Jegyzőkönyv
Szoftvertesztelés

Féléves feladat
Egységteszt – Junit

Készítette:Azari Csaba
Neptun kód : BW6XDG

2020.11.30

Feladat leírása:

A szoftvertesztelés féléves beadandóm témája az egységtesztelés megvalósítása egy konkrét példán keresztül Eclipse keretrendszerben.

Unit Test

”A komponensteszt a rendszer önálló részeit teszteli általában a forráskód ismeretében (fehér dobozos tesztelés).

Gyakori fajtái:

- unit-teszt,
- modulteszt.

A unit-teszt, vagy más néven egységteszt, a metódusokat teszteli. Adott paraméterekre ismerjük a metódus visszatérési értékét (vagy mellékhatását). A unit-teszt megvizsgálja, hogy a tényleges visszatérési érték megegyezik-e az elvárttal. Ha igen, sikeres a teszt, egyébként sikertelen. Elvárás, hogy magának a unit-tesztnek ne legyen mellékhatása.

A unit-tesztelést minden fejlett programozási környezet (integrated development environment, IDE) támogatja, azaz egyszerű ilyen teszteket írni. A jelentőségüket az adja, hogy a futtatásukat is támogatják, így egy változtatás után csak lefuttatjuk az összes unittesztet, ezzel biztosítjuk magunkat, hogy a változás nem okozott hibát. Ezt nevezzük regressziós tesztnek.

Általános jellemzői:

- A legalacsonyabb szintű tesztelés. A programot felépítő egységek tesztelése

- Unit: egy rendszer legkisebb önálló egységként tesztlehető része.
- Unit tesztekkel ellenőrizhető, hogy egy unit az elvárásoknak megfelelően működik.
- Egy unit függvényeiről ellenőrizzük, hogy különböző bemetek esetén megfelelő eredményt, vagy hibát produkálnak.
- Az egységeket egymástól izoláltan kell tesztelni.

Előnyei:

- A hibák sokkal korábban észlelhetők.
- Minden komponens legalább egyszer tesztelt.
- Az egységek elkülönítése miatt a hibák helyének meghatározása könnyű.
- A funkciók könnyen módosíthatók átalakíthatók.
- Dokumentációs szerep: példákat biztosít egyes funkciók használatára.

- Gyors: A teszteknek gyorsan kell futnia, lassú teszteket senki nem futtatja gyakran, így a hibák nem derülnek ki idejében.

- Független: A teszteknek egymástól függetlennek és bármilyen sorrendben végrehajthatónak kell lennie.

- Megismételhető: A teszteknek bármilyen környzetben, hálozat nélkül is végrehajthatónak kell lennie. A különböző futtatások eredményének meg kell egyeznie.

- Ön ellenőrző: A tesztek eredménye egy boolean, futásuk vagy sikeres, vagy sikertelen.

- Automatikus: A tesztek automatikusan, interakció nélkül futnak.

Junit (4.x)

- Junit egy unit teszt keretrendszer a Java nyelvhez.
- Annotációkkal konfigurálható.
- Elvárt eredmények ellenőrzése Assert-ekkel. „

”A teszt futása során az összes ilyen metódus futtatásra kerül, amelynek háromféle eredménye lehet:

• Sikeres végrehajtás(pass): ez azt jelenti, hogy a teszt rendben lefutott, és az ellenőrzési feltételek mind teljesültek.

• Sikertelen végrehajtás(failure): a teszt lefutott, de valamelyik ellenőrzési feltétel nem teljesült.

• Hiba(error): A teszt futása során valami komolyabb hiba merült fel, például egy Exception dobódott valamelyik tesztmetódus futása során, vagy nem volt tesztmetódus a megadott tesztosztályban.

A JUnit tesztfuttatója az összes, @Test annotációval ellátott metódust lefuttatja, azonban, ha töb ilyen is van, közöttük a sorrendet nem definiálja. Épp ezért tesztjeinket úgy célszerű kialakítani, hogy függetlenek legyenek egymástól, vagyis egyetlen tesztesetmetódusunkban se támaszkodjunk például olyan állapotra, amelyet egy másik teszteset állít be. Egy teszteset általában úgy épül fel, hogy a tesztmetódust az @org.junit.Test annotációval ellátjuk, a törzsében pedig meghívjuk a tesztelendő metódust, és a végrehajtás eredményeként kapott tényleges eredményt az elvárt eredménnyel össze kell vetni. A JUnit keretrendszer alapvetően csak egy parancssoros tesztfuttatót biztosít, de ezen felül nyújt egy API-t az

integrált fejlesztőeszközök számára, amelynek segítségével azok grafikus tesztfuttatókat is implementálhatnak. Az Eclipse grafikus tesztfuttatóját a Run → Run as → JUnit test menüpontból érhetjük el. Egy tesztmetódust kijelölve lehetőség nyílik csupán ennek a tesztesetnek a lefuttatására is. „[3]

”A JUnit a @Test annotáció mellett további annotációtípusokat is definiál, amelyekkel a tesztjeink futtatását tudjuk szabályozni. Az alábbi táblázat röviden összefoglalja ezen annotációkat. [1]

Annotáció	Leírás
@Test public void method()	A @Test annotáció egy metódust tesztmetódusként jelöl meg.
@Test(expected = Exception.class) public void method()	A teszteset elbukik, ha a metódus nem dobja el az adott kivételt
@Test(timeout=100) public void method()	A teszt elbukik, ha a végrehajtási idő 100 ms-nál hosszabb
@Before public void method()	A teszteseteket inicializáló metódus, amely minden teszteset előtt le fog futni. Feladata a tesztkörnyezet előkészítése (bemenetei adatok beolvasása, tesztelendő osztály objektumának inicializálása, stb.)
@After public void method()	Ez a metódus minden egyes teszteset végrehajtása után lefut, fő feladata az ideiglenes adatok törlése, alapértelmezések visszaállítása.
@BeforeClass public static void method()	Ez a metódus pontosan egyszer fut le, még az összes teszteset és a hozzájuk kapcsolódó @Before-ok végrehajtása előtt. Itt tudunk olyan egyszeri inicializálós lépéseket elvégezni, mint amilyen akár egy adatbázis-kapcsolat kiépítése. Az ezen annotációval ellátott metódusnak mindenképpen statikusnak kell lennie!
@AfterClass public static void method()	Pontosan egyszer fut le, miután az összes tesztmetódus, és a hozzájuk tartozó @After metódusok végrehajtása befejeződött. Általában olyan egyszeri tevékenységet helyezünk ide, amely a @BeforeClass metódusban lefoglalt erőforrások felszabadítását végzi el. Az ezzel az annotációval ellátott metódusnak statikusnak kell lennie!
@Ignore	Figyelmen kívül hagyja a tesztmetódust, illetve tesztosztályt. Ezt egyrészt olyankor használjuk, ha megváltozott a tesztelendő kód, de a tesztesetet még nem frissítettük, másrészt akkor, ha a teszt végrehajtása túl hosszú ideig tartana ahhoz, hogy lefuttassuk. Ha nem metódus szinten, hanem osztály szinten adjuk meg, akkor az osztály összes tesztmetódusát figyelmen kívül hagyja.

[1]

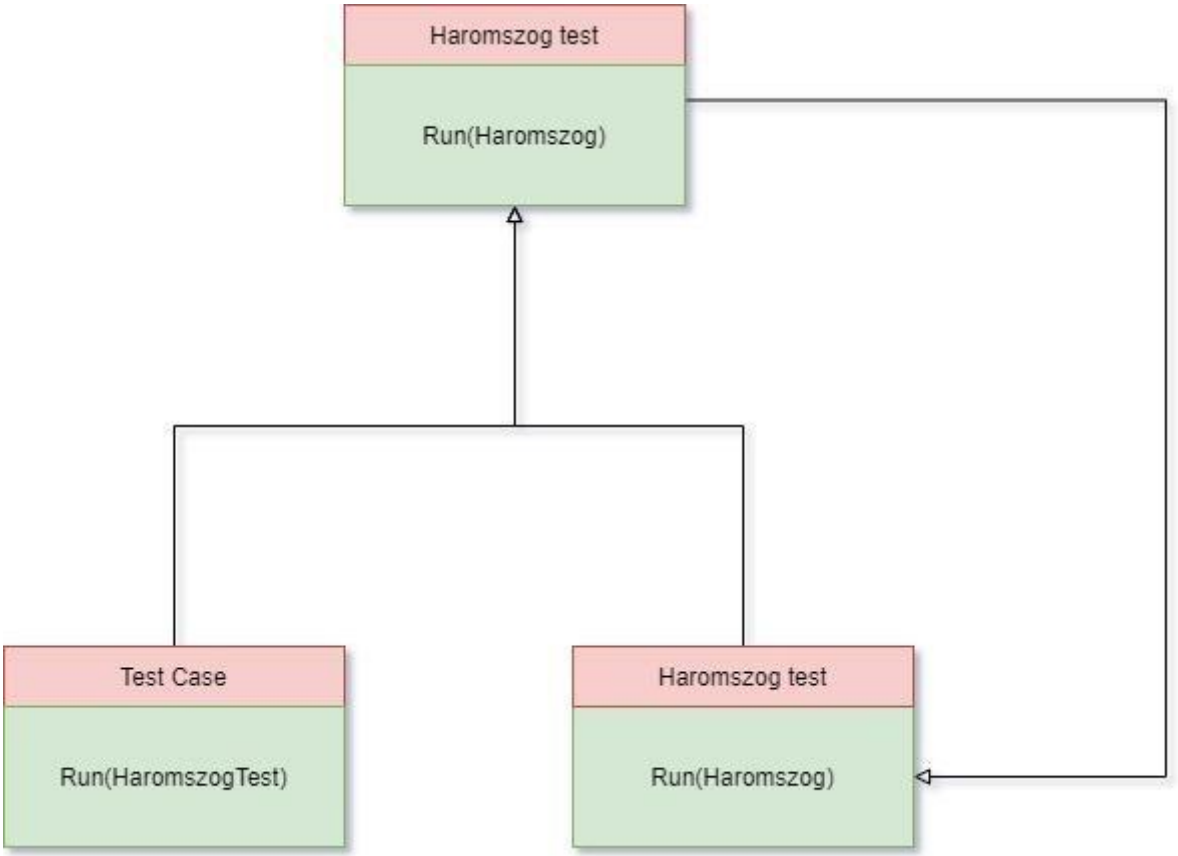
A @Before mindig a teszteset futtatása előtt, az @After utána fut le, minden egyes tesztesetre. A @BeforeClass egyszer, az első teszt, illetve hozzá tartozó @Before előtt, az @AfterClass ennek tükörképeként, a legvégén, pontosan egyszer.
Az tesztekben elhelyezett assert-tel kezdődő metódusok ellenőrzik, hogy a paraméterében megadott kifejezések igazak-e. Ha nem, akkor hibát jelez a tesztelési rendszernek. Az alábbi táblázat bemutatja, milyen egyéb ellenőrző metódusok állnak rendelkezésre JUnit tesztjeinkben:

Metódus	Leírás
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.
assertSame(Object, Object)	Ellenőrzi, hogy a két paraméter ugyanarra az objektumra mutat-e.
assertNotSame(Object, Object)	Ellenőrzi, hogy a két paraméter különböző objektumra mutat-e.
assertEquals(int, int)	Ellenőrzi, hogy a két paraméter egyenlő-e. (Az összes primitív típushoz létezik változata.)
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 azonnal sikertelenné teszi(megbuktatja).

[2]

Az összes, assert-tel kezdődő nevű ellenőrző metódusnak van olyan változata is, amelynek az első paramétere egy hibaüzenet. Ezt a tesztelő környezet írja ki akkor, amikor a teszt megbukik (azaz nem teljesül a feltétel). „ *[2]*

UML:



A diagramm saját készítésű

Forráskód:

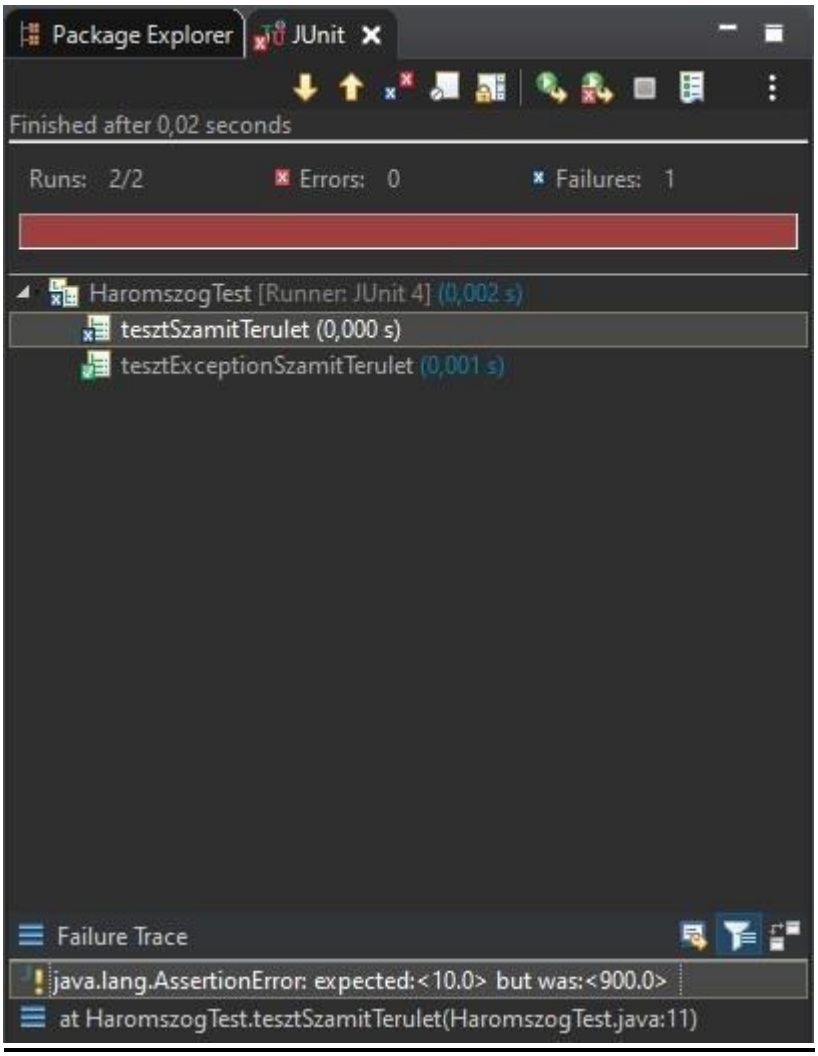
Haromszog.java

```java


public class Haromszog {

    public static double szamitTerulet(int alap, int magassag) {
        if (alap <= 0 || magassag <= 0) {
            throw new IllegalArgumentException("Nem megfelelő paraméterek");
        }
        return (alap * magassag) / 2;
    }

    public static void main(String[] args) {
        System.out.println(szamitTerulet(-1, 1));
    }
}
```

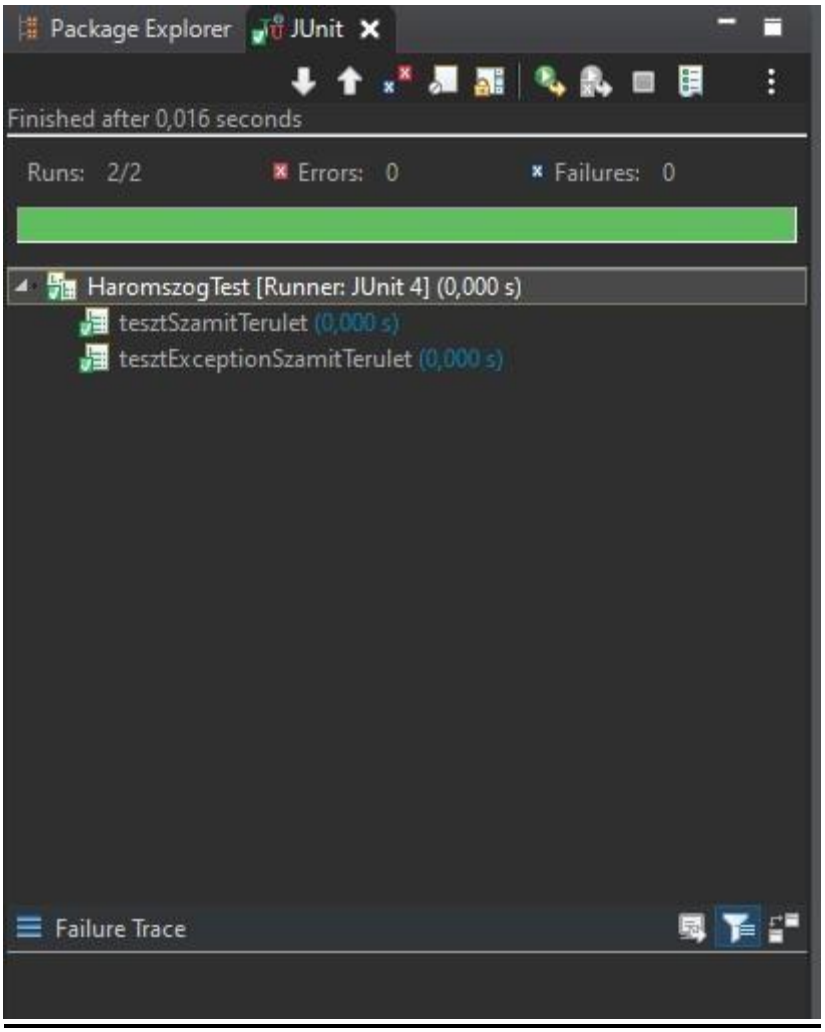
HaromszogTest.java

```java

import org.junit.Test;
import static org.junit.Assert.*;


public class HaromszogTest {

    @Test
    public void tesztSzamitTerulet() {
        assertEquals(525, Haromszog.szamitTerulet(30, 35), 0);
        assertEquals(900, Haromszog.szamitTerulet(40, 45), 0);
    }

    @Test(expected = IllegalArgumentException.class)
    public void tesztExceptionSzamitTerulet() {
        Haromszog.szamitTerulet(0, 35);
    }
}
```

Eredmény:



Jól látható, hogy az első teszt sikeresen lefutott. Az elvárt paraméterekkel megfelelő eredménnyel. A tesztSzamitTerulet viszont hibásan futott le mert nem az elvárt eredményt kaptuk.



Itt pedig minden expected paraméter helyesen lett megadva, ezért a teszt sikeresen lefutott.

Források:

[1] Egységtesztelés JUnit segítségével (unideb.hu)

[2] junit.pdf (elte.hu)

[3] Egységtesztelés – Wikipédia (wikipedia.org)

Mindegyik forrás 2020.11.29 dikén lett letöltve vagy használva.