

Jegyzőkönyv

Szoftvertesztelés

Féléves Feladat

Készítette: Varga Tamás

Neptun azonosító: I8B8WN

Feladat leírása:

A választott feladatom a szoftvertesztelés tárgyból egy **egységteszt** bemutatása java környezetben. Ezt úgy valósítom meg, hogy készítek hozzá egy elméleti leírást a kapott irodalom alapján. Az itt tanultakat bemutatom egy egyszerű gyakorlati példával amit Eclipse fejlesztői környezetben egy maven project keretén belül be is mutatok.

Egységtesztelés JUnit segítségével:

„Egy egységtesztelés tipikusan fejlesztői tesztelés: a tesztelendő programot fejlesztő programozó gyakorlatilag a programozási munkája szerves részeként készíti és futtatja az egységteszteket. Erre azért van szükség, hogy ő maga is meggyőződhessen arról, hogy amit leprogramozott, az valóban az elvárások szerint működik. Az egység tesztelésére létrehozott tesztesetek darabszáma önmagában nem minőségi kritérium: nem állíthatjuk bizonyossággal, hogy attól, mert több tesztesetünk van, nagyobb eséllyel találjuk meg az esetleges hibákat. Ennek oka, hogy a teszteseteinket gondosan meg kell tervezni. Pusztán véletlenszerű tesztadatok alapján nem biztos, hogy jobb eséllyel fedezzük fel a rejtett hibákat, azonban egy olyan tesztesettervezési módszerrel, amely például a bemenetek jellegzetességeit figyelembe véve alakítja ki a teszteseteket, nagyobb eséllyel vezet jobb teszteredményekhez. Egy fontos mérőszám a tesztlefedettség, amely azon kód százalékos aránya,

amelyet az egységteszt tesztl. Ez minél magasabb, annál jobb, bár nem éri meg meg minden határon túl növelni.

A JUnit egységtesztelő keretrendszert Kent Beck és Erich Gamma fejlesztette ki, e jegyzet írásakor a legfrissebb verziója a 4.11-es. A 3.x változatról a 4.0-ra váltás egy igen jelentős lépés volt a JUnit életében, mert ekkor jelentős változások történtek, köszönhetően elsősorban a Java 5 által bevezetett olyan újdonságoknak, mint az annotációk megjelenése. Még ma is sokan vannak, akik a 3.x verziójú JUnit programkönyvtár használatával készítik tesztjeiket, de jelen jegyzetben csak a 4.x változatok használatát ismertetjük.

A JUnit 4.x egy automatizált tesztelési keretrendszer, ami annyit tesz, hogy a tesztjeinket is programokként megfogalmazva írjuk meg. Ha már egyszer elkészítettük őket, utána viszonylag kis költséggel tudjuk őket újra és újra, automatizált módon végrehajtani. A JUnit keretrendszer a tesztként lefuttatandó metódusokat annotációk segítségével ismeri fel, tehát tulajdonképpen egy beépített annotációfeldolgozót is tartalmaz. Jellemző helyzet, hogy ezek a metódusok egy olyan osztályban helyezkednek el, amelyet csak a tesztelés céljaira hoztunk létre. Ezt az osztályt tesztosztálynak nevezzük.

A JUnit tesztfuttatója az összes, `@Test` annotációval ellátott metódust lefuttatja, azonban, ha több 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.

A JUnit a `@Test` annotáció mellett további annotációtípusokat is definiál, amelyekkel a tesztjeink futtatását tudjuk szabályozni.

Az alábbi táblázat röviden összefoglalja ezen annotációkat.

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 (bemeneti 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 inicializációs 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. ábra, forrás: <https://gyires.inf.unideb.hu/GyBITT/21/index.html>

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.

A végrehajtás tényleges eredménye és az elvárt eredmény közötti összehasonlítás során állításokat fogalmazunk meg. Az állítások nagyon hasonlóak az Állítások (assertions)” alfejezetben már megismertekhez, azonban itt nem az assert utasítást, hanem az org.junit.Assert osztály statikus metódusait használjuk ennek megfogalmazására. Ezen metódusok nevei az assert részszostringgel kezdődnek, és lehetővé teszik, hogy megadjunk egy hibaüzenetet, valamint az elvárt és tényleges eredményt. Egy ilyen metódus elvégzi az értékek összevetését, és egy AssertionError kivételt dob, ha az összehasonlítás elbukik. (Ez a hiba ugyanaz, amelyet az assert utasítás is kivált, ha a feltétele hamis.) A következő táblázat összefoglalja a legfontosabb ilyen metódusokat. a szögletes zárójelek ([]) közötti paraméterek opcionálisak.

Az Assert osztály metódusai:

Állítás	Leírás
fail([String])	Feltétel nélkül elbuktatja a metódust. Annak ellenőrzésére használhatjuk, hogy a kód egy adott pontjára nem jut el a vezérlés, de arra is jó, hogy legyen egy elbukott tesztünk, mielőtt a tesztkódot megírnánk.
assertTrue([String], boolean)	Ellenőrzi, hogy a logikai feltétel igaz-e.
assertFalse([String], boolean)	Ellenőrzi, hogy a logikai feltétel hamis-e.
assertEquals([String], expected, actual)	Az equals metódus alapján megvizsgálja, hogy az elvárt és a tényleges eredmény megegyezik-e.
assertEquals([String], expected, actual, tolerance)	Valós típusú elvárt és aktuális értékek egyezőségét vizsgálja, hogy belül van-e tűréshatáron.
assertArrayEquals([String], expected[], actual[])	Ellenőrzi, hogy a két tömb megegyezik-e
assertNull([message], object)	Ellenőrzi, hogy az objektum null-e
assertNotNull([message], object)	Ellenőrzi, hogy az objektum nem null-e
assertSame([String], expected, actual)	Ellenőrzi, hogy az elvárt és a tényleges objektumok referencia szerint megegyeznek-e
assertNotSame([String], expected, actual)	Ellenőrzi, hogy az elvárt és a tényleges objektumok referencia szerint nem egyeznek-e meg

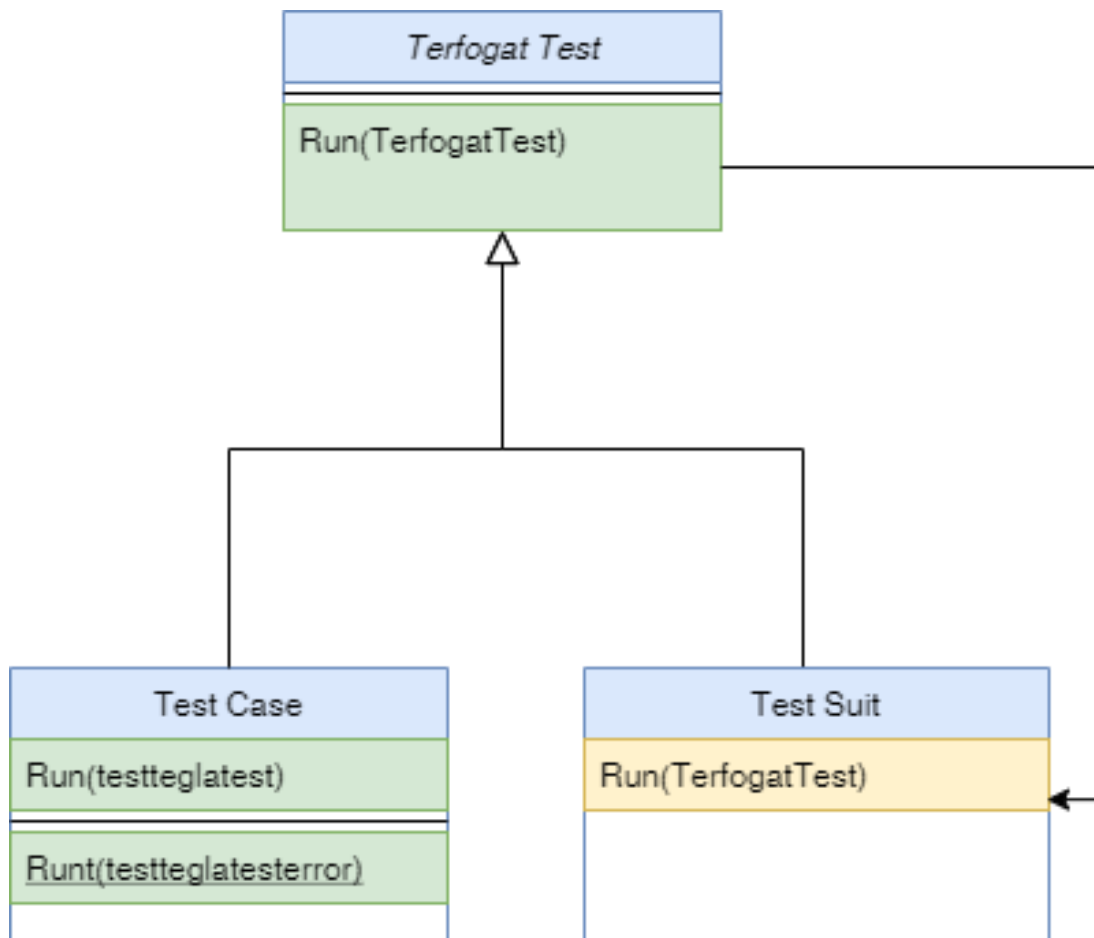
2. ábra – forrás: <https://gyires.inf.unideb.hu/GyBITT/21/index.html>

A JUnit 4-es verziójában jelent meg a parametrizált tesztek készítésének lehetősége. Ezek célja, hogy lehetővé tegyék ugyanazon tesztesetek többszöri lefuttatását, persze rendre különböző értékekkel.

Kivételek tesztelése:

Néha előfordul, hogy az elvárt működéshez az tartozik, hogy a tesztelt program egy adott ponton kivételt dobjon. Például a kivételkezeléssel foglalkozó alfejezetben így működött a push művelet, ha már tele volt a fix méretű verem: `FullStackException` kivételt vált ki, ha már elfogytak a helyek a veremben. Elvárásunkat, mely szerint kivételnek kellene bekövetkeznie, a teszteset `@Test` annotációjának `expected` paraméterének megadásával fogalmazhatjuk meg. Ilyenkor a tesztfutató majd akkor tartja sikeresnek a tesztet, ha valóban a megjelölt kivétel hajtódik végre, és sikertelennek számít minden más esetben.”^[1]

UML:

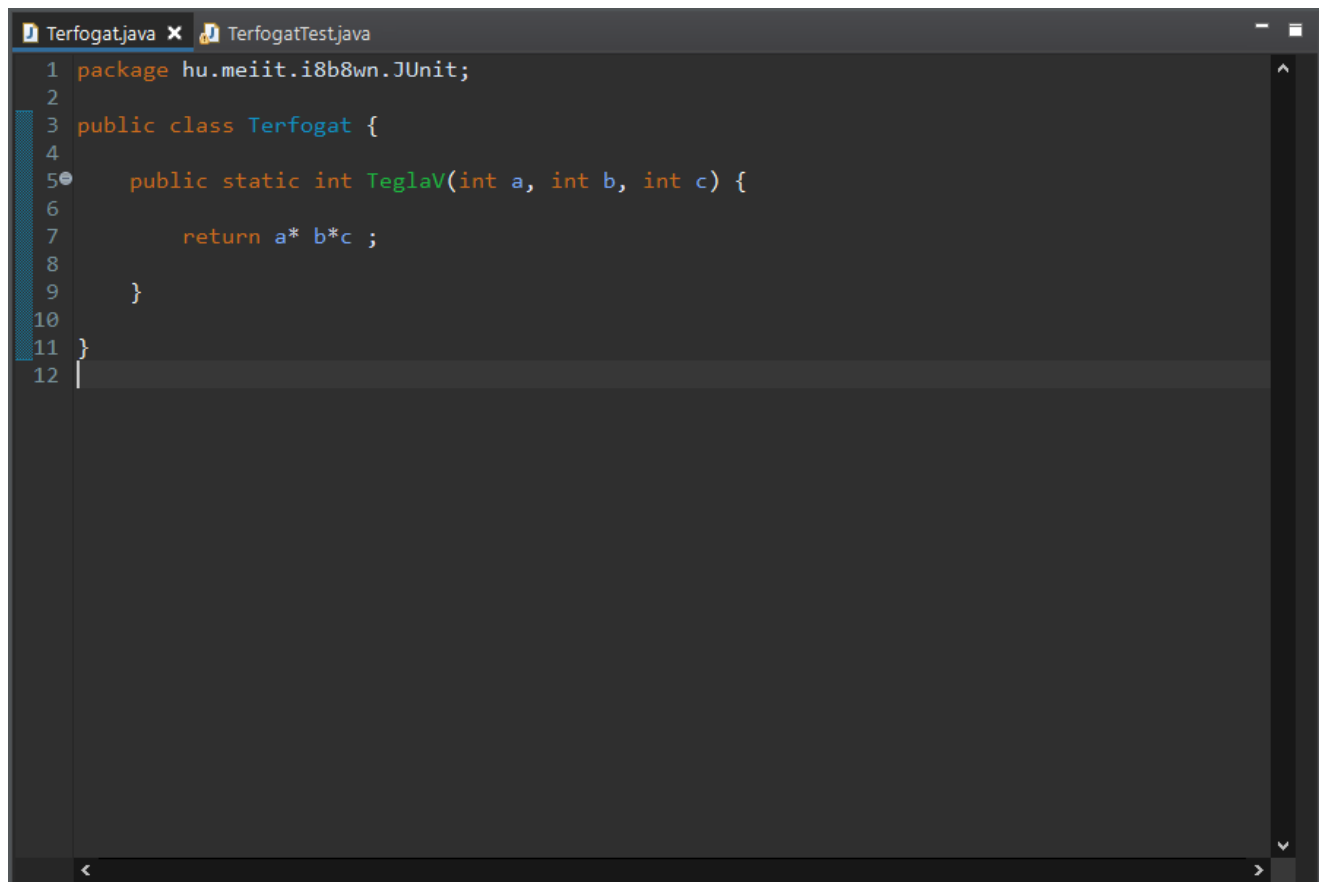


Az UML diagram saját készítés.

Gyakorlati feladat:

A feladatban egy tartály gyártás során egy téglatest alakú tartály térfogatának kiszámítását teszteljük.

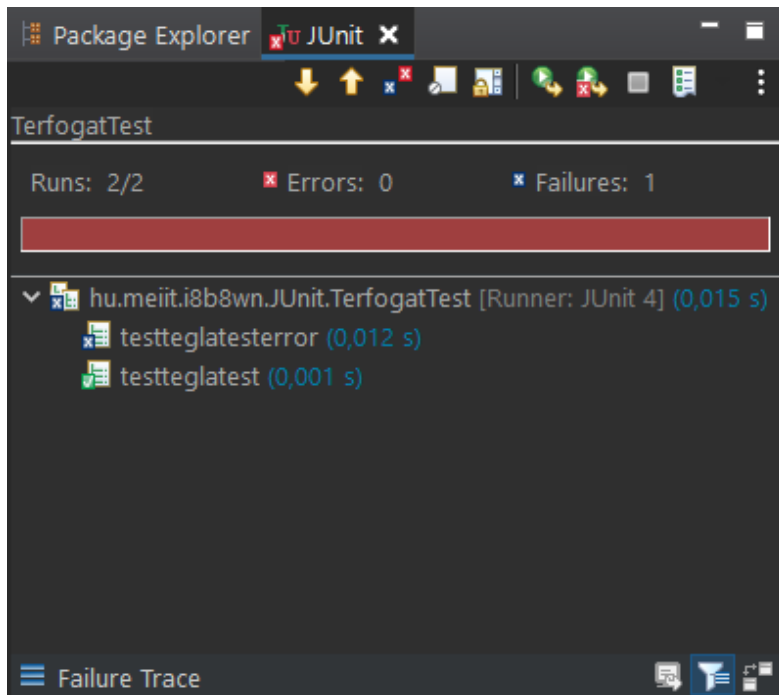
A forráskódok:



```
1 package hu.meiit.i8b8wn.JUnit;
2
3 public class Terfogatt {
4
5     public static int Teglav(int a, int b, int c) {
6
7         return a* b*c ;
8
9     }
10
11 }
12
```

```
Terfogat.java  TerfogatTest.java x
1 package hu.meiit.i8b8wn.JUnit;
2
3 import static org.junit.Assert.assertEquals;
4
5
6
7
8 public class TerfogatTest {
9     Terfogat teglatest = null;
10
11     @Before
12     public void init() {
13         teglatest = new Terfogat();
14     }
15
16     @Test
17     public void testteglatest() {
18
19         int expected = 30;
20         int result = teglatest.TeglaV(3, 2, 5);
21
22         assertEquals(expected, result);
23     }
24
25     @Test
26     public void testteglatesterror() {
27
28         int expected = 30;
29         int result = teglatest.TeglaV(4, 3, 6);
30
31         assertEquals(expected, result);
32     }
33 }
34
35
```

Eredmény:



Jól látható, hogy az első teszt a téglatest sikeresen lefutott az elvárt paramétereknek megfelelő eredménnyel. A második test a téglatesterror viszont hibásan futott le mert nem azt az eredményt kaptuk ami az elvárt lett volna.

Internetes forrás:

[1] 2020.11.28

<https://gyires.inf.unideb.hu/GyBITT/21/index.html>

Kollár Lajos, Sterbinszky Nóra: Programozási technológiák