1. fejezet

A JUnit tesztelési környezet

A JUnit egy szabad forráskódú modultesztelő rendszer, amely Java programjaink automatikus teszteléséhez nyújt segítséget. A rendszer letölthető a http://junit.org webcímről, ahol további hasznos információkat is találhatunk a JUnit használatáról és általában a tesztelésről.

A JUnit redszer használatával javíthatjuk programjaink minőségét, amivel hibakeresési időt takarítunk meg.

1.1. JUnit tesztek

A JUnit tesztek a junit.framework.TestCase leszármazottjai, és a tesztelendő kódot publikus testValami() nevű metódusokban adjuk meg. (Valami tetszőleges azonosító lehet.) 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.

Az alábbiakban példákon keresztül mutatjuk be hogyan kell JUnit teszteket írni.

1.1.1. A legegyszerűbb teszt

Az alábbi kis példa tesztprogram a java.util.HashTable isEmpty() metódusát teszteli egy üres hashtáblával:

```
import junit.framework.TestCase;
import java.util.Hashtable;

/** Hashtable tesztelő osztály */
public class HashtableTest extends TestCase {
    /** Ellenőrzi, hogy egy újonnan készített Hashtable üres-e. */
```

```
    public void testEmptyHashTable() {
        Hashtable ht = new Hashtable();
        assertTrue(ht.isEmpty());
    }
}
```

Az assertTrue metódus ellenőrzi, hogy a paraméterében megadott kifejezés igaz-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).

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). Így a fenti assert sort a következőképpen is írhattuk volna:

```
assertTrue("Az új hashtábla nem üres!", ht.isEmpty());
```

1.1.2. Több teszt egy tesztosztályban

Egy tesztosztályba több tesztet is tehetünk. Ilyenkor hasznosak lehetnek a setUp() és a tearDown() metódusok, amelyek minden egyes tesztmetódus előtt és után lefutnak. Ezekben a tesztek által használt közös változókat lehet inicializálni, fájlokat megnyitni, lezárni, stb. A tesztmetódusok végrehajtási sorrendje nem garantált, így azok nem támaszkodhatnak egymás mellékhatására! Csak annyi biztos, hogy minden egyes teszt előtt lefut a setUp(), utána pedig a tearDown() metódus. Az alábbi teszt a korábbi bővített változata:

```
/** Hashtable tesztelő osztály */
public class HashtableTest extends TestCase {
    Hashtable ht;

    /** Kezdeti beállítások, minden egyes teszt előtt lefut. */
    protected void setUp() throws Exception {
        ht = new Hashtable();
    }

    /** Ellenőrzi, hogy egy újonnan készített Hashtable üres-e. */
    public void testEmptyHashTable() {
        assertTrue("Az új hashtábla nem üres!", ht.isEmpty());
    }

    /** Ellenőrzi, hogy a táblába beszúrt és kivett elem azonos-e. */
    public void testContainsKey() {
        String kulcs = "első kulcs";
        String ertek = "első érték";
        ht.put(kulcs, ertek);
        assertEquals("A hashtáblába beszúrt érték nem azonos a kivett"
                " értékkel!", ertek, ht.get(kulcs));
    }
}
```

1.1.3. Tesztek összefűzése

A junit.framework.TestSuite osztály segítségével összetett teszteket is létrehozhatunk, amelyek több tesztosztályt is egybefoghatnak. Ehhez egy suite() nevű gyártó metódust kell csak biztosítanunk:

```
/** Példa összetett tesztek készítésére: */
public class OsszetettTeszt {

    /** Ez a gyártó metódus készíti el az összetett tesztet: */
    public static TestSuite suite() {
        TestSuite suite = new TestSuite();

        // Egyedi tesztmetódusok hozzáadása:
        suite.addTest(new HashtableTest("testContainsKey"));
        suite.addTest(new HashtableTest("testEmptyHashTable"));

        // Többszörösen összetett teszt:
        suite.addTest(MasikTeszt.suite());

        // Tesztosztály összes metódusának hozzáadása:
        suite.addTestSuite(HarmadikTeszt.class);
```

```
        return suite;
    }
}
```

1.1.4. Kivétel ellenőrzése

Korábban említettük, hogy a teszt során kapott kivétel hibát jelez. De mit kell csinálni, ha azt szeretnénk tesztelni, hogy adott kód kivált-e valamilyen kivételt? Ezt a következőképpen tehetjük meg:

```
    /** Ellenőrzi, hogy null kulcs beszúrása esetén dobódik-e
     * NullPointerException kivétel. */
    public void testNullKey() {
        try {
            ht.put(null, "érték a null kulcshoz");
        } catch (NullPointerException e) {
            return;
        }
        fail("Nem dobódott NullPointerException null kulcs esetén!");
    }
```

1.2. A tesztek futtatása

A tesztek futtatására többféle lehetőségünk van: Használhatjuk a szöveges vagy valamelyik grafikus (awt vagy swing) futtatási felületet. Arra ügyelni kell, hogy a letöltött junit.jar szerepeljen a CLASSPATH-ban.

1.2.1. Szöveges felület

A szöveges felület futtatása a parancssorból a következőképpen történhet:

```
java junit.textui.TestRunner <Tesztosztály neve>
```

Sikeres teszt esetén az OK üzenetet kapjuk:

```
$ java junit.textui.TestRunner tests.OsszetettTeszt
......
Time: 0

OK (6 tests)
```

Hiba esetén pedig a sikertelen vagy hibás tesztek üzeneteiről kapunk értesítést, hasonló formátumban mint ahogyan a le nem kezelt kivételek jelennek meg:

```
$ java junit.textui.TestRunner tests.OsszetettTeszt
.......E.F.F
Time: 0
There was 1 error:
1) testException(tests.FailTest)java.lang.NullPointerException
       at tests.FailTest.testException(FailTest.java:16)
       at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

```
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
There were 2 failures:
1) testSimplyFail(tests.FailTest)junit.framework.AssertionFailedError: i != j
        at tests.FailTest.testSimplyFail(FailTest.java:20)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
2) testAssertTrue(tests.FailTest)junit.framework.AssertionFailedError: Kivett érték != betett érték
        at tests.FailTest.testAssertTrue(FailTest.java:24)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

FAILURES!!!
```

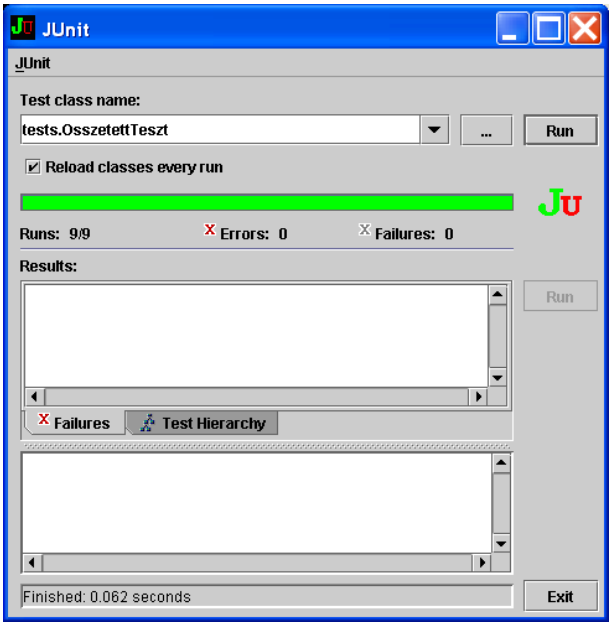
1.2.2. Grafikus felület

A grafikus futtató környezet esetén választhatunk a swing és awt verziók között. A swing verzió több funkcionalitást tartalmaz, ezért inkább annak a használatát javasoljuk. A futtatást a következő paranccsal indíthatjuk:

```
java junit.swingui.TestRunner <Tesztosztály neve>
```

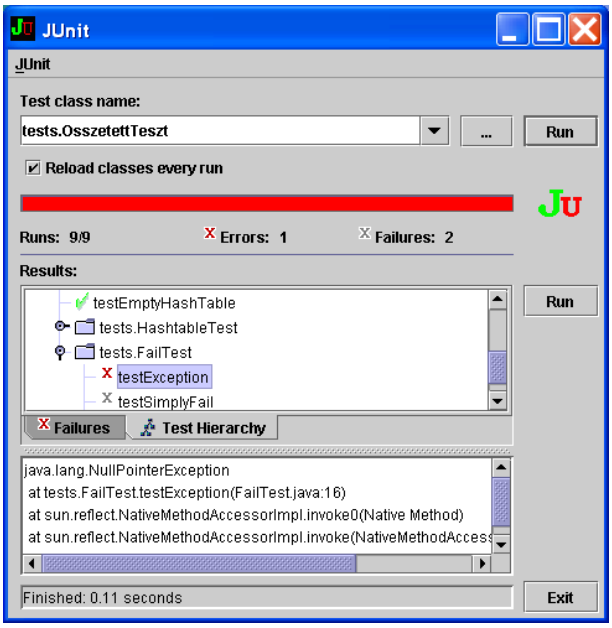
illetve

```
java junit.awtui.TestRunner <Tesztosztály neve>
```



1.1. ábra. A JUnit swing felülete sikeres teszttel.

A grafikus felületen az ablak tetején megválaszthatjuk a futtatandó tesztosztályt (amennyiben a paraméterben megadott nem megfelelő), ez alatt egy jól látható színes csík jelöli a teljes teszt eredményét:



1.2. ábra. A JUnit swing felülete egy sikertelen összetett teszttel. Képünkön a középső doboz a teljes teszthierarchiát tartalmazza, de lehetőség van csak a sikertelen tesztek megjelenítésére is.

- Sikeres futás esetén a csík színe zöldre vált.
- Hiba vagy sikertelen futás esetén a csík színe pirosra vált. Ilyenkor a csík alatti dobozban a hibás és sikertelen tesztmetódusok nevei jelennek meg. A sikertelenség okának részleteit pedig a legalsó doboz tartalmazza.

1.2.3. Futtatás API-ból

Mivel a JUnit rendszer maga is Java-ban íródott, egyszerűen beépíthetünk egy kényelmi main metódust a tesztjeinkbe:

```
public void main(String[] args) {
    junit.textui.TestRunner.run(new TestSuite(OsszetettTeszt.class));
}
```

Így a futtatást egyszerűen a következőképpen letudjuk:

```
java tests.OsszetettTeszt
```

A fenti lehetőségeken kívül a legtöbb fejlesztői környezet is támogatja JUnit tesztek futtatását és debuggolását.

1.3. Tippek és trükkök

Az alábbiakban összegyűjtöttünk néhány ajánlást, amely a JUnit tesztelés eredményességét növeli és megkönnyíti használatát.

1.3.1. Tesztelés mint minőségbiztosítási rendszer

Egy alapos tesztrendszer mindig pontos képet ad arról, milyen állapotban van a programunk. Így például segít annak a tervezésében is, hogy mikor adhatunk ki új verziót. Egy ilyen rendszerrel a háttérben bátrabban állhatunk neki nagyobb kódátszervezési feladatoknak is, hiszen ha a végén a teszt lefut, akkor biztosak lehetünk benne, hogy nem rontottunk el semmit. A tesztek mintegy minőségi tanúsítványt is jelentenek.

Tesztek írása időt takarít meg, mégpedig hibakereséssel és -javítással töltött időt. Ha egy hibajelentés érkezik a programról, érdemes az új hibát bevenni a tesztrendszerbe. Ezzel biztosíthatjuk, hogy ugyanaz a hiba nem kerül ismét vissza a programba.

1.3.2. Egy kis tesztelés – egy kis programozás

A tesztek írását érdemes párhuzamosan végezni a programok írásával. Így a munka nem olyan egysíkú, és hatékonyabban megy mindkettő munkafolyamat.

A tesztjeinket mindig tartsuk 100%-ig szinkronban a tesztelendő kóddal! Ha a tesztek nem az aktuális, hanem a két héttel ezelőtti verziót tesztelik, akkor nem sokat érnek.

Érdemes rendszeres időközönként (pl. minden éjjel automatikusan) lefuttatni a teszteket.

Az Extrém Programozási módszertan (extreme programmming — XP) nagy hangsúlyt fektet a programok tesztelésére. Eszerint a teszteket a programok előtt kell megírni, ezzel mintegy specifikálva a feladatot.

1.3.3. A JUnit rendszer egyszerűsége növeli a hatékonyságot

A JUnit rendszer egyszerűsége miatt hatékonyan írhatunk és futtathatunk teszteket. A tesztrendszer rögtön környezetet teremt a tesztelendő kód számára, így egyből a funkcionalitásra koncentrálhatunk.

A JUnit tesztek automatikusak: nem kell kézzel összehasonlítani a kapott és a várt eredményeket, a tesztek futtatása után azonnal megkapjuk a választ, hogy a programunk jó-e vagy sem.

A JUnit tesztek csoportosíthatósága miatt könnyű egybefogni egy nagyobb programrendszer összes tesztjét, és azokat egyszerre futtatni. Ugyanakkor a teszthierarchia egyes részei egyenként is futtathatók.

Sokszor a teszteket ugyanolyan csomaghierarchiában tárolják mint a tesztelendő osztályokat. Ez abból a szempontból hasznos, hogy így az osztályok csomagszintű láthatósággal rendelkező tagjai is közvetlenül tesztelhetők.

1.3.4. Speciális tesztelési feladatok

Sajnos sok hasznos tulajdonsága ellenére a JUnit rendszer nem minden tesztelési feladatra alkalmas. A JUnit honlapján(http://www.junit.org) számos kiterjesztése is elérhető.

Említésre méltó az Abbot[1] GUI tesztelési felület, amellyel swing vagy awt grafikus felületeinket lehet tesztelni. A teszt egy előre rögzített eseménysorozatot hajt végre a grafikus felületen, és közben ellenőrzéseket végez. A rendszer tartalmaz egy forgatókönyvszerkesztőt, amivel a programunkon végzett egér- és billentyűzetemenyeket tudjuk rögzíteni egy forgatókönyvbe, majd azokat módosítani, ellenőrzéseket szúrni közéjük, stb.

A JUnit további kiterjesztései lehetőséget adnak párhuzamos programok, programhatékonyság (sebesség), J2EE és JSP(Java Server Pages) programok tesztelésére is.

[1] http://abbot.sourceforge.net/