Software-Qualitätssicherung VU

Übung 3

Aufgabenstellung

1 Unit Tests (5 Punkte)

Erstellen Sie Blackbox Unit Tests (mittels JUnit 4) für die Klasse `Stack<E>` der Java API [1]. Testen Sie folgende Methoden: `empty, peek, pop, push, search` (insgesamt mindestens 15 Testfälle).

[1] http://docs.oracle.com/javase/6/docs/api/java/util/Stack.html

Hinweis: Die Benennung der Klassen, wie unter Punkt 7 vorgegeben, ist nicht optional! Achten Sie unbedingt darauf, dass die Klassen, genau so, wie Sie sie abgeben, auch kompilierbar sein müssen.

2 Test Driven Development (6 Punkte)

Entwickeln Sie folgende Klasse und dazugehörige Tests (mittels JUnit 4) nach dem Vorgehen von Test Driven Development. Schreiben Sie eine Klasse zur Dreiecksberechnung für rechtwinkelige Dreiecke, die folgende Methoden unterstützt:

- Berechnen der Hypotenuse; gegeben: Länge der beiden Katheten
- Winkelberechnung: gegeben ein Winkel, berechnen Sie den nicht bekannten Winkel
- Berechnen Sie die Fläche des Dreiecks bei gegebenen Katheten

Commiten Sie Ihre Arbeit zur Dokumentation nach jedem Schritt (Implementierung, Tests, etc.) in Ihr lokales Git-Repo. Beschreiben Sie in den Commit Messages welche Schritte Sie genau durchgeführt haben.

Hinweis: Die Benennung der Klassen, wie unter Punkt 7 vorgegeben, ist nicht optional! Achten Sie unbedingt darauf, dass die Klassen, genau so, wie Sie sie abgeben, auch kompilierbar sein müssen.

3 Bugsuche (6 Punkte)

Die folgenden Java Methoden enthalten möglicherweise Fehler. Finden Sie diese und schlagen Sie eine Möglichkeit zur Behebung vor.

3.1 Beispiel 1

Ist das Ergebnis (ret) folgender Methode immer das Selbe wie $val^{pow}|pow \geq 0$?
Begründen Sie Ihre Antwort!

```
1 public static long pow(int val, int pow) {
2     long ret = val;
3     for(int i = 1; i < pow; i++) {
4         ret = ret * val;
5     }
6     return(ret);
7 }
```

3.2 Beispiel 2

Ist das Ergebnis (res) folgender Methode immer positiv?
Begründen Sie Ihre Antwort!

```
1 public int f(int a) {
2     int res = 1 ;
3     for(int i = 2; i <= a; i++) {
4         res = res + i ;
5     }
6     return(res);
7 }
```

3.3 Beispiel 3

Ist das Ergebnis folgender Methode immer "true"?
Begründen Sie Ihre Antwort!

```
1 public boolean calculate (float a, float b) {
2     float c = a / b;
3     return(a == (c * b));
4 }
```

4 Testisolation (6 Punkte)

Folgende Klasse berechnet die Tage bis Weihnachten und benutzt dazu ein `TimeService`. Testen Sie die Methode `calculateDaysToChristmas` indem Sie das `TimeService` Objekt einmal stubben und einmal mocken. Verwenden Sie für diese Aufgabe keine Mocking-Frameworks. Schreiben Sie die dazugehörigen Tests (mittels JUnit 4) jeweils in die beiden abzugebenden Files dazu. Eventuelle Fehler in der Implementierung können mit den Tests aufgezeigt werden, Hauptziel dieser Aufgabe jedoch ist, den Unterschied zwischen Mock und Stub und deren Verwendung zu demonstrieren.

Hinweis: Die Benennung der Klassen, wie unter Punkt 7 vorgegeben, ist nicht optional! Achten Sie unbedingt darauf, dass die Klassen, genau so, wie Sie sie abgeben, auch kompilierbar sein müssen.

```
1  import java.util.Calendar;
2  import java.util.Date;
3  import java.util.GregorianCalendar;
4
5  public class ChristmasCounter {
6
7      protected TimeService   timeService;
8
9      public ChristmasCounter(TimeService timeService) {
10         this.timeService = timeService;
11     }
12
13     public long calculateDaysToChristmas() {
14         Date currentDate = this.timeService.
               getCurrentDate();
15         Calendar christmas = new GregorianCalendar();
16         christmas.setTime(currentDate);
17         christmas.set(Calendar.MONTH, Calendar.
               DECEMBER);
18         christmas.set(Calendar.DAY_OF_MONTH, 24);
19         return(this.timeService.daysBetween(
               currentDate, christmas.getTime()));
20     }
21
22 }
```

Listing 1: ChristmasCounter.java

```
1  import java.util.Date;
2
```

```java
/**
 * A very simple timeservice just for experimentation.
 *
 * @author ExampleAuthor
 */
public interface TimeService {

    /**
     * Get the current date.
     *
     * @return current date
     */
    public Date getCurrentDate();

    /**
     * Returns the number of days between to dates.
     *
     * @throws IllegalArgumentException
     *           if fromDate or toDate is not valid
     * @param dateFrom
     * @param dateTo
     * @return days between dateFrom and dateTo
     */
    public long daysBetween(Date dateFrom, Date dateTo
        );

}
```

Listing 2: TimeService.java

```java
import java.util.Date;

public class TimeServiceImpl implements TimeService {

    private static final long   ONE_DAY = 1000L * 60L
        * 60L * 24L;

    @Override
    public Date getCurrentDate() {
        return new Date();
    }

    @Override
    public long daysBetween(Date dateFrom, Date dateTo
        ) {
```

```
14          if((dateFrom == null) || (dateTo == null))
15              throw new IllegalArgumentException("date
                    mustn't be null");
16          if(dateFrom.after(dateTo))
17              throw new IllegalArgumentException("
                    fromDate mustn't be after toDate");
18          return((dateTo.getTime() - dateFrom.getTime())
                * TimeServiceImpl.ONE_DAY);
19      }
20
21  }
```

Listing 3: TimeServiceImpl.java

5 Code Coverage (6 Punkte)

Für folgende Methode sollen Testfälle (keine JUnit Tests) erstellt werden um eine spezifische Abdeckung zu erreichen. Geben Sie jeweils die drei Parameter (a, b, c) an, die Sie benötigen, um die jeweilige Coverage zu erreichen.

```
1  public static int doSomeStuff(int a, int b, int c) {
2      while (c < 0) {
3          if ((a > 0) || (b < 0)) {
4              c = a + b;
5          }
6      }
7      if (c == 0) return(c);
8      else return(a);
9  }
```

5.1 Anweisungsüberdeckung

Erstellen Sie Testfälle um eine c0 Coverage (Anweisungsüberdeckung) zu erreichen.

5.2 Zweigüberdeckung

Erstellen Sie Testfälle um eine c1 Coverage (Zweigüberdeckung) zu erreichen.

5.3 Mehrfach-Bedingungsüberdeckung

Erstellen Sie Testfälle um eine c2 Coverage (Mehrfach-Bedingungsüberdeckung) zu erreichen.

Hinweis: Die hier genannten Testarten und ihre Bezeichnungen c0-c2 entsprechen den genannten Testarten aus der Vorlesung (siehe Vorlesungsfolien aus VO Block 4) und nicht denen aus Wikipedia!

6 Theoriefragen (3 Punkte)

1. Erklären Sie die Unterschiede der Coverage Kategorien c0 - c3 (siehe Vorlesungsfolien VO Block 4, nicht Wikipedia!). Wann ist der Einsatz welcher Coverage sinnvoll?
2. Nennen Sie mindestens zwei Gründe für den Einsatz von Testdoubles (Mock, Stubs, etc.).
3. Erklären Sie den Unterschied zwischen Mock Objects und Stubs.

7 Abgabe

Folgende Dateien sind abzugeben:

- QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>.pdf
- 1 Unit Tests
  - QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>_StackTest.java
- 2 Test Driven Development
  - QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>_Triangle.java
  - QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>_TriangleTest.java
  - Dateien, die zum git-Repository gehören!
- 4 Testisolation
  - QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>_TimeService_Mock.java
  - QSVU_UEbung3_<Matrikelnummer>_ <Nachname>_<Vorname>_TimeService_MockTest.java

- QSVU_UEbung3_<Matrikelnummer>_<Nachname>_<Vorname>_TimeService_Stub.java
- QSVU_UEbung3_<Matrikelnummer>_<Nachname>_<Vorname>_TimeService_StubTest.java

Bitte verwenden Sie die Vorlage und beachten Sie, dass ausschließlich richtig benannte Dateien im angegebenen Format bewertet werden. Bitte laden Sie ihre Abgabe rechtzeitig hoch. Verspätete Abgaben werden **ausnahmslos nicht akzeptiert**!

Bei der Übung handelt es sich um eine Einzelarbeit. Plagiate werden mit 0 Punkten bewertet. Weiters behalten wir es uns vor, Studenten im Verdachtsfall zu einem **Kontrollgespräch** einzuladen.