Politechnika Krakowska
przedmiot: Systemu Odporne na Błędy
Laboratorium 3: Testy jednostkowe w języku Java
sprawozdanie

wykonał: Tomasz Fudalej

1. **Test jednostkowy -** fragment kodu sprawdzający działanie pewnego niewielkiego, dokładnie określonego obszaru funkcjonalności testowanego kodu. Głównym zadaniem testów jednostkowych jest udowodnienie, że kod działa zgodnie z przyjętym założeniem programisty.

2. **Zadanie:** zaimplementować kalkulator ONP (w odwróconej notacji polskiej) i przetestować go przy pomocy testów jednostkowych.

2.1. Stos:

a) przy obliczaniu wartości wyrażenia zapisanego w notacji ONP pomocny jest stos, stąd jego użycie w tym kalkulatorze.

b) testy jednostkowe:

- test metody zwracającej wartość *true*, jeśli stos jest pusty, a w przeciwnym wypadku - wartość *false*.

```
@Test
public void isEmpty(){
    assertEquals("Wrong returned value of isEmpty() function", stack.isEmpty(), true);
}
```

- sprawdzenie metod *pop* i *top*. Stos jest w tym momencie pusty, więc obie powinny rzucić wyjątek.

```
@Test(expected=IndexOutOfBoundsException.class)
public void testPop(){
    stack.pop();
}

@Test(expected=IndexOutOfBoundsException.class)
public void testTtop(){
    stack.top();
}
```

- Test sekwencji 1.:
  1) dodanie nowego obiektu na stos,
  2) sprawdzenie spodu stosu,
  3) sprawdzenie czy stos nie jest pusty
  4) ponowne sprawdzeni spodu stosu

```
@Test
public void test1(){
    stack = new Stack(10);
    String symbol = new String("symbol");
    stack.push(symbol);
    assertEquals(symbol, stack.top());
    assertFalse(stack.isEmpty());
    assertEquals(symbol, stack.top());
}
```

- test sekwencji 2.:
  1) dodanie obiektu na stos
  2) pobranie obiektu ze stosu
  3) porównanie przy pomocy asercji, czy pobrany element jest tym samym elementem co wcześniej dodany,
  4) sprawdzenie, czy stos jest pusty
  5) próba pobrania elementu z pustego stosu - powinno wygenerować wyjątek

```
@Test(expected=IndexOutOfBoundsException.class)
public void test2(){
    stack = new Stack(10);
    String symbol = new String("symbol");
    stack.push(symbol);
    String symbol2 = stack.pop();
    assertSame(symbol, symbol2);
    assertTrue(stack.isEmpty());
    stack.pop();
}
```

- test sekwencji 3.:
  1) dodanie trzech nowych elementów na stos
  2) pobranie trzech elementów ze stosu i sprawdzenie przy pomocy asercji, czy zostały pobrane w oczekiwanej kolejności:

```
@Test
public void test3(){
    String symbol1 = new String("symbol1");
    String symbol2 = new String("symbol2");
    String symbol3 = new String("symbol3");
    stack.push(symbol1);
    stack.push(symbol2);
    stack.push(symbol3);
    assertSame(stack.pop(), symbol3);
    assertSame(stack.pop(), symbol2);
    assertSame(stack.pop(), symbol1);
}
```

- test sekwencji 4.:
  1) dodaj obiekt typu *null* na stos
  2) pobierz element ze stosu i sprawdź, czy jest równy *null*

```
@Test
public void test4(){
    stack.push(null);
    assertNull(stack.pop());
}
```

- test sekwencji 5.:
  1) stworzenie pustego stosu
  2) próba pobrania z pustego stosu elementu
  3) blok obsługi wyjątku
  4) dodanie nowego elementu na stos
  5) pobranie elementu ze stosu i sprawdzenie przy pomocy asercji, czy stos działa poprawnie mimo wyrzucenia wyjątku

```java
@Test
public void test5(){
    stack = new Stack(10);
    try{
        stack.pop();
    }catch(Exception e){
        e.printStackTrace();
    }
    stack.push("symbol");
    assertEquals(stack.pop(), "symbol");
}
```

2.2. **Kalkulator:** posiada cztery operacje arytmetyczne, które należy przetestować: dodawanie, odejmowanie, mnożenie i dzielenie. Każde działanie zostanie przetestowane na liczbach całkowitych i rzeczywistych.

- dodawanie:

```java
@Test
public void testAdd() {
    String expression = new String("((2 + 3) + 8)");
    double expected = 13;
    assertEquals(calc.calculate(expression), expected, 0.01d);

    expression = new String("(7.1 + (2.5 + 3.2))");
    expected = 12.8f;
    assertEquals(calc.calculate(expression), expected, 0.01d);
}
```

- odejmowanie:

```java
@Test
public void testSub() {
    String expression = new String("((2 - 3) - 8)");
    double expected = -9;
    assertEquals(calc.calculate(expression), expected, 0.01d);

    expression = new String("((5.2 - 3.9) - 8.1)");
    expected = -6.8;
    assertEquals(calc.calculate(expression), expected, 0.01d);
}
```

- mnożenie:

```java
@Test
public void testMulti() {
    String expression = new String("(2 * (6 * 3))");
    double expected = 36;
    assertEquals(calc.calculate(expression), expected, 0.01d);

    expression = new String("(2.2 * (4.5 * 3.8))");
    expected = 37.62;
    assertEquals(calc.calculate(expression), expected, 0.01d);
}
```

- dzielenie:

```
@Test
public void testDiv() {
    String expression = new String("(12 / (6 / 3))");
    double expected = 6;
    assertEquals(calc.calculate(expression), expected, 0.01d);

    expression = new String("(16.1 / (6.6 / 2.9))");
    expected = 7.07;
    assertEquals(calc.calculate(expression), expected, 0.01d);
}
```

- wyrażenie mieszane:

```
@Test
public void mixedExpr(){
    String expression = new String("((11 * (6 - 3)) / 2)");
    double expected = 16.5;
    assertEquals(calc.calculate(expression), expected, 0.01d);

    expression = new String("((16.6 + (6.6 / 2.9)) * 4.5)");
    expected = 84.94;
    assertEquals(calc.calculate(expression), expected, 0.01d);
}
```

3. Wnioski:
   Program przeszedł pozytywnie wszystkie testy jednostkowe. Sprawdzone zostały wszystkie jego główne funkcjonalności, a więc operacje dodania i pobrania elementu ze stosu oraz operacje arytmetyczne kalkulatora. Nie oznacza to, że program jest w pełni odporny na błędy, np. w wyrażeniach podawanych kalkulatorowi potrzebne jest zachowanie pełnego nawiasowania. System sprawdzony został jednak pod kątem występowania nieprzewidzianych w specyfikacji błędów.