Jegyzőkönyv

Szoftvertesztelés

Egységteszt – JUnit teszt

Készítette: Pohobics Róbert

Neptun kód: GKPD4L

2020.11.29

Feladat leírása:

A program egy derékszögű háromszög átfogójának hosszát számítja ki. JUnit teszt segítségével teszteljük a programot három teszt segítségével.

Az első tesztben pontos értéket adunk meg. Az elvárt érték és az eredmény pontosan egyezik. Tehát a teszten átmegy a program.

A második tesztben az elvárt érték és az eredmény már nem pontosan egyezik meg, de a megengedett tűrésen belül van. Ezért ezen a teszten is átmegy a program. A tűrés, az összehasonlítás pontossága (assertEquals(expected, result, delta);) a delta értékkel állítható.

A harmadik teszten hibás érteket adunk meg. Ennek a tesztnek el kellene buknia. De mivel a teszt argumentumában megadjuk, hogy az az elvárt érték, hogy a teszten elbukik a program, így ezen a teszten is átmegy. „A teszt sikertelen lenne a megadott kivétel hiányában.”

„A **JUnit** egy egységteszt-keretrendszer Java programozási nyelvhez. 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.”

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.”

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 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).

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.”

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 biztosítani.”

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!"); }
```

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.”

Grafikus felület

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

java junit.swingui.TestRunner

illetve

java junit.awtui.TestRunner

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)); }
```

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.”

Tesztelés – 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."

A JUnit rendszer egyszerűsége növeli a hatékonyságot

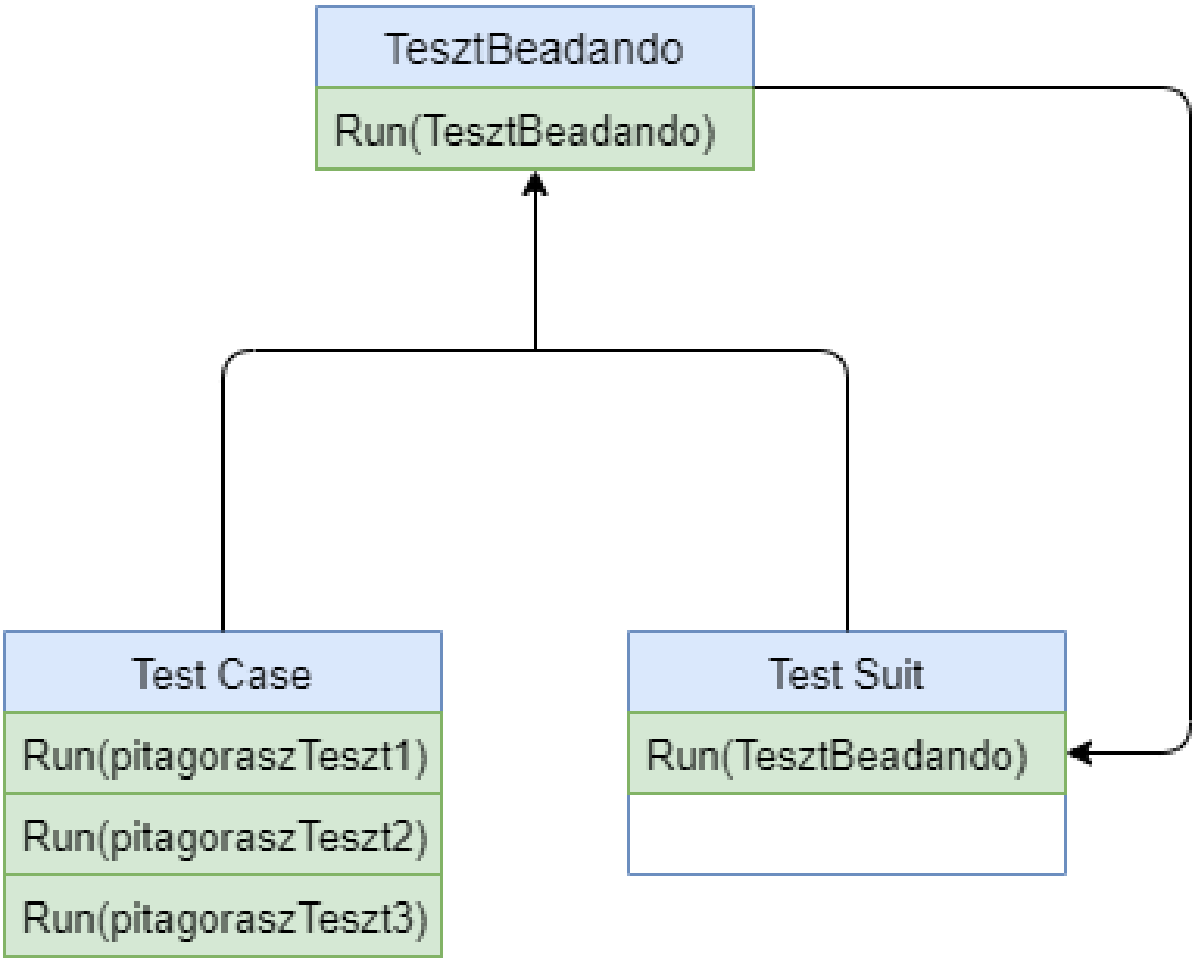
„A JUnit rendszer egyszerűsége miatt hatékonyan írhatunk és futtathatunk teszteket. A tesztrendszer rögtön környezetet teremt a tesztelendő kód számára, így egyből a funkcionalitásra koncentrálhatunk. A JUnit tesztek automatikusak: nem kell kézzel összehasonlítani a kapott és a várt eredményeket, a tesztek 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."

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 Abbot1 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 forgatókönyv szerkesztőt, amivel a programunkon végzett egér- és billentyűzeteseményeket 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.”

UML diagram:


TesztBeadando
Run(TesztBeadando)
Test Case
Run(pitagoraszTeszt1)
Run(pitagoraszTeszt2)
Run(pitagoraszTeszt3)
Test Suit
Run(TesztBeadando)

Források:

https://stackoverflow.com/questions/7554281/junit-assertions-make-the-assertion-between-floats

https://www.youtube.com/watch?v=TpvEqvSx4aI&list=PLyriihBWoulxmMv3x05WHs8R12WfCdqfH&index=3

https://hu.wikipedia.org/wiki/JUnit

https://swap.web.elte.hu/2018191_pt2e/ea08.pdf

http://java.inf.elte.hu/java-5.0/data/junit.pdf