Testkonzept

1. Einleitung

Das Testen von Quellcode ist ein fester Bestandteil der Implementierungsphase.
Jeder neue geschriebene Code muss getestet werden um im späteren Verlauf der Implementierung Fehler zu minimieren.
Tests dienen darüberhinaus auch zum überprüfen, ob das Erstellte auch dem entspricht, was erwartet wird.
Es wird dabei zwischen Komponenten-, Integrations- und Systemtests unterschieden.
Das Java-Framework JUnit bietet die Möglichkeit zum Testen von Anwendungsklassen.
Dies erlaubt das Schreiben und Ausführen von automatisierten Tests.

2. Testübersicht

2.1 **Komponententest**
Der Komponententest (auch Modultest) ist Teil des Softwareentwicklungsprozesses. Er dient zur Verifikation der Korrektheit von Modulen einer Software, z.B. von einzelnen Klassen.
Es wird kein oder nur ein sehr einfaches Zusammenspiel zwischen den Klassen getestet.
Somit wird ein ausführbares Codefragment getestet, dass das Verhalten einer Klasse darstellt und dem Programmierer eine Rückmeldung gibt ob die Klasse das geforderte Verhalten aufweist.

2.2 **Integrationstest**
Nach Abschluss des Komponententest erfolgt der Integrationstest. Hierbei wird das Zusammenspiel zwischen verschiedenen und voneinander unabhängigen Komponenten getestet.

2.3 **Systemtest**
Im Systemtest wird geprüft, ob das gesamte System den vorgegebenen Anforderungen entspricht.
Hierbei wird zwischen funktionalen - und nichtfunktionalen Systemtest unterschieden.
Im funktionalen Systemtest wird das System auf funktionale Qualitätsmerkmale wie Korrektheit und Vollständigkeit geprüft. Im nichtfunktionalen Systemtest liegt der Focus nicht auf den Funktionen sondern auf Merkmalen wie z.B. die Sicherheit, die Benutzbarkeit, die Interoperabilität, die Dokumentationsprüfung oder Zuverlässigkeit des Systems.

3. JUnit

JUnit ist ein Framework zum Testen von Java-Programmen, das besonders für automatisierte Unit-Tests einzelner Units (meist Klassen oder Methoden) geeignet ist.
Es werden dazu Testklassen erstellt, die von junit.framework.TestCase abgeleitet, und mit Testmethoden ausgestattet werden. Die Methoden haben den Präfix test.
Der Test kann Anschließend mittels run() aufgerufen werden.

Schlägt ein Test fehl so gibt JUnit die nötigen Exceptions zur Bestimmung der Ursache aus.
Einzelne Testreihen können in eine Testsuite integriert werden, um so zu umgehen jeden Klassentest einzeln ausführen zu müssen.
Für jedes Packet wird eine Testsuite erstellt, die alle Tests des Paketes und deren Unterpakete enthält. Dies ermöglicht das alle eingebetteten Tests automatisch durchgeführt werden können.

3.1 Verwendung von JUnit

Für jede zu testende Klasse, wird im selben Paket eine Testklasse angelegt .(Name: Klassenna-
Der Klassename dieser Testklasse sollte KlassennameTest.java lauten.
Diese Klasse erbt von junit.framework.TestCase.
Weiterhin ist es sinnvoll eine TestSuit zu erstellen in welcher die gesamte Funktionalität unseres Systems getestet werden kann.

Methoden der Testklasse :

- **setUp()** Diese Methode wird als erste Methode der Klasse aufgerufen. In ihr finden die Initialisierungen statt welche für alle Testmethoden gelten.
- **tearDown()** Diese Methode wird als letzte ausgeführt. Mit dieser Methode wird „aufgeräumt“ z.B. Datenbankverbindungen schließen
- **test...()** Es können beliebig viele Methoden in der Klasse erstellt werden. Sie müssen folgende Bedingungen erfüllen, um als testMethoden erkannt zu werden:
  - Name muss mit 'test' beginnen
  - dürfen keine Parameter enthalten
  - müssen 'public' und dürfen nicht 'static' sein

- **assert-Methoden** statische Methoden, die von JUnit zur Verfügung gestellt werden, um Korrektheit der Methoden zu prüfen

  - **asserTrue(boolean test)** – prüft ob übergebener Parameter 'true' ist
  - **assertNotNull(Object test)** – prüft, ob übergebenes Objekt nicht 'null' ist
  - **asserEquals(Objekt expected, Objekt actual)** – prüft , ob die beiden übergebenen Objekte übereinstimmen

Beispielklasse

```
import junit.framework.TestCase;

public class BasicTest extends TestCase {

   private int variable;
    public void setUp() {
       variable = 3;
   }

    public void testfirstThing() {
       variable = variable +1;
       assertEquals(4,variable);
```

```
   }
    public void testsecondThing() {
        variable = variable - 3;
        assertTrue(variable == 0);
    }
    public void tearDown() {
        variable = 0;
    }
}
```