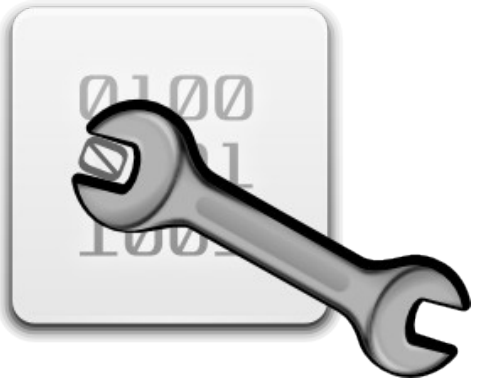
Podstawy inżynierii oprogramowania



Testy jednostkowe

Aleksander Lamża
Instytut Informatyki
Uniwersytet Śląski w Katowicach

aleksander.lamza@us.edu.pl

Zawartość

- Czym są testy jednostkowe?
- Zasady testowania
- Budowa testów
- Zasady

Testy białej skrzynki w praktyce – testy jednostkowe

Testy jednostkowe (ang. unit testing) służą do testowania elementarnych części składowych kodu.

(np. metod, klas, modułów)

Sposób testowania:

kod sprawdza kod

Kod testów jest rozwijany równolegle z kodem produkcyjnym.

Testy jednostkowe – narzędzia

Do przeprowadzania testów jednostkowych potrzebne jest odpowiednie narzędzie, tzw.

framework testów jednostkowych

Java – **JUnit**

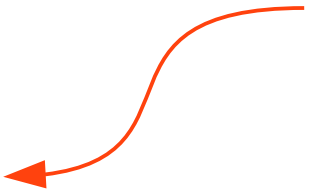
C++ – **CppUnit**

C# – **NUnit**

PHP – **PHPUnit**

Python – **PyUnit**

...



Przykłady frameworków dla różnych języków.

Obszerną listę dostępnych narzędzi można znaleźć np. w Wiki: http://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)

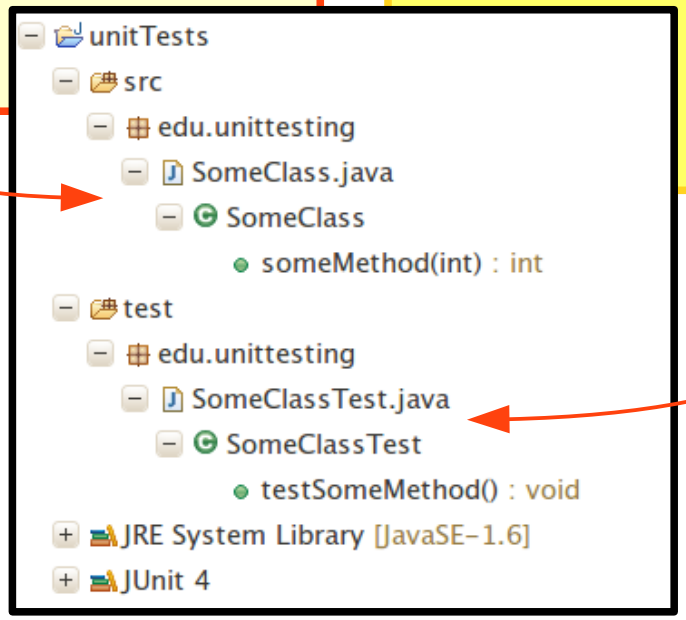
Prosty przykład testu jednostkowego

Kod produkcyjny
(folder źródłowy: *src*)

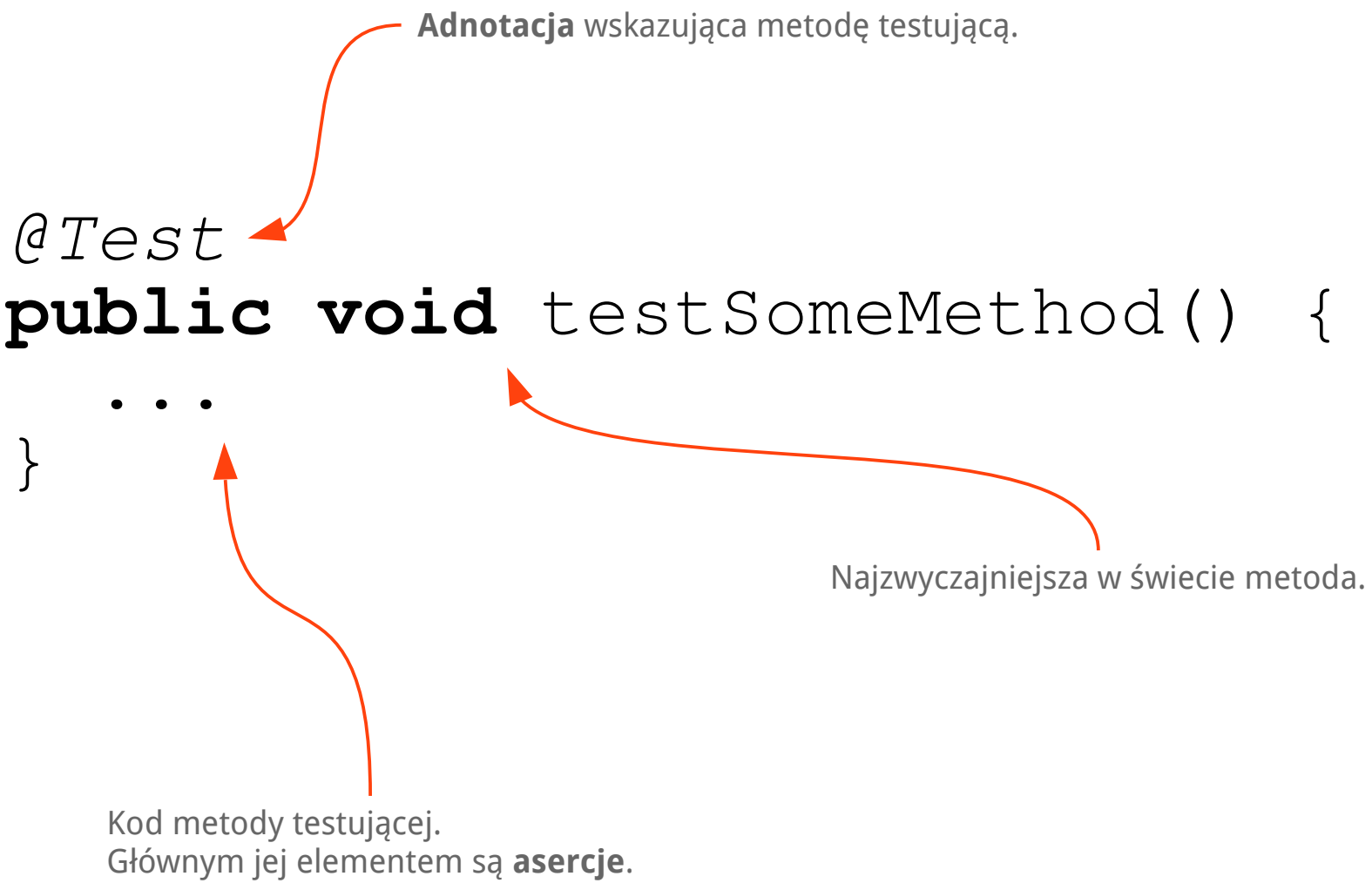
```
class SomeClass {

   ...

   public int someMethod(int a) {
      return ++a;
   }

   ...

}
```

Kod testów
(folder źródłowy: *test*)

```
class SomeClassTest {

   ...

   @Test
   public void testSomeMethod() {
      SomeClass o = new SomeClass();
      assertEquals(13, o.someMethod(12));
   }
```



Składnia testu jednostkowego



Adnotacje

```
@Test
@Test (expected = SomeException.class)
@Test (timeout = 200)
```

Umieszczenie tej adnotacji oznacza, że metoda ma być traktowana jako przypadek testowy.

@Before
@BeforeClass

Te adnotacje przydają się wtedy,
gdy przed uruchomieniem zestawu testów
(lub po jego uruchomieniu)
chcemy wykonać jakiś wspólny kod
(np. musimy utworzyć obiekty, pobrać dane itp.).

@After
@AfterClass

@Ignore

Adnotacja ta przydaje się wtedy, gdy
tymczasowo chcemy zignorować jakiś test.

Asercje

```
@Test
public void testSomeMethod() {
  SomeClass o = new SomeClass();
  assertEquals(13, o.someMethod(12));
}
```

Zakładamy, że wywołanie metody `someMethod()` obiektu `o` z argumentem **12** zwróci **wynik 13**.

```
assertEquals(expected, actual)
assertNull(object)
assertNotNull(object)
assertTrue(cond)
assertFalse(cond)
assertSame(expected, actual)
assertNotSame(expected, actual)
fail()
```

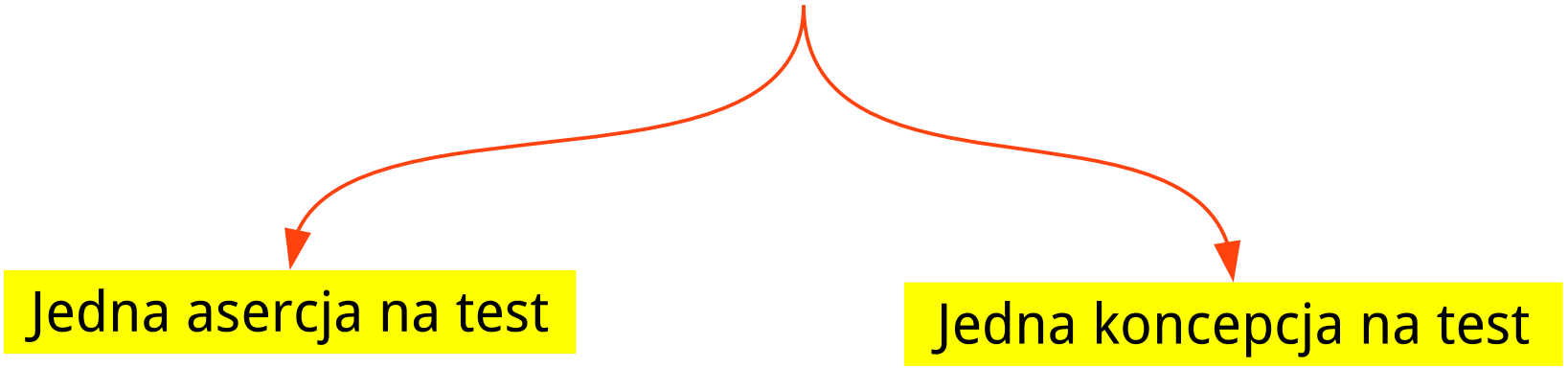
Każda asercja jako pierwszy (opcjonalny) argument może przyjąć komunikat.

Pełna lista asercji w JUnit: http://junit.sourceforge.net/junit3.8.1/javadoc/junit/framework/Assert.html

Zasady tworzenia testów jednostkowych

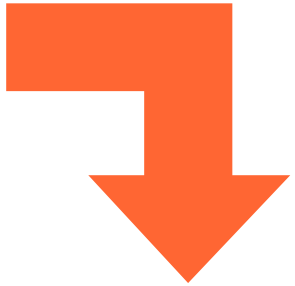
O kod testów jednostkowych **należy dbać**
tak samo jak o kod produkcyjny.

Kod testów musi być **zwięzły** i **czytelny**.



Jedna asercja na test

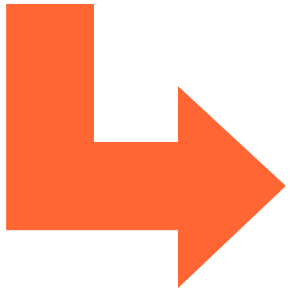
```
@Test
public void testSomeMethod() {
  SomeClass o = new SomeClass();
  assertEquals(32, o.someMethod(2, 5));
  assertEquals(1, o.someMethod(5, 0));
  assertEquals(0.5, o.someMethod(2, -1));
}
```



```
private SomeClass o;

@Before
public void setUp() {
  o = new SomeClass();
}

@Test
public void testSomeMethod() {
  assertEquals(32, o.someMethod(2, 5));
}
@Test
public void testSomeMethodZero() {
  assertEquals(1, o.someMethod(5, 0));
}
@Test
public void testSomeMethodNegative() {
  assertEquals(0.25, o.someMethod(4, -1));
}
```

Jedna koncepcja na test

```
@Test
public void testAddMinutes() {
  Time t1 = new Time(10, 30);

  assertNotNull(t1);

  Time t2 = t1.addMinutes(15);
  assertEquals(10, t2.getHours());
  assertEquals(45, t2.getMinutes());

  Time t3 = t1.addMinutes(50);
  assertEquals(11, t3.getHours());
  assertEquals(20, t3.getMinutes());

  Time t4 = t1.addMinutes(-50);
  assertEquals(9, t4.getHours());
  assertEquals(40, t4.getMinutes());
}
```



```
private Time t;

@Before
public void setUp() {
  t = new Time(10, 30);
}

@Test
public void testTimeExists() {
  assertNotNull(t);
}

@Test
public void testAddMinutesSameHour() {
  Time tt = t.addMinutes(15);
  assertEquals(10, t2.getHours());
  assertEquals(45, t2.getMinutes());
}

@Test
public void testAddMinutesNextHour() {
  Time tt = t.addMinutes(50);
  assertEquals(11, t3.getHours());
  assertEquals(20, t3.getMinutes());
}

@Test
public void testAddMinutesPrevHour() {
  Time tt = t.addMinutes(-50);
  assertEquals(9, t4.getHours());
  assertEquals(40, t4.getMinutes());
}
```

Podsumowanie

Jakie wnioski?

TESTOWAĆ

TESTOWAĆ

I JESZCZE RAZ TESTOWAĆ

!