GEIAL31H-BL

Szoftvertesztelés című kurzus

Féléves beadandó feladat

Készítette: Kiss Bence

Neptunkód: Y6JRJU

Jegyzőkönyv

Egységteszt

JUnit segítségével

"A **JUnit** egy egységteszt-keretrendszer Java programozási nyelvhez. Eredetileg az IBM által fejlesztett, de jelenleg szabad forrású szoftverként hozzáférhetőegységteszt-keretrendszer, melyet Erich Gamma es Kent Beck írták. A tesztvezérelt fejlesztés (TDD) szabályai szerint ez annyit tesz, hogy a kód írásával párhuzamosan fejlesztjük a kódot tesztelő osztályokat is (ezek az egységtesztek). Ezeken egységtesztek karbantartására, csoportos futtatására szolgál ez a keretrendszer. A JUnit-teszteket gyakran a build folyamat részeként szokták beépíteni. Pl. napi build-ek esetén ezek a tesztek is lefutnak. A release akkor hibátlan, ha az összes teszt hibátlanul lefut."[1]

"*Fő funkciói a következők:*

- tesztek automatikus futtatása egyben vagy részenként
- teszteredmény-áttekintés és kijelzés
- hierarchikus tesztstruktúra-támogatás
- tesztvégrehajtás többféle felületről

Néhány dolog, amire a JUnit nem képes:

- tesztek tervezése
- automatikus tesztprogram generálás
- lefedettség és teljesítménymérés

Az Eclipse tartalmazza a JUnit-ot. Grafikus felhasználói interfészt is kínál a tesztek futtatásához. Beállítása a Preferences ablakba történik. Általában az alapbeállításokon kívül mást nem kell beállítani. Meg lehet neki adni szűrőket, valamint beállíthatjuk, hogy milyen csomagok és osztályok jelenjenek meg a stack trace-ben.

Mit teszteljünk? Csak az API részek tesztelhetők, nincs lehetőség a GUI tesztelésére. Annak, amit tesztelni szeretnénk, láthatónak kell lennie.

Tesztesetek írása JUnit-hoz

A JUnit tesztesetekben gondolkodik, de az ő testcase fogalma egy kicsit eltér a szokásostól. Egy JUnit teszteset a tesztelendő komponens vagy komponensek bizonyos konfigurációját jelenti, amit felállítanak tesztelésre. Ebben a konfigurációban egy vagy több tesztfüggvény fut le, mind a teszteseten belül. A továbbiakban a JUnit teszteset fogalmát használjuk.
Egy JUnit teszteset egy Java osztály, amely a következőket tudja:

- a TestCase osztályból származik, ezzel jelzi a keretrendszernek, hogy futtatni kell. (Ez Junit 4-től már nem szükséges)
- felüldefiniálhatja a setUp() és tearDown() metódusokat, ezek az egyes tesztek kezdőállapotának inicializálásra és utána a takarításra szolgálnak (JUnit 4-ben @Before / @BeforeClass, valamint @After / @AfterClass annotáció)
- minden test-el kezdődő metódus tesztként lesz kezelve az osztályban
- a TestCase osztály assert kezdetű metódusaival tudja definiálni az elvárt működést, például az assertEquals azt ellenőrzi, hogy a két paramétere, az elvárt és a teszt során kapott objektum, egyenlő-e
- a testcase-eket tesztkészletekbe (testsuite) gyűjthetjük össze, így egyszerre tudjuk őket futtatni
- a tesztek futtatására grafikus és parancssori felületet is biztosít a keretrendszer, melyen megjelenik, hogy a futtatott tesztek közül melyiknek mi lett az eredménye

Azokat a beépített függvényeket, amelyeket a teszteléshez használunk, a junit.framework.Assert osztályból vesszük. Ezen assertXXX() függvények működtetik a tesztmetódusokat.

A legfontosabb assert metódusok:

•assertEquals(): két értéket hasonlítunk össze vele. A teszt sikeres, ha az értékek egyenlők

•assertFalse(): egy logikai kifejezést értékel ki. A teszt sikeres, ha a kifejezés hamis

•assertNotNull(): az objektum referenciát null-hoz hasonlítja. A teszt sikeres, ha a referencia nem null

•assertNotSame(): két objektum referencia memória címét hasonlítja össze az „==” operátor segítségével. A teszt sikeres, ha azok különböző objektumokra hivatkoznak

•assertNull(): egy objektum referenciát hasonlítja null-hoz. A teszt sikeres, ha az objektum referencia null

•assertSame(): ez a metódus szolgál a referencia egyenlőségre az „==” operátort használva. A teszt sikeres, ha ugyan arra az objektumra hivatkoznak

•assertTrue(): egy logikai kifejezést értékel ki. A teszt sikeres, ha a kifejezés igaz

Egy testcase elkészítéséhez létrehozunk egy új csomagot, a függőségekhez hozzáadjuk a junit.jarfájlt. Kiválasztjuk, hogy milyen csomagba szeretnénk tenni a tesztet, a New menüből, pedig kiválasztjuk a Junit TestCase-t, nevet adunk neki és létrejön az új testcase, melybe már írhatjuk is a tesztjeinket. "[2]

Példa egy JUnit tesztre

"Egy tetszőleges nevű tesztosztályban a teszt metódusokat meg kell jelölni a @Test annotációval. Lehetőség van definiálni olyan metódust, amely a teszt előtt (ill. utána) hajtódik végre. Ezeket @Before ill. @After annotációval kell megjelölni. A @BeforeClass annotációval megjelölt metódus az első teszt metódus meghívása előtt lefut, ill. az utolsó teszt metódus végrehajtása után az @AfterClass annotációval megjelölt metódus fut le. Az @Ignore annotációval megjelölt teszt nem fut le a teszt osztály futtatása során. Lehetőség van időtúllépést megadni ezredmásodpercben, pl. @Test(timeout=1000). Amennyiben a megadott időkeretet túllépi a teszt, a futása meg fog szakadni és Exception fog kiváltódni. "[1]

Nem létező egységtesztek

"Ez esetben nem magukkal a tesztekkel, hanem azok hiányával van a baj. Minden programozó tudja, hogy teszteket kellene írnia a kódjához, mégis kevesen teszik. Ha megkérdezik tőlük, miért nem írnak teszteket, ráfogják a sietségre. Ez azonban ördögi körhöz vezet: minél nagyobb nyomást érzünk, annál kevesebb tesztet írunk. Minél kevesebb tesztet írunk, annál kevésbé leszünk produktívak és a kódunk is annál kevésbé lesz stabil. Minél kevésbé vagyunk produktívak és precízek, annál nagyobb nyomást érzünk magunkon. Ezt a problémát elhárítani csak úgy tudjuk, ha teszteket írunk. Komolyan. Annak ellenőrzése, hogy valami jól működik, nem szabad, hogy a végfelhasználóra maradjon. Az egységtesztelés egy hosszú folyamat első lépéseként tekintendő.

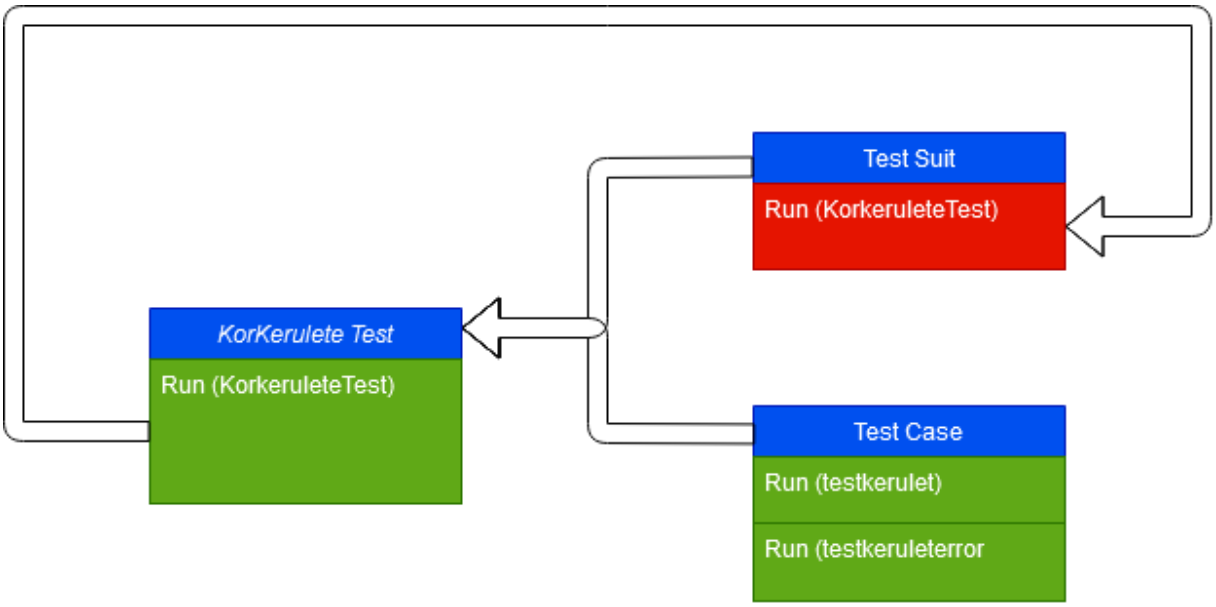
Azon túlmenően, hogy az egységtesztek a tesztvezérelt fejlesztés alapkövei, gyakorlatilag minden fejlesztési módszertan profitálhat a tesztek meglétéből, hiszen segítenek megmutatni, hogy a kódújraszervezési lépések nem változtattak a funkcionalitáson, és bizonyíthatják, hogy az API használható. Tanulmányok igazolták, hogy

az egységtesztek használata drasztikusan növelni tudja a szoftverminőséget."[3]

Gyakorlati példa:

A készített program alkalmas egy kör kerületének kiszámítására azáltal hogy megadjuk annak sugarát. Továbbá teszteljük a program helyes működését.

UML diagram:



***saját készítésű ábra**

A programok forráskódjai:

KorKerulete.java osztály:

KorKerulete.java | KorKeruleteTest.java

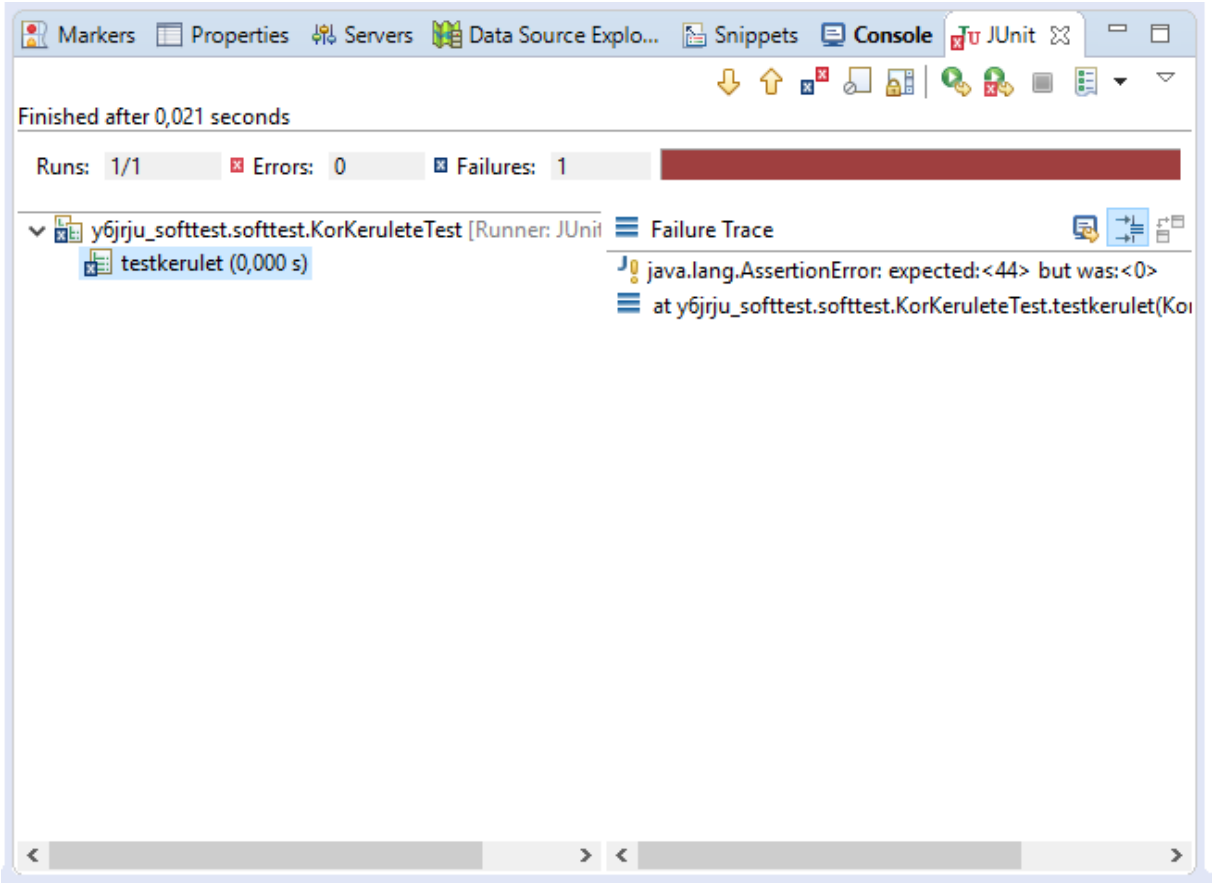
```
1  package y6jrju_softtest.softtest;
2  import java.util.Scanner;
3
4  class KorKerulete
5  {
6     public static void main(String args[])
7      {
8
9           Scanner s= new Scanner(System.in);
10
11          System.out.println("Irja be a kor sugarat:");
12
13       int r= s.nextInt();
14
15       int  korker=(22*2*r)/7 ;
16
17       System.out.println("A kor kerulete: " +korker);
18     }
19
20 public int korker(int i) {
21     // TODO Auto-generated method stub
22     return 0;
23 }
24
25
26 }
```

KorKeruleteTest.java osztály:

KorKerulete.java | KorKeruleteTest.java

```
1  package y6jrju_softtest.softtest;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8
9  public class KorKeruleteTest {
10
11     KorKerulete kerulet = null;
12
13     @Before
14     public void init() {
15         kerulet = new KorKerulete();
16
17     }
18
19     @Test
20     public void testkerulet() {
21
22         int expected = 44;
23         int result = kerulet.korker(7);
24
25         assertEquals(expected, result);
26     }
27
28     public void testkeruleterror () {
29         int expected =44;
30         int result = kerulet.korker(9);
31
32         assertEquals(expected, result);
33     }
34
35 }
36
```

A teszt futtatása utáni eredmény:



Felhasznált irodalmak:

[1] https://hu.wikipedia.org/wiki/JUnit

[2] https://core.ac.uk/download/pdf/160834571.pdf

[3] https://gyires.inf.unideb.hu/GyBITT/21/ch03s02.html