Abschlußaufgabe 1: Little Praktomat

My Solutions

Beachten Sie: Zur endgültigen Abgabe der Abschlußaufgabe werden wir im Laufe dieser Woche eine neue Instanz des Praktomaten aufsetzen. Sie werden darüber rechtzeitig an dieser Stelle informiert.

Aufgabenstellung

In dieser Aufgabe sollen Sie die Grundfunktionalität eines Systems zur Verwaltung des Übungsbetriebs (ähnlich des Praktomaten) implementieren. In einem solchen System gibt es verschiedene Arten von Benutzern: Studenten und Tutoren. Es werden Aufgaben verwaltet, die von Studenten gelöst und von Tutoren korrigiert werden. Jede abgegebene Lösung kann höchstens von einem Tutor korrigiert werden.

Zu Beginn enthält Ihr System weder vordefinierte Personen noch Aufgaben. Diese werden über eine textuelle Kommandoschnittstelle erzeugt. Dabei soll es möglich sein

- Studenten und Tutoren anzulegen, wobei jeder Student immer genau einem Tutor zugeordnet ist,
- Aufgaben anzulegen,
- eine Lösung zu einer bestimmten Aufgabe für einen bestimmten Studenten einzureichen,
- eine Korrektur für eine abgegebene Lösung zu erstellen,
- eine Notenübersicht aller korrigierten Lösungen je Aufgabe zu sehen,
- eine Zusammenfassung der Korrekturergebnisse je Aufgabe zu sehen,
- eine Zusammenfassung der Korrekturergebnisse je Tutor zu sehen und
- eine Zusammenfassung der Korrekturergebnisse je Student zu sehen.

Beachten Sie dabei, dass ein Student zu jeder Aufgabe nur einmal eine Lösung einreichen darf. Ein Nachbessern der Lösung darf also nicht möglich sein. Die Korrektur einer Lösung soll hingegen beliebig oft überschrieben werden dürfen. Es genügt, wenn sich das System die zuletzt abgegebene Korrektur zu einer Lösung merkt. Vorhergehende Korrekturen müssen nicht gespeichert werden.

Ein wichtiges Ziel dieser Aufgabe ist es, dass Sie eine saubere objektorientierte Modellierung planen und umsetzen. Überlegen Sie sich, welche Klassen hierzu nötig sind und welche Verbindungen zwischen ihnen bestehen müssen. Achten Sie dabei auf die Trennung von Ein-/Ausgabe und Funktionalität sowie die anderen aus der Vorlesung bekannten Kriterien für gutes Design. Halten Sie sich an die Java Code Conventions und dokumentieren Sie Ihre Lösung sinnvoll mit Hilfe von javadoc Kommentaren.

Verwenden Sie das Interface `AverageGrade` für alle Klassen ihrer Implementierung, für die das Berechnen einer Durchnittsnote Sinn ergibt. Diese Datei müssen Sie **nicht mit einreichen**.

```
/**
 * Interface for classes with an average grade.
 */
public interface AverageGrade {

/**
     * Returns the average grade of this object. The average grade is a
     * non-negative double value. A negative value (e.g. -1) is returned,
     * if no average grade for the current object exists.
     * @return average grade of this object, or a negative number
     * (e.g. -1) if no average grade exists.
     */
    double averageGrade();

}
```

Die Kommandoschnittstelle

ACHTUNG: Da wir automatische Tests dieser Schnittstelle machen, müssen die Ausgaben Ihrer Shell **exakt** den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Grossbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Geben Sie auch keine zusätzlichen Informationen aus. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären.

Bei Fehlermeldungen dürfen Sie den Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung **muss** aber mit `"Error! "` beginnen und darf keine Zeilenumbrüche enthalten.

Formate der Ein- und Ausgaben

Bei der Ein- und Ausgabe in der Shell werden verschiedene Daten übergeben, z.B. Namen, Matrikelnummern, usw. Entspricht eine Eingabe nicht dem hier vorgegebenen Format, dann ist immer eine Fehlermeldung auszugeben. Danach soll das Programm auf die nächste Eingabe warten.

- Die **Namen** von Personen bestehen nur aus Kleinbuchstaben ohne Leerzeichen. Tutoren müssen immer unterschiedliche Namen haben, Studenten nicht. Mehrere Studenten dürfen denselben Namen haben. Ein Student darf auch denselben Namen wie ein Tutor haben.
- Studenten werden über ihre **Matrikelnummer** identifiziert, diese ist eine positive 5-stellige Zahl.
- Der **Text** von Aufgaben, Lösungen und Korrekturen ist eine Zeichenkette, die keine Leerzeichen, Tabulatoren und Zeilenumbrüche enthalten darf. Verwenden Sie "_" anstelle von Leerzeichen. Diese Einschränkung soll Ihnen das Einlesen der Befehlszeile erleichtern.
- Eine **Note** ist eine Zahl mit ganzzahligem Wert zwischen und inklusiv 1 und 5.
- Ausgaben von **nicht ganzzahligen Werten** (Notenschnitt) sind immer mit 2 Nachkommastellen zu formatieren. Runden Sie das Ergebnis, wie aus der Schule bekannt, mit der kaufmännischen Rundungsregel (z.B. 3.124 => 3.12 und 3.125 => 3.13). Ein Punkt "." trennt dabei Vor- und Nachkommastellen.
- Generell werden wir auf sämtliche Umlaute und Sonderzeichen bei unseren Tests verzichten. Folgende Zeichen sollten Sie, zusätzlich zu Buchstaben und Zahlen, einlesen können (es dürfen auch mehr sein): `()[].,-_?!+*#`

Befehle

Der Prompt der Shell ist "praktomat> ".

Ihre Shell muss mindestens folgende Befehle kennen:

tut <name>

Legt einen neuen Tutor mit Namen <name> an und wählt diesen aus. Falls ein Tutor mit diesem Namen bereits existiert, dann wird dieser ausgewählt und kein neuer Tutor angelegt.

stud <name> <matrnr>

Legt einen neuen Studenten mit Namen <name> und Matrikelnummer <matrnr> an. Gibt eine Fehlermeldung aus, falls es bereits einen Studenten mit der gleichen Matrikelnummer gibt. Der Student wird dem zuletzt ausgewählten Tutor zugewiesen. Falls noch kein Tutor ausgewählt wurde, ist eine Fehlermeldung auszugeben und der Student wird nicht angelegt.

task <text>

Legt eine neue Aufgabe mit dem Text <text> an. Der Aufgabe wird automatisch eine ID zugewiesen. Die ID ist eine positive natürliche Zahl, mit deren Hilfe die Aufgabe eindeutig identifiziert werden kann. Sie startet bei 1 und wird je angelegter Aufgabe um 1 erhöht. Die ID <taskid> der erzeugten Aufgabe wird dabei ausgegeben:

task id(<taskid>**)**

list-students

Gibt eine Liste aller Studenten, aufsteigend numerisch sortiert nach Matrikelnummer, aus. In jeder Zeile wird ein Student in folgendem Format ausgegeben:

(<matrnr>**,**<name>**):** <tut_name>

Dabei ist <name> der Name des Studenten, <matrnr> dessen Matrikelnummer und <tut_name> der Name des Tutors, der den Studenten betreut.

submit <taskid> <matrnr> <text>

Reicht eine Lösung mit dem Text <text> zu der Aufgabe mit der ID <taskid> für den Studenten mit der Matrikelnummer <matrnr> ein. Gibt eine Fehlermeldung aus, wenn es die Aufgabe oder den Studenten nicht gibt oder wenn für den Studenten bereits eine Lösung zu dieser Aufgabe eingereicht wurde.

review <taskid> <matrnr> <grade> <text>

Erstellt eine Korrektur für die Lösung der Aufgabe mit ID <taskid> des Studenten mit der Matrikelnummer <matrnr>. Die Korrektur enthält die Note <grade> (1-5 ganzzahlig) und einen Text <text>. Der Ersteller der Korrektur ist automatisch der Tutor, der den betreffenden Studenten betreut.

Falls der Student, die Aufgabe oder die Lösung nicht existieren, dann wird eine Fehlermeldung ausgegeben. Das Überschreiben einer existierenden Korrektur ist aber erlaubt.

Wenn die Korrektur erfolgreich erstellt oder überschrieben wurde, dann ist folgende Meldung auszugeben:

<tut_name> **reviewed (**<matrnr>**,**<name>**) with grade** <grade>

Dabei ist <tut_name> der Name des Tutors, der den Studenten betreut, <name> der Name des Studenten, <matrnr> dessen Matrikelnummer und <grade> die Note der Korrektur.

list-solutions <taskid>

Listet alle eingereichten Lösungen für die Aufgabe mit der ID <taskid> auf. Die Ausgabe ist dabei numerisch aufsteigend nach der Matrikelnummer des Studenten sortiert. Jede Zeile hat folgendes Format:

(<matrnr>**,**<name>**):** <text>

Dabei ist <name> der Name des Studenten, <matrnr> dessen Matrikelnummer und <text> der Text der eingereichten Lösung.

results

Gibt die Resultate für alle Aufgaben, numerisch aufsteigend sortiert nach ihrer ID, aus. Je Aufgabe ist dabei folgendes auszugeben:

- Eine Kopfzeile mit der ID <taskid> und dem Text der Aufgabe <task_text> im Format:

  task id(<taskid>**):** <task_text>

- Eine Liste der bewerteten Lösungen, numerisch aufsteigend sortiert nach Matrikelnummer. Eingereichte Lösungen ohne Bewertung sollen ignoriert werden. Jede Zeile der Ausgabe ist wie folgt zu formatieren:

  <matrnr>**:** <grade>

  Dabei ist <matrnr> die Matrikelnummer des betreffenden Studenten und <grade> die Note, mit der seine Lösung bewertet wurde.

summary-task

Gibt eine Zusammenfassung der Resultate der Aufgaben, aufsteigend sortiert nach ihrer ID, aus. Je Aufgabe ist dabei folgendes auszugeben:

- Eine Kopfzeile mit der ID <taskid> und dem Text der Aufgabe <task_text> im Format:

task id(<taskid>**):** <task_text>

- Eine Zusammenfassung im Format:

submitted: <#submitted>
reviewed: <#reviewed>
average grade: <avg>
distribution: <#grade1>**x1,** <#grade2>**x2,** <#grade3>**x3,** <#grade4>**x4,** <#grade5>**x5**

  - <#submitted> ist die Anzahl der eingereichten Lösungen und <#reviewed> die Anzahl der korrigierten Lösungen.
  - <avg> ist der Notendurchschnitt aller korrigierten Lösungen. Dieser ist stets mit exakt 2 Nachkommastellen formatiert und verwendet einen Punkt "." zur Trennung von Vor- und Nachkommastellen. Also z.B. 3.14 oder 1.00. Verwenden Sie die kaufmännische Rundung, wie zuvor im Abschnitt "Formate der Ein- und Ausgaben" beschrieben. **Achtung:** Wenn noch keine Korrekturen für die entsprechende Aufgabe vorliegen, dann geben Sie statt dessen ein Minuszeichen "-" aus.
  - <#grade**N**> ist die Anzahl der Lösungen, die mit der Note **N** bewertet wurden.

summary-tutor

Gibt Informationen zu den einzelnen Tutoren aus. Die Ausgabe wird alphabetisch aufsteigend nach dem Namen des Tutors sortiert. Je Tutor ist folgendes auszugeben:

<name>**:** <#studs> **students,** <#todo> **missing review(s), average grade** <avg>

Dabei ist <name> der Name des Tutors, <#studs> die Anzahl der betreuten Studenten und <#todo> die Anzahl der noch zu bewertenden Lösungen, d.h. die Anzahl der eingereichten, aber noch nicht korrigierten Lösungen (aller Aufgaben) der Studenten, die der Tutor betreut. <avg> ist der Durchschnitt aller Noten, die der Tutor vergeben hat. Behandeln Sie die Ausgabe des Notenschnitt wie zuvor beim **summary-task**-Befehl beschrieben.

summary-student

Gibt Informationen zu den einzelnen Studenten aus. Die Ausgabe wird numerisch aufsteigend nach dem gerundeten Notenschnitt des Studenten sortiert. Runden Sie den Notenschnitt mit der kaufmännischen Rundungsregel auf 2 Stellen nach dem Komma. Bei gleichem gerundeten Notenschnitt wird numerisch aufsteigend nach Matrikelnummer sortiert. Die Studenten ohne Note sind numerisch aufsteigend sortiert nach Matrikelnummer am Ende der Liste auszugeben. Je Student ist folgendes auszugeben:

(<matrnr>**,**<name>**):** <avg>

Dabei ist <matr> die Matrikelnummer des Studenten, <name> dessen Name und <avg> der Notenschnitt aller korrigierten Lösungen des Studenten. Behandeln Sie die Ausgabe des Notenschnitt wie zuvor beim **summary-task**-Befehl beschrieben.

reset

Initialisiert den Praktomat neu. Der Zustand des Praktomaten ist danach wie bei einem Neustart des Java-Programms.

quit

Beendet das Programm.

Beispiel

Beispiel eines Programmablaufs:

```
praktomat> tut florian
praktomat> stud bernd 23442
praktomat> stud andrea 12345
praktomat> stud hans 32000
praktomat> tut martin
praktomat> stud tina 66666
praktomat> stud tina 43222
praktomat> stud georg 12345
Error! a student with id 12345 already exists.
praktomat> task Abschlussaufgabe_Praktomatmodell
task id(1)
praktomat> task Dies_ist_eine_Zusatzaufgabe
task id(2)
praktomat> list-students
(12345,andrea): florian
(23442,bernd): florian
(32000,hans): florian
(43222,tina): martin
(66666,tina): martin
praktomat> review 1 66666 2 Ganz_gut_geloest,_weiter_so!
Error! (66666,tina) has not submitted a solution to task id(1)
praktomat> submit 1 66666 Die_Loesung_ist_42
praktomat> review 1 66666 2 Ganz_gut_geloest,_weiter_so!
martin reviewed (66666,tina) with grade 2
praktomat> submit 1 23442 Glaub_es_ist_42
praktomat> submit 1 32000 Keine_Ahnung
praktomat> list-solutions 1
(23442,bernd): Glaub_es_ist_42
(32000,hans): Keine_Ahnung
(66666,tina): Die_Loesung_ist_42
praktomat> review 1 32000 5 Das_war_wohl_nix.
florian reviewed (32000,hans) with grade 5
```

```
praktomat> results
task id(1): Abschlussaufgabe_Praktomatmodell
32000: 5
66666: 2
task id(2): Dies_ist_eine_Zusatzaufgabe
praktomat> summary-task
task id(1): Abschlussaufgabe_Praktomatmodell
submitted: 3
reviewed: 2
average grade: 3.50
distribution: 0x1, 1x2, 0x3, 0x4, 1x5
task id(2): Dies_ist_eine_Zusatzaufgabe
submitted: 0
reviewed: 0
average grade: -
distribution: 0x1, 0x2, 0x3, 0x4, 0x5
praktomat> summary-tutor
florian: 3 students, 1 missing review(s), average grade 5.00
martin: 2 students, 0 missing review(s), average grade 2.00
praktomat> summary-student
(66666,tina): 2.00
(32000,hans): 5.00
(12345,andrea): -
(23442,bernd): -
(43222,tina): -
praktomat> reset
praktomat> results
praktomat> quit
```

Allgemeine Hinweise zur Implementierung

Die Anzahl der Studenten, Tutoren und Aufgaben wird zur Laufzeit des Programms verändert. Ihre Implementierung sollte daher keine feste Maximalanzahlen annehmen, sondern mit beliebig vielen Studenten, Tutoren und Aufgaben umgehen können.

In Ihrer Implementierung dürfen Sie Klassen aus den Paketen `java.lang.*` und `java.util.*` verwenden. Insbesondere sind also Klassen aus dem Java Collections Framework erlaubt, z.B. `LinkedList`, `TreeMap` oder `HashMap`. Beachten Sie auch, dass Java eine einfache Möglichkeit zum Sortieren von Listen über `Collections.sort()` zur Verfügung stellt.

Sie dürfen ebenfalls die aus der Übung bekannte Klasse `Terminal` verwenden.

Zum Formatieren von Gleitkommazahlen dürfen Sie die Klasse `DecimalFormat` verwenden:

```
DecimalFormat df = new DecimalFormat("0.00", DecimalFormatSymbols.getInstance(Locale.US));
 df.setRoundingMode(RoundingMode.HALF_UP);
 System.out.println(df.format(2.32123322));
```

Abgabe der Lösung im Praktomat

Reichen Sie bei der Abgabe Ihrer Lösung in jedem Fall folgende Dateien mit ein:

- Eine Datei `Shell.java`, die die main-Methode enthält.
- Eine Datei `Tests.txt`, die eigene Testfälle enthält. Dokumentieren Sie die Testfälle in Form von Programmabläufen, wie im obigen Beispiel.

Reichen Sie das Interface `AverageGrade` sowie die Klasse `Terminal` **nicht** mit ein.

Achten Sie bei der Bearbeitung dieser Abschlussaufgabe auf

- korrekte Java-Syntax
- Einhaltung der Java Code Conventions
- Einrückungen
- Einhaltung vorgegebener Interfaces
- Umsetzung des Geheimnisprinzips
- ausführliche Dokumentation des Quelltextes
- adäquates Verhalten öffentlicher Methoden bei ungültigen Parametern
- sinnvolle Modellierung der Klassenstruktur

Praktomat wird für diese Abschlussaufgabe folgende automatische Prüfungen auf Ihren Lösungen ausführen. **Fett** markierte Prüfungen müssen bestanden werden, damit die Lösung eingereicht werden kann:

- **Maximale Zeilenbreite:** 120 Zeichen
- **Übersetzen (Java)**
- **Checkstyle WS (1/2)**: Überprüfen korrekter Whitespace-Setzung
- **Checkstyle JCC (2/2)**: Überprüft, ob Namenskonventionen eingehalten wurden und ob die eingereichten Klassen mit Javadoc-Kommentaren versehen wurden.
- Funktionalitätstest: Testet die Funktionalität Ihres Programmes.
  - **Einfache Tests**: Testet das Programm auf minimale Funktionalität.
  - Erweiterte Tests: Schwierigere und umfassendere Tests.

Bewertung Ihrer Abgabe

Beide Aufgaben werden jeweils in den Kategorien *Funktionalität* und *Programmiermethodik* mit einer Punktezahl von 7 bis 0 bewertet. Zum Bestehen der Abschlussaufgaben müssen

- beide Aufgaben erfolgreich in Praktomat eingereicht sein (d.h. die oben fett markierten Prüfungen müssen bestanden werden) **und**
- der Durchschnitt über beide Aufgaben in *Funktionalität* besser oder gleich 4 Punkte sein **und**
- der Durchschnitt über beide Aufgaben in *Programmiermethodik* besser oder gleich 4 Punkte sein.

Bei der Berechnung der *Endnote* werden die Punkte in *Funktionalität* **doppelt** gewichtet.

Bei Abschreiben werden beide Abschlussaufgaben mit **nicht bestanden** bewertet. Auch dann, wenn nur eine der Aufgaben abgeschrieben wurde.

Beispiel 1:

- **Aufgabe 1**: Funktionalität: 7, Programmiermethodik: 5
- **Aufgabe 2**: Funktionalität: 5, Programmiermethodik: 3

Es gilt also für den Durchschnitt: **Funktionalität**: 6, **Programmiermethodik**: 4 ⇒ Bestanden!

Beispiel 2:

- **Aufgabe 1**: Funktionalität: 7, Programmiermethodik: 3
- **Aufgabe 2**: Funktionalität: 7, Programmiermethodik: 4

Es gilt also für den Durchschnitt: **Funktionalität**: 7, **Programmiermethodik**: 3,5 ⇒ **Nicht** bestanden!

Beispiel 3:

- **Aufgabe 1**: Funktionalität: 5, Programmiermethodik: 1
- **Aufgabe 2**: Funktionalität: 6, Programmiermethodik: 3

Es gilt also für den Durchschnitt: **Funktionalität**: 5,5, **Programmiermethodik**: 2 ⇒ **Nicht** bestanden!

Beispiel 4:

- **Aufgabe 1**: Funktionalität: 6, Programmiermethodik: 7
- **Aufgabe 2**: Funktionalität: 7, Programmiermethodik: 2

Es gilt also für den Durchschnitt: **Funktionalität**: 6,5, **Programmiermethodik**: 4,5 ⇒ Bestanden!