JEGYZŐKÖNYV

Szoftvertesztelés

Féléves feladat

Készítette: **Kiss Máté**

Neptunkód: **DLN82Q**

A jegyzőkönyv elkészítéséhez Tompa Tamás dokumentációja szolgált alapul. https://users.iit.uni-miskolc.hu/~tompa/

A feladat leírás: A szoftvertesztelés célja az esetleges hibák kiszűrése, minimalizálása mielőtt a szoftver végleges megjelenése megtörténne. Szoftvertesztelésre mindig lesz szükség, hiszen emberek írják a forráskódot. Emiatt biztos lesz benne hiba. Ilyen hibák közül megkülönböztetünk emberi hibát és meghibásodást. Emberi hiba a helytelen használatból ered, míg a meghibásodás a rendszer nem megfelelő működéséből fakad, eltér az elvárt eredménytől. Tesztelési technikák között megkülönböztetünk feketedobozos, fehérdobozos, fejlesztői, valamint átviteli tesztet.
Fekete dobozos tesztelés esetén a fejlesztő nem látja a forráskódot, ezért szükséges a futtatható szoftver. Adott bement – kimenet párokat tesztel. Fehérdobozos tesztelésnél ismert a forráskód.
Fejlesztői tesztelésnél megkülönböztetünk komponenstesztet integrációs tesztet és rendszertesztet. A komponensteszt csak a rendszer egy komponensét teszteli önmagában.
Ennek része az általam választott Unit-teszt, a modulteszt és az integrációs teszt. Az integrációs teszt már kettő vagy több komponens együttműködési tesztje. Amely lehet komponens és komponens közötti vagy rendszer és rendszer közötti. Ezt a tesztet érdemes minél hamarabb elvégezni, mert minél nagyobb az integráció mértéke annál nehezebb meghatározni, hogy a fellelt hiba honnan származik.
A tesztet mely az egész rendszert teszteli, tehát minden komponenst együtt, rendszertesztnek hívjuk. A kész szoftvert teszteli, hogy megfelel-e a követelményspecifikációnak, a funkcionális specifikációnak, valamint a rendszertervnek.

1. feladat

1a) Az általam választott témakör a Unit-teszt. Adott paraméterekre ismerjük a metódus visszatérési értékét. A Unit-teszt megvizsgálja, hogy tényleges visszatérési értéke megegyezik-e az elvárttal. Ha igen a teszt sikeres, egyébként sikertelen. Minden fejlett programozói környezet támogatja.
A tesztek elvégzésére a JUnit keretrendszert lehet alkalmazni. A JUnit egy nyílt forráskódú modultesztelő rendszer, Java programok automatikus teszteléséhez. A JUnit rendszer használatával a programok minősége javítható, amivel hibakeresési idő takarítható meg. A JUnit alap eszközei a tesztosztály és a tesztmetódus. A tesztosztály egy POJO (Plain Old Java Project), melynek metódusai annotációkkal vannak ellátva. Ilyen annotáció a @Test, amellyel ellátott tesztmetódusok számítanak teszt metódusnak, ezek lesznek végrehajtva. Egy teszt futtatása során az összes ilyen metódus végrehajtásra kerül, amelynek háromféle eredménye lehet sikeres végrehajtás

(pass), sikertelen végrehajtás (failure) vagy hiba (error).
Sikeres végrehajtás esetén, a teszt rendben lefutott és az összes feltétel teljesült.
Sikertelen végrehajtásnál a teszt lefut, azonban van olya ellenőrzési feltétel, amely nem teljesült.
Hiba esetén a teszt futása során komolyabb hiba lépett fel (pl. Exception).

A JUnit használatához egy (akár) Maven projekt-en belül szükséges implementálni a JUnit dependenciáját (függőség). Ezt a dependenciát a projekt pom.xml fájljában kell megadni az alábbi módon:

```
<dependency>
    <groupID>junit</groupID>
    <artifactID>junit</artifactID>
    <version> „az aktuális verzió”</version>
    <scope>test</test>
</dependency>
```

A tesztelést Assert osztályok segítségével lehet elvégezni. Ezek az osztályok viselkedéseket ellenőriznek, logikai kérdéseket lehet velük végre hajtani. Ilyen osztályok a következők:

- assertTrue(boolean) – Ellenőrzi, hogy a paraméter igaz-e
- assertFalse(boolean) – Ellenőrzi, hogy a paraméter hamis-e
- assertNull(Object) – Ellenőrzi, hogy a paraméter null-e
- assertNotNull (Object) – Ellenőrzi, hogy a paraméter nem null
- assertNotSame(Object, Object) - Ellenőrzi, hogy a két paraméter különböző objektumra mutat-e
- assertEquals(int, int) – Ellenőrzi, hogy a két paraméter egyenlő-e
- assertEquals(double, double, double) - Ellenőrzi, hogy a két double paraméter egyenlő-e a megadott tolerancián belül. (A tolerancia a harmadik double paraméter.)
- assertEquals(Object, Object) - Ellenőrzi, hogy a két paraméter egyenlő-e. (Az equals() metódust használva.)
- fail() – A tesztet sikertelenné teszi

Ezen osztályok használatához annotáció szükséges. Az annotáció segít metaadatokat beépíteni a forráskódba. Az annotációk a programkód elemeihez rendelhetők, és a @ jellel kezdődik, melyet az annotáció neve követ.

Ezek az annotációk a következők:

- @Test public void method() – egy tesztmetódust jelöl meg
- @Test (expected = Exception.class) public void method () – elvárt kivétel megadása, a teszt elbukik ha nem dobódik az elvárt kivétel
- @Test (timeout =100) public void method() – A teszt elbukik, ha a végrehajtási idő 100 ms-nál hosszabb
- @Before public void method() – teszteseteket inicializáló metódus, minden teszteset előtt lefut, feladata a tesztkörnyezet előkészítése
- @After public void method() – minden egyes teszteset végrehajtása után lefut, feladata az ideiglenes adatok törlése
- @BeforeClass public static void method() - egyszer fut le, még az összes teszteset és a hozzájuk kapcsolódó @Before-ok végrehajtása előtt. Itt lehet egyszeri inicializálós lépéseket megadni, pl. adatbázis-kapcsolat kiépítése. Ilyen annotációval ellátott metódusnak statikusnak kell lennie!
- @AfterClass public static void method() - egyszer fut le, miután az összes tesztmetódus, és a hozzájuk tartozó @After metódusok végrehajtása befejeződött. Egyszeri tevékenységet helyezünk ide, amely a @BeforeClass metódusban lefoglalt erőforrások felszabadítását végzi el. Az ilyen annotációval ellátott metódusnak statikusnak kell lennie!
- @Ignore - Figyelmen kívül hagyja a tesztmetódust, vagy a tesztosztályt. Olyankor használható, ha megváltozott a tesztelendő kód, de a tesztesetet még nem frissítettük vagy abban az esetben, ha a teszt végrehajtása túl hosszú ideig. Ha osztály szinten van definiálva, akkor az osztály összes tesztmetódusát figyelmen kívül hagyja

Parametrizált tesztek

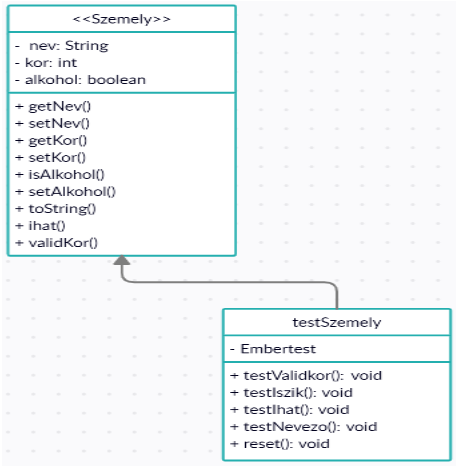
A parametrizált tesztek a JUnit 4-es verziójában jelent meg. Célja, hogy lehetővé tegyék ugyan azon tesztesetek többszöri lefuttatását, azonban minden lefutását alkalmával különböző adatokkal. Az ehhez szüksége annotáció a @RunWith(Parametrized.class).

Kivételek kezelése

Vannak olyan esetek, amikor a normál működésből adódik, hogy kivétel keletkezik. Ilyen lehet egy Push művelet, például FullStackException dobása metódusban, vagy akár saját kivétel dobása. Ilyenkor adott kivétel keletkezését várjuk és ha ez bekövetkezik, a teszt elbukhat. Azonban, ha cél a kivétel keletkezése, azt jelezni kell.

Például @Test(expected=FullStackException.class). Ebben az esetben akkor bukik el a teszt, ha nem teljesül az adott elvárt kivétel.

1b) UML



1c)

Az általam készített forráskódban az látható, hogy a Szemely.java osztályban deklarálom egy adott személy nevét, életkorát, és hogy szeretne-e alkoholt inni.

Szemely.java

```
package com.dln82q.Beadando;

public class Szemely {

	private String nev; // A személy neve
	private int kor; // A személy életkora
	private boolean alkohol; // A személy alkoholt vesz-e

	public String getNev() { // Visszaadja a személy nevét
		return nev;
	}

	public void setNev(String nev) { // Beállítja a személy nevét
		this.nev = nev;
	}
```

```
	public int getKor() { // Visszaadja a személy életkorát
		return kor;
	}

	public void setKor(int kor) { // Beállítja a személy életkorát
		this.kor = kor;
	}

	public boolean isAlkohol() {
		return alkohol;
	}

	public void setAlkohol(boolean alkohol) {
		this.alkohol = alkohol;
	}

	@Override
	public String toString() {
		return "Szemely [nev=" + nev + ", kor=" + kor + ", alkohol=" +
alkohol + "]";
	}

	public boolean ihat() { // Eldönti, hogy a személy fogyaszthat-e már
alkoholt
		if (getKor() > 18) {
			return true;
		}
		System.out.println("A személy nem szeretne alkoholt inni");
		return false;
	}

	public boolean validKor() {
		if (getKor() > 0) {
			return true;
		} else
			return false;
	}

}
```

testSzemely.java

```
package com.dln82q.Beadando;



/*import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertTrue;*/

import static org.junit.Assert.*;

import org.junit.After;
```

A jegyzőkönyv elkészítéséhez Tompa Tamás dokumentációja szolgált alapul. https://users.iit.uni-miskolc.hu/~tompa/

```
import org.junit.Before;

import org.junit.Test;


public class testSzemely {


   private Szemely Embertest = new Szemely();


   @Before // Minden teszteset előtt lefut

   public void init() {

          Embertest.setNev("Béla");  //A személy neve

          Embertest.setKor(25);           //A személy életkora -> legális
életkor 18

          Embertest.setAlkohol(true);     //A személy szeretne-e alkoholt inni

   }


   @Test

   public void testValidKor() { //Le ellenőrzi, hogy valós életkort adott-e
meg, ami ami a Személy osztályban validkor() metódus definiál

          assertTrue("A személy valós életkort adott meg?",
Embertest.validKor()); //Az Embertest.validkor() visszatérési értékét
ellenőrzi, hogy igaz-e



   }


   @Test

   public void testIszik() // Teszteset, hogy a személy szeretne-e alkoholt
inni

   {

          assertTrue("A személy szeretne alkoholt inni?",
Embertest.isAlkohol()); // Az Embertest.isAlkohol() visszatérési értékét
ellenőrzi, hogy igaz-e

   }
```

```
    @Test

    public void testIhat() // Teszteset, hogy a személy elmúlt-e 18 éves

    {

          assertTrue("A személy ihat alkoholt?", Embertest.getKor() > 18); //Ha
az Embertest.getkor() nagyobb mint 18, akkor a visszatérési értéke igaz, és a
teszteset lefut

    }



    @Test

    public void testNevezo()  //A személy átnevezése Andrásra

    {

          Embertest.setNev("András"); //A név átállítása Andrásra

          assertEquals("A személy valós neve.", Embertest.getNev(), "András");
//Az elvárt névvel (András) összehasonlítja az újonnan megadott nevet

    }



    @After  //Minden teszteset után alapértékekre állítja az Embertest-et

    public void reset()

    {

          Embertest.setKor(0);

          Embertest.setNev("Teszt");

          Embertest.setAlkohol(false);

    }



}
```