P. Thiemann und K. Anton Sommersemester 2010

Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2010/

Betreutes Java-Programmieren 4

2010-05-10

Anfang Loggen Sie sich in der normalen Linux-Umgebung ein.

Starten und Speichern Starten Sie Eclipse, indem Sie in einem Terminalfenster `eclipse` eingeben. Wenn Eclipse Sie fragt, wo der Workspace hin soll, antworten Sie bitte `/home/`*`ihrloginname`*`/workspace`, was normalerweise auch schon voreingestellt sein sollte.

Drumherum Unter der URL `http://nonopapa:8080/teaching/java/2010/` steht Ihnen das Vorlesungsmaterial zur Verfügung. Die Sun-Java-Doku gibt es unter `http://nonopapa:8080/javadoc/`

Bei Fragen zur Rechnerbenutzung, zu den Aufgaben oder zu diesen Hinweisen wenden Sie sich bitte sofort an einen der Tutoren.

Ende Erzeugen Sie ein ZIP- oder tgz-Archiv Ihres Eclipse-Projects: Rechtsklick aufs Project, "Export...", im Ast "General" die Zeile "Archive File" auswählen, prüfen, dass links oben genau Ihr Project angekreuzt ist, im Feld "To archive file" einen Dateinamen eingeben, der Ihren Namen enthaelt: zB `/home/mustermm/Desktop/abgabe3-mustermm.zip` für Max Mustermann, "Finish" klicken. Dabei sollte ein ZIP-Archiv enstehen, das Sie am Ende dem Tutor mailen. Nur ein Attachment pro Mail!

Dieses Blatt geben Sie bitte den Tutoren zurück.

Eclipse-Project. Heute brauchen Sie das Eclipse-Projekt nicht selbst anzulegen – Sie starten mit einem halbfertigen Projekt. Laden Sie zunächst `http://nonopapa:8080/teaching/java/2010/supp/skel4-imp.zip` herunter (skel wie "Skelett"). Dann Kontextmenü im Package-Explorer aufmachen, "Import...", "General" aufklappen, "Existing projects into workspace" auswählen, "Next" klicken, oben "Select Archive File" auswählen, "Browse" drücken, das runtergeladene `skel4.zip` suchen, dann "Finish" klicken. Sie sollten jetzt ein Project "abgabe4" haben.

Test-First. Es ist fast immer eine gute Idee, den Test zuerst zu schreiben: beginnen Sie mit der Implementation einer Methode erst, wenn Sie schon Testfälle haben, die Sie laufen lassen können. Dazu kann es notwendig sein, die zu testende Klasse schon mal hinzuschreiben, aber die Methoden leer zu lassen.

Diese Woche. Wir implementieren imperative Datenstrukturen am Beispiel einer geordneten verketteten Liste. In Highscore-Listen werden alle `Participants` mit ihrem `name` und ihrem `score` gespeichert, und zwar absteigend nach Score (damit der Sieger als erstes kommt). Das implementieren wir als verkettete Liste[1]. **Package:** `lists`

Aufgabe 1 (Participants, 2+4 Punkte)

(a) Schreiben Sie eine Klasse `Participant` mit Feldern `name` und `score`. Die Klasse soll keine öffentlichen Felder haben, sondern den Zugriff auf die Felder nur mit Gettern (Methoden wie `String getName()`) ermöglichen. Sie soll auch das Interface `IParticipant` aus der Vorlage implementieren.

(b) Die Klasse `Participant` soll eine Methode `boolean lessThan(IParticipant)` haben. Und zwar soll sie folgendes erfüllen:

- Wenn `a.score` < `b.score`, soll `a.lessThan(b)` `true` liefern.
- Wenn `a.score` = `b.score` und `a.name` im Alphabet vor `b.name` kommt (wie man mit `String.compareTo` herausfindet), soll `a.lessThan(b)` `true` liefern.

[1] Eigentlich enthält die Java-Standardbibliothek viele fertige Datenstrukturen, die wir nehmen könnten – diese Woche bauen wir es aber selbst.

- Sonst soll das Ergebnis `false` sein.

Schreiben Sie vier sinnvolle Testfälle und *dann* die Methode.

Aufgabe 2 (Liste, 3+4+1+6)

(a) Für die einfach verkettete Liste brauchen wir Knoten: ein Interface `IOrderedListNode` steht schon bereit, die Klassen `EmptyNode` und `ConsNode` für leere und nichtleere Knoten schreiben bitte Sie. In Methoden, deren Aufruf verboten ist, können Sie den Befehl

```
throw new UnsupportedOperationException("irgendeine gute Fehlermeldung");
```

unverstanden hineinschreiben (nutzen Sie Ctrl+Space für den Namen!)

(b) Wir folgen dem Muster für imperative Datenstrukturen, d.h. es gibt eine Klasse für die Gesamtliste und andere für Knoten. Das Interface `IHighScoreList` für die Gesamtliste steht schon bereit. Implementieren Sie die Klasse `HighScoreList`, die das Interface implementiert, aber *nur so weit*, bis keine roten Fehlermarkierungen mehr zu sehen sind.

Schreiben Sie dann mindestens sechs Testfälle, in denen Sie verschiedene Folgen von `Participant`s in eine `HighScoreList` stecken und den Rückgabewert von `asCommaSeparatedString()` mit Ihrem Erwartungswert vergleichen.

(c) `HighScoreList` soll ein Feld `head` haben, das auf den ersten Listenknoten zeigt. Frisch erzeugte `HighScoreList`s sollen dort einen leeren Listenknoten haben.

(d) Implementieren Sie `insert` und `asCommaSeparatedString` so weit, dass die Testfälle passen,

a) in denen keine Elemente eingefügt werden (erst mal)

b) in denen Participants in aufsteigender Reihenfolge der Scores eingefügt werden (dann)

c) alle (wenn es so weit funktioniert)

Für `asCommaSeparatedString` lohnt sich ein Blick in `StringBufferDemo`.

Praktische Eclipse-Tastenkombinationen

- Ctrl+Space: den Identifier unterm Cursor vervollständigen
- Ctrl+1: Quick Fix (leichtere Reparaturen automatisch vornehmen, z.B. alle fehlenden Methoden des Interfaces implementieren)
- F3: gehe zur Definition der Klasse/Methode/... unterm Cursor
- Ctrl+Shift+T: gehe zu Klasse, gegeben die Grossbuchstaben des Klassennamens
- Ctrl+Shift+G: liste alle Verwendungen der Klasse/Methode/... unterm Cursor auf
- Alt+Shift+J: kommentiere das markierte Sprachelement
- Alt+Shift+S: diverse Source-Transformationen, z.B. “Generate Constructor using Fields”
- Alt+Shift+R: Rename
- Und einfach mal das Source- und das Refactor-Menü durchstöbern, man lernt eigentlich immer was dazu.