

Aufgabe 11-1: Durchsichten

Auf der letzten Seite finden Sie eine Checkliste zur Durchsicht von Java-Code.

- a) Gegen welche der dort aufgeführten Prüfpunkte (*checks*) verstoßen Sie selbst gelegentlich bei Ihrer Programmierarbeit?
- b) Nennen Sie mindestens einen Punkt, der nicht automatisiert geprüft werden kann und erklären Sie, warum das nicht geht.
- c) Nennen Sie mindestens einen Punkt, der Defekte aufdeckt, deren potenziellen Auswirkungen (= Versagensfälle) durch Testen schwer zu entdecken sind und erläutern Sie diesen.
- d) Welche Vorteile haben Durchsichten im Allgemeinen im Vergleich zu dynamischen Methoden, also Tests? Beschreiben Sie mindestens drei solcher Vorteile.

Aufgabe 11-2★: Testtechniken anwenden und bewerten

Im Jahr 1968 hat Edsger Dijkstra eine kurze, aber berühmt gewordene Notiz über die Gefährlichkeit von Sprunganweisungen in Programmiersprachen unter dem Titel "*Goto considered harmful*" verfasst.

- a) Eine Internetsuche wird Sie diesen Artikel schnell finden lassen. Lesen Sie ihn.
- b) Charakterisieren Sie seine Argumentation und geben Sie sie in groben Zügen wieder. Stimmen Sie ihr zu und finden Sie sie überzeugend?

Aufgabe 11-3^Δ: Testfallerstellung mit Äquivalenzklassen

Eine Form von Black-Box-Tests sind Tests mit Äquivalenzklassen. Als Äquivalenzklassen bezeichnet man beim Software-Testen *disjunkte* Mengen von Programmeingaben, deren jeweilige Werte zu einem *gleichartigen* Verhalten der Software führen. Da es sich um Black-Box-Tests handelt, ist hier gleichartiges *vermutetes* Verhalten gemeint.

Für das folgende Problem wurde bereits eine Software entwickelt. Sie sind mit einem Black-Box-Tests beauftragt. Da Ihnen die Gesamtzahl aller möglichen Testfälle zu hoch ist, entschließen Sie sich, mittels Äquivalenzklassenbildung eine Auswahl vorzunehmen.

Die Benotung eines Uni-Kurses setzt sich aus der Bepunktung zweier Klausuren zusammen. Bei der ersten Klausur waren maximal 40 Punkte zu erreichen; bei der zweiten 60 Punkte. Es gilt folgendes Bewertungsschema:

- Wer in einer der beiden Klausuren weniger als 20 Punkte erreicht, der fällt durch (Note F).
- Wer in beiden Klausuren je mindestens 20 Punkte erreicht, erhält mindestens die Note D. (Es gibt keine Note E.)
- Ab insgesamt 60 Punkten gibt es die Note C.
- Ab insgesamt 75 Punkten wird die Note B vergeben.
- Die Bestnote A erhält, wer mindestens 90 Punkte in der Summe beider Klausuren erhält.

Es werden nur ganze Punkte vergeben. Student/inn/en schreiben immer beide Klausuren mit.

Das Softwaresystem erwartet zwei ganzzahlige Eingaben für die Punktzahlen der beiden Klausurergebnisse und liefert eine Note zurück.

- a)** Halten Sie sich an die obige Definition und definieren Sie Äquivalenzklassen für dieses Problem. Erläutern Sie insbesondere Ihre zugrundeliegenden *Vermutungen*.

Achten Sie darauf, dass Ihre Äquivalenzklassen den Eingaberaum vollständig abdecken. Wenn Sie dabei unsicher sind, machen Sie sich eine grafische Skizze des zweidimensionalen Raums, der durch die beiden Bepunktungen aufgespannt wird.

Sie können davon ausgehen, dass nur gültige Eingaben (also z.B. nicht mehr als die jeweilige Maximalpunktzahl, keine negativen Eingaben) erfolgen.

- b)** Formulieren Sie für jede Ihrer Äquivalenzklassen genau einen Testfall.

Eine Erweiterung der Äquivalenzklassen-Methode ist die *Grenzwertbetrachtung*. An der Grenze zwischen zwei Äquivalenzklassen „schlägt“ das (vermutete) Verhalten der Software um: Das hieße in einem eindimensionalen Fall z.B. dass der Wert $alter=17$ noch in die Klasse „Kind“ gehört, während der um eins größere Wert $alter=18$ in die Klasse „Erwachsene/r“ fällt.

Aus einer eindimensionalen Grenzfallbetrachtung würden hier also zwei Testfälle entstehen. Bei *mehrdimensionalen* Äquivalenzklassen (in dieser Aufgabe: zwei Dimensionen) werden die Grenzen komplizierter.¹ Anstelle einer vollständigen Grenzwertbetrachtung beschränken wir uns hier die einfachen Übergänge zwischen je zwei Äquivalenzklassen.

- c)** Verschaffen Sie sich einen Überblick über die Grenzverläufe Ihrer Äquivalenzklassen aus Aufgabe **a)**. Definieren Sie für jeden ununterbrochenen Grenzverlauf zwischen zwei Klassen genau zwei Testfälle: Einen für die erste, einen für die zweite Klasse.

Zur Veranschaulichung sehen Sie hier eine Darstellung von zweidimensionalen Äquivalenzklassen (Rechtecke) mit den hervorgehobenen Testfällen (Kreise):

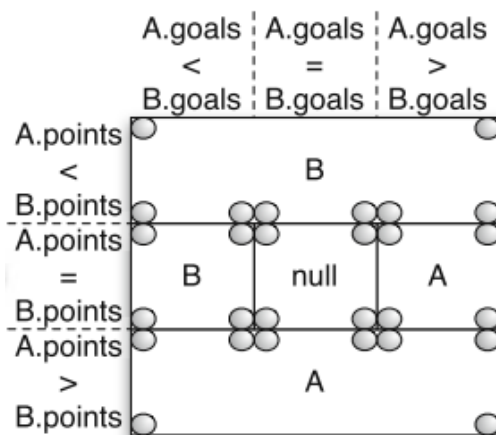
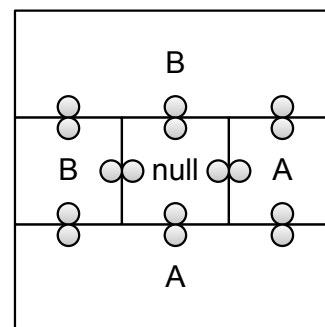


Abbildung 4.17 aus Hoffmann (2013), mit vollständiger Grenzfallbetrachtung



Analoges Szenario, mit vereinfachter Grenzfallbetrachtung

- d)** Installieren Sie die „Eclipse IDE for Java Developers“ (<http://www.eclipse.org>) auf Ihrem Computer. Laden Sie sich das vorbereitete Java-Projekt *grader-project.zip* aus dem KVV herunter und importieren Sie es in Eclipse („Import existing project“ und wählen Sie die ZIP-Datei aus). Die oben beschriebene Funktionalität ist in der Methode `Grader.grade(int firstExam, int secondExam)` implementiert; sie liegt nur als Bibliothek, nicht aber im Quellcode vor.

Implementieren Sie die Testfälle der Aufgaben **b)** und **c)** als JUnit-Testfälle in der vorhandenen Testklasse `GraderTest.java`. Gruppieren Sie Ihre Testfälle sinnvoll in Testmethoden und benennen Sie diese aussagekräftig. Geben Sie Ihre Testklasse mit ab.

Führen Sie die Testfälle aus: Haben sie Versagen aufgedeckt? Falls ja, stellen Sie mindestens zwei Hypothesen über den/die zu Grunde liegende/n Defekt/e auf.

¹Lesen Sie bei Bedarf z.B. Abschnitt 4.3.1 aus Hoffmanns „Software-Qualität“ (2013) nach; dort wird allerdings die hier relevante Aspekt von abhängigen Parametern ausgeklammert. Diese Lücke schließen können Sie bei Interesse mit Abschnitt 4.2 von Kleukers „Qualitätssicherung durch Softwaretests“ (2013). Beide Bücher sind aus dem FU-Nutz unter <http://link.springer.com> im Volltext verfügbar.

Checkliste zur Durchsicht von Java-Programmen

Arithmetik

1. Sind Überlauf oder Unterlauf während der Berechnung möglich?
2. Für jeden Ausdruck mit mehr als einem Operator: Sind die Annahmen über die Ausführungsreihenfolge korrekt?
3. Wurden Klammern eingesetzt um Mehrdeutigkeit zu vermeiden?
4. Ist „Division by Zero“ möglich?
5. Geht man fälschlicherweise davon aus, dass Fließkomma-Arithmetik genau ist?

Schleifen

6. Werden vor einer Schleife alle beteiligten Variablen richtig initialisiert oder enthalten sie alle bereits einen gültigen Wert?
7. Werden alle Schleifen auf jeden Fall beendet?

Abzweigungen

8. Hat jedes switch-Statement einen default-Fall?
9. Sind fehlende break-Statements in switch-Blöcken korrekt und gesondert kommentiert?
10. Sind alle else-Zweige richtig behandelt? Wenn einer if-Bedingung der else-Zweig fehlt, wird der Fall richtig behandelt, wenn die if-Bedingung nicht erfüllt wird?

Datenfluss

11. Kann eine Variable unter Umständen null sein und wird dieser Fall abgefangen?
12. Werden Parameterwerte auf gültige Wertebereiche (entsprechend der Vorbedingungen) überprüft?
13. Müssen Objekte mit equals () oder direkt mit == verglichen werden?
14. Kann eine Typanpassung (casting) fehlschlagen?

Arrays

15. Ist das Indexieren von Arrays außerhalb des gültigen Bereichs möglich?

Funktionsaufrufe

16. Reagiert der Aufrufer einer Methode auf alle möglichen Werte, die zurückgegeben werden können, inkl. Ausnahmen (Exceptions)?
17. Erfüllt der Aufrufer einer Methode die Voraussetzungen für die Parameter der Methode?
18. Kann ein Stack-Overflow bei rekursiven Funktionen auftreten?

Multithreading

19. Sind alle Zugriffe von mehreren Threads auf dieselben Variablen synchronisiert? Muss die Variable als volatile deklariert werden?
20. Besteht eine Verklemmungsgefahr?
21. Werden Threads, die mit wait () warten, irgendwann mal aufgeweckt?
22. Werden InterruptedExceptions behandelt?
23. Erfolgt der Zugriff auf eine Variable in zwei getrennten synchronized-Blöcke vom selben Thread und ist das korrekt?
24. Wird ein Objekt von einem Thread freigegeben (auf null gesetzt) und von einem anderen zugegriffen?

Ausnahmebehandlung

25. Wird der Kontrollfluss richtig fortgesetzt, wenn eine Ausnahme auftritt oder abgefangen wird?
26. Werden eventuelle Ausnahmen beim Zugriff auf externe Ressourcen abgefangen?
27. Werden alle Ausnahmen, die in den von dieser Methode aufgerufenen Methoden aufgeworfen werden können, abgefangen?

Externe Ressourcen

28. Werden Eingabe-, Ausgabe-Ströme und externe Ressourcen (Datenbank-Connections, Sockets etc.) wieder geschlossen?
29. Werden gepufferte Daten „geflusht“?
30. Werden Ausnahmen (Exceptions) bei Ein- und Ausgabe richtig behandelt?