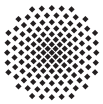

Universität Stuttgart

RSS
Reliable Software Systems

Übungsblatt 13

Programmierung und Softwareentwicklung (WiSe 2019/2020)

Abgabe: Fr. 31.01.2020, 23:59 Uhr — Besprechung: KW 6

- Bitte lösen Sie die Übungsaufgabe in **Gruppen von 2 Studenten**.
- Dieses Übungsblatt besteht aus zwei Teilen (A+B). Teil A ist in der Präsenzübung zu lösen. Teil B ist in Heimarbeit (Gruppe von 2 Studenten) zu lösen und rechtzeitig abzugeben.
- Geben Sie .java-Dateien nur im UTF-8 Encoding ab. Ändern Sie das Textdateiencoding von Eclipse auf UTF-8 ab, bevor Sie die Unterlagen herunterladen. Abhängig von Ihrem Betriebssytem müssen Sie möglicherweise auch nichts tun. (Tutorial: `https://youtu.be/07Rj8jw8cE8`)
- Geben Sie zu Beginn der Dateien Ihre Namen (Vor- und Nachname), die Matrikelnummern und die E-Mail-Adressen an. Nutzen Sie bei Java-Dateien die korrekte JavaDoc-Syntax.
- Benennen Sie die Dateien nach dem folgenden Schema:
  1. **PSE[ÜB-Nr]-[Nachnamen der Teammitglieder]-[Nachname des Tutors].pdf**
  2. ***.java**: Alle Java-Dateien, die die Tests für Aufgabe 2 enthalten

Lernziel: Dieses Übungsblatt dient dazu, dass Sie die in der Vorlesung vorgestellten Konzepte zum Testen anhand von Äquivalenzklassen und JUnit vertiefen und anwenden. Zudem kennen Sie bereits ein umfangreiches Set von Stilrichtlinien, welche Sie anwenden, üben und vertiefen sollen.

Vorbereitung: Stellen Sie sicher, dass Sie sich mit den Vorlesungsunterlagen zum Testen vertraut gemacht haben und die Übungsblätter 11 und 12 absolviert haben.

Punkte: Dieses Übungsblatt enthält zwei Teile. Teil A mit 2 Aufgaben und Teil B mit 2 Aufgaben. Im Teil B können Sie bis zu 41 Punkte erzielen.

Style: Bitte halten Sie die in der Vorlesung vorgestellten Style-Regeln ein. Dazu gehören auch JavaDoc sowie Vor- und Nachbedingungen. Der Style Ihrer Implementierung wird mit bis zu 50% bewertet.

Unterlagen: Alle relevanten Unterlagen, für die Bearbeitung des Übungsblatts finden Sie wie gehabt in unserem git-Repository: `https://github.com/RSS-PSE-WS1920/`

UML Cheat Sheets: Kurzübersichten der UML-Notation für Klassen-, Sequenz- und Objektdiagramme finden Sie im ILIAS-Ordner zu den Übungsblättern.

Viel Erfolg!

1 Teil A - Präsenzaufgaben

Aufgabe 1 Black-Box-Testen

Es sollen Black-Box-Testfälle für die im Folgenden angegebenen Operationen aus der Klasse `java.lang.Math` aus dem JDK definiert werden, die eine vollständige Abdeckung des Eingabebereichs basierend auf Äquivalenzklassen erfüllen. Geben Sie jeweils die Menge der Testfälle an, in dem Sie die Äquivalenzklasse kurz beschreiben sowie einen Repräsentanten und das erwartete Ergebnis definieren. Orientieren Sie sich an der Dokumentation, die Sie unter `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Math.html` finden.

- `public static int abs(int a)`
- `public static int max(int a, int b)`
- `public static int incrementExact(int a)`
- `public static double sqrt(double a)`

Aufgabe 2 Testen mit JUnit

In dieser Aufgabe sollen Tests mittels JUnit 4 erstellt und ausgeführt werden.

(a) Importieren Sie das gegebene Projekt in Eclipse. Im Ordner `test/` ist eine Klasse `TestJavaMath` enthalten. Darin befindet sich bereits ein JUnit-Test für die Operation `java.lang.Math.min(int a, int b)`. Machen Sie sich anhand dieser Klasse mit dem generellen Aufbau von JUnit-basierten Testklassen sowie dem Ziel des gegebenen Testfalls vertraut.

(b) Unter `http://www.vogella.com/tutorials/JUnit/article.html` finden Sie eine Beschreibung der Funktionsweise von JUnit und der Benutzung innerhalb von Eclipse. Machen Sie sich anhand dieser Dokumentation mit der Funktionsweise vertraut und führen Sie den in der Klasse `TestJavaMath` gegebenen Testfall aus.

(c) Implementieren Sie mindestens fünf Ihrer Testfälle aus Aufgabe 1 mittels JUnit (durch Hinzufügen zur Klasse `TestJavaMath`) und führen Sie diese aus.

(d) Betrachten Sie nun die Klasse `TestPerson`, die zwei Testfälle für die ebenfalls gegebene Klasse `Person` enthält. Führen Sie die Tests aus und Sie werden sehen, dass beide erfolgreich sind.

(e) Die Klasse `Person` ist fehlerhaft. Begeben Sie sich auf Fehlersuche im Konstruktor der Klasse `Person`. Warum wurde der Fehler mit den Tests aus `TestPerson` nicht erkannt?

(f) Erstellen Sie einen Testfall in der Klasse `TestPerson`, der aufgrund des eben festegestellten Programmierfehlers bei der Ausführung fehlschlägt.

(g) Korrigieren Sie den Programmierfehler und führen Sie den entsprechenden Testfall erneut durch. Dieser sollte nun erfolgreich sein.

2 Teil B - Heimarbeit

Aufgabe 1 Vererbung und Polymorphie (15 Punkte)

Erstellen Sie für die unten aufgeführten Bedingungen eine Vererbungshierarchie als UML-Klassendiagramm. Die Vererbungshierarchie soll die minimale Lösung der nachfolgenden Vererbungsbedingungen enthalten. Verwenden Sie nur die angegebenen Klassen.
Achten Sie bei Ihrer Abgabe auf Lesbarkeit. Sie können Ihre Abgabe zum Beispiel mit dem Online-Tool *draw.io* (`https://www.draw.io/`) erstellen.

Beispiel:

1. Es gibt die Klassen Bix, Foo, Quaz sowie die Schnittstelle (Inteface) Bar.
2. Foo ist ein Subtyp von Bar.
3. Quaz ist ein Subtyp von Foo.

Die Bedingungen des Beispiels ergeben folgendes Klassendiagramm 1:

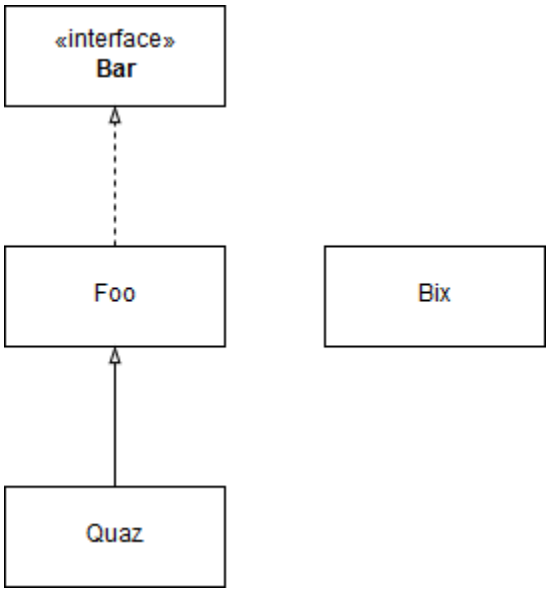

Abbildung 1: Beispiel des Klassendiagramms zu den Bedingungen 1, 2 und 3

Die Bedingungen der Aufgabe sind:

1. A und B sind abstrakte Klassen.
2. Es kann sowohl Entitäten vom Typ G als auch vom Typ H geben.
3. H und F implementieren kein Interface.
4. B ist Subtyp von C und F.
5. D ist Subtyp von B, E und H.
6. G ist Subtyp von C.
7. B ist kein Subtyp von einer der angegebenen Klassen.
8. Alle Objekte können Variablen vom Typ C zugewiesen werden.
9. D implementiert genau ein Interface direkt.
10. B ist kein Subtyp von H.
11. Alle Objekte vom Typ D sind auch vom Typ F.
12. E ist kein Subtyp von B.
13. Es gibt keine redundanten Vererbungen.

Aufgabe 2 Unit-Tests

In dieser Aufgabe geht es um die Definition von Testfällen, den Begriff der Testabdeckung, sowie die Erstellung und Ausführung von Tests mittels JUnit 4.

(a) *(9 Punkte)* Es sollen Black-Box-Testfälle für den folgenden Konstruktor der Klasse `Hamster` definiert werden. Ziel ist die Abdeckung des Eingabebereichs basierend auf Äquivalenzklassen.

```
/**
 * Create and initialize a new hamster object with the given
 * parameters.
 * @param territory The territory this hamster lives in.
 * @param location The location in the territory, where the
 *                 hamster starts.
 * @param direction The direction in which the hamster looks
 *                  initially.
 * @param newGrainCount The number of grain objects initially
 *                      placed into the hamster's mouth.
 */
public Hamster(final Territory territory, final Location location,
               final Direction direction, final int newGrainCount)
```

Geben Sie sechs Testfälle an, indem Sie die Äquivalenzklasse kurz beschreiben, sowie einen Repräsentanten (Beispielwerte für alle Parameter des Konstruktors) und das erwartete Ergebnis definieren.

(b) *(4 Punkte)* Der Quellcode der Opertation `putGrain` der Klasse `Hamster` sieht wie folgt aus:

```
/**
 * Drop a random grain object from the hamster's mouth.
 * @throws MouthEmptyException when the hamster does not carry
 * any grain.
 */
public void putGrain() {
    if (this.internalHamster.getGrainInMouth().isEmpty()) {
        throw new MouthEmptyException();
    }
    this.game.processCommandSpecification(
            new PutGrainCommandSpecification(
                    this.internalHamster,
                    this.internalHamster.getGrainInMouth().get(0)));
}
```

Skizzieren Sie eine Menge von Testfällen, mit der eine vollständige Anweisungsüberdeckung erreicht wird. Begründen Sie kurz.

(c) *(1 Punkt)* Importieren Sie das gegebene Projekt in Eclipse. Im Ordner `test` ist eine Klasse `TestHamster` enthalten. Darin befindet sich bereits ein JUnit-Test für den Hamster (`testConfiguredHamsterOnTerritory`). Machen Sie sich mit dem Aufbau des Tests vertraut. Führen Sie den Test mittels JUnit in Eclipse aus. Sie werden feststellen, dass der Test fehlschlägt, was an einem Fehler im Testcode liegt. Beheben Sie den Fehler. Überprüfen Sie Ihre Korrektur durch erneute Ausführung, welche nun erfolgreich sein sollte.

(d) Es sollen Testfälle für die Operationen `move()` und `pickGrain()` der Klasse `Hamster` definiert und implementiert werden. Ziel ist die Überdeckung bezüglich der Operationen-Spezifikation mittels Äquivalenzklassen.

i. *(6 Punkte)* Geben Sie entsprechende Testfälle an, indem Sie den benötigten Zustand des Hamsters vor der Ausführung der getesteten Operation skizzieren und das erwartete Ergebnis definieren.

ii. *(6 Punkte)* Implementieren Sie für die Operationen `move()` und `pickGrain()` jeweils zwei Ihrer Testfälle mittels JUnit. Hierzu legen Sie nach dem Schema des Beispieltests zunächst ggf. ein geeignetes Territorium an.

3 Teil L - Lehramtsteil

Dieser Teil enthält eine Präsenzaufgabe die speziell für Lehramtsstudierende konzipiert ist. Das bedeutet allerdings nicht, dass "Nicht-Lehramtsstudierende" diese nicht auch bearbeiten dürfen. Sie bezieht sich auf den Vorlesungsstoff und die Übungsaufgaben, betrachtet diese aber aus einer didaktischen Perspektive.

Aufgabe 1 Testen in der Schule

Teil der Softwareentwicklung sind Tests, die automatisch bestimmte Fehlerquellen testen und ggf. erkennen können. Diese Aufgabe befasst sich mit den Black-Box-Tests und JUnit im Unterricht.

(a) Ein wichtiger Schritt beim Bestimmen von Testfällen ist das Einordnen in Äquivalenzklassen. In der Schulmathematik werden Äquivalenzklassen (zumindest in Baden-Württemberg) aber nicht angesprochen. Diskutieren Sie mit Ihrem Team, wie man den Schülerinnen und Schülern das Einteilen in solche trotzdem beibringen könnte.

(b) Bei der Einteilung in die Äquivalenzklassen gibt es Grenzfälle. Diskutieren Sie mit Ihrem Team, wie man den Schülerinnen und Schüler aufzeigen kann, dass das Testen der Grenzfälle ebenfalls wichtig ist.

(c) Gehen Sie davon aus, dass Sie im Unterricht die Testklassen, wie bei den anderen Aufgaben dieses Blatter, mit JUnit implementieren. Diskutieren Sie mit Ihrem Team, welche JUnit-Operationen für den Unterricht wichtig sind und wie man diese einführen könnten. Überlegen Sie sich außerdem, welche Befehle man im Unterricht eventuell weglassen könnte und warum (*"didaktische Reduktion"*).

Hinweise zu den Übungen:

- Die Abgabe erfolgt im ILIAS.
- Durch die Teilnahme am Übungsbetrieb können Sie sich für die Teilnahme an der Klausur qualifizieren.
  - Bestehen von min. 80% aller Übungsblätter.
  - Ein Übungsblatt gilt als bestanden, wenn 50% der Punkte erreicht wurden.
  - Teilnahme an min. 80% der Übungen.
  - Bestehen der Scheinklausur.

Viel Erfolg!