Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2011/

Java-Übung Blatt 9 (Generics and Collections)

2011-07-04

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Exportieren Sie das Projekt als ZIP-Datei mit Namen *accountname-projectname*`.zip`. Beispiel: Peter Lustig (`lustigp`) gibt `ex02_1` ab als Datei `lustigp-ex02_1.zip`
- ZIP-Export in Eclipse: Rechtsklick auf die Wurzel des Projects, "Export...", "General/Archive File..."
- Mailen Sie die ZIP-Files an Ihren Tutor.

Abgabe: Montag, 11. Juli 2011, um 9.00 Uhr.

Aufgabe 1 (Equality, 4 Punkte)
Projekt: `ex09_1`. Package: `equality`.
Wir betrachen die `equals`-Methode. Holen Sie sich das Skelett der Aufgabe von der Vorlesungsseite. Im Projekt befinden sich eine Reihe von Klassen, die von `Vehicle` abgeleitet sind.
Ihre Aufgabe ist, es eine korrekte Implementierung von `equals` in allen `Vehicle`-basierten Klassen zu erstellen. Sie sollen dabei auf die Semantik achten: Welche Felder der Klassen sind relevant für die Gleichheit? Wann würde man von zwei gleichen Fahrzeugen sprechen? Ihre Implementierung der `equals`-Methode soll das "natürliche" Verständis von Gleichheit bei Fahrzeugen repräsentieren. Achten Sie darauf keinen Code zu duplizieren.
Schauen Sie sich zunächst noch einmal die Anforderungen an eine korrekte `equals`-Methode an. Denken Sie auch daran, dass Ihre `equals`-Methode die Methode `Object.equals` überschreiben muss.
Testen Sie die wichtigsten Szenarios. Insbesondere sollen Instanzen aller Klassen miteinander verglichen werden. Vergleichen die gleiche Instanzen und unterschiedliche Instanzen miteinander. In welchen Fällen erwarten Sie bei unterschiedlichen Instanzen, dass die `equals`-Methode `true` zurückliefert. Stellen Sie auch sicher, dass `hashCode` korrekt implementiert ist. Beachten Sie den Zusammenhang zur `equals`-Methode.

Aufgabe 2 (Einfach verkettete Listen, 4 Punkte)
Projekt: `ex09_2`. Package: `lists`.
In den wenigen Javakurswochen haben Sie funktionale einfach-verkettete Listen schon oft gesehen. In dieser Übung brauchen wir nochmal eine, und zwar eine generische, also eine, die sich für beliebige Elementtypen `E` instantiieren lässt.
Spätestens jetzt sollten Sie sich die Eclipse-Zaubertasten für Vervollständigung (Ctrl+Space), Erzeugen von Gettern und Settern (Alt+Shift+S R), Erzeugen eines Konstruktors für Felder (Alt+Shift+S O) und allgemeine Quick-Fixes (Ctrl+1) angewöhnen, sonst dauert es zu lange. Nutzen Sie ebenfalls die in den Eclipse-Dialogen angebotene Generierung von Kommentaren. Schreiben Sie

- ein Interface `IList<E>` mit Methoden `isEmpty()`, `getFirst()`, `getRest()` mit sinnvollen Typen.
- eine Klasse `EmptyList<E>`, die `IList<E>` als leere Liste implementiert. Die `get...`-Methoden dürfen `null` zurückliefern oder eine Exception werfen.
- eine Klasse `ListNode<E>`, die `IList<E>` als nichtleere Liste, bestehend aus einem ersten Element vom Typ `T` und einem Rest vom Typ `IList<E>`, implementiert.

Aufgabe 3 (Multimap, 4 Punkte)
Projekt: `ex09_3`. Package: `collections`.

Das Java Collections Framework enthält keine Implementierung für eine `Map`, die einen Schlüssel auf mehrere Werte abbildet, ein sogenannte `MultiMap`. Allerdings kann eine solche relativ leicht auf Basis einer `HashMap` realisiert werden. Implementieren Sie eine Klasse `MultiValueMap`, die das folgende `MultiMap`-Interface implementiert:

```
1  /**
2   * A <tt>Map</tt>−like interface for an object that maps keys to a collection
3   * of values.
4   *
5   * @param <K> the type of keys maintained by this map
6   * @param <V> the type of mapped values
7   */
8  public interface MultiMap<K, V> {
9
10     /**
11      * Checks whether the map contains the value specified.
12      *
13      * @param value the value to search for
14      * @return true if the map contains the value
15      */
16     boolean containsValue(Object value);
17
18     /**
19      * Gets the collection of values associated with the specified key.
20      *
21      * @param key
22      * the key to retrieve
23      * @return the Collection of values, implementations should return null for
24      * no mapping, but may return an empty collection
25      */
26     Collection<V> get(K key);
27
28     /**
29      * Adds the value to the collection associated with the specified key.
30      *
31      * Unlike a normal Map the previous value is not replaced. Instead the new
32      * value is added to the collection stored against the key.
33      *
34      * @param key
35      * the key to store against
36      * @param value
37      * the value to add to the collection at the key
38      * @return typically the value added if the map changed and null if the map
39      * did not change
40      */
41     Collection<V> put(K key, V value);
42
43     /**
44      * Removes all values associated with the specified key.
45      *
46      * @param key
47      * the key to remove values from
48      * @return the Collection of values removed, implementations should return
49      * null for no mapping found, but may return an empty collection
```

```
50      */
51     V remove(Object key);
52
53     /**
54      * Removes a specific value from map.
55      *
56      * The item is removed from the collection mapped to the specified key.
57      * Other values attached to that key are unaffected.
58      *
59      * @param key
60      * the key to remove from
61      * @param item
62      * the item to remove
63      * @return the value removed (which was passed in), null if nothing removed
64      */
65     V remove(java.lang.Object key, java.lang.Object item);
66
67     /**
68      * Gets the number of keys in this map.
69      *
70      * @return the number of key−collection mappings in this map
71      */
72     int size();
73
74     /**
75      * Gets a collection containing all the values in the map.
76      *
77      * @return a collection view of the values contained in this map
78      */
79     Collection<V> values();
```

Testen Sie Ihre Implementierung mit mindestens 2 Testfällen pro Methode.

Nachholaufgaben für Thema 7:

Mit dieser Aufgabe können Sie den Stoff von Thema 7 (dem Thema von Blatt 7) wiederholen und nochmal Punkte für Thema 7 erwerben.

Aufgabe 4 (Exceptions und Covarianz, 4 Punkte)
Projekt: `ex07_4` Package: `exceptions`

In Java weisen Arrays einige Besonderheiten auf: Betrachten wir Java-Arrays als eingebaute parametrisierte Klassen, dann ist insbesondere interessant, dass diese im Gegensatz zu anderen generischen Klassen covariant sind. Covarianz bedeutet in diesem Zusammenhang, dass ein Array vom Typ `T1[]` einer Referenz vom Typ `T2[]` zugewiesen werden kann, wenn `T1` von `T2` abgeleitet ist.

(a) Zeigen Sie anhand eines Beispiels, wo der Unterschied zwischen einem Array und einer `ArrayList<E>` beim Zuweisen einer mit `T1` parametrisierten `ArrayList`-Instanz an eine mit `T2` parametriesierten `ArrayList`-Referenz liegt, wenn `T1` von `T2` abgeleitet ist. Tipp: Im Fall einer `ArrayList<E>` sollte ihr Code nicht kompilieren.

(b) Zeigen Sie nun, welche Probleme die Covarianz bei Arrays haben kann. Tipp: Schauen Sie sich in der Java-Sprachdefinition das Kapitel über Arrays an.

(c) Nutzen Sie einen erzwungenen Cast mit dem Typcast-Operator, damit die Zuweisung auch im Fall der (invarianten) `ArrayList<E>` kompiliert. Beschwert sich der Compiler immer noch, dass kein Cast möglich ist? Machen Sie dem Compiler glaubhaft, dass der erzwungene Cast zur Laufzeit möglich sein kann! Der Code müsste nun

mit einer Warnung (die Sie außerhalb dieser Aufgabe nicht ignorieren sollten) `Type safety. Unchecked cast ...` kompilieren. Tipp: Sie benötigen eine Refererenz, die sowohl kompatibel zum Typ des Ziels Ihrer Zuweisung, als auch zur Quelle Ihrer Zuweisung ist.

(d) Im vorigen Schritt haben Sie die Typsicherheit ausgehebelt. Demonstrieren Sie in einem Beispiel, welche negativen Konsequenzen das haben kann. Tipp: Probleme tauchen möglicherweise erst bei spätereren Operationen auf der Collection auf, als die im Fall von Arrays der Fall war.

Auch wenn Sie die Aufgabe nur unvollständig gelöst haben, sollten Sie sicherstellen, dass Ihr abgegebenes Projekt kompilierbar ist. Kommentieren Sie dazu die Zeile aus, in der der Typ-Check fehlschlägt.