Java tesztelés

Készítette: Goldschmidt Balázs, Budai Péter, BME IIT, 2019.

1. A *Complex* osztály letöltése

a) Hozzunk létre egy új Java projektet *complextest* néven!
b) Hozzunk létre egy *complex* nevű csomagot a projektben!
c) Töltsük le a *Complex* osztály forráskódját a tárgy honlapjáról, és a fájlt másoljuk a projekt *src/complex* nevű mappájába!
d) Fordítsuk le és futtassuk a programot!
e) Nézzük meg, hogy a *Complex* osztály mely metódusokkal rendelkezik!

2. A *Complex* osztály tesztelése

*A **JUnit** egy nyílt forráskódú Java osztálykönyvtár, mely mára a Java nyelven írt programok egység-tesztelésének (unit testing) egyik legnépszerűbb megoldásává vált. A JUnit keretrendszer olyan osztályokat tartalmaz, melyek kényelmesebbé, átláthatóbbá és megismételhetővé teszik a programok alapvető egységeinek (osztályoknak és metódusoknak) a tesztelését.*

a) A JUnit keretrendszert általában a különböző fejlesztő-környezetek is támogatják, így az Eclipse is. Ha használni szeretnénk, a JUnit osztálykönyvtárát hozzá kell adnunk a projektünkhöz. Ennek eléréshez a projekt nevén jobb egérgombbal való kattintásra felugró menüből válasszuk a legalsó, **Properties** menüpontot, majd a megnyíló ablakban navigáljunk a **Java Build Path**, **Libraries** fülhöz, és jelöljük ki a **Classpath** elemet. Ezután a jobb oldali **Add Library...** gombra klikkelve egy felugró ablakot kapunk. Az újabb ablakban válasszuk ki a **JUnit** könyvtárat, a következő képernyőn pedig a legördülő menüből a **JUnit 4** verziót. A korábbi verziót másképpen kell használni, a legfrissebb verzió pedig speciális nyelvi elemeket is alkalmaz, így ezen laborgyakorlat során nem foglalkozunk velük. Végül az **Apply and Close** gombbal zárjuk be az ablakot.

b) Egy szoftver fejlesztése során a tesztosztályokat általában külön kezeljük az alkalmazás logikától, hiszen azok nem kerülnek majd bele a lefordított és összecsomagolt **.jar** fájlba. Ezért az Eclipse projektünkben létre kell hozni egy új forráskönyvtárat a teszt-állományok számára. Ehhez kattintsunk jobb egér-gombbal a projekten, majd válasszuk a **New**, **Source Folder** lehetőséget. A könyvtár neve legyen **test**.

   A tesztállományokat itt ugyanolyan package struktúrába szervezhetjük, mint a programunk többi részét. A konvenció szerint a tesztosztályokat ugyanabba a csomagba helyezzük el, ahol a tesztelt osztály is található. Így a tesztosztályok hozzáférhetnek az alapértelmezett (*default*, *package*) láthatóságú elemekhez is, a külön forráskönyvtár miatt azonban nem kerülnek bele a végleges szoftverbe.

c) Hozzunk létre egy új **JUnit Test Case**-t a **test** könyvtárunkon belül!
   - *test* folderen jobb klikk, majd **New** és **JUnit Test Case**.
   - A felbukkanó ablakban a teszt verziója legyen **New JUnit 4 test**.
   - A csomagnév legyen **complextest**, a teszt neve legyen **ComplexTest**.
   - A **Finish** gombra kattintva véglegesítsünk.

d) Készítsük el az első tesztesetünket. A példa kedvéért most meglehetősen felületesek leszünk, és csak egyetlen számpárral próbáljuk ki az összeadás műveletet.
   - a tesztelő osztály legyen a fent létrehozott *ComplexTest*, nyissuk meg!
   - a tesztmetódus neve legyen *testAdd*
   - a tesztmetódusban hozzunk létre 2 komplex számot (c1 és c2):
     c1 = 2+3i és *c2 = 4+5i*
   - c1 értéke vegye fel a két komplex szám összegét
   - hozzuk létre az elvárt eredményt tartalmazó komplex számot (c3):
     c3 = 6+8i
   - ellenőrizzük (assert!), hogy az elvárt eredmény és valós eredmény egyezik-e! Ugyanazt az eredményt várjuk-e *assertEquals* és *assertSame* használata esetén? Melyiket kell jelen esetben alkalmaznunk?

e) Ha elkészült a tesztosztályunk, akkor az Eclipse **Package Explorer** ablakában jobb egérgombbal rákattintunk a projekt nevére, és kiválasztjuk a **Run As**/**JUnit Test** lehetőséget. A fejlesztőkörnyezet le fogja futtatni nekünk az osztályban található teszteket, és a *package explorer* mellett új fülön (*JUnit*) meg is jeleníti az eredményt.

3. Újabb tesztek felvétele

a) A tesztünkhöz adjunk tesztesetet a szorzásra is (a metódus neve: *testMult*), és ezt is futtassuk! A komplex szorzás definíciója, ha a két szám *a+bi* és *c+di* alakú:

$$(a + bi)(c + di) = (ac - bd) + (ad + bc)i$$

$$\text{pl.: } (\mathbf{2} + \mathbf{3}\boldsymbol{i})(\mathbf{4} + \mathbf{5}\boldsymbol{i}) = (8 - 15) + (12 + 10)i = -\mathbf{7} + \mathbf{22}\boldsymbol{i}$$

b) Mit tapasztalunk?
c) Ha szükséges, javítsunk a *Complex* osztály kódján, amíg a tesztek sikeresen nem futnak.
d) Írjunk tesztet a komplex szám abszolút értékével visszatérő *abs()* metódusra is! A tesztben ellenőrizzük, hogy *0,1+0,1i* abszolút értéke $\sqrt{0{,}02}$ lesz-e. Használjunk megfelelő *assertEquals* metódust!

4. Exception dobásának ellenőrzése

a) Implementáljuk a *Complex* osztály *div* metódusát (ami jelenleg egy fix értéket ad vissza)! A metódus két komplex szám hányodást kell kiszámítsa. Ez az alábbi képlet felhasználásával könnyen megvalósítható, ahol a két komplex szám *a+bi* és *c+di* alakú:

$$\frac{\boldsymbol{a+bi}}{\boldsymbol{c+di}} = \frac{a+bi}{c+di}\cdot\frac{c-di}{c-di} = \frac{(a+bi)(c-di)}{c^2+d^2} = \frac{\boldsymbol{ac+bd}}{\boldsymbol{c^2+d^2}} + \frac{\boldsymbol{bc-ad}}{\boldsymbol{c^2+d^2}}\boldsymbol{i}$$

b) Készítsünk tesztmetódust (*testDiv*) a *ComplexTest* osztályban, amely ellenőrzi, hogy a műveletet jól hajtottuk-e végre! Ehhez segítségként használhatjuk a *Complex* osztály *mult* metódusát:

$$\frac{x}{y} = z \;\Leftrightarrow\; x = z\cdot y$$

c) Definiáljunk egy *DivisionByZero* nevű új kivételosztályt!
d) Ha a *div* metódusban implementált kifejezésben a nevező (c^2+d^2) 0 lenne, akkor dobjunk *DivisionByZeroException*-t!
e) Készítsünk tesztmetódust (*testDivByZero*) a *ComplexTest* osztályban, amely ellenőrzi, hogy ha a (0+0i) számmal osztunk, az elvárt kivételt kapjuk-e!
f) Futtassuk a két tesztet!

5. Tesztkörnyezetek (fixture) használata

A JUnit keretrendszerben a tesztmetódusok egymástól függetlenül kerülnek végrehajtásra, az egyik teszt eredménye nem befolyásolja a másikét. Ezt úgy éri el a JUnit, hogy minden tesztmetódus végrehajtásához egy teljesen új példányt hoz létre a tesztosztályból.

*Összetettebb komponensek esetén gyakori eset, hogy egy adott művelet teszteléséhez a vizsgált objektumot előzőleg egy meghatározott kiindulási állapotba kell hozni. Az ehhez szükséges (néha nagy számú) lépések valójában logikailag nem tartoznak a teszthez. Ráadásul ha több teszteset is felhasználja ugyanazt a kiinduló állapotot, ezeket minden egyes tesztmetódusban meg kellene ismételni. A JUnit ezért lehetőséget ad arra, hogy ezeket a tesztinicializáló (és esetleg -lezáró) kódrészleteket külön metódusokba helyezzük el a tesztosztályon belül, melyeket aztán minden tesztmetódus előtt (és után) meghív. Ily módon a tesztek számára egy úgynevezett tesztkörnyezetet (**test fixture**) alakíthatunk ki. Az inicializáló és lezáró metódusokat szintén annotációk (**@Before** és **@After**) jelölik.*

a) Alakítsuk át a tesztosztályunkat úgy, hogy a *testAdd, testMult,* és *testDiv* metódusokban ne kelljen elvégezni a *Complex* objektumok létrehozását, ezt helyezzük át egységesen egy speciális tesztkörnyezet-inicializáló metódusba (*setUp*)! Ebben a példában ezzel sokat nem nyerünk, de az elvet jól demonstrálja.

b) Figyeljünk arra, hogy a *setUp* metódusban létrehozott *Complex* példányokat elérhetővé kell tenni a tesztmetódusok számára, ezért attribútumként kell hivatkoznunk rájuk. Ha mindent jól csináltunk és újra lefuttatjuk a teszteket, akkor az eredmény nem változott, a teszt továbbra is sikeres.

6. Paraméteres tesztelés

*Főkent különböző algoritmusok fejlesztése során tipikus feladat, hogy egy bizonyos műveletet több különböző bemeneti adatra is le kell tesztelni, hogy megbizonyosodjunk arról, a program szélsőséges esetekben is helyesen működik. Ilyenkor minden bemeneti adathoz külön tesztmetódust kellene készíteni, ami szélesebb paramétertartománynál már kezelhetetlenné válik. Szerencsére a JUnit kínál megoldást ennek kikerülésére is a paraméteres tesztek (*parametrized test*) formájában.*

*Paraméteres teszt esetén a tesztosztály egy, a megfelelő annotációval (**@Parameters**) ellátott statikus metódusán keresztül adjuk meg a bemeneti adatsorokat, és az osztályban szereplő tesztmetódusokat pontosan annyiszor fogja lefuttatni a keretrendszer, ahány ilyen különböző bemeneti adatsor van.*

a) Készítsünk új tesztosztályt *ParamTest* néven!
b) A parametrikus tesztek speciálisak abból a szempontból, hogy az alapértelmezett *JUnit* végrehajtó motor nem képes lefuttatni őket. Ezért a tesztosztályunkat egy speciális annotációval *(@RunWith*) kell ellátni, melynek a paraméterében megadhatjuk, hogy a JUnit azt a végrehajtó osztályt (*org.junit.runners.Parameterized*) használja, amely fel van készülve a parametrikus tesztek kezelésére (lásd az előadásfóliákat).
c) Legyen a *ParamTest* osztálynak három *Complex* típusú attribútuma (*c1*,*c2*,*res*)! Ezeket konstruktorból lehessen állítani! Ezeket az attribútumokat fogják majd az egyes tesztmetódusok a futásuk során elérni és használni.
d) Hozzunk létre egy külön metódust, amivel tesztparamétereket tudunk generálni (*data*)! A statikus metódus **Object** tömbök kollekciójával (**Collection<Object[]>**) térjen vissza. A kollekcióban szereplő **Object[]** típusú elemek pedig egy-egy adatsort tartalmaznak (azért tömb, mert előfordulhat, hogy egy teszthez több paraméter tartozhat, például bemeneti értékek és elvárt kimeneti érték, mint a mi példánkban is lesz).
e) A visszaadandó **Object[]** típusú tömbök elemszámának (**.length**) meg kell egyeznie a tesztosztály konstruktora argumentumainak számával, ugyanis a JUnit a tömb elemeit fogja sorban értékül adni a konstruktor argumentumainak, amikor példányosítja a tesztosztályt. A metódusban az alábbi komplexszám-hármasokat hozzuk létre és adjuk vissza:

1. komplex szám **Ez lesz *c1* értéke**	**2. komplex szám** **Ez lesz *c2* értéke**	**3. komplex szám** **Ez lesz *res* értéke**
3+2i	1+4i	4+6i
1-4i	-1+4i	0+0i
5,3-1,2i	2+0i	7,3-1,2i

f) Legyen egy tesztmetódusunk, amelyik az összeadást (*add*) ellenőrzi: *c1* és *c2* összege megegyezik-e *res* értékével!
g) Legyen egy tesztmetódusunk, amelyik a kivonást (*sub*) ellenőrzi: *res* és *c1* különbsége *c2* értékével kell egyezzen!
h) Futtassuk a teszteket!

7. Kódfedettség-vizsgálat

a) A projekten jobb egérgombbal klikkelve a felugró menüből válasszuk a **Coverage As...** menüpont **JUnit Test** almenüpontját.
b) Nyissuk meg a Complex osztályt, és nézzük meg, mely sorai milyen színűek! Mely részeket sikerült tesztelni?
c) Készítsünk tesztet minden nem tesztelt metódushoz és végrehajtási ághoz, hogy a fedettség 100% legyen!