Programowanie zespołowe

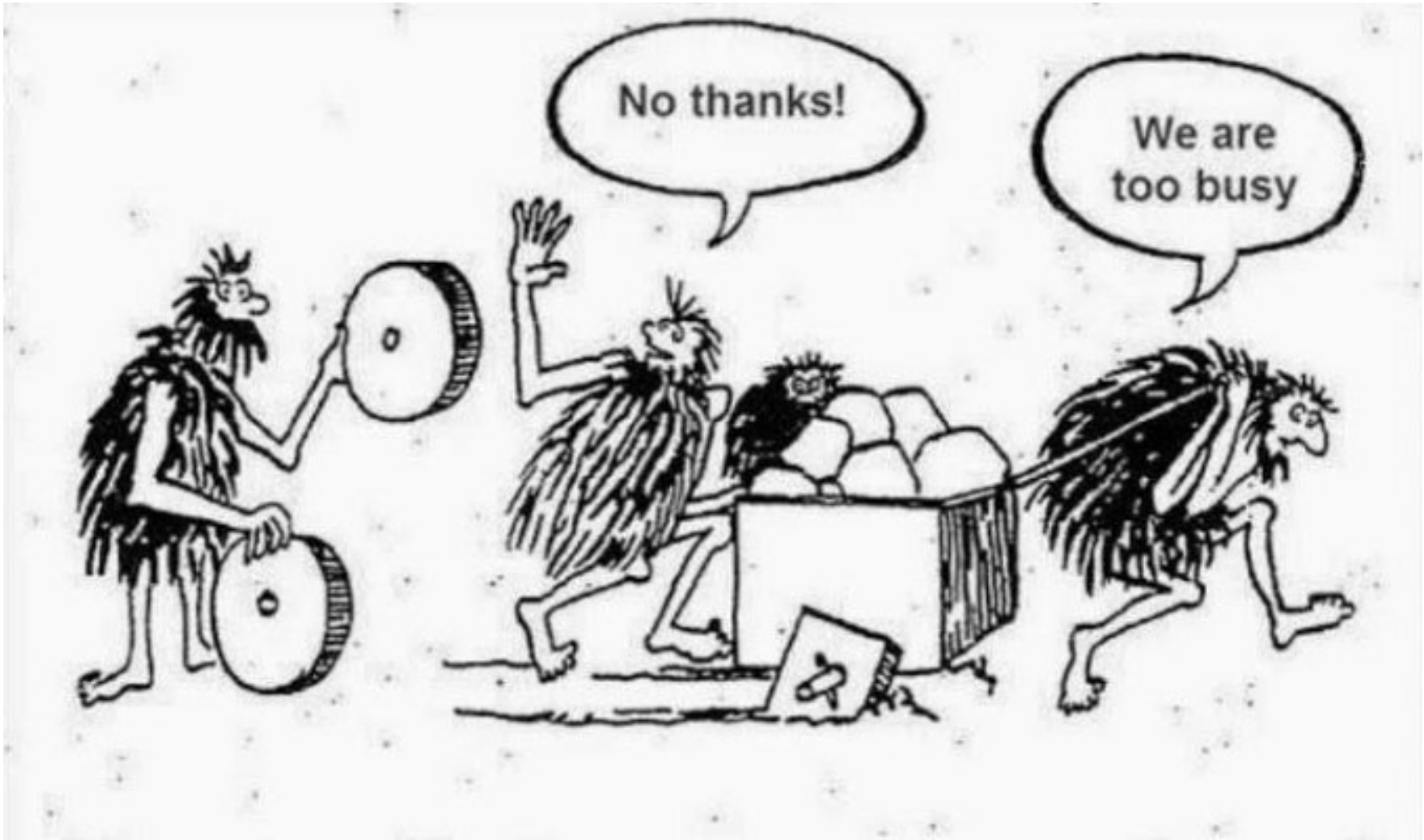
Laboratorium 3 - podstawy testów jednostkowych

mgr inż. Krzysztof Szwarc

krzysztof@szwarc.net.pl

Sosnowiec, 7 marca 2017

Brak czasu na testy?

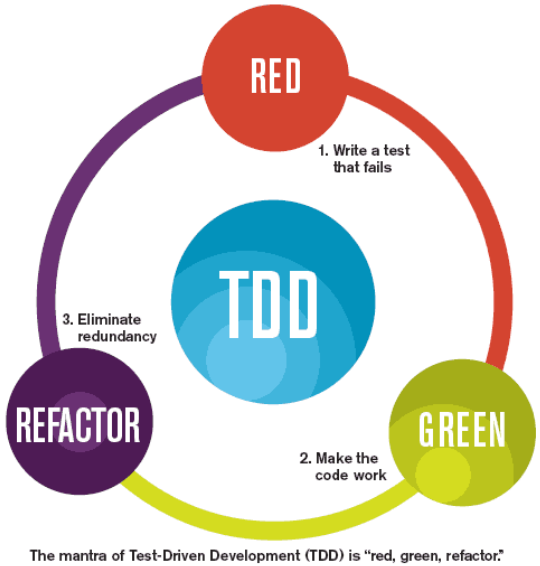


https://media.licdn.com

TDD

TDD (Test-driven development) jest techniką wytwarzania oprogramowania polegającą na powtarzaniu trzech kroków:

1. Napisz automatyczny test sprawdzający dodawaną funkcjonalność (aktualnie nie powinien się on udać).
2. Zaimplementuj funkcjonalność (test powinien się udać).
3. Zrefaktoryzuj kod.



https://s-media-cache-ak0.pinimg.com

Testy jednostkowe

Testy jednostkowe

Test jednostkowy (unit test) – metoda testowania kodu za pomocą testów weryfikujących poprawność działania pojedynczych elementów (w OOP najczęściej metod). Sprawdzają one czy realizowane jest zamierzone działanie kodu. Zwykle są wykonywane wielokrotnie w trakcie cyklu powstawania oprogramowania, aby sprawdzić czy funkcjonalność działa pomimo zmian w kodzie.

Pokrycie kodu

Pokrycie kodu

Pokrycie kodu (code coverage) – procent linijek kodu wykonywanych podczas wszystkich testów (bez klamer, deklaracji itp.). Zakłada się, że dobrą praktyką jest pokrycie $70 - 85\ \%$ kodu.

Jakie elementy pokryć testami jednostkowymi?

- Testujemy przede wszystkim logikę biznesową.
- Do testowania kontrolerów wykorzystujemy zazwyczaj testy integracyjne.
- Nie testujemy getterów i setterów.

Asercja

Asercja

Asercja (assertion) – predykat (forma zdaniowa zwracająca wartość logiczną) wskazujący, że w danym miejscu kodu powinna wystąpić wartość true, np. wyrażenie assertEquals(zmienna, 2) zakłada, że zmienna ma wartość 2 (wtedy test zostanie zakończony sukcesem).

Wzorzec AAA

Arrange–Act–Assert

Wzorzec AAA (Arrange–Act–Assert) określa strukturę testu jednostkowego, zapewniając uporządkowanie kodu. Zakłada on następujący podział testu:

- Arrange - dane wejściowe.
- Act - operacje (działanie na metodzie/funkcji/klasie, która zostaje poddana testowi).
- Assert - asercja (sprawdzenie czy wynik jest zgodny z oczekiwanym).

Wybrane frameworki do testowania

- PHPUnit.
- JUnit.

PHPUnit

Klasy testowe korzystające z PHPUnit muszą dziedziczyć po klasie PHPUnit_Framework_TestCase. Aby sprawdzić czy wartość zwracana przez metodę testowanego obiektu jest równa oczekiwanej stosujemy różne rodzaje asercji.

Przegląd najważniejszych rodzajów asercji w PHPUnit

Rodzaj	Opis	Przykład
assertTrue()	Sprawdzamy czy zmienna ma wartość true	$this->assertTrue(2==2);
assertEquals ()	Sprawdzamy czy oba parametry są sobie równe	$this>assertEquals(2,2);
assertContains ()	Sprawdzamy czy pierwszy parametr jest elementem tablicy (drugi parametr)	$this>assertContains(2,$tablica);
assertInstanceOf ()	Sprawdzamy czy zmienna jest obiektem danej klasy	$this->assertInstanceOf('DateTime', $date);

Przykład - plik CalculatorClass

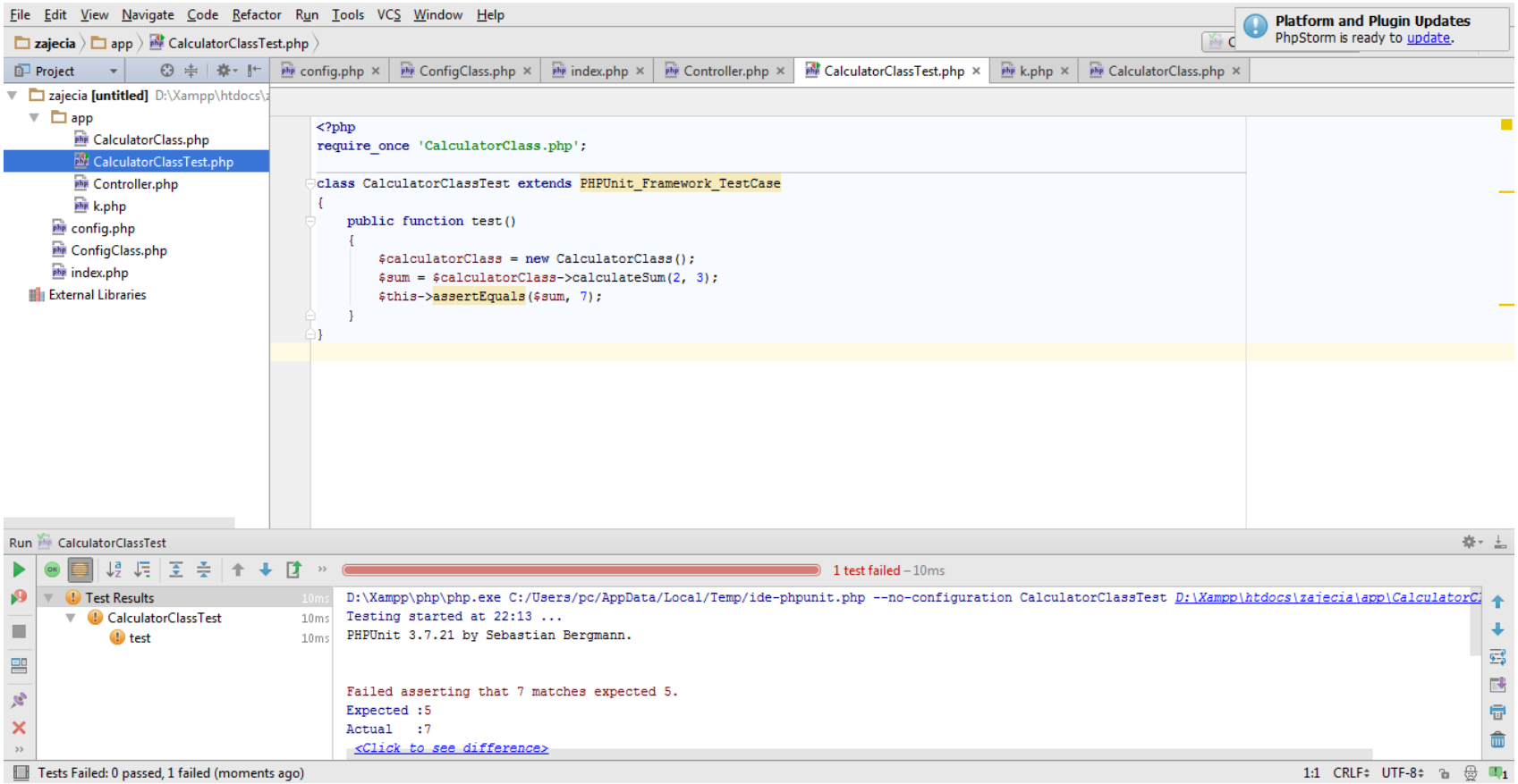
```
class CalculatorClass {
        public function calculateSum($x, $y){
                return $x+$y;
        }
}
```

Przykład - plik CalculatorClassTest

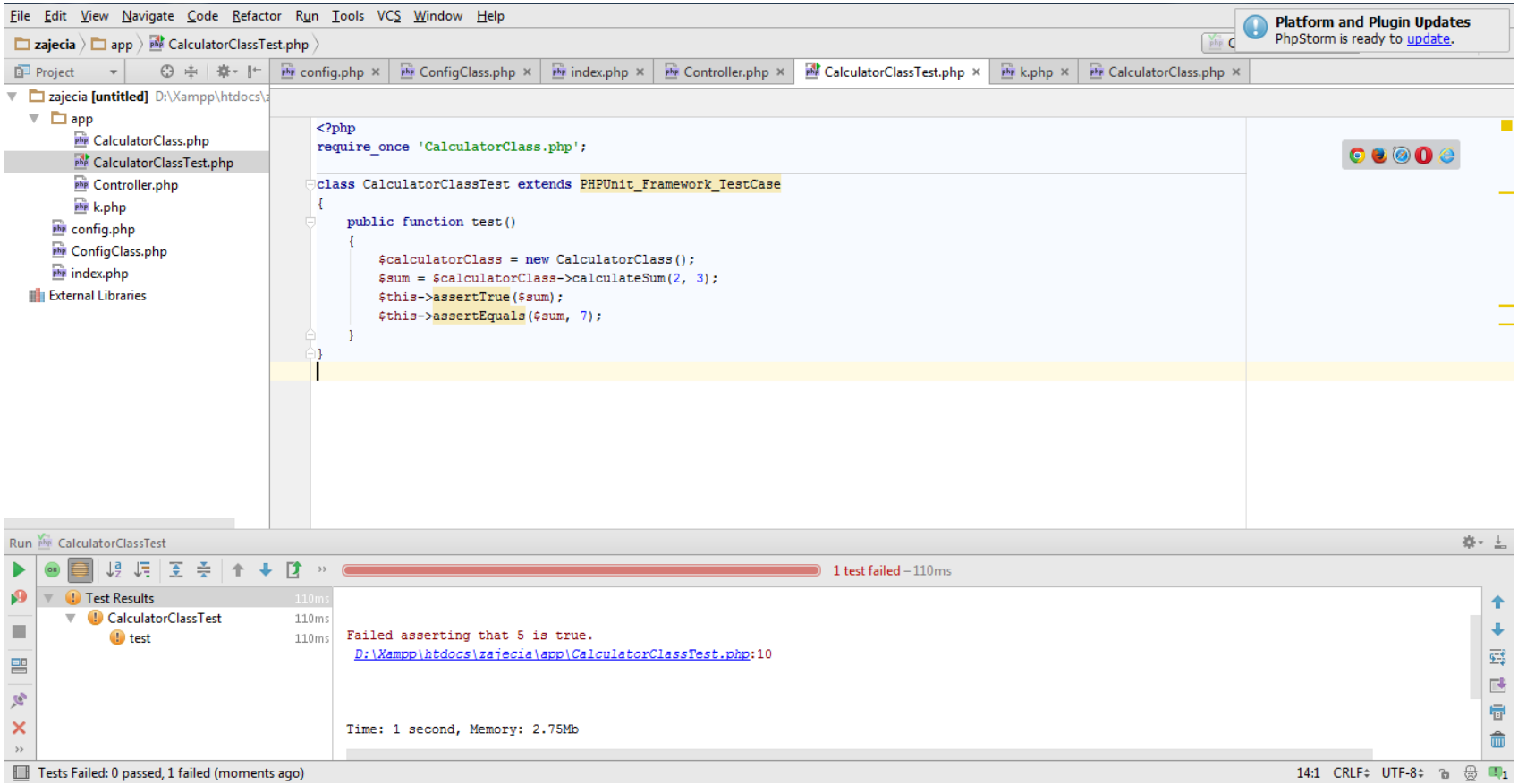
```
require_once 'CalculatorClass.php';

class CalculatorClassTest extends
    PHPUnit_Framework_TestCase {
    public function calculateSumTest(){
        // Arrange
        $calculatorClass = new CalculatorClass();
        // Act
        $sum = $calculatorClass->calculateSum(2,
            3);
        // Assert
        $this->assertEquals($sum, 7);
    }
}
```

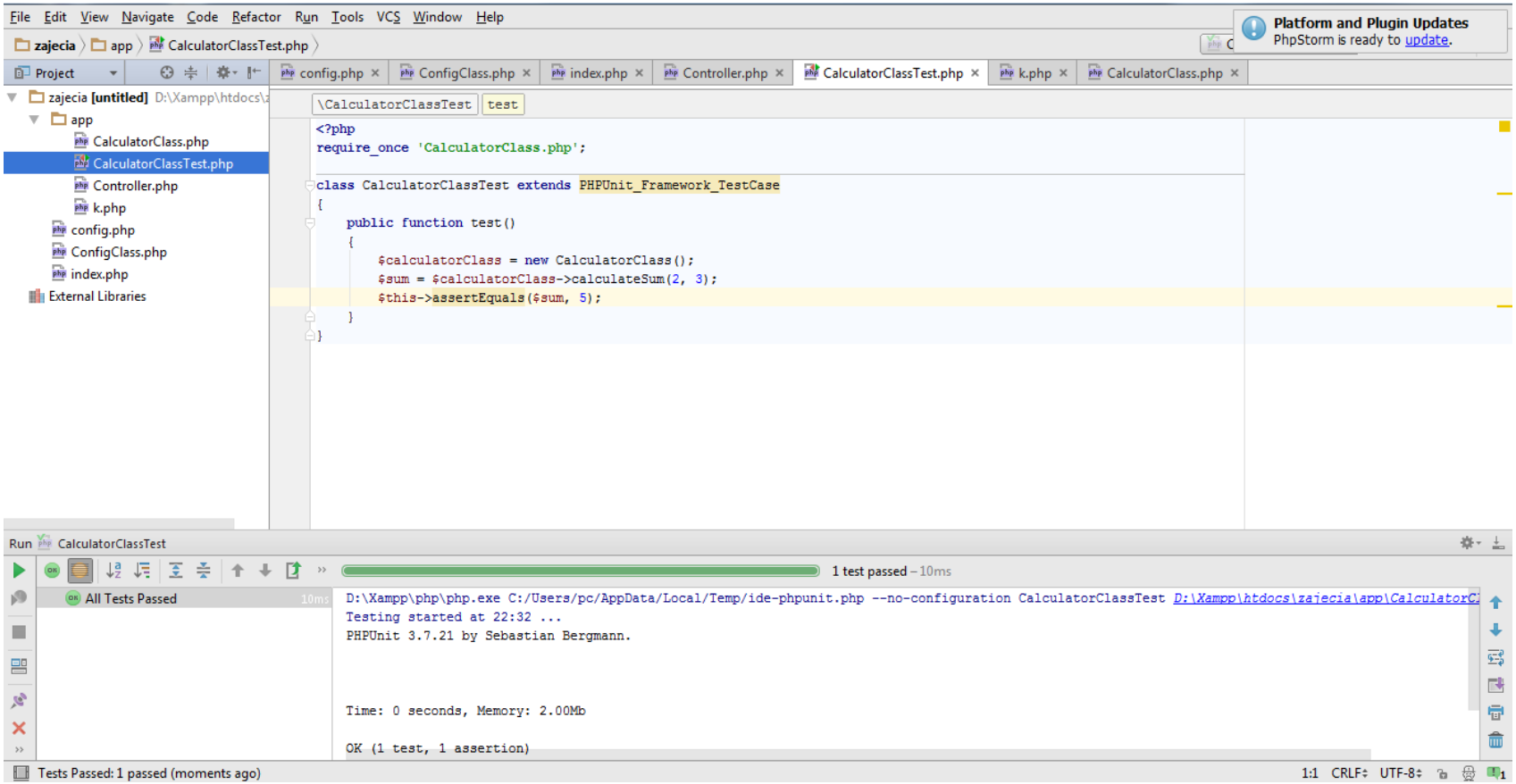
Przykład w PhpStorm



Przykład 2 w PhpStorm



Przykład 3 w PhpStorm



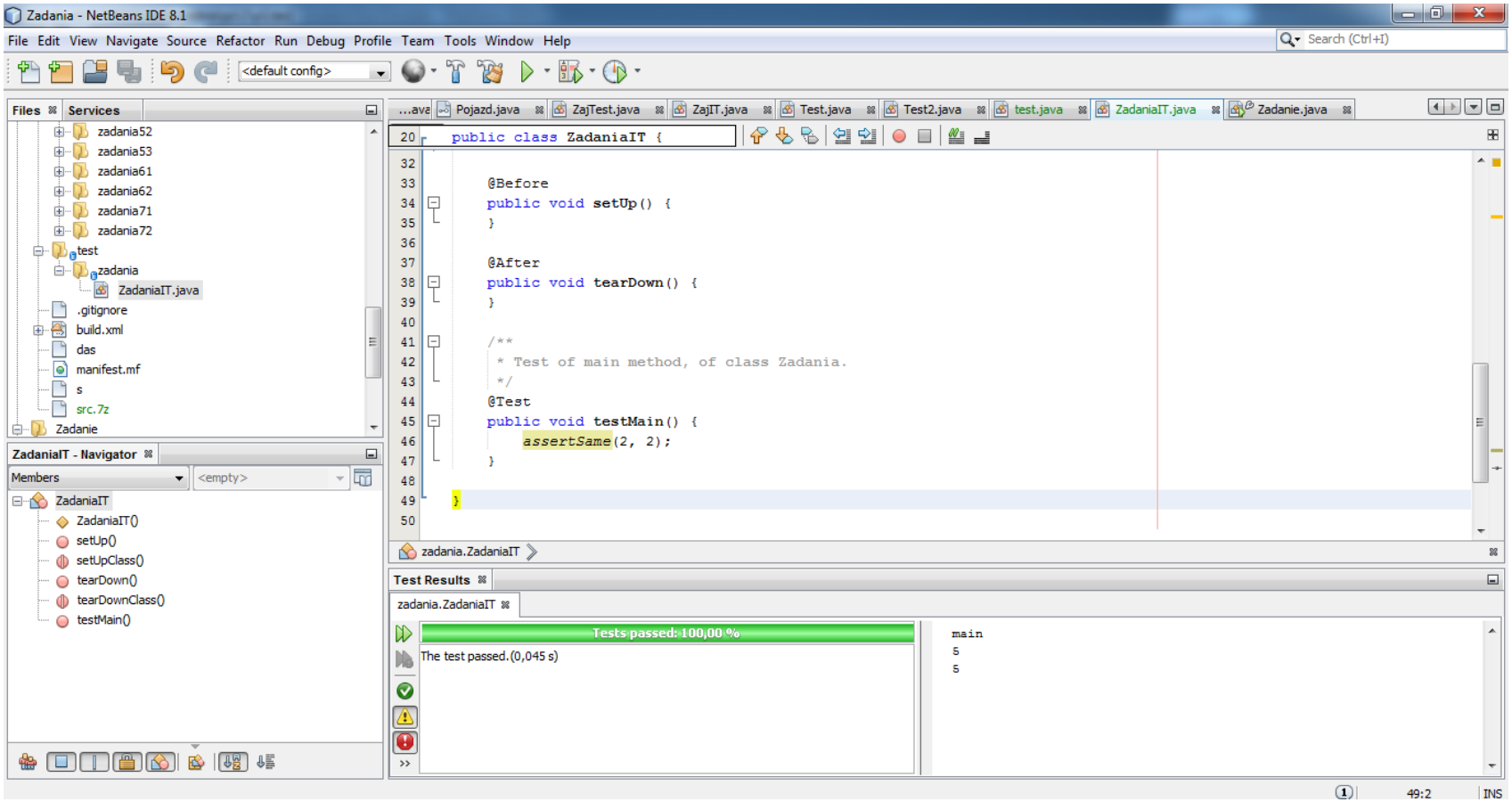
Przegląd możliwych testów w JUnit

Oznaczenie	Opis
@Test	Oznaczenie metody jako test
@Test(expected=Exception.class)	Oznaczenie metody jako test, która powinna zwrócić wyjątek Exception
@Test(timeout=200)	Oznaczenie metody jako test, która powinna się wykonać w < 200 ms
@Before	Metoda wykonywana przed każdym testem
@After	Metoda wykonywana po każdym teście
@BeforeClass	Metoda wykonywana raz przed wszystkimi testami
@AfterClass	Metoda wykonywana raz po wszystkich testach

Przegląd najważniejszych rodzajów asercji w JUnit

Rodzaj	Opis	Przykład
fail()	Zawsze zróci false	fail();
assertTrue()	Sprawdzamy czy parametr ma wartość true	assertTrue(zmienna);
assertEquals()	Sprawdzamy czy oba parametry są sobie równe	assertEquals(2,2);
assertNull()	Sprawdzamy czy obiekt ma wartość null	assertNull(obiekt);
assertSame()	Sprawdzamy czy oba parametry są identyczne	assertSame(2,2);

Przykład w NetBeansie



Zadania

1. Skonfiguruj dowolny Framework testowy dla wybranego IDE.
2. Napisz program kalkulator realizujący operacje dodawania, odejmowania, mnożenia, dzielenia, potęgowania i pierwiastkowania. Nastepnie pokryj wszystkie metody testami jednostkowymi.
3. Napisz i pokryj testami jednostkowymi następujące metody: tworzącą dwuwymiarową tablicę o losowej wielkości (z zakresu [1,10]) zawierającą rózną liczbę wierszy i kolumn, wypełniającą ją losowymi wartościami z przedziału [0,10] (za wyjątkiem elementów znajdujących się na przekątnej tablicy - mają one mieć wartość 1) oraz zwracającą największą wartość znajdującą się w tablicy.

Zadania cz. 2

4. Pobierz program ze strony `http://szwarc.net.pl/program.zip` i napisz do niego testy jednostkowe.

Dziękuję za uwagę