": Stadtteil " + j + ": ");
      System.out.print("Kinder: " +
         be.Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst() + " ");
      System.out.println("Erwachsene: " +
         be.Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd());
    }}}}
```

3. Typberechnungen

```
class Bevoelkerungsentwicklung {

  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Bevoelkerungsentwicklung() { ... }

  Vector<Vector<Double>> Kinder_Quote () {
    Vector<Vector<Double>> Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
    for (int i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
      Vector<Double> Stadtteil_KQ = new Vector<Double>();
      for (int j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
        double kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
            /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
            + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
        Stadtteil_KQ.addElement(kq);
      }
      Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
    }
    return Jahr_Stadtteil_KQ;
  }
}
```

Ziel

```
class Bevoelkerungsentwicklung {

  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
    Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
    for (i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
       Stadtteil_KQ = new Vector<Double>();
       for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
          kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
              /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
              + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
          Stadtteil_KQ.addElement(kq);
       }
       Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
    }
    return Jahr_Stadtteil_KQ;
  }
}
```

Typ–Inferenz–Algorithmus

Idee: Typberechnung einer Methode

$$\texttt{Rtyp name (ty}_1\ p_1, \ldots, \texttt{ty}_n\ p_n)\ \{\ stmts\ \}.$$

1. Man ermittelt alle bekannten Typen (Typen der Attribute)
2. Man belegt die gesuchten Typen zunächst mit Typvariablen:
   - $p_1 : \alpha_1$
   - $p_2 : \alpha_2$
   - ...
   - $p_n : \alpha_n$
   - $\texttt{name} : \alpha_1 \times \ldots \times \alpha_n \rightarrow \alpha_R$
3. Man fügt diese Typ–Annahmen zu den Typ–Annahmen der bekannten Methoden hinzu.
4. Man geht durch den abstrakten Syntaxbaum der Klasse durch und unifiziert die Typ–Annahmen miteinander.
5. Als Ergebnis erhält man (mehrere) Unifikatoren. Diese wendet man auf die angenommenen Typvariablen an und erhält so, (mehrere) Typen der Methode.

Typ–Annahmen für das Beispiel

$$O_{\texttt{Vector<a>}} = \{\, \texttt{addElement} : \texttt{a} \rightarrow \texttt{void} \,\}$$

und

$$V_1 = \{\, \texttt{Kinder\_Quote} :\rightarrow \alpha_1,$$
$$\texttt{Jahr\_Stadtteil\_Kinder\_Erw} : \texttt{Vector<Vector<Pair<Long,Long>>>} \,\}.$$

Der Typ für α_1 wird nun Schritt für Schritt hergeleitet.

– $V_1 = \{$ Kinder_Quote $:\longrightarrow \alpha_1$,

Jahr_Stadtteil_Kinder_Erw : Vector<Vector<Pair<Long,Long>>>,

Jahr_Stadtteil_KQ : Vector<Vector<Double>> $\}$.

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
    Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
    for (i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
       Stadtteil_KQ = new Vector<Double>();
       for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
          kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
              /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
              + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
          Stadtteil_KQ.addElement(kq);
       }
       Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
    }
    return Jahr_Stadtteil_KQ;
  }
}
```

– $V_1 = \{$ Kinder_Quote $:\longrightarrow \alpha_1$,

Jahr_Stadtteil_Kinder_Erw : Vector<Vector<Pair<Long,Long>>>,

Jahr_Stadtteil_KQ : Vector<Vector<Double>>,

i : int $\}$.

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
    Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
    for (i = 0; i < 9; i++) {// Jahr 0 bis Jahr 9
       Stadtteil_KQ = new Vector<Double>();
       for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
          kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
              /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
              + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
          Stadtteil_KQ.addElement(kq);
       }
       Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
    }
    return Jahr_Stadtteil_KQ;
  }
}
```

- $V_1 = \{$ Kinder_Quote $:\rightarrow \alpha_1$,
  Jahr_Stadtteil_Kinder_Erw : Vector<Vector<Pair<Long,Long>>>,
  Jahr_Stadtteil_KQ : Vector<Vector<Double>>,
  i : int,
  Stadtteil_KQ : Vector<Double> $\}$.

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
    Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
    for (i = 0 i < 9; i++) {// Jahr 0 bis Jahr 9
       Stadtteil_KQ = new Vector<Double>();
       for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
          kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
              /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
              + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
          Stadtteil_KQ.addElement(kq);
       }
       Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
    }
    return Jahr_Stadtteil_KQ;
  }
}
```

- $V_1 = \{$ Kinder_Quote $:\longrightarrow \alpha_1$,
  Jahr_Stadtteil_Kinder_Erw : Vector<Vector<Pair<Long,Long>>>,
  Jahr_Stadtteil_KQ : Vector<Vector<Double>>,
  Stadtteil_KQ : Vector<Double>,
  i : int, j : int, kq : double $\}$.

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
     Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
     for (i = 0 i < 9; i++) {// Jahr 0 bis Jahr 9
        Stadtteil_KQ = new Vector<Double>();
        for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
           kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
               /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
               + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
           Stadtteil_KQ.addElement(kq);
        }
        Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
     }
     return Jahr_Stadtteil_KQ;
  }
}
```

– $V_1 = \{$ Kinder_Quote :⟶ Vector<Vector<Double>>,
Jahr_Stadtteil_Kinder_Erw : Vector<Vector<Pair<Long,Long>>>,
Jahr_Stadtteil_KQ : Vector<Vector<Double>>,
Stadtteil_KQ : Vector<Double>,
i : int, j : int, kq : double $\}$.

```
class Bevoelkerungsentwicklung {
  Vector<Vector<Pair<Long,Long>>> Jahr_Stadtteil_Kinder_Erw;

  Kinder_Quote () {
     Jahr_Stadtteil_KQ = new Vector<Vector<Double>>();
     for (i = 0 i < 9; i++) {// Jahr 0 bis Jahr 9
        Stadtteil_KQ = new Vector<Double>();
        for (j = 0; j < 4; j++) {//Stadtteil 0 bis Stadtteil 4
           kq = Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue()
               /  (Jahr_Stadtteil_Kinder_Erw.get(i).get(j).snd().doubleValue()
               + Jahr_Stadtteil_Kinder_Erw.get(i).get(j).fst().doubleValue());
           Stadtteil_KQ.addElement(kq);
        }
        Jahr_Stadtteil_KQ.addElement(Stadtteil_KQ);
     }
     return Jahr_Stadtteil_KQ;
  }
}
```

Der Algorithmus

- beruht auf vielfacher Anwendung der klassischen Unifikation
  (klassische Unifikation lässt sich quasi–linear implementieren)
- es gibt mehrere Unifikatoren ⇒ eine Methode kann mehrere Typen haben

  (wurde als Studienarbeit implementiert und bei UNIF'04 in Cork vorgetragen)
- sehr komplex (np–vollständig)

Geschichte von generischen Typen in Java

- Pizza als Erweiterung von Java wird in Januar 1997 u.a. in Schwarzenberg vorgestellt. (Entwicklung an der Uni Karlsruhe)
- Generic–Java (GJ) als *Tochter* von Pizza wird im März 1998 vorgestellt.
- Im Mai 1999 kündigt Sun an, generische Typen ins Java aufzunehmen.
- Im Mai 2001: Sun Prototype mit generischen Typen.
- Im Januar 2003 werden generische Typen für Java 1.5 angekündigt.
- Im Mai 2004 wird eine Beta–Version zur Verfügung gestellt.
- Ab Herbst 2004 werden generische Typen in den Lehrveranstaltungen an der BA in Horb eingesetzt.

5. Zusammenfassung, Fazit und Ausblick

Zusammenfassung

1. Java 1.5: Die Basis–Typen können in der boxed und unboxed Darstellung weitgehend automatisch konvertiert werden.
2. Java 1.5 enthält Typvariablen und generische Typen.
3. Es sind erheblich weniger Type–Casts nötig.
4. Manche Laufzeitfehler werden zu Compilationsfehler.
5. Es gibt einen Typinferenz–Algorithmus.

Bewertung von generischen Typen in **Java**

- Es werden Type–Cast Fehler bereits zur Compilationszeit gefunden.
- Erfahrung mit *spezifischen* Typsystemen: Wenn der Typ eines Programms korrekt ist, entspricht es oft auch seiner Spezifikation.
- Typberechnung dient der Bequemlichkeit ohne die posistiven Eigenschaften des spezifischen Typsystems zu verlieren.

Ausblick

- Implementierung des Typinferenz–Algorithmus.
- Einbau des Systems in Programmierumgebungen wie Eclipse oder Netbeans.