

V&V alapfogalmak

JUnit, Mockito

Simon Károly

simon.karoly@codespring.ro

- Verifikáció és validáció – verification and validation - V&V: ellenőrző és elemző folyamatok amelyek biztosítják, hogy a szoftver megfelel a specifikációjának és kielégíti a kliens igényeit.
- V&V (Boehm)
 - Verification: Are we building the product right?
 - Validation: Are we building the right product?
- Statikus (átvizsgálás, automatizált kódelemzés, helyesség bizonyítás) és dinamikus (tesztelés)
- Elfogadási szint:
 - a V&V célja nem a tökéletes program.
 - Az elfogadási szintet befolyásoló tényezők: szoftverfunkció, felhasználói elvárások, piaci környezet.
- Debugging: a V&V megállapítja, hogy vannak-e hibák, a debugging behatárolja és kijavítja ezeket.
 - Regressziós tesztelés szükségessége

Statikus módszerek

- Szoftver forrásreprezentációjának átvizsgálása:
 - Szerepkörök, módszerek
 - az IBM szakértői által javasolt módszer
 - Cleanroom (formális specifikáció, inkrementális fejlesztés, strukturált programozás, statikus verifikáció + statisztikai tesztelés)
 - A hibákra elszigeteltként tekintünk
 - Hibák listája (checklist): adat, vezérlés, I/O, interfész, tárkezelés, kivételkezelés stb.
- Automatizált statikus elemzés
- Helyességbizonyítás
 - Algoritmusok helyességének bizonyítása (pl. Floyd, Hoare módszer)
 - Modellellenőrzés (temporális logikák szerepe)
 - További módszerek
 - Előnyök és hátrányok
- Kritikus rendszerek
 - Megbízhatóság validálása (működési profilok szerepe)
 - Megbízhatóság előrejelzése (ROCOF – Rate of Occurrence of Failures, függvények, extrapoláció)
 - Biztonságosság szavatolása, biztonsági indoklások (pl. ellentmondáson alapuló bizonyítások)
 - Folyamat szavatolása (standardok, szabályok, minősített intézmények/szakemberek, jelentések)
 - Védettség értékelése (tapasztalat/eszköz alapú, "tiger teams", formális verifikáció)
 - Minősítési rendszerek

Dinamikus módszerek

- Hiányosság tesztelés és statisztikai tesztelés
- Komponens (modul) tesztelés és integrációs tesztelés
- Funkció-orientált és objektumorientált rendszerek tesztelése
- Fekete doboz és üveg (fehér) doboz tesztelés (stuktúrateszt)
- Módszerek:
 - Ekvivalenciaosztályozás
 - Útvonal tesztelés
 - Interfésztesztelés (paraméter, osztott memóriájú, procedurális, üzenettovábbító)
 - Stressztesztelés
 - Stb.
- Integrációs tesztelés:
 - Inkrementális tesztelés
 - Fentről lefele vs. lentől felfele
- OO tesztelés:
 - Szintek (metódusok, osztályok, objektumcsoportok, rendszer)
 - Forgatókönyv alapú tesztelés

További módszerek, eszközök

- Tesztelési eszközrendszerek: tesztmenedzserek, tesztadat generátorok, előrejelzők, állomány összehasonlító, jelentésgenerátorok, dinamikus elemzők/profilerek, szimulátorok
- Hibakövető/hibajelentő rendszerek (Jira, Redmine, Bugzilla stb.)
- Statikus elemzők: IDE integrált funkciók, külső plug-in-ok (pl. Eclipse: EclEmma, FindBugs stb.)
- Profiler-ek: beépített IDE eszközök, plug-in-ok, dedikált alkalmazások (YourKit, VisualVm, Eclipse Memory Analyzer stb.)
- Automatizált GUI tesztelőeszközök (pl. Selenium)
- Elfogadási tesztek leíró nyelvek (pl. Cucumber)
- Összetett code review eszközök (pl. SonarQube)
- Stb.

SonarQube

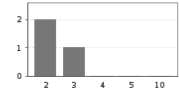
Dashboards Projects Measures Issues

Settings Simon Károly Search

- RegionRank default
- Dashboard
 - Hotspots
 - Reviews
 - Time Machine
 - SCM Stats
 - Issues
- TOOLS
- Components
 - Issues Drilldown
 - Design
 - Libraries
 - Clouds
 - Compare
- sonarqube

Version 1.0.0-SNAPSHOT - May 25 2013 11:39 Time changes...

LCOM4
1.0 /class
2.9% files having LCOM4>1



Alerts : Rules compliance < 85, Critical violations > 5


Lines of code
4,684
6,274 lines
1,692 statements
102 files

Classes
107
42 packages
412 methods
86 accessors

Package tangle index
7.4%
> 7 cycles

Dependencies to cut
4 between packages
5 between files

Complexity
1.6 /method
6.3 /class
6.6 /file
Total: 678



Methods Files

Issues
646
Rules compliance
74.6%

Blocker
Critical
Major
Minor
Info

0
21
245
348
32

Unit Tests Coverage
2.6%
2.7% line coverage
2.2% branch coverage

Unit test success
50.0%
1 failures
1 errors
4 tests
1 skipped
5.9 sec

Comments
4.6%
227 lines
32.0% docu. API
223 undocu. API

Duplications
1.1%
68 lines
4 blocks
2 files

Key: com.regionrank.regionrank:default
Language: Java
Profile: Codespring Default (version 29)
Alerts: RSS Feed



	May 24 2013	May 25 2013 1.0.0-SNAPSHOT
Lines of code	4,684	4,684
Lines	6,274	9,274
Statements	1,692	1,692
Files	102	102
Classes	107	107
Methods	412	412
Accessors	86	86

	May 24 2013	May 25 2013 1.0.0-SNAPSHOT
Comments (%)	4.6%	4.6%
Comment lines	227	227
Public documented API (%)	32.0%	32.0%
Public undocumented API	223	223

	May 24 2013	May 25 2013 1.0.0-SNAPSHOT
Issues	545	545
Blocker issues	9	9
Critical issues	21	21
Major issues	245	245
Minor issues	348	348
Weighted issues	1,188	1,188

	May 24 2013	May 25 2013 1.0.0-SNAPSHOT
Complexity	678	678
Complexity method	1.6	1.6
Complexity class	6.3	6.3
Complexity file	6.6	6.6

	May 24 2013	May 25 2013 1.0.0-SNAPSHOT
Coverage	2.6%	2.6%
Line coverage	2.7%	2.7%
Branch coverage	2.2%	2.2%
Unit tests success (%)	50.0%	50.0%
Unit tests failures	1	1
Unit tests errors	1	1
Unit tests duration	5.9 sec	5.9 sec

Profile Codespring Default Time changes...

Severity	Count
Blocker	0
Critical	21
Major	245
Minor	348
Info	32

Rule	Count
Immutable Field	43
Correctness - Class defines fields that are used only as locals [fb-contrib]	21
Non-transient non-serializable instance field in serializable class	15
Malicious code vulnerability - May expose internal representation by incorporating reference to mutable object	12
Local Variable Name	12
Uncommented Empty Constructor	10

Project	Count	Project	Count	Project	Count
RegionRank Admin UI default	28	com.regionrank.ui.admin.layout.region	12	RegionLayout	12
RegionRank Admin WidgetSet default	6	com.regionrank.ui.admin.window	7	UploadLayout	6
RegionRank Backend default	5	com.regionrank.component.shapeeditor.leaflet	6	ShapeEditorComponent	6
Region Rank Client UI default	4	com.regionrank.ui.admin.layout.common	6	AddCategoryWindow	4
		com.regionrank.ui.client.layout	3	AddShapeWindow	3
		com.regionrank.backend.auth.event	2	AuthenticationEvent	2

	RegionRank default 1.0.0-SNAPSHOT May 25 2013	Red Dog 1.0.0-SNAPSHOT Jul 11 2012
Lines of code	4,684	5,405
Complexity	678	788
Comments (%)	4.6%	5.5%
Duplicated lines (%)	1.1%	2.8%
Issues	646	67
Coverage	2.6%	8.5%

```
private Map<String, List<double[]>> shapes = new HashMap<>();
```

Immutable Field | Open | about 1 month

Private field 'shapes' could be made final; it is only initialized in the declaration or constructor.

Name	Rules compliance	Coverage	Duplicated lines (%)	Build time	Package dependencies to cut
Region Rank Client UI default	82.3%	0.0%	0.0%	May 25 2013	1
RegionRank Admin UI default	74.7%	0.0%	0.0%	May 25 2013	1
RegionRank Admin WidgetSet default	48.3%	0.0%	1.3%	May 25 2013	0
RegionRank API default	68.2%	0.0%	0.0%	May 25 2013	0
RegionRank Backend default	81.4%	8.2%	2.4%	May 25 2013	2
RegionRank Parent default				May 25 2013	

Unit testing

Unit testing

- Unit: egy program legkisebb különállóan tesztelhető része (procedurális nyelv esetén egy függvény, objektumorientált nyelv esetén egy metódus).
- A tesztesetek egymástól függetlenek, a tesztek a programozók írják.
- Unit test: egy „szerződés”, melynek egy adott programrésznek eleget kell tennie.
- Minden (fontosabb) metódushoz el kell készíteni egy ilyen tesztesetet.
- Automatizálás: unit-testing keretrendszer.
- Refactoring támogatás
- Test-driven development, extreme programming
- Korlátok:
 - Hiányosság tesztelési technika: a hibák jelenlétét mutathatjuk ki, nem bizonyíthatjuk hiányukat.
 - Kevésbé alkalmas integrációs tesztelésre, vagy teljesítménnyel kapcsolatos problémák kimutatására.
 - Teszt kód hosszúsága (boolean decision – true/false – 2 sor).
 - Jól szabályozott fejlesztési folyamat és verziókövető jelenlétének igénye.

JUnit

- Kent Beck: SUnit (Smalltalk)
- Kent Beck, Erich Gamma: JUnit
- Unit-testing keretrendszer
- Hello World JUnit 3.x-ben:

```
public class HelloWorld extends TestCase {  
    public void testMultiplication() {  
        // Testing if 3*2=6:  
        assertEquals("Multiplication", 6, 3*2);  
    }  
}
```

- Hello World JUnit 4.x-ben:

```
public class HelloWorld {  
    @Test  
    public void testMultiplication() {  
        // Testing if 3*2=6:  
        TestCase.assertEquals("Multiplication", 6, 3*2);  
    }  
}
```

- ```
@Test
public void simpleAdd() {
 Money m12CHF = new Money(12, "CHF");
 Money m14CHF = new Money(14, "CHF");
 Money expected = new Money(26, "CHF");
 Money result = m12CHF.add(m14CHF);
 assertTrue(expected.equals(result));
}
```

- Kivételek kezelése, várt kivételek:

```
@Test(expected = IndexOutOfBoundsException.class)
public void empty() {
 new ArrayList<Object>().get(0);
}
```

# Fixture

- Ha több tesztet akarunk lefuttatni ugyanazon az objektum-halmazon (fixture):
  - Minden objektumhoz hozzárendelünk egy field-et.
  - Bevezetünk egy `@org.junit.Before`-al jegyzett metódust, ami inicializálja az objektumokat.
  - Bevezetünk egy `@org.junit.After`-el jegyzett metódust, ami felszabadítja a lefoglalt erőforrásokat.

```
• public class MoneyTest {
 private Money f12CHF;
 private Money f14CHF;
 private Money f28USD;
 @Before
 public void setUp() {
 f12CHF= new Money(12, "CHF");
 f14CHF= new Money(14, "CHF");
 f28USD= new Money(28, "USD");
 }
}
```

# Test suite

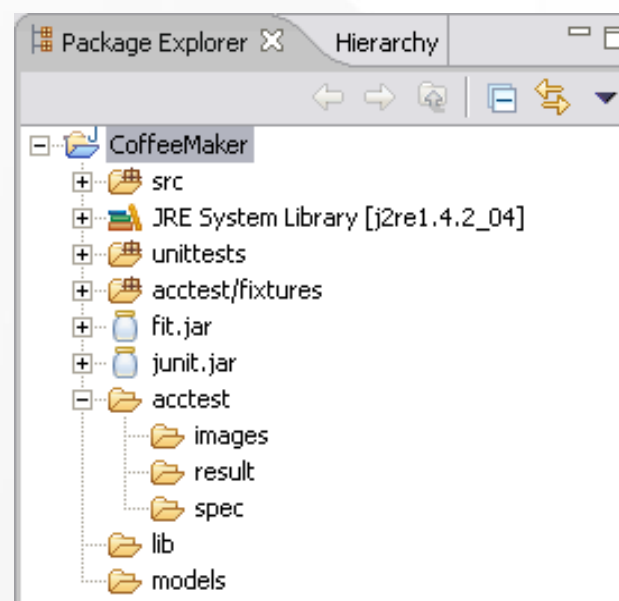
- Ha egy olyan programot szeretnénk készíteni/futtatni, ami az összes tesztünket lefuttatja, a megoldás test suite készítése:

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class SampleTestSuite {
 public static Test suite() {
 TestSuite suite = new TestSuite("Sample Tests");
 // Add one entry for each test class or test suite.
 suite.addTestSuite(SampleTest.class);
 // For a master test suite, use this pattern.
 // (Note that here, it's recursive!)
 suite.addTest(AnotherTestSuite.suite());
 return suite;
 }
}
```

# Eclipse és JUnit

- Eclipse: JUnit támogatás
- Név-konvenciók:
  - Test Case Class: [classname]Test.java
  - Test Case Method: test[methodname]
  - Test Suite Eclipse-ben: AllTests.java
- Javaslat: a teszt kód elválasztása a forráskódtól, valamint a JUnit és FIT tesztek elválasztása.
- FIT: Ward Cunningham által bevezetett JUnit-on alapuló, azt kiterjesztő eszköz automatizált elfogadási tesztek készítésére. A tesztesetek html táblákban vannak megjelenítve, így a felhasználók által könnyebben átláthatóak/módosíthatóak (Eclipse: FitRunner plug-in)



# Eclipse: JUnit példa

- 1. lépés: projekt létrehozása, a junit.jar hozzáadása a projekthez, a junit teszteseteket tartalmazó könyvtár létrehozása (a java build path source részénél is).
- 2. lépés: a tesztelendő osztály létrehozása (esetünkben MyClass.java):

```
public class MyClass {
 public int multiply(int x, int y) {
 return x + y;
 }
}
```

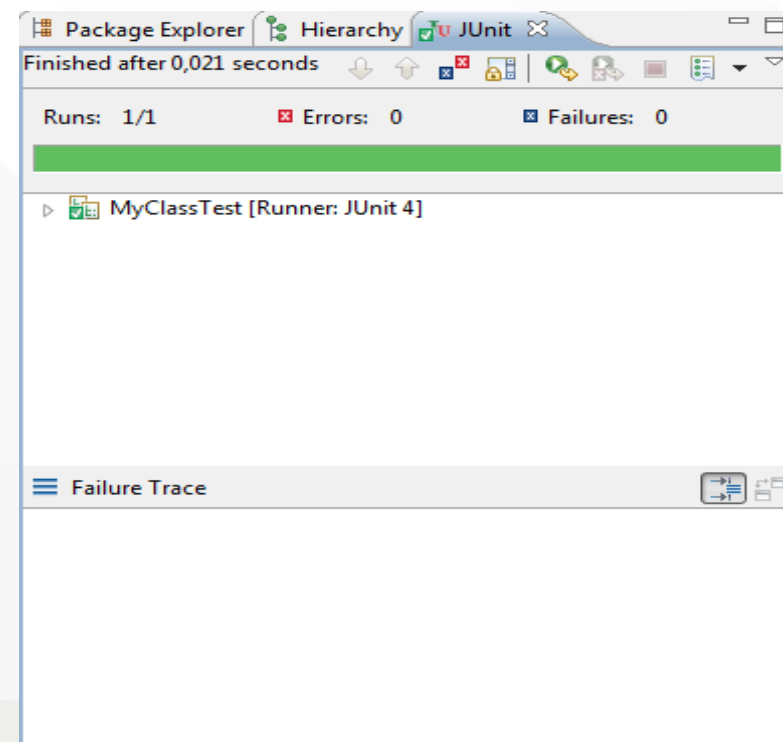
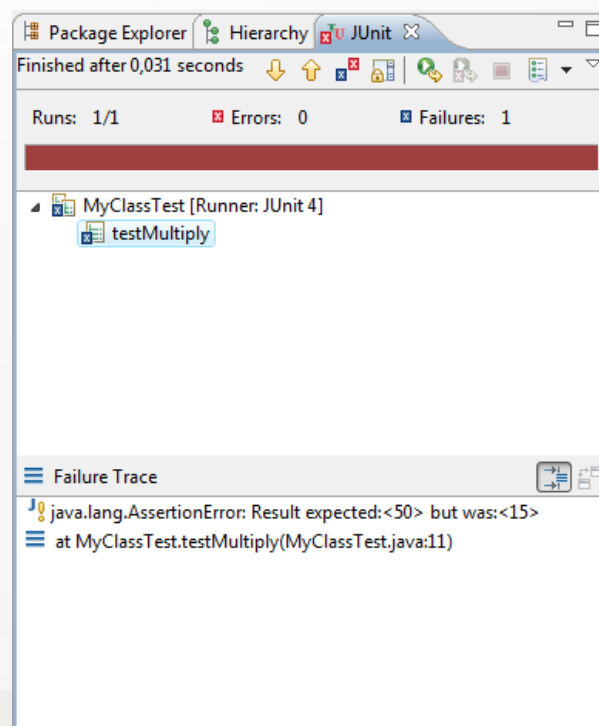
- 3. lépés: a megfelelő teszteseteket tartalmazó osztály generálása (jobb-click a MyClass.java-ra, new JUnit Test Case, a könyvtár nevének beállítása).

# Eclipse: JUnit példa

- 4. lépés: a tesztek implementációja

```
import static org.junit.Assert.*;
import org.junit.Test;
public class MyClassTest {
 @Test
 public void testMultiply() {
 MyClass tester = new MyClass();
 assertEquals("Result", 50, tester.multiply(10, 5));
 }
}
```

- 5. lépés: a teszt futtatása (jobb-click a MyClassTest.java-ra, run as JUnit Test).



# Eclipse: fixture példa

- `import junit.framework.TestCase;`

```
public class SampleTest extends TestCase {
 private java.util.List emptyList;

 /**
 * Sets up the test fixture. (Called before every test case method.)
 */
 protected void setUp() {
 emptyList = new java.util.ArrayList();
 }

 /**
 * Tears down the test fixture. (Called after every test case method.)
 */
 protected void tearDown() {
 emptyList = null;
 }

 public void testSomeBehavior() {
 assertEquals("Empty list should have 0 elements", 0, emptyList.size());
 }

 public void testForException() {
 try {
 Object o = emptyList.get(0);
 fail("Should raise an IndexOutOfBoundsException");
 } catch (IndexOutOfBoundsException success) { }
 }
}
```



# Unit testing

- `@Test public void method()`  
Jelzi, hogy teszt metódusról van szó.
- `@Before public void method()`  
A metódus minden teszt előtt végre lesz hajtva. A tesztelési környezet előkészítésére alkalmazzuk, pl. bemeneti adatok beolvasása, példányosítások.
- `@After public void method()`  
Minden teszt után végre lesz hajtva, lefoglalt erőforrások felszabadítására alkalmazzuk.
- `@BeforeClass public void method()`  
A tesztek futtatása előtt lesz végrehajtva, előkészítő műveletek elvégzésére alkalmazzuk (pl. adatbázis kapcsolat megnyitása).
- `@AfterClass public void method()`  
Az összes teszt lefuttatása után lesz végrehajtva, erőforrások felszabadítására alkalmazzuk (pl. adatbázis kapcsolat bezárása).
- `@IgnoreWill`  
A teszt metódus figyelmen kívül hagyása, pl. ha a hozzátartozó kód változott és a teszt kódja még nem került megfelelő frissítésre.
- `@Test(expected=IllegalArgumentException.class)`  
Teszteli, hogy a metódus egy bizonyos típusú kivételt dob-e.
- `@Test(timeout=100)`  
Sikertelen ha a metódus futás tovább tart mint 100 millisec.

# Unit testing

- `fail(String)`  
„Sikertelenné teszi a metódust” → segítségével, hogy elérünk-e egy bizonyos kódrészt.
- `assertTrue(true)`  
Igaz (true) értéket ad.
- `assertEquals([String message], expected, actual)`  
Ellenőrzi, hogy az értékek ugyanazok-e (tömbök esetében a referenciát ellenőrzi, nem az értékeket).
- `assertEquals([String message], expected, actual, tolerance)`  
Float és double értékek esetében megszabhatjuk a tolerancia küszöböt (hány tizedesnek kell megegyeznie).
- `assertNull([message], object)`  
Ellenőrzi, hogy null objektumról van-e szó.
- `assertNotNull([message], object)`
- `assertSame([String], expected, actual)`  
Ellenőrzi, hogy a két változó ugyanarra az objektumra mutat-e.
- `assertNotSame([String], expected, actual)`
- `assertTrue([message], boolean condition)`  
Ellenőrzi, hogy a feltétel igaz-e
- ```
try {a.shouldThrowException(); fail("Failed")}  
catch (RuntimeException e) {assertTrue(true);}
```


Alternatíva kivételek ellenőrzésére

- Deklaratíván definiált szabályok, megfelelést ellenőrző matcher objektumok.

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.*;

import junit.framework.TestCase;

public class BiscuitTest extends TestCase {
    public void testEquals() {
        Biscuit theBiscuit = new Biscuit("Ginger");
        Biscuit myBiscuit = new Biscuit("Ginger");
        assertThat(theBiscuit, equalTo(myBiscuit));
    }
}
```

Hamcrest Matchers

Hamcrest 1.3 Quick Reference

[Core](#)[Library](#)

General purpose

```
is(T)
equalTo(T)
not(T) : Matcher<T>

anything()
anything(String) : Matcher<Object>

any(Class<T>)
instanceOf(Class<?>)
isA(Class<T>) : Matcher<T>

nullValue() : Matcher<Object>
nullValue(Class<T>) : Matcher<T>
notNullValue() : Matcher<Object>
notNullValue(Class<T>) : Matcher<T>

sameInstance(T)
theInstance(T) : Matcher<T>

isIn(Collection<T>)
isIn(T[])
isOneOf(T...)
hasToString(String)
hasToString(Matcher<? super String>) : Matcher<T>
```

Combining multiple matchers

```
is(Matcher<T>)
not(Matcher<T>) : Matcher<T>

allOf(Matcher<? super T>...)
allOf(Iterable<Matcher<? super T>>)
anyOf(Matcher<? super T>...)
anyOf(Iterable<Matcher<? super T>>) : Matcher<T>

both(Matcher<? super LHS>)
either(Matcher<? super LHS>) : Matcher<LHS>

describedAs(String, Matcher<T>, Object...) : Matcher<T>
```

Strings

```
containsString(String)
startsWith(String)
endsWith(String) : Matcher<String>

equalToIgnoringCase(String)
equalToIgnoringWhiteSpace(String) : Matcher<String>

isEmptyString()
isEmptyOrNullString() : Matcher<String>

stringContainsInOrder(Iterable<String>) : Matcher<String>
```

Iterables

```
everyItem(Matcher<U>) : Matcher<Iterable<U>>

hasItem(T)
hasItem(Matcher<? super T>) : Matcher<Iterable<? super T>>

hasItems(T...)
hasItems(Matcher<? super T>...) : Matcher<Iterable<T>>

emptyIterable() : Matcher<Iterable<? extends E>>
emptyIterableOf(Class<E>) : Matcher<Iterable<E>>

contains(E...)
contains(Matcher<? super E>...)
contains(Matcher<? super E>)
contains(List<Matcher<? super E>>) : Matcher<Iterable<? extends E>>

containsInAnyOrder(T...)
containsInAnyOrder(Collection<Matcher<? super T>>)
containsInAnyOrder(Matcher<? super T>...)
containsInAnyOrder(Matcher<? super E>) : Matcher<Iterable<? extends E>>

iterableWithSize(Matcher<? super Integer>)
iterableWithSize(int) : Matcher<Iterable<E>>
```

Collections

```
hasSize(int)
hasSize(Matcher<? super Integer>) : Matcher<Collection<? extends E>>

empty() : Matcher<Collection<? extends E>>
emptyCollectionOf(Class<E>) : Matcher<Collection<E>>
```

Arrays

```
array(Matcher<? super T>...) : Matcher<T[]>

hasItemInArray(T)
hasItemInArray(Matcher<? super T>) : Matcher<T[]>

arrayContaining(E...)
arrayContaining(List<Matcher<? super E>>)
arrayContaining(Matcher<? super E>...) : Matcher<E[]>

arrayContainingInAnyOrder(E...)
arrayContainingInAnyOrder(Matcher<? super E>...)
arrayContainingInAnyOrder(Collection<Matcher<? super E>>) : Matcher<E[]>

arrayWithSize(int)
arrayWithSize(Matcher<? super Integer>)
emptyArray() : Matcher<E[]>
```

Maps

```
hasEntry(K, V)
hasEntry(Matcher<? super K>, Matcher<? super V>) : Matcher<Map<? extends K, ? extends V>>

hasKey(K)
hasKey(Matcher<? super K>) : Matcher<Map<? extends K, ?>>

hasValue(V)
hasValue(Matcher<? super V>) : Matcher<Map<?, ? extends V>>
```

Beans

```
hasProperty(String)
hasProperty(String, Matcher<?>)
samePropertyValuesAs(T) : Matcher<T>
```

Comparables

```
comparesEqualTo(T extends Comparable<T>)
greaterThan(T extends Comparable<T>)
greaterThanOrEqualTo(T extends Comparable<T>)
lessThan(T extends Comparable<T>)
lessThanOrEqualTo(T extends Comparable<T>) : Matcher<T>
```

Numbers

```
closeTo(double, double) : Matcher<Double>
closeTo(BigDecimal, BigDecimal) : Matcher<BigDecimal>
```

Classes

```
typeCompatibleWith(Class<T>) : Matcher<java.lang.Class<?>>
```

EventObjects

```
eventFrom(Object)
eventFrom(Class<? extends EventObject>, Object) : Matcher<EventObject>
```

DOM

```
hasXPath(String)
hasXPath(String, NamespaceContext)
hasXPath(String, Matcher<String>)
hasXPath(String, NamespaceContext, Matcher<String>) : Matcher<org.w3c.dom.Node>
```

Created by Marc Philipp, <http://www.marcphilipp.de>

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License, <http://creativecommons.org/licenses/by-sa/3.0/>



Hamcrest – custom matcher

- ```
package org.hamcrest.examples.tutorial;

import org.hamcrest.Description;
import org.hamcrest.Factory;
import org.hamcrest.Matcher;
import org.hamcrest.TypeSafeMatcher;

public class IsNotANumber extends TypeSafeMatcher<Double> {

 @Override
 public boolean matchesSafely(Double number) {
 return number.isNaN();
 }

 public void describeTo(Description description) {
 description.appendText("not a number");
 }

 @Factory
 public static <T> Matcher<Double> notANumber() {
 return new IsNotANumber();
 }
}
```
- ```
public void testSquareRootOfMinusOneIsNotANumber() {
    assertThat(Math.sqrt(-1), is(notANumber()));
}
```


Mockup Frameworks

- Teszt objektumok (test doubles): dummy ("üres" objektum, pl. fordító elvárásainak megfelelő paraméter), fake (leegyszerűsített implementáció), stub (hardcode-olt viselkedés), mock (helyettesítő, adott implementációval és a viselkedés futási idejű ellenőrzésével), spy (mock, valós objektumhoz rendelt proxy: bizonyos metódusok szimuláltak, mások valósak, lehetőség van a futási idejű viselkedésellenőrzésre)
- Mock Object: minták/sablonok/vázak, amelyek valós objektumok viselkedését szimulálják
- Motiváció:
 - Eszközök (pl. szenzorok, időzítők stb.) szimulációja
 - Nehezen reprodukálható állapotok szimulációja (pl. meghibásodások)
 - Erőforrás-igényes műveletek szimulációja (pl. adatbázis műveletek esetében)
 - Még nem létező komponensek viselkedésének szimulációja
- Megoldás: a szimulált objektumával azonos interfész biztosítása, mockup/**mocking**/mock keretrendszerek alkalmazása
- Java mockup keretrendszerek: **Mockito**, Jmock, EasyMock stb.

Mockito

- Mocks & spies
- mockito-all-x.x.x.jar
- Mock objektum létrehozása:
 - `MyClass myMock = Mockito.mock(MyClass.class);`
 - `@Mock`
`private MyClass myMock;`
 - Megjegyzések:
 - Mockito annotációk használata esetén: `MockitoAnnotations.initMock(testClass)` metódushívás szükséges, vagy a `MockitoJUnit4Runner` JUnit runner alkalmazása
 - (nem final) osztályok esetében is alkalmazható a Mock, nem csak interfészek esetében
- Stub metódusok megírása:
 - ...
`import static org.mockito.Mockito.mock;`
`import static org.mockito.Mockito.when;`
...
`@Test`
`public void shouldReturnGivenValue() {`
 `MyClass myMock = mock(MyClass.class);`
 `when(myMock.getTheNumber()).thenReturn(TEST_NUMBER);`
 `int number = myMock.getTheNumber();`
 `assertEquals(number, TEST_NUMBER);`
}

Mockito

- Mockito metódusok:
 - `thenReturn(T valueToBeReturned)`
 - `thenThrow(Throwable toBeThrown), thenThrow(Class<? Extends Throwable> toBeThrown)`
 - `then(Answer answer), thenAnswer(Answer answer)`
 - `thenCallRealMethod()`
 - Megjegyzés: a nem void metódusok a megfelelő típusú "üres" értéket térítenek vissza (null, 0, false, empty collection)
- A tesztek (a given-when-then mintához hasonlóan) három fázisra bonthatóak:
 - Arrange(given) – mock inicializálás
 - Act (when) – művelet végrehajtása
 - Assert (then) – ellenőrzés
 - Megjegyzés: BDDMockito kiterjesztéssel a given/willReturn szintakszis is alkalmazható
- Argumentumok ellenőrzése:
 - `given(someMock.someMethod(anyInt(), contains("some_string"), eq("some_string"))).willReturn(true);`
 - Mockito matchers: `any()`, `any(Class<T> c)`, `anyBoolean()`, `anyByte()`, ... , `anyCollection()`, `anyList()`, ..., `anyCollectionOf(Class<t> c)`, ..., `anyVararg()`, `eq(T value)`, `isNull()`, `isNull(Class<t> c)`, `isNotNull()`, `isNotNull(Class<t> c)`, `isA(Class<t> c)`, `matches(String regexp)`, `startsWith(String s) + endsWith`, `contains`, `aryEq(T[] value)`, `cmpEq(Comparable<T> value)`, `gt(value) + geq`, `lt`, `leq`, `argThat(org.hamcrest.Matcher<t> matcher)`, `booleanThat(Matcher<Boolean> + byteThat`, ..., `and(first, second)`, `or(first, second)`, `not(first)` stb.
- Stubbing void methods:
 - `doThrow(Class<? Extends Throwable> toBeThrown), doAnswer(Answer answer), doCallRealMethod(), doNothing(), doReturn(Object toBeReturned)`
- Specifikus válaszok: Answer interfész implementációja, az answer metódus megírása

Sample

```
import static org.mockito.BDDMockito.*;
import static org.mockito.Mockito.*;

@Test
public void shouldReturnGivenValue() {
    //given
    MyClass myMock = mock(MyClass.class)
    given(myMock.getSomeNumber()).willReturn(TEST_NUMBER);

    //when
    int number = myMock.getSomeNumber();

    //then
    assertEquals(number, TEST_NUMBER);
}
```

- Viselkedés futási idejű ellenőrzése:

- `MyClass myMock = mock(MyClass.class);`
`myMock.doSomething();`
`verify(myMock).doSomething();`
- Alapértelmezetten azt ellenőrzi, hogy egy adott metódus (adott argumentummal) egyszer és csakis egyszer volt-e meghívva. Ez `VerificationMode` megadásával változtatható:
 - `times(int wantedNumberOfInvocations)`, `never()`, `atLeastOnce()`, `atLeast(int minNumberOfInvocations)`, `atMost(int maxNumberOfInvocations)`, `only()`, `timeout(millis)`
 - `verify(myMock, never()).doSomething();`
- `InOrder` API a hívások sorrendjének ellenőrzésére
- Paraméterek átadásának ellenőrzése: `ArgumentCaptor` osztály

- Spying: spy objektum valós objektumhoz rendelése, egyes metódusok átírásának lehetősége, futási idejű ellenőrzés

- `MyClass realObject = new MyClass();`
`realObject.setSomeProperty(ORIGINAL_PROPERTY);`
`MyClass mySpy = spy(realObject);`
`willDoNothing().given(mySpy).setSomeProperty(anyInt());`
`mySpy.setSomeProperty(NEW_PROPERTY);`
`verify(mySpy).setSomeProperty(NEW_PROPERTY);`
`assertEquals(mySpy.getSomeProperty(), ORIGINAL_PROPERTY);`

- Annotációk: `@Mock`, `@Spy`, `@Captor`, `@InjectMocks`

- Forrás és további dokumentáció: Marcin Zajaczkowski: Mockito - A Simple, Intuitive Mocking Framework