Programozási nyelvek II. JAVA

9. gyakorlat

2017. november 13-17.

Tartalom

Szoftver minőségbiztosítás

Adott: Specifikáció (követelmények halmaza)

Cél: A követelményeket teljesítő ("helyes") program

Megközelítések:

- **Tesztelés:** Futtatás során ellenőrizzük a követelmények teljesülését
- **Formális verifikáció:** A helyesség (automatikus) levezetése a (formális) specifikációból
- **Program szintézis:** Helyes program levezetése a specifikációból
- **És még mások** ...

Tesztelés

Mit tesztelünk?

- **feketedoboz (blackbox)**: A funkcionalitást teszteljük, az implementáció ismerete nélkül
- fehérdoboz (whitebox): Az implementációt teszteljük a kód ismeretében.

Tesztelés

Mekkora egységet tesztelünk?

Egységteszt Legkisebb önállóan működő alapegységeit (osztály, metódus)

Integrációs teszt Összetartozó egységek együttműködését (csomag).

Rendszer teszt Rendszer, alkalmazás egészét fejlesztői oldalról. Nem-funkcionális követelmények, használati esetek.

Átvételi teszt Üzembehelyezett rendszer, alkalmazás egészét felhasználói oldalról.

Egységteszt

A rendszer legkisebb önállóan működő egységeit teszteli (osztály, metódus).
Célok:

- Követelmények teljesülésének ellenőrzése kimeneten
- Funkcionalitás (vagy annak hiányának) dokumentálása
- Kód változtatásakor hibák detektálása (ld. Test driven development, Continous integration)

Elvárások:

- 1:n kapcsolat: Egy teszt pontosan 1 egységet tesztel, de 1 egységre több teszt is juthat.
- Az egységek egymástól, környezettől, felhasználótól függetlenek (nincs mellékhatás)
- Minőségmutató: lefedettség (tesztelt publikus metódusok száma)

JUnit 4

- Egységteszt-keretrendszer JAVA-hoz.
- Letöltés és tutorial
- Két kódkönyvtár (`.jar`)
  - junit-4.12.jar
  - hamcrest-core-1.3.jar
- Az útvonalnak fordításkor és futtatáskor szerepelnie kell a *classpath*-ban (`-cp` kapcsoló).
- Útvonal elválasztók
  - Linux: kettőspont (:)
  - Windows: pontosvessző (;)

Példafuttatás

- Hozzuk létre a `SimpleTest.java` tesztfájlt a két .jar fájl mellett.
- Könyvtárszerkezet:
  - `SimpleTest.java`
  - `hamcrest-core-1.3.jar`
  - `junit-4.12.jar`
- Fordítás és futtatás (Linux):

```
$ javac -cp .:junit-4.12.jar:hamcrest-core-1.3.jar
    SimpleTest.java

$ java -cp .:junit-4.12.jar:hamcrest-core-1.3.jar
        org.junit.runner.JUnitCore SimpleTest
```

SimpleTest.java

```
import org.junit.Test;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.fail;
// statikus import: metodus explicit osztalyhivatkozas
    nelkul hasznalhato

public class SimpleTest {
  @Test
  public void testSuccess() {
    assertTrue(1 == 1);
  }

  @Test
  public void testFail() {
    assertTrue(0 == 1);
  }
}
```

További eszközök:

```
@Test
public void manualSuccess() {
  return;
}

@Test
public void manualFail() {
  fail("Always fails");
  assertTrue(1 == 1);
}

@Test(timeout=2000)
public void abortWhenInfiniteLoop(){
  while(true){}
}
```

További eszközök:

```
@Test
public void testUnexpectedException() {
  throw new RuntimeException("Varatlan hiba tortent");
}

@Test(expected = RuntimeException.class)
public void testExpectedException() {
  throw new RuntimeException("Varakozasnak megfelelo
      hiba tortent");
}

@Test
public void testExpectedException2() {
  try{
    int x = 3/0;
    fail("Nem tortent kivitel");
  } catch(ArithmeticException e){}
}
```

A tesztek szerkezete

Előkészítés-Kivitelés-Ellenőrzés (Arrange-Act-Assert, AAA)

```
@Test
public void testStringIsReversed() {
  // arrange //
  String input="Hello world";

  // act //
  String result = MyLib.reverse(input);

  // assert //
  assertEquals("dlrow olleH", result);
}
```

Módszertan

Jelölés

p tesztelő, f program implementáció, X inputtér, $T \subseteq X$ a tesztesetek halmaza. $\forall x \in X : p(f, x) \in \{0, 1\}$.

Definíció

Az f program *átment a teszteken*, ha $\forall t \in T : p(f, T) = 1$.

Feladat (Tesztek tervezése)

Keressük p tesztelőt, amivel:
f program átmegy a teszteken $\iff$ f program teljesíti a követelményeket.

Módszertan

Néhány módszer fekete-doboz teszteléshez

- Decision table testing
- All-pairs testing
- **Equivalence partitioning**
- **Boundary value analysis**
- Cause–effect graph
- Error guessing
- State transition testing
- Use case testing
- User story testing
- Domain analysis
- Syntax testing
- Combining technique

Ekvivalencia particionálás

```
public static int add(int a, int b){...}
```

Kimerítő tesztelés: minden lehetséges paraméterezésre teszteljük az outputot.

- Az `int` típusú egésszámokat a memóriában 32 hosszú bitsorozatok reprezentálják $\Longrightarrow$ Az `int` értékhalmazának mérete: $|int| = 2^{32}$.
- Az inputtér mérete 2 paraméter esetén: $|int \times int| = |int| \cdot |int| = |int|^2$
- Az inputtér mérete 2 paraméter esetén n paraméter (pl. lista, tömb) esetén: $|int|^n$
- A paraméterek számának és típusának függvényében ez nagyon nagy szám is lehet $\Longrightarrow$ **A kimerítő tesztelés nem hatékony.**

Ekvivalencia particionálás

Eml.: A paraméterek számának és típusának függvényében $|int|^n$ nagyon nagy szám is lehet $\Longrightarrow$ **A kimerítő tesztelés nem hatékony.**

- **Ötlet (Ekvivalencia particionálás):** Az inputteret osszuk fel ekvivalencia-osztályokra.

- Két elem kerüljön egy osztályba, ha a program a két értéken várhatóan nagyon hasonló módon (vagy ugyanolyan természetű hibával) fut le.
- Innentől elegendő csak az osztályok reprezentánsait tesztelnünk.
- Ideális esetben a partíciók uniója kiadja az inputteret.

Ekvivalencia particionálás

Például

- `int`: Pozitív számok, negatív számok, 0.
- `String`: Csak számokat tartalmazó, csak betűket tartalmazó, kisbetűs, nagybetűs, stb.

Fontos: A partíciók megfelelő kialakítását a szakterület és a feladat határozza meg, nincs általános módszer.

Határeset elemzés

A partíciók határán levő értékek gyakran különlegesek, ezekre érdemes külön is megvizsgálni a program működését. Például

- `int`: `Integer.MAX_VALUE`,1,0,-1, `Integer.MIN_VALUE`.
- `String`: Üres, (csak) szóközöket tartalmazó, null-referencia, stb.

Tippek

1. Ismerd meg a feladat szakterületét
2. Vedd számba milyen input/output értékek tartozhatnak az egységhez.
3. Particionáld az inputteret, vizsgáld az egyes partíciók határelemeit is.
4. Particionáld az outputteret (beleértve a hibákat is), vizsgáld meg, hogy a program kimenetén megjelenhetnek-e a partíciók egyes elemei. Ebben segítség a végrehajtási utak megbecslése.
5. Kereshetsz általános (pl. algebrai) tulajdonságokat is a követelményben leírt egységre, melyeknek a teljesülését ellenőrizni lehet (sanity check).
6. Írd meg a JUnit teszteket

Feladatok szerkezete

1. Adott egy követelményleírás és egy letölthető kódkönyvtár, melynek ismerjük a publikus metódusait (az *interfészt*).
2. Tervezd meg a teszteket, amelyekkel jellemezhető a program működése, hatásköre!
3. Implementáld a teszteket JUnit segítségével!

Feladat (Greeter)

Követelmények

- A program egy nevet (`String`) vár paraméterként, és egy üdvözlő szöveget ad vissza úgy, hogy a paraméter elé konkatenálja a `"Hello, "` karakterláncot.
- Amennyiben a paraméter üres vagy `null`, a metódus `IllegalArgumentException` kivételt dob.

Feladat (Greeter)

Interfész

```
public class Greeter{
  public static String greet(String name){...}
}
```

Kódkönyvtár

- Letölthető erről a linkről.

Megoldás (Greeter)

Tesztelési terv

- Inputtér: sztringek
- Outputtér: sztringek, kivétel
- Particionálás: alfanumerikus, speciális karaktereket tartalmazó (ezen belül szóköz, sorvége, escape),...
- Sztring határesetek: üres, szóközzel kezdődő/végződő, csak szóközökből álló, nagyon hosszú string,...

Megoldás (Greeter)

- `Greeter.class`
- `GreeterTest.java`
- `hamcrest-core-1.3.jar`
- `junit-4.12.jar`

```
$ javac -cp .:junit-4.12.jar:hamcrest-core-1.3.jar
    GreeterTest.java
$ java  -cp .:junit-5.12.jar:hamcrest-core-1.3.jar
    org.junit.runner.JUnitCore GreeterTest
```

Feladat (Triangle)

Követelmények

- Adott egy háromszög három oldala. Az oldalak alapján sorold a háromszöget az alábbi osztályok közül a megfelelőbe.
- Egyenlő oldalú (EQUILATERAL), egyenlő szárú (ISOSCELES), egyéb (SCALENE).
- Ha több is megfelelő, az itt megadott sorrend szerint a legkorábbi osztályba sorold.
- Ha az oldalak nem háromszöget írnak le, a program dobjon IllegalArgumentException kivételt.

Feladat (Triangle)

Interfész

```
public enum TriangleType{
  EQUILATERAL, ISOSCELES, SCALENE
}
public class Triangle{
  public Triangle(int a, int b, int c){...}
  public TriangleType classify(){...}
}
```

Kódkönyvtár

- TriangleType.java és Triangle.class letölthető a megfelelő linkekről.

Megoldás (Triangle)

Tesztelési terv

- Inputtér: egésszám hármasok
- Outputtér: `EQUILATERAL, ISOSCELES, SCALENE`
- Sanity check
- Háromszög vagy nem háromszög (háromszög-egyenlőtlenség)
- Paraméterek sorrendje
- `int`-hármasok particionálása, határesetei, elfajzott esetek (0 oldalhosszú háromszög, (0,0,0) egyenlőoldalú-e, stb...)

Feladat (Adder)

Követelmények

- A program két számot vár paraméterként és visszaadja azok összegét.
- Fiktív körülmények miatt a paraméterek és a visszatérési érték típusa is karakterlánc (String). Az átadni kívánt számokat tehát előbb ilyen módon el kell kódolni, majd az eredményt megfelelően dekódolni kell.
- Ha a karakterlánc nem értelmezhető számként, a program dobjon IllegalArgumentException kivételt.

Feladat (Adder)

Interfész

```
public class Adder {
  public String add(String a, String b){...}
}
```

Kódkönyvtár

- Letölthető erről a linkről.