Übung Test Driven Development

Szenario

Für ein Seminar soll die Notenverwaltung realisiert werden. Die Notenverwaltung muss die folgenden Anforderungen erfüllen:

- Jedem Studenten soll eine eindeutige Id zugewiesen werden.
  - Bei Ids handelt es sich um Integers.
  - Die kleinste Id ist Eins.
  - Ids werden fortlaufend vergeben. Beispiel 1, 2, 3, 4 ...
- Die Anzahl der Teilnehmer ist begrenzt auf 20 Studenten.
- Dem Seminar sollen Studenten hinzugefügt werden können.
- Einem Studenten soll eine Note zugeordnet werden können. Wobei die Id für die Zuordnung verwendet wird.
- Mit Hilfe der Id soll die Note eines Studenten ermittelt werden.
- Alle Studenten und Ihre Noten sollen auf die Konsole ausgegeben werden.

Vorgehen

Es sollen zuerst nicht alle Klassen und Funktionalitäten implementiert werden, sondern es soll iterativ entwickelt werden, wobei zuerst die Tests geschrieben werden.

Beispiel:

Einfacher Taschenrechner der zwei Zahlen addieren kann.

Ersten Test erstellen

```
public class TestCalculator  {
     public static void main(String[] argv) {
          testCreateCalculator();
     }

     public static void  testCreateCalculator() {
          assertTrue(new Calculator() != null);
     }

     public static void assertTrue(boolean isTrue) {
          if(isTrue) {
               return;
          } else {
               throw new RuntimeException(„TEST FAILED.“);
```

```
        }
    }
}
```

Zu testende Klasse erstellen

```
public class Calculator {
    public Calculator() {
        // Nothing to-do
    }
}
```

Zweiten Test erstellen

```
public class TestCalculator  {
    public static void main(String[] argv) {
        testCreateCalculator();
        testAdd() ;
    }

    public static void  testCreateCalculator() {
        assertTrue(new Calculator() != null);
    }

    public void testAdd() {
        assertTrue (2 == new Calculator().add(1,1));
        // ODER
        if(2 == new Calculator().add(1,1)) {
            System.out.println(„OK“);
        } else {
            System.out.println(„FAILED: testAdd()“);
        }
    }

    public static void assertTrue(boolean isTrue) {
        if(isTrue) {
            return;
        } else {
            throw new RuntimeException(„TEST FAILED.“);
```

```
            }
        }
}
```

Zu testende Klasse erweitern

```
public class Calculator {
    public Calculator() {
        // Nothing to-do
    }

    public int add(int left, int right) {
        return left + right;
    }
}
```

Aufgabe 1

Entwickelt iterativ die Notenverwaltung für ein Seminar und erstellt dabei zuerst die Tests. Siehe Abschnitt *Vorgehen*.

Nach jedem erstellten Test sollen alle Dateien übersetzt werden und die Tests ausgeführt werden. Neue Funktionalitäten und Tests werden erst implementiert, wenn alle Tests funktionieren.

Für die Tests sollen asserts und eigene Vergleichsfunktionen verwenden werden, siehe *testAdd()* im Abschnitt *Vorgehen*. Beide Varianten sollten mindestens einmal verwendet werden.

Aufgabe 2

Baue absichtlich Fehler in Deinen Code ein und prüfe, ob sie von Deinen Tests erkannt werden. Korrigiere die Fehler wieder.

Aufgabe 3 OPTIONAL

Verwende für Deine Test-Klasse das JUnit-Testframework (http://www.junit.org).

Methoden ähnlichen zu `assertTrue()` aus Aufgabe 1 und 2 werden von JUnit schon angeboten, siehe http://junit.sourceforge.net/javadoc_40/org/junit/Assert.html.

Um alle Dateien zu übersetzen muss `javac` wie folgt aufgerufen werden:

```
javac -cp .:junit.jar TestSeminar.java
```

Um JUnit-Tests auszuführen muss der Test wie folgt aufgerufen werden:

```
java -cp .:junit.jar TestSeminar
```

Beispiel:

```
import junit.framework.TestCase;
```

```
public class TestSeminar extends TestCase {

     private Calculator  toBeTestedCalculator;

     public static void main(final String[] args) {
          junit.swingui.TestRunner.run(TestSeminar.class);
     }

     protected void setUp() throws Exception {
          this.toBeTestedCalculator = new Calculator();
     }

     public final void  testCreateCalculator() {
          assertNotNull(this.toBeTestedCalculator);
     }

     public void testAdd() {
          assertEquals (2, this.toBeTestedCalculator .add(1,1));
     }
}
```

Aufgabe 4 OPTIONAL

Baut wieder Fehler in Euren Code und Eure Tests ein und führt die JUnit-Tests aus.

Um alle Dateien zu übersetzen muss `javac` wie folgt aufgerufen werden:

```
javac -cp .:junit.jar TestSeminar.java
```

Um JUnit-Tests auszuführen muss der Test wie folgt aufgerufen werden:

```
java -cp .:junit.jar TestSeminar
```