

Objektumorientált programozás

Unit tesztelés

Goldschmidt Balázs
balage@iit.bme.hu

Unit tesztek

- Verifikáció és validáció több szinten is történik
 - rendszerteszt
 - integrációs teszt
 - egységteszt (unit teszt)
 - stb.
- Egy implementációs egység tesztelése a *unit* teszt
 - OO környezetben a unit tipikusan az osztály/objektum
- Automatizálás és megismételhetőség fontos
 - regressziós tesztelés

Unit tesztek

- A szoftver egy kis részét teszteljük
 - Egy osztály vagy metódus
 - Minden nem-triviális metódushoz
- Tesztek legyenek függetlenek
 - Nincs állapotuk
 - Ne hasson az egyik eredménye a másikra
- Fejlesztő és tesztelő legyen különböző személy
 - szerencsés, ha külön csoportok végzik

Tesztelés – klasszikusan

- Code review (statikus tesztelés)
 - Hasznos, ha betartjuk a kódolási szabályokat
 - Nem mindig elég
- Manuális tesztelés
 - Tesztelő alkalmazást kell írni
 - Egyszerű
 - Karbantarthatatlanná válik
 - Nem szervezett, rendezett
 - Eredmények nem koherensek

Tesztelés – kézzel

- `System.out.println()`
 - Szegény ember debuggere 😊
 - Egyszerűen megvalósítható
 - Minden tele lesz kiíratással
 - hogyan kapcsoljuk ki?
 - A kimenet tipikusan olvashatatlan egy idő után
 - Kézi beavatkozás lehet szükséges

Tesztelés – kézzel

- Debugger használatával
 - IDE támogatás
 - változók, állapot követésére
 - Lassú
 - Körülményes, ha többszálú alkalmazásunk van
 - Minden módosítás után újra kell végezni
 - Emberi közreműködést igényel

Tesztelő környezetek

- XUnit sok nyelvre és környezetre elérhető
 - CppUnit (C++)
 - unittest (python)
 - stb.
- JUnit
 - open source Java tesztelő keretrendszer
 - JAR fájlként is elérhető
 - a tesztek Java-ban készülnek
 - IDE adhat beépített támogatást (pl. Eclipse)
 - külön ablakok, perspektívák, stb.

JUnit tulajdonságai

- *Assertion*-ök az eredmények kiértékeléséhez
 - szabványos módon leírt elvárások
 - valós és elvárt eredmény összehasonlítása
- *Test fixture*-ök tesztek közös adataihoz
 - a közös részeket ki lehet emelni
- *Test runner* megoldások a teszt lefuttatásához
 - automatizálás könnyen előállítható
 - regresszió kényelmesen
 - minden teszt újrafuttatható egy gombnyomásra

JUnit példa: Fraction

```
public class Fraction {
    private int num, den;

    public Fraction(int p, int q) { num = p; den = q; }

    public int getNum() { return num; }
    public int getDen() { return den; }
    public double doubleValue() { return 1.0*num/den; }

    public Fraction add(Fraction f) {
        int d = den*f.den, n = num*f.den+f.num*den;
        return new Fraction(n,d);
    }
    public Fraction mult(Fraction f) {
        int d = den*f.den, n = num*f.num;
        return new Fraction(n,d);
    }
}
```

Példa teszt

■ Naív megoldással

- Objektumok létrehozása – inicializálás, fixture
- Metódusaik meghívása (működtetés)
- Elért állapotot ellenőrzése

```
public class MyTest {  
    public static void main(String[] args) {  
        Fraction f1 = new Fraction(5,2);  
        Fraction f2 = new Fraction(3,2);  
        f1 = f1.add(f2);  
        if (f1.getNum() != 8 || f1.getDen() != 2)  
            System.out.println("add f1 failed");  
        if (f2.getNum() != 3 || f1.getDen() != 2)  
            System.out.println("add f2 failed");  
    }  
}
```

Működtetés

Inicializálás

Ellenőrzés

Példa JUnit teszt

```
public class MyFractTest {
    Fraction f1, f2;
    @Before
    public void setUp() {
        f1 = new Fraction(5,2);
        f2 = new Fraction(3,2);
    }
    @Test
    public void testAdd() {
        f1 = f1.add(f2);
        assertEquals("add f1.num", 8, f1.getNum());
        assertEquals("add f1.den", 2, f1.getDen());
        assertEquals("add f2.num", 3, f2.getNum());
        assertEquals("add f2.den", 2, f2.getDen());
    }
}
```

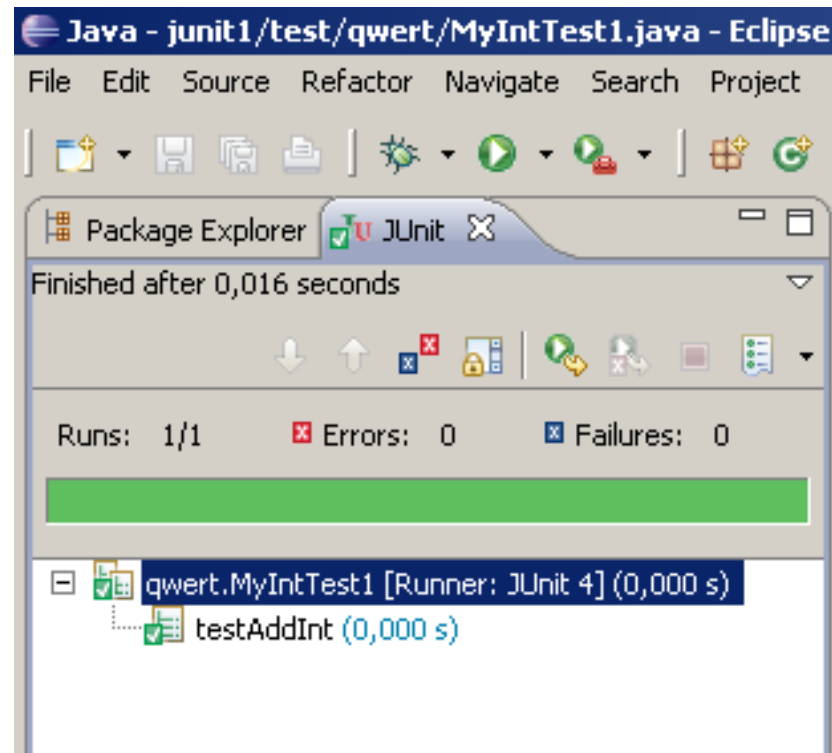
Inicializálás

Ellenőrzés

Működ-
tetés

JUnit Eclipse-ben

- Java Build Path/Libraries/Add Library/JUnit 4
- Run As/JUnit Test



Teszt-metódusok

■ Megkötések

- Minden teszt egy metódusban van implementálva
- Nincs paramétere és visszatérési értéke
- Publikusak
- Annotálni kell: **@Test**
- Nem definiált, de determinisztikus lefutás
 - sorrend ismeretlen, de mindig ugyanaz
 - osztály-annotáció, ha rendezést akarunk (v4.11):
@FixMethodOrder(MethodSorters.NAME_ASCENDING)

Fixture metódusok

■ Bevezetés

- azonos objektumokon akarunk különböző tesztek futtatni
 - közös inicializálás, lebontás, stb.
 - fixture metódus ezt a feladatot végzi el
- a tesztek függetlenek!
 - mindegyik saját objektum-készletet kap
- az objektumokat példányváltozók referálják

Fixture metódusok 2

■ Típusaik

□ **@Before**

- minden teszt előtt lefut, felépíti a kontextust

□ **@After**

- minden teszt után lefut, lebontja a kontextust

□ **@BeforeClass / @Afterclass**

- első/utolsó teszt előtt/után futnak le
- erőforrás-intenzív inicializálás esetén javasolt

Példa JUnit teszt

```
public class MyFractTest {
    Fraction f1, f2;
    @Before
    public void setUp() {
        f1 = new Fraction(5,2);
        f2 = new Fraction(3,2);
    }
    @Test
    public void testAdd() {
        ...
    }
    @Test
    public void testMult() {
        ...
    }
}
```


Fixture és teszt metódusok

- Végrehajtási sorrend 2 teszt esetén:

@BeforeClass metódusok

@Before metódusok

@Test metódus #1

@After metódusok

@Before metódusok

@Test metódus #2

@After metódusok

@AfterClass metódusok

Eredmények kiértékelése

msg a hiba esetére

■ Hogyan ellenőrizzük az eredményt?

- static void **assertTrue**([String msg,] boolean condition)
- static void **assertFalse**([String msg,] boolean condition)
- static void **assertNull**([String msg,] Object object)
- static void **assertNotNull**([String msg,] Object object)
- static void **assertSame**([String msg,] Object exp, Object act)
- static void **assertNotSame**([String msg,] Object unexp, Object act)
- static void **assertEquals**([String msg,] X exp, X act)
- static void **assertArrayEquals**([String msg,] X exp, X act)
- static void **fail**([String msg])

Assert példák

```
public class MyFractTest {
    Fraction f1, f2;
    @Before
    public void setUp() {
        f1 = new Fraction(5,2);
        f2 = new Fraction(3,2);
    }
    @Test
    public void testAdd() {
        Fraction old1 = f1, old2 = f2;
        Fraction res = new Fraction(8,2);
        f1 = f1.add(f2);
        assertEquals("add f1 result", res, f1);
        assertNotSame("add f1 change", old1, f1);
    }
}
```

Eredmények kiértékelése

■ Lebegőpont összehasonlítás speciális

- egészek összehasonlíthatók: `==`
- objektumok összehasonlíthatók: `equals`, `==`
- lebegőpontos ábrázolás kerekítési hibával jár

```
// Complex.equals segédfüggvényei:  
private static double delta = 1e-6; // hibahatár  
private static boolean close(double a, double b) {  
    return Math.abs(a-b) < delta; // |b-a| < delta  
}
```

- `assertEquals double` esetén megkaphatja a deltát is

```
assertEquals(3.0, x, 1e-8); // |x-3.0| < 1e-8
```

Teszt futtatása

■ Parancssorból

- `java org.junit.runner.JUnitCore TestClass1 [...other test classes...]`

■ Alkalmazásból

- `org.junit.runner.JUnitCore.
runClasses (TestClass1.class, ...);`

■ IDE-ből

- klikkeljünk a *run tests...* menüpontra

Tesztek eredménye

■ Success

- Sikeres futás: az elvárt eredményt kaptuk

■ Failure

- valós és elvárt eredmény eltér
- elég, ha az egyik *assertion* hibás lesz

■ Error

- nem várt kivétel dobódik futás közben

■ Ignore

- tesztet figyelmen kívül hagytuk (*assume* vagy *@Ignore*)

Tesztek eredménye 2

- Kivételek kezelése

```
@Test(expected=NumberFormatException.class) ...
```

- Timeout kezelése

```
@Test(timeout=100) ...
```

- Teszt figyelmen kívül hagyása

```
@Ignore("még dolgozunk rajta") @Test ...
```

- ha *Assume.assumeXXX* metódust használjuk, akkor a *failure* helyett *ignore*-t kaphatunk

- *assertNotNull(obj)* → *assumeNotNull(obj)*

- ha *obj* null, a teszt eredménye *ignore* (*failure* helyett)

Exception ellenőrzése

```
public class Fraction {
    private int num, den;
    //...
    public Fraction div(Fraction f) throws DivisionByZero {
        if (f.num == 0) throw new DivisionByZero();
        int n = num*f.den, d = den*f.num;
        return new Fraction(n,d);
    }
}
```

```
public class MyFractTest {
    //...
    @Test(expected=DivisionByZero.class)
    public void testDivExc() {
        Fraction t1 = new Fraction(2,1);
        Fraction t2 = new Fraction(0,1);
        t1 = t1.div(t2);
    }
}
```


Tesztosztályok leszármazása

- Tesztek végrehajtása a leszármazottakban
 - alulról felfele a hierarchiában
 - először a legutolsó leszármazott tesztjei, aztán az őse
- Before metódusok végrehajtása
 - fentről lefele a hierarchiában
- After metódusok végrehajtása
 - alulról felfele a hierarchiában

Paraméteres tesztelés

- Ugyanaz a teszt különböző bemenetekkel
 - tesztadatok és tesztmetódusok descartes-szorzata fut le (minden adat minden metódussal)

```
@RunWith(value = Parameterized.class)
public class ParamTest {
    private int a, b;
    public ParamTest(int a1, int b1) {a = a1; b = b1;}

    @Parameters
    public static collection<Object[]> data() {...}

    @Test public void runTest() {...}
}
```

Paraméterek

Ctr inicializál

Adatok

teszt-metódus

Paraméteres teszt példa

```
@RunWith(value = Parameterized.class)
public class RecipTest {
    private int d,n;
    public ParamTest(int a, int b) {d = a; n = b;}
    @Parameters public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
            {1,2},{2,3},{5,4}
        });
    }
    @Test
    public void runTest() {
        Fraction f = new Fraction(d,n);
        Fraction r = new Fraction(n,d);
        assertEquals("recip test", r, f.reciprocal());
    }
}
```

Teszt-csomagok (suite)

- Csomagoljuk a teszteket!

```
@RunWith (Suite.class)
@Suite.SuiteClasses ({
    TestFeatureLogin.class,
    TestFeatureLogout.class,
    TestFeatureNavigate.class,
    TestFeatureUpdate.class
})
public class FeatureTestSuite {
    // üres gyűjtőosztály
}
```

Tesztek kategorizálása

- Teszteket elláthatjuk kategória-jelöléssel
 - címkézés
 - könnyen csoportosíthatók a különböző tesztek
- A kategóriák egyszerű annotációk
 - kényelmes a használat
- Mind osztály, mind metódus szinten használható

Kategória-példa

```
public interface FastTests {}  
public interface SlowTests {}
```

```
public class A {  
    @Test public void a() { ... }  
  
    @Category(SlowTests.class)  
    @Test public void b() { ... }  
}
```

```
@Category({SlowTests.class, FastTests.class})  
public class B {  
    @Test public void c() { ... }  
}
```

kategóriák jelölése

Kategóriák és csomagok

- Teszt-csomagok definiálhatók kategóriák alapján

```
@RunWith(Categories.class)
@IncludeCategory(SlowTests.class)
@ExcludeCategory(FastTests.class)
@SuiteClasses({ A.class, B.class })
public class SlowTestSuite {
    // Csak a SlowTests kategóriájú tesztek
    // A és B tesztesztályban
}
```

ExternalResource

- Különböző teszt-osztályok közös inicializálása

- az inicializáló kód egy *ExternalResource*-ba kerül

- *public void before()*: minden teszt előtt lefut

- *public void after()*: minden teszt után lefut

- minden teszthez újonnan jön létre

- hozzá kell adni a kívánt teszt-osztályhoz

```
@Rule public ExternalResource resource =  
    new MyExternalResource ();
```

- osztályszintű Rule (hasonlóan a *BeforeClass*-hoz)

```
@ClassRule ...
```


JUnit konvenciók

- Tesztek és forráskód elkülönítése
 - Külön mappába (**src** vs. **test**)
 - Release változatban nincs teszt
- A teszt és forrásosztályok azonos csomagban
 - a hozzáférési jogok így tiszták
- Ha lehet, egy forrásosztály egy tesztosztállyal
 - Nem kötelező betartani 😊

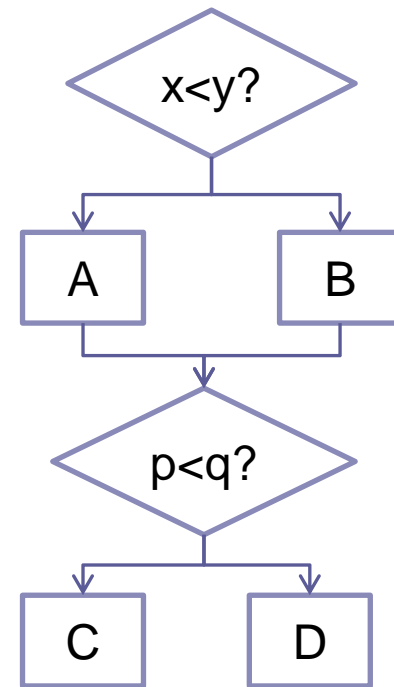
Tesztfedettség

- A kód mekkora részét teszteltük?

- fedettség = $\text{tesztelt} / \text{összes}$
- kivételeket is beleértjük
- statikus vs dinamikus tesztek
 - minden lehetséges út bejárva?

- Cél: 100% fedettség

- de még így is lehet hiba ☹️



Futásidejű ellenőrzések

- Nem tekintjük tesztnek
- *Design by contract*
 - prekondíció
 - amit a metódus elvár a hívótól
 - invariáns
 - ami a metódushívás hatására nem változik
 - posztkondíció
 - ami a metódushívás végén előáll
 - Néhány nyelvben beépítve (pl. Eiffel)

Design by contract példa

```
public interface Stack<T> {  
    /** t-t a verem tetejére teszi */  
    void push(T t);  
    /** leemeli a legfelső elemet */  
    T pop();  
    /** visszaadja a legfelső elemet */  
    T top();  
    /** visszaadja a tárolt elemek számát */  
    int size();  
}
```

Design by contract példa

■ `void push(T t)`

- *invariáns*: a korábbi elemek sorrendje ne változzon
- *posztk.*: `t` a verem tetején

■ `void pop()`

- *prek.*: verem nem üres
- *invariáns*: a legfelső elem kivételével a többiek megőrzik a sorrendjüket
- *posztk.*: legfölső elem lekerül a tetőről

Design by contract példa

■ `T top()`

- *prek.* : verem nem üres
- *invariáns* : minden elem a helyén marad
- *posztk.* : a visszaadott elem a legfölsővel azonos

■ `int size()`

- *invariáns* : minden elem a helyén marad
- *posztk.* : visszaadott érték azonos az elemek számával

Tesztek vs Design by contract

- Tesztek generálhatók a DbC leírásból
 - pre, inv, poszt formálisan specifikálандó
 - feltételekből tesztesetek készíthetők
- Teszteket fejlesztés közben futtatjuk
 - mielőtt használatba kerül a szoftver
- DbC normál használat közben
 - mi a teendő, ha valamelyik feltétel sérül?
 - főleg prototipizálásakor alkalmazzuk

Java: *assert* kulcsszó

- Futásidejű feltétel-ellenőrzés
 - *assert* kifejezés;
 - pl. `assert (stack.size() > 0);`
 - *assert* kifejezés1: kifejezés2;
 - pl. `assert (i % 5 == 0 : i);`
 - ha hiba van, az *i* bent lesz az üzenetben
- Amikor az *assert* kifejezése hamis lesz
 - *AssertionError* dobódik
 - Error típus -> nem kell és nem is lehet kezelni

Java *assert* szabályok

- Ne használjuk publikus metódus paraméterének ellenőrzéséhez
 - inkább *IllegalArgumentException*, *NullPointerException*, stb.
- Assert-kifejezésbe ne tegyünk működést
 - pl. `assert (stack.pop() == x); // NE!`
`Object o = stack.pop();`
`assert (o==x); // OK`
- Engedélyezésükhöz
 - `javac -source 1.4 MyClass.java`
 - különben *assert* nem kulcsszó

Java *assert* engedélyezése

■ Csomagok és osztályok esetén

- *-enableassertions* vagy *-ea*
- *-disableassertions* vagy *-da*
 - argumentumok (pl. *-ea:hu.bme.iit...*)
 - *none*: teljes alkalmazásra
 - **packageName** . . . : csomagra és alcsomagjaira
 - . . . : a névtelen csomagra
 - **classname**: egy adott osztályra

■ A rendszerosztályok esetén

- *-enablesystemassertions* vagy *-esa*
- *-disablesystemassertions* vagy *-dsa*

JUnit 5

■ Módosulások

- `@Before`, `@After` → `@BeforeEach`, `@AfterEach`
- `@BeforeClass`, `@AfterClass` → `@BeforeAll`, `@AfterAll`
- `@Test(expected=NumberFormatException.class)`
→ `assertThrows(Exception.class, ...)`
- `@Test(timeout = 1)`
→ `assertTimeout(Duration.ofMillis(1), ...)`
- `@Category(osztály)` → `@Tag(string)`
- ...