
Programmieren in Java<http://proglang.informatik.uni-freiburg.de/teaching/java/2013/>

Java-Übung Blatt 8 (Iterator, Generics)

2013-06-12

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Identifier und Kommentare bitte auf *englisch*!
- Schreiben Sie *sinnvolle* Kommentare
- Laden Sie Ihre Lösungen mit subversion (svn) ins Übungssystem hoch. Den entsprechenden Pfad finden Sie online.
- Das Übungssystem kann überprüfen, ob Sie Ihr Quelltext den Anforderungen genügt und ob Sie alle Klassen erstellt haben, etc. Nutzen Sie dies!
- Sollte das Übungssystem Ihre Lösungen ablehnen, dann werden sie *nicht* korrigiert!

Akzeptanzkriterien:

- Compiliert erfolgreich
- Checkstyle bringt keine Fehler
- Alle Packages, Klassen, Interfaces, Methoden, Typen, Argumente sind exakt wie auf dem Übungsblatt gefordert.
- Ihr korrigierender Tutor wird die Korrektur Ihrer Abgabe in Ihr svn-Repository unter dem Namen `Feedback-<login>-ex<XX>.txt` comitten. (<login> ist dabei Ihr myAccount name und <XX> die Kennziffern des Übungsblatts)
- Zusätzlich können Sie Ihre Gesamtpunktzahl im Übungsportal einsehen.

Abgabe: Freitag, 28. Juni 2013, um 23.59 Uhr.

Aufgabe 1 (Stream Editor, 10 Punkte)Projekt: `ex08_1`. Package: `stream`.

Sie modellieren in dieser Aufgabe das Grundgerüst eines einfachen Stream-Editors, mit dem eine Folge von Wörtern modifiziert werden kann. Die Grundbausteine hierzu sollen verkettete String-Iteratoren sein, d.h. Instanzen des aus der Vorelesung bekannten `Iterator<String>` Interfaces.

Ein Hinweis vorweg: Da wir Ströme aus Worten betrachten möchten, die nur einmal durchlaufen werden, macht die `remove` Methode des `Iterator` Interfaces keinen Sinn. Implementieren Sie diese für diese Aufgabe immer folgendermaßen (dies ist übrigens das Standardverfahren für `Iterator`-Implementierungen ohne `remove` Funktionalität):

```
1 @Override
2 public void remove() {
3     throw new UnsupportedOperationException();
4 }
```

Die konkrete Aufgabe ist nun:

1. (4 Punkte) Schreiben Sie zunächst einen String-Iterator `Words`, der als Eingabestrom dient. Die Klasse soll einen Konstruktor enthalten, der einen `String` übergeben bekommt und diesen in die einzelnen, durch Leerzeichen getrennten Wörter zerlegt (Tipp: nehmen Sie geeignete Methoden aus `java.lang.String` zur Hilfe). Über die einzelnen Wörter soll dann mit `hasNext` und `next` navigiert werden können. Testen Sie (mit JUnit) alle Methoden aus dem `Iterator`-Interface, bis auf `remove` (s.o.).
2. (6 Punkte) Es sollen nun verschiedene Arten Modifikatoren realisiert werden, die mit Hilfe des `Iterator<String>`-Interfaces auf Eingabeströmen (wie z.B. `Words`) arbeiten und dieses Interface auch selbst implementieren (um wiederum anderen Modifikatoren als Eingabestrom zu dienen). Implementieren Sie mindestens die folgenden Modifikatoren:

- Änderung der Schreibweise zu Großschreibung (Klasse `UpperCase`);
z.B. wird ("aaa", "bbb", "ccc") zu ("AAA", "BBB", "CCC").
- Ersetzen eines Wortes durch ein anderes (Klasse `ReplaceWord`);
z.B. wird ("aaa", "bbb", "ccc") zu ("ddd", "bbb", "ccc"), wenn ein `ReplaceWord` benutzt wird, der "aaa" gegen "ddd" ersetzt.
- Herausfiltern eines bestimmten Wortes (Klasse `RemoveWord`);
z.B. wird ("aaa", "bbb", "ccc") zu ("bbb", "ccc"), wenn ein `RemoveWord` benutzt wird, der das Word "aaa" herausfiltert.

Insbesondere sollen beliebige Modifikatoren miteinander kombinierbar sein.

Testen Sie (mit JUnit) ihre Implementierungen und verschiedene Kombinationen von Modifikatoren auf interessanten Eingabedaten.

Aufgabe 2 (Iterator Transformer, 13 Punkte)

Projekt: `ex08_2`. Package: `iter`. In der Vorlesung haben Sie gesehen, wie die Elemente von Collections mittels `ITransform` Implementierungen beliebig verändert werden können. Das `ITransform`-Interface finden sie auch noch einmal im Template zu dieser Aufgabe.

1. (6,5 Punkte) In dieser Aufgabe sollen Sie die `Transform` Klasse für Iteratoren implementieren. Beispiel: Eine `map` Operation mit Verdopplungs-`ITransform` auf einem Integer-Iterator der die Zahlen 1,2,3,4 liefert, ergibt einen Ausgabeiterator, der die Zahlen 2,4,6,8 liefert.

Die Signatur von `Transform` soll also wie folgt aussehen:

```

1 class Transform {
2     /**
3      * Applies an ITransform from type T to S on each element
4      * of an iterator.
5      *
6      * @param source The input elements.
7      * @param trans The transform.
8      * @return Provides a new iterator with the transformed elements.
9      *
10     * @param <T> The type of the input elements.
11     * @param <S> The type of the output elements.
12     */
13     public static <T, S> Iterator<S> map(Iterator<T> source,
14                                         ITransform<T, S> trans);
15 }

```

2. (6,5 Punkte) Eine weitere manchmal interessante Operation auf Iteratoren sind sogenannte Scans. Ein Scan wendet eine zweistellige Operation sukzessive auf eine Reihe Elemente an, und gibt dabei alle Zwischenergebnisse zurück.

Ein Beispiel für einen Scan mit der Operation „+“ (Addition) wäre die Umwandlung der Integer Reihe 1, 2, 3, 4 in 1, 3, 6, 10 (also $0 + 1, 1 + 2, 3 + 3, 6 + 4$). Die 0 muss hier als Startwert (oder „neutrales Element“) extra definiert werden.

Ein anderes Beispiel ist der Scan mit der Operation „add“ (entsprechend der `add` Methode des `List` Interfaces): Die Reihe 1, 2, 3, 4 würde in `[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]` umgewandelt werden¹. Das neutrale Element ist hier die leere Liste „`[]`“.

Die binären Operationen sollen, ähnlich wie die einstelligen Transformationen aus dem ersten Teil, durch Objekte repräsentiert werden. Das Interface hierzu soll

¹`[x, y, z]` symbolisiert hier, als abkürzende Schreibweise, ein `List` Objekt mit den Elementen `x`, `y` und `z`. Ausgeschrieben hieße das etwas wie `Arrays.asList(new int[] {x, y, z})`. Die leere Liste (`new ArrayList<Integer>()`) schreiben wir als `[]`

ScanOp heißen. Definieren Sie ein **IScanOp** Interface; Sie benötigen zwei Typparameter.

Implementieren Sie zusätzlich die **Scan** Klasse für Iteratoren. Dabei hat **Scan** folgende Signatur:

```
1 class Scan {
2     /**
3      * Applies a fold from types T and S to S on elements of an Iterator<T>
4      * with a start value, yielding all intermediate results in the process.
5      *
6      * @param source The source iterator.
7      * @param start The start value, or "neutral element"
8      * @param op The binary operation.
9      * @return The resulting iterator after the scan.
10     * @param <T> The type of the input elements.
11     * @param <S> The type of the output elements.
12     */
13     public static <T,S> Iterator<S> scan(Iterator<T> source, S start,
14         IScanOp<T,S> op);
15 }
```

Testen Sie Ihren Code mit JUnit, und mindestens drei verschiedenen **IScanOp** Implementierungen!

Aufgabe 3 (Generischer Baum, 10 Punkte)

Projekt: `ex08_3`. Package: `gentree`. In Aufgabe `ex05_1` haben Sie eine Baumstruktur implementiert deren innere Knoten Werte vom Type `int` waren. In dieser Aufgabe ist das Ziel, den Baum generisch zu implementieren, sodass die inneren Knoten Werte eines beliebigen (wenn auch gleichen) Typs tragen dürfen. Im Template finden Sie eine Musterlösung zur Aufgabe `ex05_1` die Sie anpassen können. Sie können aber auch Ihre eigene (funktionierende) Abgabe zu Blatt 5 nutzen.

1. (7 Punkte) Passen Sie nun die Implementierung des Baumes so an, dass der Typ der Daten der Inneren Knoten generisch ist. Zur Darstellung der Daten in der **draw**-Methode, kann die **toString()** Methode verwendet werden; diese ist für alle Objekte definiert und steht deshalb auch generisch deklarierten Objekten zur Verfügung.

Zusatzaufgabe für Ästheten (unbepunktet): Die Darstellung der Knoten ist nun natürlich etwas zerstört, da sich die Größe der Knotenkreise nicht dem Inhalt anpasst. Korrigieren Sie das. Den benötigten Platz für einen String können Sie mit folgendem Code berechnen²:

```
1     Graphics g; // g bekommen Sie z.b. in der paintComponent Methode
2     g.getFontMetrics().stringWidth("Hello");
```

2. (3 Punkte) Erweitern sie das (nun generische) **ITree** Interface mit einer Methode **customDraw**. Diese soll analog zu **draw** funktionieren, aber dem Aufrufer der Methode die Möglichkeit bieten, die Darstellung der einzelnen Knoten auf dem **ICanvas** Interface als weiteren Parameter zu spezifizieren (durch Implementierung und Instanziierung eines generischen Interfaces). Implementieren Sie außerdem die **draw** Methode mittels **customDraw**, um Codeduplizierung zu vermeiden. (Tipp: Hier ist das *Strategy*-Pattern gefragt. Es wurde in der Vorlesung vorgestellt und auch in den obigen Aufgaben implizit benutzt)

Testen durch ausprobieren reicht; geben Sie Ihre Testprogramme aber mit ab!

²siehe auch <http://docs.oracle.com/javase/tutorial/2d/text/measuringtext.html>