Proseminar Programmiermethodik
Universität Innsbruck - Institut für Informatik
B. Murauer, D. Silbernagl, M. Tschuggnall

13. Juni 2017

Übungsblatt 9

Aufgabe 1 (Debugging – 2 Punkte)

The most effective debugging tool is still careful thought, coupled with judiciously placed print statements. — Brian Kernighan, 1979
Das ist zum Glück schon eine Weile her, und mittlerweile haben wir durchaus einige Verbesserungen bezüglich unserer Debugging-Tools erfahren. Eclipse bietet einen sehr brauchbaren Debugger an, der eine Vielzahl von bequemeren Möglichkeiten bietet, als in jeder Zeile ein `System.out.println` zu schreiben. Unter anderem gibt es verschiedene Möglichkeiten, Breakpoints zu setzen. Machen Sie sich mit den verschiedenen Arten von Breakpoints vertraut und sehen Sie sich den Code in den Dateien `Course.java`[OLAT] und `Student.java`[OLAT] an. In `Course` gibt es eine main-Methode, die aus einer Datei `students.data`[OLAT] Studenten generiert.
Anschließend wird der Mittelwert der Kursnoten berechnet und ausgegeben.

Ein mäßig motivierter Mitarbeiter hat aber anscheinend bei der Eingabe der Daten ein bisschen geschlafen und diese sind dadurch fehlerhaft. Finden Sie mit Hilfe des Debuggers die unten beschriebenen Datensätze und korrigieren Sie diese wie angegeben. Geben Sie Ihre Angaben als Textdatei[1] ab.

Hinweis: *Für diese Aufgabe muss die Klasse 'Student' im default-Package liegen, da die Datensätze aus Serialisierten Student-Objekten besteht, deren Package-Name sich nicht verändern darf.*
Hinweis: *Verändern Sie nichts an den Dateien, außer die Aufgaben verlangen es explizit. Oft wird man gegebenen Code debuggen, der nicht veränderbar ist (z.B. Java-Lang-Files), um dessen Funktionalität besser zu verstehen.*

a) 1 Punkt - Beim ersten Aufrufen der Main-Methode fliegt gleich eine Nullpointer-Exception. Aber wo? Finden Sie die ID des Eintrags, bei dem eine Nullpointer-Exception geworfen wird mit Hilfe eines Exception-Breakpoints. Ändern Sie anschließend die `getGrade`-Methode, dass statt eines `null`-Wertes der Wert 3 zurückgegeben wird (aber nur, wenn `grade` auch wirklich null ist). Notieren Sie den Mittelwert der Noten.

b) 0.5 Punkte - Bei einem Eintrag entspricht die vergebene Note nicht dem Österreichischen Notensystem (liegt also nicht zwischen 1 und 5). Finden Sie den Vor- und Nachnamen des Eintrags, indem Sie einen Breakpoint mit Conditions verwenden.

c) 0.5 Punkte - Sie können mit dem Debugger während der Laufzeit Variablen verändern. Nutzen Sie diese Möglichkeit und bessern Sie die Note des betroffenen Eintrags aus Aufgabe b) auf "2" aus. Wie lautet jetzt der Notendurchschnitt des Kurses?

[1] TEXT-Datei. Bevorzugt UTF-8-kodiert. ASCII-Art willkommen, Umlaute im Dateinamen nicht.

Aufgabe 2 (Refactoring – 2 Punkte)

Neben den Debugging-Funktionen bietet Eclipse auch sehr nützliche Tools zum Umstrukturieren von Code an. Verschaffen Sie sich einen Überblick über die verfügbaren Funktionen und vergleichen Sie diese mit Informationen über Refactoring auf `https://refactoring.com` und `https://sourcemaking.com/refactoring`. Nach einer Inspektion des Codes in `Refactor.java` (OLAT) steht fest: er ist kaum lesbar. Aber er produziert hübsche ASCII-Bildchen auf der Konsole. Falls Sie Schwierigkeiten haben, die Bedeutung der einzelnen Variablen zu verstehen, empfiehlt es sich die Wiki-Seite der Mandelbrot-Menge[2] durchzulesen. Mit dem vollen Repertoire an Methoden im Gepäck können Sie nun mit beliebig schwerwiegenden Änderungen den Code verändern, um folgende Punkte zu erreichen:

a) 1 Punkt - Im Code werden sehr offensichtlich und ganz eindeutig komplexe Zahlen verwendet. Finden Sie sie, ergänzen Sie die Aufgabe um eine separate Klasse für komplexe Zahlen und implementieren Sie Methoden zum Quadrieren und Auslesen des Betrags, die jeweils chaining[3] unterstützen. Ersetzen Sie dann alle möglichen Operationen und Variablen mit den entsprechenden Versionen der komplexen Zahl.

b) 1 Punkt - Strukturieren Sie den Code so um, dass folgende Punkte erfüllt sind:
   - Keine Methode darf mehr als 15 Zeilen lang sein
   - Alle Parameter der Ansicht der Mandelbrot-Menge (also alle Variablen die die Position und Zoom-Level des angezeigten Ausschnitts definieren) müssen als Felder vorliegen. Geben Sie allen Variablen sinnvolle(re) Namen.

Aufgabe 3 (Refactoring – 6 Punkte)

In den beiden Dateien `Game.java` (OLAT) und `Player.java` (OLAT) ist ein einfaches Schere-Stein-Papier-Spiel implementiert. Für die folgenden Aufgaben können Sie den Code beliebig ändern, durch neue Klassen ergänzen, etc.

a) 3 Punkte - Ändern Sie den Code, damit folgende Punkte erfüllt sind:
   - Keine Methode darf länger als 15 Zeilen lang sein.
   - Die Anzahl der verfügbaren Optionen soll möglichst variabel gehalten sein. Es soll zum Beispiel **auf eine einfache Weise** möglich sein, das klassische Schema um die zwei Optionen *Lizard* und *Spock*[4] und die entsprechenden Gewinnkombinationen zu erweitern.

b) 1 Punkt - Ändern Sie den Code so, dass dieser interaktiv wird und die Wahl der Optionen über eine Eingabe erfolgt.

c) 2 Punkte - Ändern Sie den Code so, dass ein Spieler entweder gegen einen anderen menschlichen Spieler oder gegen eine einfache KI[5] spielen kann, die zufällig eine Option auswählt. Überlegen Sie sich, welche Möglichkeiten der Objektorientierung, die Sie bereits kennen, am besten geeignet ist (evtl auch für zukünftige Anforderungen). Die Wahl des Gegnertyps soll dabei am Spielanfang über eine Eingabe erfolgen.

Wichtig: Laden Sie bitte Ihre Lösung (.txt, .java oder .pdf) in OLAT hoch und geben Sie mittels der Ankreuzliste auch unbedingt an, welche Aufgaben Sie gelöst haben. Die Deadline dafür läuft am Vortag des Proseminars um 23:59 (Mitternacht) ab.

[2] `http://de.wikipedia.org/wiki/Mandelbrot-Menge`
[3] `https://en.wikipedia.org/wiki/Method_chaining`
[4] `http://bigbangtheory.wikia.com/wiki/Rock_Paper_Scissors_Lizard_Spock`
[5] `https://de.wikipedia.org/wiki/Künstliche_Intelligenz`