Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2015/

Java-Übung Blatt 9 (Generics, Anonyme Klassen)

2015-06-29

Hinweise

- Ändern Sie nicht die Schreibweise von Bezeichnen, die auf dem Übungsblatt vorgegeben sind. Dies betrifft sämtliche global sichtbaren Namen von Eclipse-Projekten, Paketen, Klassen, etc.
- Bezeichner 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.
- Sollte das Übungssystem Ihre Einreichung nicht übersetzen können, dann wird sie nicht korrigiert und Sie erhalten keine Punkte.
- Die Korrektur Ihrer Abgabe wird unter dem Namen `Feedback-<login>-ex<NN>_<X>.txt` in das jeweilige Projektverzeichnis eingecheckt. Dabei ist `<login>` Ihr *myAccount*-Name und `<NN>_<X>` der Projektname.
- Sie können Ihre Gesamtpunktzahl im Übungsportal einsehen.
- Dokumentieren Sie Ihren Zeitaufwand, den gefühlten Schwierigkeitsgrad, sowie Probleme beim Lösen der Aufgabe oder auch Anregungen und Vorschläge zur Verbesserung in der Datei `<Projektverzeichnis>/erfahrungen.txt`

Hinweis: Dieses PDF stammt aus dem Archiv `ex09.zip`. Dieses Archiv enthält weiterhin noch Skelett-Projekte, die für die Bearbeitung der Aufgaben hilfreich sind.

Exercise 1 (Paare und Ordnung, 8 Punkte)

Projekt: `ex09_1`. Package: `pairs`. In der Vorlesung wurden ja bereits als Beispiel für Generics ein Datentyp für sogenannte Paare gezeigt. Ein Paar ist ein Objekt, das genau zwei andere Objekte zusammenfasst. Im Unterschied zu einer Collection können die beiden Objekte aber unterschiedliche (auch nicht-verwandte) Typen haben.

1. (2 Punkte) Definieren Sie den Paar-Typ als eine generische Klasse `Pair`. Ihr Konstruktor erwartet zwei Objekte beliebigen Typs und man kann mit den Methoden `fst()` und `snd()` auf das Erste bzw. Zweite der übergebenen Objekte zugreifen. Implementieren Sie außerdem geeignete `hashCode` und `equals` Methoden.

2. (2 Punkte) Schreiben Sie eine Klasse `CompPair` die von `Pair` ableitet und zusätzlich das `Comparable` Interface implementiert. Stellen Sie sicher, dass Code wie der folgende compiliert:

```
1  package pairs;
2
3  interface Vehicle extends Comparable<Vehicle> {}
4  interface Car extends Vehicle {}
5  interface Train extends Vehicle {}
6
7  public class CompileMe {
8      public static void test(Car c1, Car c2, Train t1, Train t2) {
9          new CompPair<Car, Train>(c1, t1)
10             .compareTo(new CompPair<Car, Train>(c2, t2));
11     }
12 }
```

   Beachten Sie außerdem die in der Vorlesung vorgestellten Einschränkungen und Konventionen für `Comparable`.

   Definieren Sie auch einen Konstruktor für `CompPair` dem man gewöhnliche `Pair`s übergeben kann, sofern deren Komponenten vergleichbar sind.

   Testen Sie in ihren Unit-Tests auch die korrekte „Zusammenarbeit" mit den Collection-Klassen; verwenden Sie dazu z.B. die Collection-Klasse `TreeSet` (welche Mengen aus geordneten Elementen implementiert) und überprüfen Sie, ob die typischen Mengen-Operationen wie erwartet funktionieren und ob der Iterator des `TreeSet`s die Elemente in der richtigen Reihenfolge ausgibt.

3. (2 Punkte) Paare können im Rückgabetyp von sog. zip-Operationen verwendet werden: Eine zip-Funktion nimmt zwei `Collection`s als Argument und gibt eine `Collection` aus

Paaren als Ergebnis zurück. Die Paare des Ergebnisses bestehen genau aus den kombinierten Elementen der Argument-`Collection`s. Falls die Argument-`Collection`s eine „Reihenfolge" der Elemente vorgeben, soll diese im Ergebnis eingehalten werden. Falls die Argument-`Collection`s ungleicher Länge sind, soll das Ergebnis die Länge des kürzeren Arguments haben. Beispiel in Pseudocode :

```
zip([1,2,3,4,5],["Hallo", "Welt", "wie", "geht's"])
                        ==
[(1, "Hallo"), (2, "Welt"), (3, "wie"), (4, "geht's")]
```

(`[1,2,3]` symbolisiert hier eine Collection mit den Elementen `1`, `2` und `3`; `(1,2)` symbolisiert ein Paar mit Komponenten `1` und `2`)

Implementieren sie die zip-Funktion als generische, statische Methode `zip` der Klasse `pairs.Zip`.

4. (2 Punkte) Implementieren Sie ein alternative Version von `zip`, bei der man über einen zusätzlichen Parameter angeben kann, von welchem `Collection`-Typ das Ergebnis sein soll. Für diesen Zusatz-Parameter werden Sie ein Interface definieren müssen, das angibt wie man eine `Collection` von entsprechendem Typ baut. Nutzen Sie Generics damit dieser Typ auch statisch als Rückgabewert auftritt.

   Das folgende Beispiel demonstriert, was mit der oben beschriebenen Funktion möglich sein soll:

```
1 ArrayList<Pair<Integer,String>> zipResult1 =
2     zip(ArrayListBuilder.make(), Arrays.asList(1,2,3), Arrays.asList("4","5","6"));
3 Set<Pair<Integer,String>> zipResult1 =
4     zip(SetBuilder.make(), Arrays.asList(1,2,3), Arrays.asList("4","5","6"));
```

   (ArrayListBuilder.make() und SetBuilder.make() sind hier Factory-Methoden für Objekte die zum oben beschriebenen Zusatz-Parameter passen)

Testen Sie ihre Implementierungen mit JUnit. Die obigen Aufgaben können und sollen ohne die Verwendung von Typcasts gelöst werden.

Exercise 2 (Anonyme Klassen, 2 Punkte)
Projekt: `ex09_2`. Package: `anonymous`.
Im Skelett finden Sie die von Entwickler `anonymous` entwickelte Klasse `HelloWorld`, eine grafische Variante des bekannten Hallo-Welt-Programms. Dieses Programm gibt beim Klick auf einen Button „Hello World #n!" aus, wobei n für den n-ten Weltgruß steht. Der Handler für die Klick-Aktion ist hier als anonyme Klasse implementiert, was ein verbreitetes Anwendungsszenario anonymer Klassen darstellt.

Weiterhin enthält das Skelett noch die Klasse `HelloWorldMockAnonymous`, eine Version von `HelloWorld`, die daraufhin entstanden ist, dass Entwickler `nick` benannten Klassen gegenüber anonymen Klassen den Vorzug gibt. Das Program benutzt die konkrete Handler-Klasse `MockAnonymousClassHandler`.

Da die Entwicklung auch im Hallo-Welt-Projekt weiter ging, entstand aus einer Initiative von Entwickler `anonymous` heraus bald die Klasse `HelloWorldResetPartlyMockAnonymous`, die auf `nick`s Implementierung aufbaut und als neues Feature einen zusätzlichen Reset-Button enthält, mit dem der interne Zähler zurückgesetzt werden kann — natürlich mit dem Ziel, dass nach einem Reset die Ausgabe wieder mit „Hello World #1!" beginnt. Der neue Handler ist wiederum mittels anonymer Klasse umgesetzt.

Sie werden feststellen, dass das Programm nicht wie erwartet funktioniert: Ein Klick auf „Reset" hat keinen Einfluss auf die Ausgabe. Entwickler `anonymous` behauptet, das Problem liege nur an `nick`s `MockAnonymousClassHandler` und präsentiert kurzerhand die Klasse `HelloWorldReset`, die anstandslos funktioniert.

Ihre Aufgabe besteht nun darin Entwickler `nick` beizustehen und das Programm `HelloWorldResetPartlyMockAnonymous` zu reparieren.

1. Reparieren Sie den konkreten `MockAnonymousClassHandler`, ohne diesen jedoch durch eine anonyme Klasse (oder, falls Sie diese bereits kennen, einen λ-Ausdruck) zu ersetzen.

2. Erklären Sie was genau das Problem in `nick`s `MockAnonymousClassHandler` ist und erklären Sie, wie Sie dieses gelöst haben. Gehen Sie auch darauf ein, warum `anonymous`' `HelloWorldReset` funktioniert.