Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2011/

Java-Übung Blatt 7 (Exceptions und Iteration)

2011-06-20

Hinweise

- Schreiben Sie Identifier *genau so*, wie sie auf dem Blatt stehen (inklusive Groß- und Kleinschreibung), nicht nur ungefähr.
- Exportieren Sie das Projekt als ZIP-Datei mit Namen *accountname-projectname*.zip. Beispiel: Peter Lustig (lustigp) gibt ex02_1 ab als Datei lustigp-ex02_1.zip
- ZIP-Export in Eclipse: Rechtsklick auf die Wurzel des Projects, "Export...", "General/Archive File..."
- Mailen Sie die ZIP-Files an Ihren Tutor.

Abgabe: Montag, 27. Juni 2011, um 9.00 Uhr.

Aufgabe 1 (Testen, 4 Punkte)

Projekt: ex07_1. Package: testing.

Klassen sind schwieriger zu testen, wenn sie Abhängigkeiten zu anderen Klassen haben. Wenn die Abhängigkeiten so formuliert sind, dass sie sich auf ein Interface statt zu einer konkreten Klasse beziehen, kann man das Interface für den Test durch einen Dummy[1] implementieren.

Holen Sie sich das Skeleton. Zu testen ist die Klasse ErrorLogger. Sie soll Fehlermeldungen mit Zeitstempel in Logdateien schreiben. Dabei gibt es eine Logdatei für kritische Fehler und eine Logdatei für alle Fehler. Die einzige öffentliche Methode ist logError; sie hat aber nur Seiteneffekte, keinen Rückgabewert. Wir müssen also im Test prüfen, dass die Seiteneffekte stattgefunden haben. ErrorLogger hat Abhängigkeiten zu einer Zeitquelle und zwei Ausgabeströmen; wir werden sie alle durch Dummies ersetzen, die mit dem Testfall zusammenarbeiten.

(a) IClock ist eine Uhr, die die aktuelle Uhrzeit als SimpleTime zurückgibt. Damit unsere Tests zu jeder Uhrzeit funktionieren, möchten wir im Test beherrschen, welche Zeit zurückgeliefert wird. Implementieren Sie IClock als Klasse DummyClock, die immer wieder dieselbe, im Konstruktor angegebene Uhrzeit zurückliefert.

(b) IOutputStream entspricht einer zum Schreiben geöffneten Datei: man kann dorthin Strings schreiben. Im Test möchten wir nachher prüfen, welche Strings dorthin geschrieben wurden. Implementieren Sie IOutputStream als DummyOutputStream. Im Inneren soll ein StringBuilder die per write hereinkommenden Strings aufsammeln. Mit einer Methode String getContents() sollen Testklassen nachher die gesamte bisherige Ausgabe auslesen können.

(c) Lognachrichten bestehen immer aus 1. Stunden (zweistellig, ggf. führende Nullen) 2. Doppelpunkt 3. Minuten (ebenso) 4. Doppelpunkt 5. Sekunden (ebenso) 6. Doppelpunkt 7. ein Leerzeichen 8. (nur bei kritischen Fehlern im normalen Log:) dem Wort CRITICAL, gefolgt von Doppelpunkt und einem Leerzeichen. 9. Fehlermeldung 10. Newline-Zeichen \n

Beispiel: logError(true,"Feuer!") abends um fünf nach neun soll im normalen Log die Zeile

```
21:05:00: CRITICAL: Feuer!
```

[1] In der Testing-Community unterscheidet man noch zwischen Stubs, Mocks, Fakes etc.; diese feinen Unterschiede kehren wir hier unter den Teppich.

und im kritischen Log

```
21:05:00: Feuer!
```

hinterlassen.

Implementieren Sie nun folgende Testfälle. Beide sollten einen Fehler in `ErrorLogger` offenbaren:

- Wenn man einen nichtkritischen Fehler loggt, soll im kritischen Log nichts neues stehen und im normalen Log ein Eintrag mit korrektem Zeitstempel und korrektem Fehler. Achten Sie auch darauf, dass die Newlines da sind!
- Wenn man einen kritischen Fehler loggt, soll er mit korrektem Zeitstempel im kritischen Log stehen und mit korrektem Zeitstempel und `CRITICAL`-Präfix im normalen Log stehen.

Zum Nachtisch können Sie sich überlegen, wie Sie `ErrorLogger` testen würden, wenn die Abhängigkeit zur Uhr und zu den Ausgabelogdateien hartkodiert wären.

Aufgabe 2 (Exceptions, 4 Punkte)
Projekt: `ex07_2`. Package: `except`.
Der Hersteller eines Eiscremeautomaten hat mal wieder Fehlerbehandlung bis zuletzt vernachlässigt und braucht jetzt Ihre Hilfe.

Der Eiscremeautomat besteht aus Dispenser und Kontrolleinheit. Die Kontrolleinheit hat Knöpfe und Display. Der Dispenser besteht aus einem Spender für leckere kegelförmige Waffeln (im folgenden "Cones"), einer Pumpe für Softeis (es gibt der Einfachheit halber nur eine Geschmacksrichtung[2]) und einer Mühle, die Schokoraspeln raspelt.

Der Hersteller des Dispensers gibt an, dass vier Arten von Dispenser-Fehlern passieren können:

1. Der Vorrat an Cones ist leer – dann muss man Cones nachfüllen.
2. Der Vorrat an Eis ist leer – dann muss man Eis nachfüllen.
3. Die Pumpe ist verstopft – dann muss man den Service rufen.
4. Die Mühle ist verklemmt – dann muss man den Service rufen.

Holen Sie sich das Skelett.

(a) Definieren Sie Exceptionklassen für jede der vier Fehlersituationen. Richten Sie Superklassen ein, so dass man Nachfüllprobleme (1,2) von Serviceproblemen (3,4) unterscheiden kann und alle Dispenser-Fehler unter einer gemeinsamen Superklasse `DispenserException` zusammengefasst sind.

(b) Im Skelett finden Sie ein Interface `IceCreamDispenser` und eine Klasse `IceCreamControl`. Rüsten Sie die Fehlerbehandlung nach.

- Die Methoden von `IceCreamDispenser` sollen *möglichst genau* beschreiben, welche Exceptions geworfen werden können. Fügen Sie entsprechende `throws`-Klauseln ein.
- Die `make...`-Methoden von `IceCreamControl` sollen Exceptions an ihren Aufrufer weiterleiten. In der Signatur soll aber nur drinstehen, dass sie Dispenser-Fehler weiterleiten, nicht eine vollständige Liste der möglichen Dispenser-Fehler (das wäre zu viel Detailwissen an falscher Stelle).
- Die `handleKeypadInput`-Methode soll sämtliche Dispenserfehler auffangen und behandeln, indem eine entsprechende Message angezeigt wird und der Automat in Störung geht (`ok=false`). Wortlaut der Messages: steht im Kommentar.

[2] Leider Erdbeer.

(c) Implementieren Sie folgende Testfälle (mit Hilfe der Dummy-`IceCreamDispenser`-Klasse, in der noch Ihre Exceptions fehlen):

a) Wenn die Schoko-Mühle klemmt und man auf dem Keypad 'm' drückt, wird `CALL SERVICE` angezeigt und der Automat ist in Störung.

b) Wenn Eis leer ist, wirft ein `makeMaxiCone`-Aufruf eine Eisvorrat-leer-Exception. Um das zu testen, können Sie entweder mit `try` und `catch` und `Assert.fail()` arbeiten oder in der Junit-Doku nach `@Test(expected=...)` suchen.

Aufgabe 3 (Wrapperklasse für Operationen, 4 Punkte)
Projekt: `ex07_3` Package: `ops`
Im Skelett finden Sie eine Klasse `Patient` und Klassen, die eine verkettete Liste von `Patient` (mit Name, Behandlungszeit in Minuten) implementieren (`IPatientList` und Co). Schreiben Sie nach dem Vorbild in der Vorlesung eine Wrapper-Klasse `PatientListOps`, die die Operation `int waitingTime()` anbietet: diese soll die Behandlungszeiten `duration` aller Patienten der Liste aufsummieren.

Sie dürfen wählen, ob Sie die Aufgabe iterativ oder rekursiv lösen. Schreiben Sie aber in den Klassenkommentar von `PatientListOps`, welche der Herangehensweisen sie gewählt haben.

Tests: zwei Testfälle.

Nachholaufgaben für Thema 5: Abstraktion

Mit dieser Aufgabe können Sie den Stoff von Thema 5 (dem Thema von Blatt 5) wiederholen und nochmal Punkte für Thema 5 erwerben.

Aufgabe 4 (Fuhrpark, 4 Punkte)
Projekt: `ex05_4` Package: `fuhrpark`
Schreiben Sie eine Twodeedoo-Welt, in der verschiedene Fahrzeuge von links nach rechts über den Bildschirm fahren. Es soll mindestens Pkw und Lkw geben, wobei es bei den Lkw mindestens zwei Sorten gibt: solche mit Kastenaufbau und solche mit Pritsche.

Repräsentieren Sie die verschiedenen Arten von Fahrzeugen durch Klassen, die sich auf ein `ICanvas` zeichnen können. Nutzen Sie alle Gelegenheiten, gemeinsame Funktionalität in abstrakte Oberklassen zu verschieben.

Test: Ausprobieren reicht.