Politechnika Wrocławska
Katedra Informatyki Technicznej

Inżynieria Oprogramowania

# Testy jednostkowe z użyciem biblioteki JUnit i frameworku JMockit

*Magdalena Biernat*
*Mateusz Bortkiewicz*

Opiekun
prof. dr hab. inż. Jan Magott

26 stycznia 2018

# 1 Wprowadzenie

Sprawozdanie dotyczy dwunastego laboratorium. Na tych laboratoriach zaczęliśmy pisać testy stworzonego uprzednio kodu.

# 2 Laboratorium

## 2.1 Cel

Celem tego laboratorium było nabycie umiejętności przygotowywania testów jednostkowych przy użyciu biblioteki JUnit i frameworku JMockit w środowisku NetBeans.

## 2.2 Przebieg

### 2.2.1 Stworzenie klasy danych

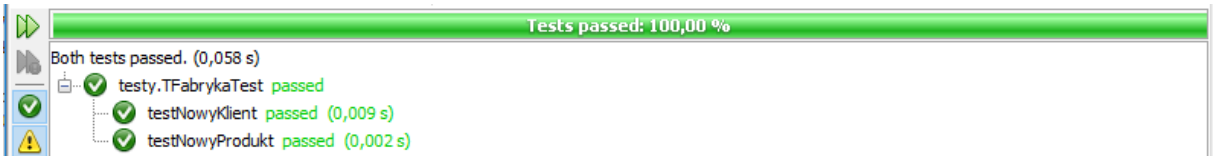
Pierwszy krokiem tego laboratorium (prócz podłączenia samych bibliotek do projektu) było stworzenie klasy, która będzie przechowywać dane, które będą użyte w testach. My stworzyliśmy tę klasę dla instancji z klas: *TProdukt*, *TPozycja*, *TKlient* i *TWypozyczenie*. Przykładowo dane do testowania w klasie *TFabryka* i *TKlient*:

```
public Object dane_klientow[][] = new Object[][]{
  {"","","",new byte[]{6,0,0,7,0,7,1,5,5,1,0}, "675876654",""},
  {"Adam", "Kowalski", "", new byte[]{6,3,0,6,1,4,0,7,0,7,4}, "675876654", "mb@mb.pl"},
  {"Krysia","Momo", "",new byte[]{2,1,1,0,2,9,1,4,1,4,3}, "5656566565", "oaka@ww.pl"},
  {"Marysia","Nowak", "", null, "123456789", ""},
  {"Laptop", "Lenovo", "", null, "","support@lenovo.pl"},
  {"Trololo", "Nudno", "Solilandia", new byte[]{6,5,0,7,0,6,1,3,5,6,1}, "", "walidacj@ju.com"},
  {"Co tam", "Anony", "Trawa", null, "", ""},
  {"Tutaj", "będzie", "błąd", new byte[]{7,1,0,5,0,9,0,0,3,4,1}, "123", "123"}
};
```

Dzięki tym danym testowaliśmy metodę *nowyKlient()* z klasy TFabryka.

### 2.2.2 Testy w JUnit

Testy przebiegły pozytywnie, 7 z 8 przykładów otrzymało wynik pozytywny, ostatni klient otrzymał wynik pozytywny z uwagi na zamierzony efekt - wpisaliśmy zły PESEL, ale że jako to było działanie zamierzone, testy wypadły pomyślnie. Podobne testy tyczyły się metody *nowyProdukt()* z tej samej klasy.




Rysunek 1: Wyniki testu klasy TFabryka

Oczywiście należy pamiętać o tym, że testy mają pomóc w wykryciu błędów. W rezultacie wcześniej przeprowadzonego testu, udało się nam znaleźć błąd w metodzie *obliczSumeKontrolna()* z klasy *TKlient*, liczącej liczbę kontrolną w numerze PESEL. Źle dobrano współczynniki prazy mnożeniu pozycji. Tak było przed:

```
public byte obliczSumeKontrolna(byte[] PESEL)
{
  int suma = 1* (PESEL[0] + PESEL[4] + PESEL[8]) +
      3 * (PESEL[1] + PESEL[5] + PESEL[9]) +
      7 * (PESEL[2] + PESEL[6]) +
      9 * (PESEL[3] + PESEL[7]);

  return (byte) (suma % 10);
}
```

Tak po zastosowaniu poprawki:

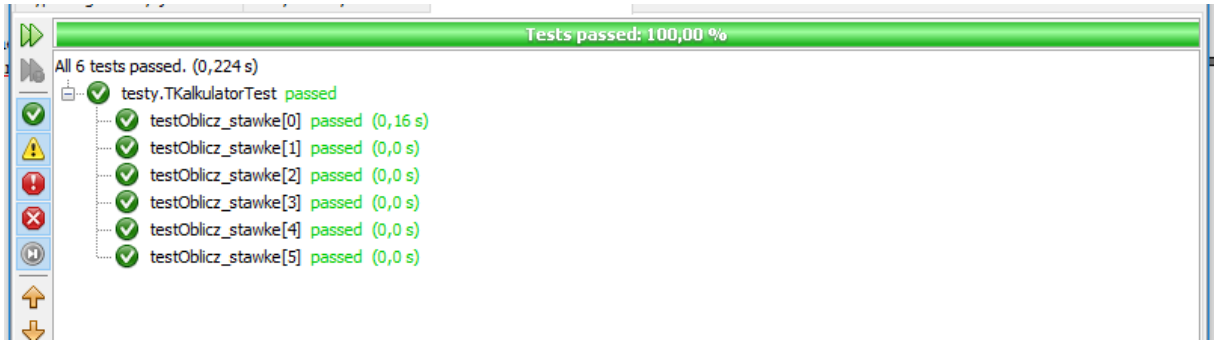
```
public byte obliczSumeKontrolna(byte[] PESEL)
{
  int suma = 9* (PESEL[0] + PESEL[4] + PESEL[8]) +
      7 * (PESEL[1] + PESEL[5] + PESEL[9]) +
      3 * (PESEL[2] + PESEL[6]) +
      1 * (PESEL[3] + PESEL[7]);

  return (byte) (suma % 10);
}
```

Kolejnym testem był test klasy *TKalkulator*. Testowaliśmy metodę *ObliczStawke()*. Użyliśmy tym razem znacznika **@Parameterized**. Dało to nam możliwość przetestowania wszystkich przypadków bez tworzenia pętli.

```
@Parameterized.Parameter
public int pos1;

@Parameterized.Parameters
public static Collection<Object[]> data(){
  Object[][] data1 = new Object[][]{{0},{1},{2},{3},{4},{5}};
  return Arrays.asList(data1);
```

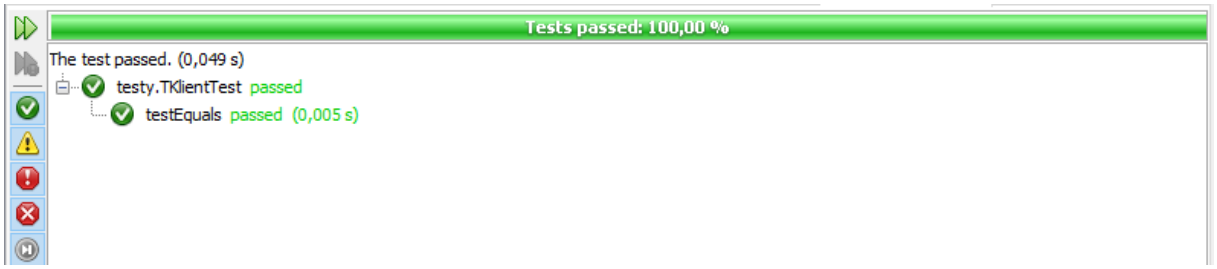
```
}

/**
 * Test of oblicz_stawke method, of class TKalkulator.
 */
@Test
public void testOblicz_stawke() {
  System.out.println("oblicz_stawke");
  TKalkulator instance = new TKalkulator();
  float expResult = (float)round(dane.ceny[pos1],1);
  float result = (float)round(instance.oblicz_stawke(7, (float)dane.dane_produktow[pos1][2]),1);
  assertEquals(expResult, result, 0.0f);
}
```

Rezultat testów był pozytywny.

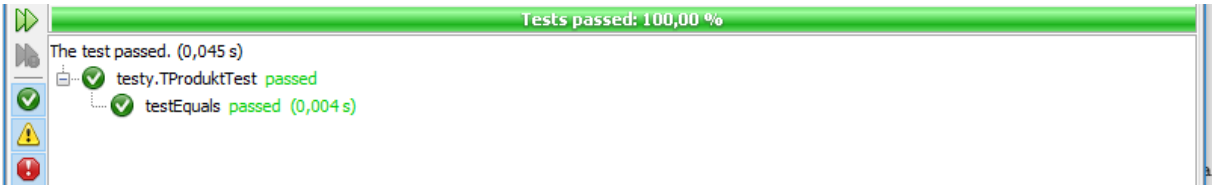



Rysunek 2: Test metody ObliczStawke

Ostatnim krokiem dla testów z JUnit był test metod klasy *TWypozyczalnia*. Początkowo testy wypadały negatywnie. Zdecydowaliśmy się na testy jednostkowe dla klas *TKlient* i *TProdukt* z uwagi na podejrzenie o brak poprawności. Testy jednakże wypadły pozytywnie w obu przypadkach.




Rysunek 3: Test metody Equals w klasie TKlient




Rysunek 4: Test metody Equals w klasie TProdukt

Szukaliśmy rozwiązania gdzie indziej. Powodem były błędy w kilku fragmentach kodu. Poprawiono je w następujący sposób:

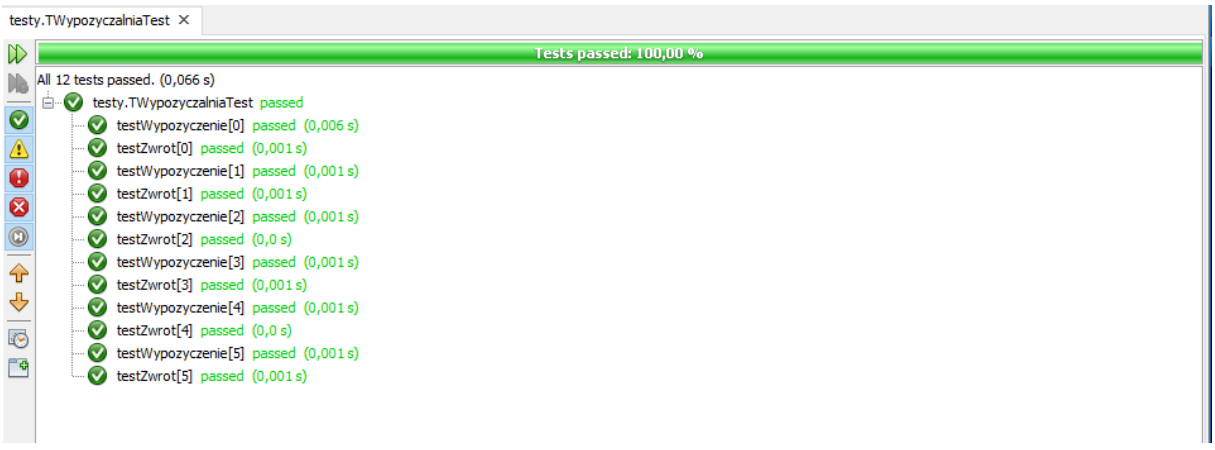
- Typ listy *wypozyczenia* został zmieniony z ArrayList na LinkedList
- W metodzie *sprawdz_dostepnosc()* klasy TMagazyn, znacznik return został zanegowany.

```
public boolean sprawdz_dostepnosc(TProdukt produkt)
{
  produkt = szukaj_TProdukt(produkt);
  if(produkt == null)
    return false;
  return !produkt.getWypozyczony();
}
```

  Powodem błędu było to, że metoda ta zwraca dostępność danego produktu, a parametr *wypozyczony* tej klasy ma przeciwne znaczenie.

- Stworzono listę wypozyczen do testowania w klasie testowej Dane. Dało to możliwość prawidłowego testowania metody *zwrot()*.

Po wszystkich zmianach przeprowadzono testy i wypadły one pozytywnie.



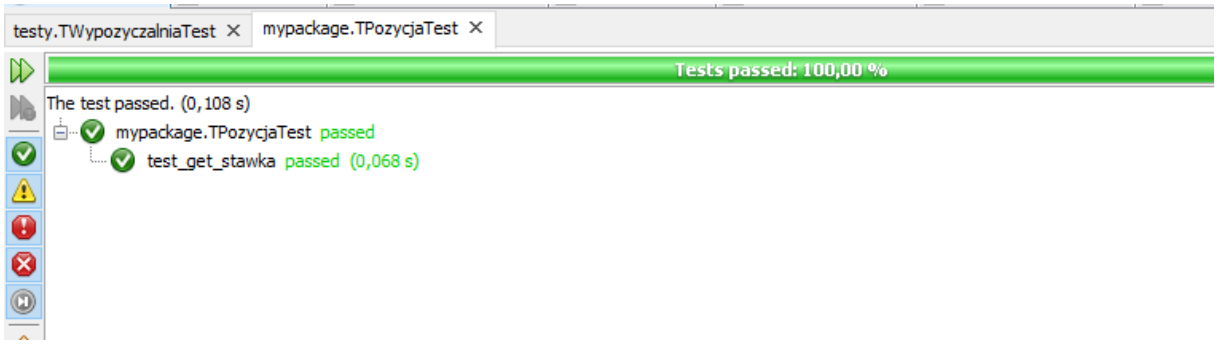

Rysunek 5: Testy metod wypożyczenie i zwrot w klasie TWypozyczalnia

### 2.2.3 Testy przy użyciu JMockit

Testy w JMockit przeprowadzono na metodzie *get_stawka()* klasy TPozycja. Jako **@Capturing** oznaczono instancję klasy TProdukt, a jako *Expectations()* przyjęto zwrot z metody get_stawka() na 3.3.

```
package mypackage;

import mockit.Capturing;
import mockit.Expectations;
import mockit.Injectable;
import mockit.Tested;
import mockit.integration.junit4.JMockit;
import org.junit.Test;
import org.junit.runner.RunWith;
import static org.junit.Assert.*;

/**
 *
 * @author mateu
 */
@RunWith(JMockit.class)
public class TPozycjaTest {
    TFabryka fabryka = new TFabryka();

    @Capturing(maxInstances = 1)
    TProdukt produkt1;

    @Test
    public void test_get_stawka() {
        new Expectations(){
            {
                produkt1.getStawka();
                result = 3.3F;
            }
        };

        TProdukt produkt11 = fabryka.nowyProdukt(new Object[]{new TTytul(), 1, 3.3F, true, TProdukt.
        assertEquals(3.3F, produkt11.getStawka(), 0F);

        TPozycja pozycja1 = fabryka.nowaPozycja(new Object[]{produkt11, produkt11.getStawka(), false
        assertEquals(pozycja1.getStawka(), 3.3F, 0F);
    }
}
```

Testy wypadły pozytywnie.




Rysunek 6: Testy metody get_stawka w klasie TPozycja

# 3 Podsumowanie

Testy z użyciem technologii JUnit i JMockit dały nam ogląd na testowanie w ogóle. Doszliśmy do kilku wniosków, a są to m.in., że praca testera jest niedoceniana. Drugim wnioskiem było to, że jedne zajęcia na testy to za mało. W dwie godziny nie można przyswoić nieczytelnej instrukcji, skonfigurować środowiska i wykonać testów. Mamy nadzieję, że sprawa planu laboratorium się ułoży w przyszłości, a kolejne roczniki będą mogły więcej wynieść z tych zajęć.

Pliki testów zostały umieszczone w katalogu ze sprawozdaniem