**Programmieren in Java**

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

# Java-Übung Blatt 5 (Rekursive Klassen)

2015-05-27

## 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`

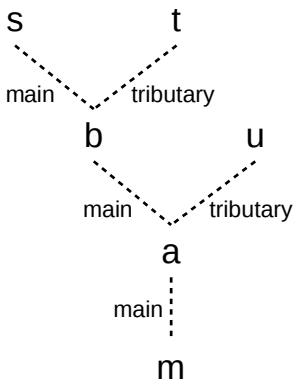
*Hinweis:* Dieses PDF stammt aus dem Archiv `ex05.zip`. Dieses Archiv enthält weiterhin noch Skelett-Projekte, die für die Bearbeitung der Aufgaben hilfreich sind.

**Exercise 1** (Flusssystem, 2,5 Punkte)
Projektverzeichnis: `ex05_1`. Package: `riversystems`.

Das Umweltministerium überwacht die Wasserqualität in einem Flussnetz. Ein Flussnetz besteht aus einem Fluss, seinen Nebenflüssen, den Nebenflüssen von Nebenflüssen und so fort.

Der Ort, an dem ein Nebenfluss in einen Fluss fließt, wird Zusammenfluss genannt. Das Ende eines Flusses - der Teil, der in ein Meer fließt - wird Mündung genant. Der initiale Teil eines Flussabschnittes wird Quelle genannt.

Beispiel:




Das Beispiel hat 3 Quellen (`s`, `t` und `u`) und 2 Nebenflüsse an den Orten `a` und `b`. Die Mündung des gesamten Netzes ist bei dem Ort `m`. Die Karte stellt das Netz dar, als ob es einen Hauptfluss mit zwei direkten Nebenflüssen gibt. Das Beispiel lässt allerdings auch andere Interpretationen des Flussnetzes zu.

1. Erstellen Sie zunächst ein Klassendiagramm, wie Sie es bereits aus den vorhergehenden Übungsblättern gewohnt sind.[1] Verwenden Sie das Konzept der rekursiven Klassen, damit ein beliebig großes Flussnetz aufgebaut werden kann. Geben Sie das Diagramm zusammen mit Ihrem Code im Projekt ab. Jeder Teil eines Flusssystems enthält Daten zur Lokalisierung, nämlich kartesische Koordinaten und einen Namen. Für ein Flusssystem soll feststellbar sein, ob eine bestimmte Koordinate flussaufwärts erreichbar ist. Stellen Sie außerdem sicher, dass keine Flusssysteme erstellt werden können, in denen verschiedene Teile auf gleichen Koordinaten liegen. *Hinweis:* Verwenden Sie ein Interface `IRiver`, das als Vereinigung von zwei bestimmten Klassen dient. Überlegen Sie selbst, welche Klassen das sind.

[1] Verwenden Sie die UML-Konventionen aus der Vorlesung. Sie können ein Tool Ihrer Wahl, ASCII-Art (siehe `http://c2.com/cgi/wiki?UmlAsciiArt`) verwenden oder eine ordentliche, handgeschriebene Zeichnung einscannen. Speichern Sie Ihre Lösung bitte als pdf, txt, jpg oder png ab (in das Projektverzeichnis; neben dem src-Ordner) und übertragen Sie diese ins Daphne-SVN.

2. Implementieren Sie ausgehend von Ihrem Diagramm Klassen, die ein Flussnetz beschreiben. Alle Klassen, die das Flusssystem repräsentieren, sollen *immutable* sein, d.h. ihre Objekte sollen nach der Instanzierung nicht mehr verändert werden können.

3. Überlegen Sie sich nun ein eigenes Beispielflussnetz. Dieses soll mindestens so groß und komplex sein wie das obige Beispiel. Erstellen Sie hierfür eine Klasse `RiverSystemExample`, die ihr Beispielflussnetz in einer Methode erstellen soll.

**Exercise 2** (Decorator-Pattern, 5 Punkte)
Projektverzeichnis: `ex05_2`. Package: `geometry`.

Benutzen Sie das mitgelieferte Skelett-Projekt zu dieser Aufgabe. Das Projekt ähnelt der Lösung zu Aufgabe 2.1.

In der Vorlesung haben Sie rekursive Klassen kennengelernt. Sie sollen mit Hilfe des *Decorator-Patterns* Transformationen auf geometrischen Figuren implementieren. Eine Transformation ist ein *Dekorierer*[2] für Figuren[3], welcher selbst auch eine Figur ist und in der `contains`-Methode die übergebenen Punkte modifiziert an das Kind weiterreicht. *Tipp:* Um Codeduplizierung zu vermeiden, schreiben Sie eine abstrakte Klasse DecoratedShape und leiten Sie die Transformationen davon ab.

Implementieren Sie drei Dekorierer:

**Versatz** Transformiert den Punkt auf eine Art, die das Objekt verschoben erscheinen lässt.

**Skalierung** Multipliziert die Koordinaten des Punkts mit einem festen Wert und simuliert so eine Vergrößerung oder Verkleinerung vor.

**Rotation** Verschiebt den Punkt um einen Winkel auf einer Kreisbahn um den Ursprung. Das Objekt erscheint so rotiert.

Erörtern Sie außerdem folgende Punkte und geben Sie Ihre Antwort als Text- oder PDF-Datei in das Projektverzeichnis neben dem src-Ordner ab.

- Welche Vor- und Nachteile hat die Verwendung des Decorator-Patterns zur Implementierung von Shape-Transformationen?

- Einer der Dekorierer stellt im Prinzip die gleiche Funktionalität wie die `offset`-Methode bereit. Welche Vor- und Nachteile würden sich ergeben wenn man diesen Dekorierer zur Implementierung von `offset` benutzt (bzw. `offset` zu Gunsten des Dekorieres entfernt)?

*Hinweis:* Die o.g. Punkte überschneiden sich thematisch; Sie müssen nicht für jeden eine explizite Antwort entwickeln.

**Exercise 3** (Mengenoperationen, 2,5 Punkte)
Projektverzeichnis: `ex05_3`. Package: `geometry`.

Bevor Sie diese Aufgabe bearbeiten, sollten Sie Aufgabe 2 bearbeiten. Insbesondere der Versatzdekorierer ist hilfreich bei der Implementierung dieser Aufgabe. Benutzen Sie das mitgelieferte Skelett-Projekt zu dieser Aufgabe.

In den bisherigen `geometry` Aufgaben wurden geometrischen Figuren als Menge von Punkten definiert. Das heißt eine Figur bestimmt über ihre `contains`-Methode welche Punkte zu ihr gehören. Für diese mengenbasierte Definition von Figuren ist auch sinnvoll, die üblichen Mengenoperationen zu implementieren. Zur Erinnerung:

$$A \cap B := \{p \mid p \in A \wedge p \in B\} \qquad \text{Schnitt}$$
$$A \cup B := \{p \mid p \in A \vee p \in B\} \qquad \text{Vereinigung}$$
$$A \setminus B := \{p \mid p \in A \wedge p \notin B\} \qquad \text{Differenz}$$

Implementieren Sie diese Operationen. Entwickeln Sie JUnit-Tests, die alle Operationen abdecken. Testen Sie einige wesentliche Fälle (im Fall der Schnitt-Operation z.B. $A \cap B = \emptyset$, $A \cap B = A$, $A \cap B = B$, $A \cap B \subset A \wedge A \cap B \subset B$) mit interessanten Punkten.
*Hinweis:* Es bietet sich an, eine abstrakte Klasse CompoundShape, die das Interface Shape implementiert und selbst drei Kinder vom Typ Shape hat, zu entwickeln.

[2] Das Dekorierer-Entwurfsmuster wird am Montag, dem 01.06.2015 in der Vorlesung behandelt.
[3] Das heißt: Die Klasse hat ein Feld vom Typ `Shape`.