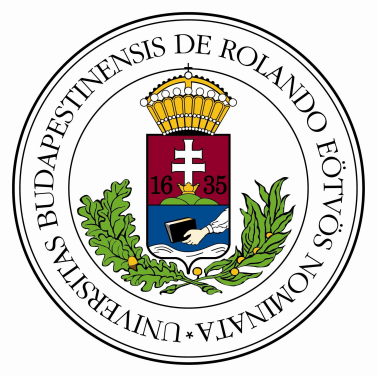
Programozási nyelvek – Java

Tesztelés



Kitlei Róbert

ELTE Eötvös Loránd Tudományegyetem

Egységteszt (Unit test)

- A program legkisebb, önálló részeinek kipróbálása
  - Egység lehet: metódus, **osztály**, komponens/modul
  - Nem egységteszt, ha külső függőségei vannak
    - Ilyen pl.: fájlrendszer, adatbázis, hálózat használata
- Kis, gyorsan lefutó, független tesztek
  - Futási időben működik
  - Fekete dobozos: az egység belső szerkezete nem ismert
    - Csak az osztály publikus interfészét (metódusait) használja
- Funkcionális helyességet tesztel: a lefutás az elvárt eredményt adja-e
  - Nem cél: hatékonyság tesztelése

Egységteszt: helyesség

- Nem *bizonyítja*, csak *alátámasztja* a helyességet
- Regressziók felfedése: hamar kiderül, ha hibás a kód
- Egyúttal dokumentálja, mi az elvárt működés
  - Együtt fejlődik a kóddal: ezt a fordítóprogram „érti'' és ellenőrzi
  - A szöveges dokumentáció elavulhat
- Lefedettség (code coverage)
- Sok hibát megelőz még fejlesztés alatt
  - Nagyobb munkaigény kezdetben
  - Olcsóbb lehet az utólagos hibajavításnál
    - Az éles rendszer jobban működik

Egységteszt: módszerek

- Tesztvezérelt fejlesztés (test driven development, TDD)
  1. Új teszteset hozzáadása, ami még "piros" (sikertelen)
  2. Kód írása/fejlesztése: minden teszteset legyen "zöld" (sikeres)
  3. A kód minőségének javítása (refaktorálás): minden "zöld"
- Egyéb tesztelési megközelítések
  - Naplózás, kiírások használata
  - Hibakeresés (debugging)
  - Összetettebb: integrációs ~, teljesítmény~, stressz/terhelési ~, automatizált ~, véletlenített/tulajdonság alapú ~, mock ~, folyamatos ~ (CI/CD), ...
  - Felhasználói élmény: elfogadási ~, biztonsági ~, használati ~, lokalizációs ~, ...
  - Formális helyességbizonyítás

Egységtesztelő: így használandó

- Egy tesztelő metódus egyetlen vizsgálatot tartalmaz
- A lehető legegyszerűbb szerkezet: ciklus, elágazás, véletlen, ... nélkül
- Saját kódot teszteljünk, ne könyvtárakat
- Lebegőpontos típusok tesztelése: az eredménynek lehet pontatlansága
  - Extra paraméter: tűréshatár (delta)
- Számítás adatainak struktúrája: egyszerűtől bonyolultig
  - `null`
  - üres szöveg, `0`
  - konstruktorhívás, majd getter
  - kis, pozitív értékek
  - egy-két lépéssel összeállított adatok
  - negatív/szokatlan/extrém értékek
    - pl. `Integer.MAX_VALUE` vagy `Double.MIN_VALUE`
    - kivételek
  - hosszabb "történet", több hívással

Egységtesztelés: FIRST

- Fast: μs-ms
- Isolated: egymástól és külvilágtól elkülönülő
- Repeatable: megismételhető
  - Nincsenek mellékhatások
  - Nincs nemdeterminisztikus futás
- Self-verifying: önellenőrző
  - Minden teszt elbukhat
  - Minden bukásnak pontosan egy oka lehet
- Timely: a kóddal együtt bővülnek/fejlődnek a tesztek
- vagy Thorough: lásd előző fólia

Outline

JUnit

- Java nyelvű megvalósítások közül a legnépszerűbb
- A jelenleg legújabb kiadás: JUnit 5, 1.9.2 verzió
- Innen letölthető a `jar` fájl
  - A letöltött fájl átnevezhető rövidebb névre, pl. `junit5.jar`
- Tesztelendő osztály: system under test (SUT)
  - Tegyük fel, hogy a `time.Time` osztályt teszteljük
  - A SUT kódja a `time/Time.java` fájlban van
  - A tesztelő kód a `time/TimeTest.java` fájlba kerül
- Fordítás: `javac -cp junit5.jar time/TimeTest.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTest`

JUnit teszteset: Arrange-Act-Assert

```java
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    // Step 1: Arrange
    Time sut = new Time(0, 0);
    // Step 2: Act
    int hour = sut.getHour();
    // Step 3: Assert
    assertEquals(0, hour);
  }
}
```

JUnit teszteset: Arrange-Act-Assert, röviden

```java
package time;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class DemoTest {
  @Test
  void testHour00_00() {
    assertEquals(0, new Time(0, 0).getHour());
  }
}
```

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }

org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)
```

JUnit teszteset kimenete

- Fontos: az elvárt érték az első paraméter
  - Ez mindig egy konstans legyen, ne számított érték

```
@Test
void wrongResultTest() { assertEquals(5, 2+2); }

org.opentest4j.AssertionFailedError: expected:<5> but was:<4>
  ... (sok, érdektelen információ)
  at testing.DemoTest.wrongResultTest(DemoTest.java:9)
  ... (még több sor)

@Test
void wrongOrderTest() { assertEquals(2+2, 5); }

org.opentest4j.AssertionFailedError: expected:<4> but was:<5>
  at testing.DemoTest.wrongOrderTest(DemoTest.java:9)
```

JUnit: ritkábban használatos eszközök

```
fail();

assertEquals("y", "x", "expected to be y");

assertEquals("y", "x", () -> "Also expected to be y");
```

```
... AssertionFailedError
    at time.JUnitDemoTest.testFail(JUnitDemoTest.java:19)
...: expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV1(JUnitDemoTest.java:24)
...: Also expected to be y ==> expected: <y> but was: <x>
    at time.JUnitDemoTest.testMessageV2(JUnitDemoTest.java:29)
```

JUnit: ritkábban használatos eszközök

```java
@Test
public void testTrue() {
  assertTrue(2 + 2 == 4);
}

@Test
public void testFalse() {
  assertFalse("it's true" == "it's " + true);
}
```

- Az `assertEquals` jobb: precízebb a hibaüzenet
- Figyelem: a == nem helyes egyenlőségvizsgálat a `String` típuson!
  - Az ellenpárja, != szintén rossz

```
...: expected: <false> but was: <true>
    at time.JUnitDemoTest.testFalse(JUnitDemoTest.java:14)
```

JUnit: paraméterezett teszt: azonos működés több adaton

```java
@CsvSource("this is some text,4")
@ParameterizedTest
public void testSplit(String text, int partCount) {
    assertEquals(partCount, text.split(" ").length);
}

@DisplayName("Computing the Fibonacci numbers")
@ParameterizedTest(name = "fib({0}) = {1}")
@CsvSource({"13,6", "21,7"})
public void testFib(int expected, int num) {
  assertEquals(expected, Fibonacci.fib(num));
}
```

```
'-- Computing the Fibonacci numbers [OK]
  +-- fib(6) = 13 [OK]
  '-- fib(7) = 21 [OK]
```

JUnit: paraméterezett tesztek szövegblokkokkal

Forrás: JUnit 5 dokumentációja

```
@ParameterizedTest(name = "[{index}] {arguments}")
@CsvSource(useHeadersInDisplayName = true, textBlock = """
  FRUIT,         RANK
  apple,         1
  strawberry,    700_000
  'lemon, lime', 0xF1
  """)
public void testWithCsvSource(String fruit, int rank) {
  // ...
}
```

Kimenet:

```
[1] FRUIT = apple, RANK = 1
[2] FRUIT = strawberry, RANK = 700_000
[3] FRUIT = lemon, lime, RANK = 0xF1
```

JUnit: kivételek

```
@Test
public void testInvalidTime() {
  InvalidTimeException exception =
    assertThrows(InvalidTimeException.class, () -> {
      new Time(123, 456);
    };
    assertEquals("/ by zero", exception.getMessage());
  }
}
```

- `() -> { ... }`: a kivételt potenciálisan kiváltó kódrészlet ide kerül
- A `.class` tekinthető speciális adattagnak
- Itt megengedett két `assertX` írása is egy tesztelő metódusba
  - Sokszor nincs üzenet, akkor változó sem szükséges

JUnit: tömbök

- Tömbök tesztelése: külön `assertArrayEquals` művelettel
  - `assertEquals` nem jó
  - Más adatszerkezetek jól működnek

```
@Test
public void testFibArray() {
  int[] fibs = Fibonacci.fibsUpTo(6);
  assertArrayEquals(new int[] { 1, 1, 2, 3, 5, 8 }, fibs);
}
```

JUnit: életciklus

```
public class TimeTest {
  private Time time;

  @BeforeEach
  public void beforeEach() {
    time = new Time(12, 34);
  }

  @Test void test1() { assertEquals(12, time.getHour()); }
  @Test void test2() { assertEquals(34, time.getMin());  }
  @Test void test3() { assertEquals(35, time.inc().getHour()); }
}
```

- `@BeforeEach`: tesztesetek ismétlődő adatainak közös beállítása
  - A tesztesetek nem zavarják egymást, mert mindig újrainicializál
- `@AfterEach`: pl. átmeneti fájlok törlésére
- `@BeforeAll`, `@AfterAll`: ritkán használatos

Outline

CheckThat

- A szokásos JUnit tesztek a kód funkcionalitását vizsgálják
- Ez az eszköz a kód szerkezetét ellenőrzi
- Használata intuitív
- A megvalósító kód túlmutat a félév anyagán, nem kell megérteni

CheckThat példa

```
package time;

import static check.CheckThat.Condition.*;
import check.CheckThat;

import org.junit.jupiter.api.Test;

public class StructureTest01_Time {
  @Test
  public void test1() {
    CheckThat...
  }

  ...
}
```

CheckThat példa

```
CheckThat.theClass("time.Time")
   .thatIs(NOT_ABSTRACT, PUBLIC)
   .hasConstructorWithParams("int", "int")
     .thatIs(PUBLIC);
}

CheckThat.theClass("time.Time")
   .hasFieldOfType("hour", "int")
     .thatIs(PRIVATE, NOT_STATIC, MODIFIABLE)
     .has(GETTER, SETTER);

CheckThat.theClass("time.Time")
   .hasMethodWithParams("getEarlier", "Time")
     .thatIs(PUBLIC, NOT_STATIC)
     .thatReturns("Time");
```

CheckThat hibaüzenetek

```
org.opentest4j.MultipleFailuresError: Multiple Failures (1 failure)
    ...: Nincsen megfelelő GETTER metódus
                                ehhez az adattaghoz: Time.hour
```

További üzenetek:

```
...: A Time.hour visszatérése nem megfelelő
...: A Time.hour láthatósága nem megfelelő
```

- Egy változóval angolra is állítható

CheckThat használata

```
package time;
import org.junit.platform.suite.api.*;

@Suite
@SelectClasses({
  StructureTest01_Time.class,
  StructureTest02_WorldTimes.class
  ,TimeTest.class
  ,WorldTimesTest.class   // (*)
})
public class TestSuite {}
```

- Fordítás: `javac -cp junit5.jar time/TimeTestSuite.java`
- Futtatás: `java -jar junit5.jar -cp . -c time.TimeTestSuite`
- A tesztelő kódhoz nem kell hozzányúlni
  - Ha még csak a `Time` osztály van készen, `(*)` kikommentezendő

CheckThat használata, elkülönülő tesztelő kód

```
root
+ project
  + src
    + time
      + Time.java
  + test
    + time
      + StructureTest01_Time.java
      + StructureTest02_WorldTimes.java
      + TestSuite.java
      + TimeTest.java
```

```
root
+ tester
  + junit5.jar
  + check
    + CheckThat.java
```

- Továbbra is ugyanabban a csomagban van a SUT és a tesztelő
- Fordítás: `javac -cp ../tester/junit5.jar;../tester test/time/*.java src/time/*.java`
- Futtatás: `java -jar ../tester/junit5.jar -cp ../tester;test;src -c time.TestSuite`

CheckThat használata, elkülönülő tesztelő kód

```
'-- JUnit Platform Suite [OK]
  '-- TestSuite [OK]
    '-- JUnit Jupiter [OK]
      '-- StructureTest01_Time [OK]
        +-- test1() [OK]
        +-- test2() [OK]
        '-- test3() [OK]
```