

Errata zum Buch "Fortgeschrittene Programmierung mit Java 5"

Autor: Johannes Nowak
dpunkt.verlag, Heidelberg
ISBN 3-89864-306-9

Stand: 12. Januar 2006

Die meisten der im Folgenden beschriebenen Fehler wurden von einem äußerst aufmerksamen Leser entdeckt: Herrn Dominik Gruntz. Die Fehler sind großenteils "Flüchtigkeitsfehler" – aber es haben sich auch einige "richtige", harte Fehler eingeschlichen. Vielen Dank, Herr Gruntz.

Dank auch an Herrn Ulrich Grude, der mir einige Anmerkungen zu den C++-Templates hat zukommen lassen.

Im Folgenden beziehen sich negative Zeilenzahlen auf das Ende der Seite (vom Ende an aufwärts).

Seite 1, Zeile -7, und Seite 12, unteres Drittel

```
enum Season { SPRING, SUMMER, AUTUMN, WINTER; }
```

Nach `winter` sollte kein Semikolon stehen.

Seite 11, Zeile -14 bis -11

```
public static final Season1 SPRING = new Season(0);
```

`SPRING` etc. ist nicht vom Typ `season1` (den gibt's gar nicht – den gab's aber mal in einer Testversion), sondern vom Typ `season`.

Seite 13

Hier sollte noch die statische Methode `valueOf` erwähnt werden, die der Compiler ebenfalls generiert. Sie kann wie folgt angewendet werden:

```
enum Season { SPRINT, SUMMER, ... }  
  
System.out.println (Season.SPRING == Season.valueOf ("SPRING"));
```

Es wird `true` ausgegeben.

Seite 14, Zeile -5 bis -1

Hier sollte erwähnt werden, dass die beiden folgenden Zeilen denselben Output erzeugen:

```
System.out.println(season);  
System.out.println(season.getName());
```

Seite 18, Zeile 13

Es heißt natürlich nicht:

```
(new int[] { 1, 2 } )
```

sondern:

```
(new int[] { 1, 2 })
```

Seite 45, Zeile 13

Der Compiler generiert nicht:

```
A m (Object t)
```

sondern:

```
A m (A t)
```

Seite 47, Zeile -9

Statt:

Sowohl in **B** als auch in **A** ist ...

muss es heißen:

Sowohl in **B** als auch in **C** ist ...

Seite 50, Zeile 14

Statt:

```
this.m((A)obj;
```

muss es heißen:

```
this.m((A)obj);
```

Seite 51, Zeile 8

Statt:

```
public void C.m(java.lang.Object)
```

muss es heißen:

```
public java.lang.Object C.m(java.lang.Object)
```

Seite 57

Die Zeile

```
D<? super C> <-- D<? super B> <-- D<? super A> <-- D<?>
```

enthält einen Fehler. Richtig ist vielmehr:

```
D<?> <-- d<? super C> <-- D<? super B> <-- D<? super A>
```

Danke an Herrn Ch. Sahnwaldt, der mich auf den Fehler aufmerksam gemacht hat.

Seite 78, Figur

Die Beschriftung ist unvollständig. Statt:

```
GenericArra
```

muss es heißen:

```
GenericArrayType
```

Seite 83, Zeile -3

Statt:

```
getArgumentTypes
```

muss es heißen:

```
getActualTypeArguments
```

Seite 122, Zeile -14

Kein Fehler. Aber statt der Klassen `MethodFilter` und `FieldFilter` hätte man eine Klasse `Filter<M extends Member>` schreiben können.

Seite 129, Zeile -9

Die Methode `getChildName` ist ein Relikt. Sie kann gestrichen werden. Statt dessen gibt's die Methode `toString (T node)`.

Seite 130, Zeile 1

Bei der Methode `getAttributeName` ist `return ""` sinnvoller.

Seite 158

Die Methode `getParameterizedType` ist algorithmisch noch nicht vollständig durchdacht. Sie funktioniert aber in allen "Standard-Situationen". Es sind aber "vertrackte" Fälle denkbar, die es erforderlich machen, den Algorithmus präziser zu fassen.

Seite 160, Zeile 13

Statt:

```
p4.isAssignableFrom(p1)
```

muss es heißen:

```
p3.isAssignableFrom(p1)
```

Seite 197, Mitte

"Unmittelbar nach ihrer Erzeugung ist sie signalisiert ..."

In diesem Satz fehlt das wichtige Wort "NICHT":

Unmittelbar nach ihrer Erzeugung ist die Condition NICHT signalisiert.

Seite 205, Zeile -20 (Mitte)

"Zwei `Producer`, die beide darauf warten, dass ein Platz in der `queue` frei wird. Der `consumer` holt ein Produkt ab. Beide `Producer` werden aufgeweckt - aber natürlich kommt nur einer der beiden zum Zuge. Der andere wartet erneut. Am Ende der `put`-Methode ruft dann der erste `Producer` `notifyAll` auf – mit dem Ergebnis, dass der zweite `Producer` erneut aufwacht, und nichts tun kann, als erneut zu warten..."

Herr Dominik Gruntz stellt die Sache richtig:

"Diese Aussage ist richtig, aber die Schlussfolgerung nicht. Der andere `Producer` der aufgeweckt worden ist wartet auf den synchronisations-Lock. Wenn dann der erste `Producer` `notifyAll` aufruft, so geht dieser Aufruf ins leere, da der andere `Producer` noch nicht das `wait` aufrufen konnte. Wenn der erste `Producer` dann den Synchronisationsblock verlässt, dann kommt der zweite `Producer` dran und wird feststellen, dass die `Queue` weiterhin voll ist und erst dann wird er `wait` aufrufen. Aber die Aussage, dass mit dem `notifyAll` sowohl `Producer` als auch `Consumer` aufgeweckt werden stimmt natürlich schon."

Seite 209, Zeile 1

"Zusätzlich zu der bereits in `queue` spezifizierten parameterlosen `offer`-Methode... "

Es bigt dort keine parameterlose `offer`-Methode. Es ist natürlich die erste `offer`-Methode (mit EINEM Parameter) gemeint.

Seite 215, Zeile -6

Herr Dominik Gruntz ist etwas verwundert:

"Ich verstehe nicht, warum die Elemente in umgekehrter Reihenfolge produziert werden, denn mit `products.add` werden die Produkte am Ende eingefügt, d.h., zuvorderst ist `Product 0`, dann `Product 1`, etc. und in dieser Reihenfolge werden die Elemente mit dem Iterator auch besucht und produziert."

Er wundert sich zu Recht.

Der Programmtext ist falsch. Er muss lauten (so steht es auch in der Original-Source - ich habe aber wohl eine frühere bzw. eine Testversion in das Manuskript übertragen):

```
for (int i = 5; i >= 0; i--)  
    products.add ("Product " + i);
```

Dann wird die Ausgabe verständlich.

Dann sollte allerdings auch in der viertletzten Zeile dieser Seite auch nicht von "Product 9", "Product 8" etc. die Rede sein, sondern von "Product 5", "Product 4" etc.

Seite 229-231

Die Klasse `CancellableTask` war in der Version 1.5.0 der J2SE noch enthalten, aber nicht mehr in der endgültigen Version.

Seite 239

Im Konstrukt der Klasse `Counter` befindet sich folgende Zeile:

```
this.count = count;
```

Es muss dort natürlich heißen:

```
this.count = initialCount;
```

Der Fehler wurde auch in den Sourcen beseitigt.

Danke an Herrn P. Saratchev, der den Fehler entdeckt hat.

Seite 253

Herr Ulrich Grude kritisiert zu Recht, dass ich nicht all die Möglichkeiten beschreibe, die die C++-Templates bieten – dass auch die Templates "Einschränkungen" kennen. Das ist richtig (und ich denke hierbei nur an die ATL und an die STL). Ich halte aber dagegen: Diese "Einschränkungen" sind nur implizit – sie können nicht explizit in der Schnittstelle ausgedrückt werden. Man muss die Implementierung lesen, um die impliziten Einschränkungen zu erkennen. C++-Templates müsste man natürlich wesentlich ausführlicher behandeln.

Seite 254, Abschnitt "Template-Funktionen"

Ich behaupte dort, es könnten nur globale Template-Funktionen definiert werden. Was mich zu diesem Satz veranlasst hat, ist mir völlig unerklärlich. Natürlich können auch Member-Funktionen als Template-Funktionen definiert werden.

Generell

Herr Ulrich Grude kritisiert zudem zu Recht, dass ich immer von Container-Klassen rede, wenn ich Collection-Klassen meine. Ich nehme aber an, dass aus dem Kontext hervorgeht, dass ich natürlich nicht die AWT-Container-Klasse meine, sondern eben "Sammlungs"-Klassen.