Übung 01: Reflection und Annotationen

Abgabetermin: 14. 3. 2022 8:30

Übung 1: Unittest-Framework (JayUnit) (24 Punkte)

"Schon wieder eine LinkedList zu korrigieren...", seufzen Sie. Die Arbeit als TutorIn hatten Sie sich definitiv spannender vorgestellt. Wenn es doch nur ein Testframework gäbe, um die Funktionalität der Programme automatisiert zu prüfen…

Leider haben Sie aber selbst in der zugehörigen Softwareentwicklung 2 Stunde nicht gut aufgepasst, um sich an das JUnit-Framework zu erinnern. Also bleibt nichts Anderes übrig, als selbst kreativ zu werden:

Erstellen Sie einfaches Framework für Unittests – *JayUnit* – um sich die Arbeit als TutorIn zu erleichtern. Schreiben Sie anschließend eine Testklasse, die die Funktionalität der LinkedList mit JayUnit prüft.

Aufbau des JayUnit Frameworks

Die Hauptklasse `JayUnit` soll eine öffentliche Methode

```
public static void runTests(Class<?> testClass)
```

bereitstellen, der eine Testklasse übergeben wird. Die Methode `runTests` soll alle in der Testklasse enthaltenen Testmethoden ausführen und das Ergebnis auf der Konsole ausgeben.

Eine Testklasse definiert eine Reihe von Methoden, die mit Annotationen versehen sein können. Das Framework soll folgende Annotationen bereitstellen:

- `@BeforeTests`: Die mit dieser Annotation annotierte Methode wird vor jeder Testmethode ausgeführt. `@BeforeTests`-Methoden dürfen keine Parameter und Rückgabewerte haben. Pro Testklasse darf nur eine Methode mit dieser Annotation versehen sein.
- `@MyTest`: Annotation für Methoden, die von JayUnit zum Testen herangezogen werden sollen. Testmethoden haben keine Parameter und keine Rückgabewerte. `@MyTest` hat einen optionalen Parameter `ignore` (default: `false`), der eine mit `@MyTest` annotierte Methode, beim Durchlaufen der Testklasse ignoriert.
- `@ExpectException`: Optionale Annotation um JayUnit mitzuteilen, dass eine bestimmte Exception erwartet wird. Hat als Parameter `exception` eine Klasse, die von `Exception` ableiten muss.

Ihnen werden zwei Exception Klassen zur Verfügung gestellt:

- `TestClassException`: `RuntimeException`, für Fehler in der Testklasse
- `TestFailedException`: Wird in den Testmethoden geworfen, wenn ein Test fehlschlägt.

LinkedList

Im Download zur Aufgabe befindet sich eine Klasse für eine LinkedList Implementierung, die von einem Kursteilnehmer oder einer Kursteilnehmerin in Softwareentwicklung 2 erstellt wurde.

Vorgehen und Anforderungen

Im Folgenden sind die genauen Anforderungen angegeben:

`JayUnit.runTests`

- erzeugt eine Instanz der Testklasse (leerer Konstruktor)
- führt vor jedem Aufruf einer Testmethode die `@BeforeTests` Methode aus (falls vorhanden)
- falls mehrere `@BeforeTest` Methoden vorhanden sind, soll eine `TestClassException` geworfen werden
- führt alle mit `@MyTest` annotierten Methoden aus (die nicht mit `ignore` gekennzeichnet sind). Tests, die eine `TestFailedException` werfen, schlagen fehl. Tests, die „durchlaufen" sind erfolgreich (siehe Hinweis zu Exceptions). Das Ergebnis jedes Tests wird auf der Kommandozeile ausgegeben (`"testX ok"`, `"testY failed"` oder `„testZ ignored"`).
- Tests, die eine Exception erwarten, sind nur dann erfolgreich, wenn genau diese Exception geworfen wurde (siehe Hinweis zu Exceptions)
- Wird eine Exception gefangen, die weder in der `ExpectException` Annotation spezifiziert, noch eine `TestFailedException` ist, soll diese mit geeigneter Fehlermeldung in eine `TestClassException` geschachtelt und weitergeworfen werden

Annotations

- Erzeugen Sie die geforderten Annotationen `@BeforeTests`, `@MyTest` und `@ExpectException` im Package `jayunit`

Assertions

- Implementieren Sie die statischen Hilfsmethoden `assertEquals`, `assertTrue` und `assertFalse`, die von der Testklasse mittels static import verwendet werden sollen.

Klasse `LinkedListTest`

- Testen Sie die LinkedList Implementierung mit Integer als generischem Typparameter.
- Nutzen sie die Annotationen für Test-Methoden, die von JayUnit ausgeführt werden sollen und die JayUnit Assertions, um `TestFailedExceptions` zu werfen.
- Sie können annehmen, dass Typen und Namen von Feldern und Methoden der zu testenden Klasse vorgegeben sind, wodurch Sie (ausnahmsweise) auch interne Zustände für die Tests miteinbeziehen können, um Testfälle isoliert zu implementieren.
- Reflection-spezifische Exceptions können einfach weitergeworfen werden und sollen von JayUnit als Fehler in der Testklasse interpretiert werden (`TestClassException`)
- Folgende Methoden sind zu implementieren:
  - Methode `init`, die vor jedem Test eine neue `LinkedList` erzeugt
  - Methode `testEmpty` soll folgende Bedingungen für eine neu erzeugte Liste prüfen: `head==null` und `size==0`
  - Methode `testSize` soll prüfen, ob die Länge der Liste richtig zurückgegeben wird. Setzen Sie `size` zuvor mittels Reflection, um den Testfall von anderen Methoden zu isolieren.
  - Methode `testAdd` soll prüfen, ob ein mittels `add` zu einer leeren Liste hinzugefügtes Element in der Liste ist. Verwenden Sie für die Überprüfung kein `contains`.

- Methode testContains soll die Funktionalität der `contains` Methode isoliert (ohne Methode `add`) prüfen. Fügen Sie zuerst mittels Reflection ein Element in eine leere Liste ein.
- Methode `testAddAllNull` soll einen illegalen Aufruf der `addAll` Methode mittels `null` Parameter testen. Dieser Test soll eine `IllegalArgumentException` erwarten (spezifiziert über die Annotation `ExpectException`)
- Methode `fancyButNotYetFinishedTest` stellt einen leerer Test dar, der mit `ignore=true` parametrisiert sein soll und daher nicht ausgeführt wird

Vorgabe und Test

Ihnen wird ein ein schlankes Codegerüst zur Verfügung gestellt, in dem bereits die Exceptions und restliche Klassenstruktur (ohne Annotationen) vorgegeben sind. Im package `main` finden Sie eine Klasse `Runner`, die JayUnit mit der von Ihnen implementierten `LinkedListTest` Klasse aufruft. Die Ausgabe soll sinngemäß wiefolgt aussehen:

```
testEmpty OK.
testSize OK.
testAdd failed: Size should be 1.
testContains OK.
testAddAllNull OK.
fancyButNotYetFinishedTest ignored.
```

Abgabe

Abzugeben ist das gesamte Projekt, direkt aus der IDE als .zip exportiert (Eclipse: Rechtsklick auf das Projekt → Export… → General → Archive File → als .zip speichern). Die .class Files brauchen nicht abgegeben werden (kann im Exportmenü abgewählt werden).

Hinweise:

- Exceptions, die in einer über Reflection aufgerufenen Methode geworfen werden, sind in eine `InvocationTargetException` gekapselt und können mit `e.getCause()` extrahiert werden.
- Sie dürfen natürlich beliebig viele Hilfsmethoden anlegen