

**POLITECHNIKA KRAKOWSKA**  
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ



# **Testy jednostkowe w języku JAVA**

Systemy odporne na błędy  
Sprawozdanie z laboratorium nr 3

Maria Guz  
Karol Pątko  
Wojciech Maludziński

# Wstęp

Testy jednostkowe są metodami testowania pojedynczych elementów takich jak funkcje czy też obiekty. Głównym zadaniem testów jednostkowych jest potwierdzenie, że kod działa w poprawny, założony sposób. Gdy wywoływany jest test, wykonuje się dana funkcja a potem w zależności od zastosowanej asercji porównywana z wartością oczekiwaną. Testy jednostkowe są zalecane, aby pisany kod był niezawodny. Im wcześniej wykryje się błąd, tym łatwiej i taniej go usunąć.

## Cel zajęć

Celem zajęć laboratoryjnych nr 3 jest przedstawienie działania testów jednostkowych oraz wykonanie przez studentów zadania praktycznego ukazującego ich użycie. Testy jednostkowe zostały wprowadzone do sprawdzenia stosu i kalkulatora ONP.

## Treść zadania

*Dla kalkulatora ONP stworzonego na zajęciach z “Programowania Obiektowego” stwórz klasę reprezentującą stos do przechowywania obiektów typu String. Wyposaź program w mechanizm odporności na błędy:*

- *w przypadku stosu, który nie był jeszcze używany, metoda isEmpty() powinna zwracać wartość true, a metody top() i pop() generować wyjątek,*
- *rozpocznij od pustego stosu, odłóż na stosie łańcuch tekstowy, wywołując metodę push(). Wywołaj kilkakrotnie metodę top() i sprawdź, czy za każdym razem zwraca właściwy łańcuch. Wywołaj metodę isEmpty() i sprawdź, czy metoda zwróciła wartość logiczną false,*
- *usuń łańcuch ze stosu, wywołując metodę pop(), i sprawdź czy zwróciła ten sam łańcuch, który wcześniej umieściłeś na stosie (tu nie wystarczy użyć samej asercji assertEquals(), musimy wywołać asercję assertEquals(), aby sprawdzić czy jest to ten sam obiekt). Wywołanie metody isEmpty() powinno zwrócić wartość logiczną true. Wywołaj ponownie metodę pop() i sprawdź czy wygenerowała tym razem wyjątek,*
- *wykonaj powyższy test, odkładając tym razem wiele elementów na stosie. Sprawdź czy wywołania metody pop() zwracają odpowiednie elementy we właściwej kolejności (ostatni element odłożony na stosie powinien zostać zwrócony jako pierwszy),*
- *odłóż na stosie wartość null i zdejmij ją ze stosu za pomocą metody pop(). Sprawdź czy rzeczywiście otrzymałeś wartość null,*

- sprawdź czy po wyrzuceniu wyjątku stos nadal działa poprawnie,
- przeprowadź osobne testy poprawności dla operacji arytmetycznych wykonywanych przez kalkulator ONP.

## Rozwiązanie zadania

Kod został rozdzielony na pliki `Item.java`, `Main.java`, `OnpCalc.java`, `Priority.java`, `Stack.java` oraz `OnpCalcTest.java` i `StackTest.java`. Klasa `Main` zawiera próbę wyświetlenia działania matematycznego przekonwertowanego na odwrotną notację polską za pomocą statycznej funkcji `infixToOnp()` z klasy `OnpCalc`. Klasa `Item` służy do przechowywania wartości i poprzednika elementu stosu. Klasa `OnpCalc` zawiera funkcję zmieniającą zapis infiksowy na ONP (`infixToOnp`) oraz funkcję `calculateOnp()`, która oblicza wynik działania na podstawie podanego ONP. Klasa `Priority` posiada enum do szeregowania priorytetów, funkcję `priorityComparator()` do porównywania priorytetów oraz `getPriority()` do wyłuskiwania priorytetu. Klasa `Stack` implementuje stos, zawiera metody `push()`, `pop()`, `top()` oraz `isEmpty()`. Klasy `OnpCalcTest` i `StackTest` zawierają testy jednostkowe i zostały omówione w późniejszej części dokumentu.

### Main.java

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            System.out.println(OnpCalc.infixToOnp("2+2*3.3"));  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

### Item.java

```
public class Item {  
    private String value;  
    private Item previous;  
  
    public String getValue() {  
        return value;  
    }  
  
    public void setValue(String value) throws Exception {  
        this.value = value;  
    }  
  
    public Item getPrevious() {  
        return previous;  
    }  
}
```

```

    public void setPrevious(Item previous) {
        this.previous = previous;
    }
}

```

## OnpCalc.java

```

import java.util.regex.Pattern;
import com.company.Priority.*;

import static com.company.Priority.getPriority;
import static com.company.Priority.priorityComparator;

public class OnpCalc {

    static String infixToOnp(String infix) throws Exception {

        String regular = ".*\\/(^[^.]|\\$|\\.(0{4,}.*|0{1,4}([^0-9]|$))).*";
        Pattern pattern = Pattern.compile(regular);
        if(pattern.matcher(infix).matches()){
            throw new Exception("DivideByZero");
        }

        StringBuilder onp = new StringBuilder();
        Stack stack = new Stack();

        StringBuilder nextCharacter;
        String character;
        PRIORITY characterPriority;

        while (!infix.isEmpty()) {
            character = infix.substring(0, 1);
            infix = infix.length() > 1 ? infix.substring(1) : "";

            characterPriority = getPriority(character);

            switch (characterPriority) {
                case VARIABLE:
                    nextCharacter = new StringBuilder(character);

                    if (!infix.isEmpty()) {
                        character = infix.substring(0, 1);

                        while (getPriority(character) == PRIORITY.VARIABLE) {
                            if (infix.length() > 1) {
                                infix = infix.substring(1);
                            } else {
                                infix = "";
                            }
                            nextCharacter.append(character);
                            if (infix.isEmpty()) {
                                break;
                            }
                            character = infix.substring(0, 1);
                        }
                    }
                    onp.append("_").append(nextCharacter).append("_");
                    break;

                case PRIORITY_0:
                    stack.push(character);
                    character = infix.substring(0, 1);
                    if (character.equals("-")) {
                        stack.push("~");
                        infix = infix.substring(1);
                    }
                    break;

                case PRIORITY_1:
                case PRIORITY_2:
                case PRIORITY_3:
                    if (character.equals("(")) {
                        while (!stack.top().equals("(")) {
                            onp.append(stack.top());
                            stack.pop();
                        }
                    }
            }
        }
    }
}

```

```

        stack.pop();
        break;
    }
    if ((onp.length() == 0) && character.equals("-")) {
        stack.push("~");
        break;
    }
    while (!stack.isEmpty()) {
        if (priorityComparator(characterPriority, getPriority(stack.top()))) {
            stack.push(character);
            break;
        } else {
            onp.append(stack.top());
            stack.pop();
        }
    }
    if (stack.isEmpty())
        stack.push(character);
    break;
default:
    break;
}
}

while (!stack.isEmpty()) {
    onp.append(stack.top());
    stack.pop();
}

return onp.toString();
}

public static String calculateOnp(String onp) throws Exception {
    onp = onp + "|";
    Stack stack = new Stack();
    float out, temp;
    StringBuilder tempString;
    String character = onp.substring(0, 1);
    onp = onp.substring(1);

    while (!character.equals("|")) {
        tempString = new StringBuilder();
        switch (character) {
            case "_":
                character = onp.substring(0, 1);
                onp = onp.substring(1);
                do {
                    tempString.append(character);
                    character = onp.substring(0, 1);
                    onp = onp.substring(1);
                } while (!character.equals("_"));
                stack.push(tempString.toString());
                break;
            case "+":
                temp = Float.parseFloat(stack.pop());
                out = Float.parseFloat(stack.pop()) + temp;
                stack.push(out + "");
                break;
            case "-":
                temp = Float.parseFloat(stack.pop());
                out = Float.parseFloat(stack.pop()) - temp;
                stack.push(out + "");
                break;
            case "~":
                out = -Float.parseFloat(stack.pop());
                stack.push(out + "");
                break;
            case "*":
                temp = Float.parseFloat(stack.pop());
                out = Float.parseFloat(stack.pop()) * temp;
                stack.push(out + "");
                break;
            case "/":
                temp = Float.parseFloat(stack.pop());
                if (temp == 0) {
                    throw new Exception("You can not divide by zero");
                }
                out = Float.parseFloat(stack.pop()) / temp;
                stack.push(out + "");
                break;
            case "%":
                temp = Float.parseFloat(stack.pop());
                out = Float.parseFloat(stack.pop()) % temp;

```

```

        stack.push(out+"" );
        break;
    case "^":
        temp = Float.parseFloat(stack.pop());
        out = (float) Math.pow(Float.parseFloat(stack.pop()), temp);
        stack.push(out+"" );
        break;
    }
    character = onp.substring(0, 1);
    if (onp.length() > 1)
        onp = onp.substring(1);
    else
        onp = "";
}

return stack.top();
}
}
}

```

## Priority.java

```

public class Priority {

    public enum PRIORITY {
        PRIORITY_0,
        PRIORITY_1,
        PRIORITY_2,
        PRIORITY_3,
        VARIABLE,
    }

    static boolean priorityComparator(PRIORITY priority1, PRIORITY priority2) {
        boolean ret = false;

        switch (priority1) {
            case PRIORITY_0:
                ret = false;
            case PRIORITY_1:
                if(priority2 == PRIORITY.PRIORITY_0){
                    ret = true;
                }
                break;
            case PRIORITY_2:
                if(priority2 == PRIORITY.PRIORITY_0 || priority2 == PRIORITY.PRIORITY_1){
                    ret = true;
                }
                break;
            case PRIORITY_3:
                if(priority2 == PRIORITY.PRIORITY_0 || priority2 == PRIORITY.PRIORITY_1 || priority2 ==
PRIORITY.PRIORITY_2){
                    ret = true;
                }
                break;
            default:
                ret = true;
                break;
        }
        return ret;
    }

    static PRIORITY getPriority(String symbol) {
        PRIORITY priority = null;
        switch (symbol) {

            case "(":
                priority = PRIORITY.PRIORITY_0;
                break;

            case "-": case "+": case ")":
                priority = PRIORITY.PRIORITY_1;
                break;

            case "~": case "*": case "/": case "%":
                priority = PRIORITY.PRIORITY_2;
                break;

            case "^":
                priority = PRIORITY.PRIORITY_3;
                break;
        }
    }
}

```

```

        default:
            priority = PRIORITY.VARIABLE;
            break;
    }

    return priority;
}
}

```

## Stack.java

```

public class Stack {
    private Item top;

    public Stack() {
        top = null;
    }

    public String top() throws Exception {
        if(isEmpty()) {
            throw new Exception("Stack is empty.");
        }
        return top.getValue();
    }

    public String pop() throws Exception {
        if(isEmpty()) {
            throw new Exception("Stack is empty.");
        }

        String ret = top.getValue();
        top = top.getPrevious();

        return ret;
    }

    public void push(String value) throws Exception {
        Item newItem = new Item();

        try {
            newItem.setValue(value);
        } catch (Exception e) {
            throw new Exception(e);
        }

        if(isEmpty()){
            top = newItem;
            top.setPrevious(null);
        } else {
            newItem.setPrevious(top);
            top = newItem;
        }
    }

    public boolean isEmpty() {
        return top == null;
    }
}

```

## OnpCalcTest.java

```

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

import static org.junit.jupiter.api.Assertions.*;

public class OnpCalcTest {

    @ParameterizedTest()
    @CsvSource({
        "0+0, _0_0_+",
        "1.1+7, _1.1_7_+",
        "2+3-1, _2_3_+1_-",
        "2+3-4, _2_3_+4_-",
        "2*3*4, _2_3_*4_*",
    })

```

```

        "2/3*4, _2_3/_4_+",
        "2^4^6, _2_4_^_6_^"
    })
    void
    test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithSameOperatorPriority_thenShouldReturnExpectedValue(String
    equation, String expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.infixToOnp(equation);

        //then
        assertEquals(expected, result);
    }

    @ParameterizedTest()
    @CsvSource({
        "0+0*2, _0_0_2_+",
        "1.1+7/4, _1.1_7_4_/+",
        "2+3^1, _2_3_1_^+",
        "2^3+4, _2_3_^_4_+",
        "2*3-4, _2_3_*_4_-+",
        "2/3^4, _2_3_4_^/"
    })
    void
    test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDifferentOperatorPriority_thenShouldReturnExpectedValue(Str
    ing equation, String expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.infixToOnp(equation);

        //then
        assertEquals(expected, result);
    }

    @ParameterizedTest()
    @CsvSource({
        "2+2/0.000",
        "2/0.000+1",
        "2*3/0.0",
        "3+3+3+3+-4^2/0.0*6",
        "2^0/0.00"
    })
    void test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByZero_thenShouldThrowException(String
    equation) throws Exception {
        //given
        //given from CsvSource

        //when
        //no result

        //then
        assertThrows(Exception.class, () -> OnpCalc.infixToOnp(equation));
    }

    @ParameterizedTest()
    @CsvSource({
        "2+2/0.0001",
        "2/0.0010+1",
        "2*3/0.02",
        "3+3+3+3+-4^2/3*6",
        "2^0/1"
    })
    void
    test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByAnythingExceptZero_thenShouldNotThrowException(Stri
    ng equation) throws Exception {
        //given
        //given from CsvSource

        //when
        //no result

        //then
        assertDoesNotThrow(() -> OnpCalc.infixToOnp(equation));
    }

    @ParameterizedTest()
    @CsvSource({
        "(2+2), _2_2_+",
        "(2+2)*3, _2_2_+_3_+",

```



```

        "3*(2+2), _3_2_2_+*",
        "6^(2+2)*(9+9), _6_2_2_+^_9_9_+*",
        "(2+2)/((4*4)-3), _2_2_+4_4_*_3_-/",
    })

    void test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithBracket_thenShouldGiveExpectedValue(String
equation, String expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.infixToOnp(equation);

        //then
        assertEquals(expected, result);
    }

    @ParameterizedTest()
    @CsvSource({
        "_2_2_+, 4.0",
        "_2_2_+_3_*, 12.0",
        "_3_2_2_+, 12.0",
        "_6_2_2_+^_9_9_+*, 23328.0",
        "_3_3_*_9_/_4_^, 1.0",
    })
    void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNaturalNumber(String equation, String
expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.calculateOnp(equation);

        //then
        assertEquals(expected, result);
    }

    @ParameterizedTest()
    @CsvSource({
        "_2_2_+_8_/, 0.5",
        "_2_2_+_3_*_5_/, 2.4",
        "_3_10_/, 0.3",
        "_0.2_2_*, 0.4",
    })
    void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNotNaturalNumber(String equation, String
expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.calculateOnp(equation);

        //then
        assertEquals(expected, result);
    }

    @ParameterizedTest()
    @CsvSource({
        "_2_2_-",
        "_2_2_/_1_-",
        "_3_9_*_27_-",
        "_8_4_*_2_/_16_-",
        "_6_6_*_4_/_9_/_1_-",
    })
    void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueEqualZero(String equation) throws
Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.calculateOnp(equation);

        //then
        assertEquals("0.0", result);
    }
}

```

## StackTest.java

```
import org.junit.Test;

import static org.junit.Assert.assertTrue;
import static org.junit.jupiter.api.Assertions.*;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class StackTest {

    // PUNKT 1; 3 testy
    @Test
    public void test_whenStackIsNotUsed_givenNewStack_thenIsEmptyReturnTrue() {
        //given
        Stack stack = new Stack();

        //when
        //stack not used

        //then
        assertTrue(stack.isEmpty());
    }

    @Test
    public void test_whenStackIsNotUsed_givenNewStack_thenTopMethodThrowException() {
        //given
        Stack stack = new Stack();

        //when
        //stack not used

        //then
        assertThrows(Exception.class, () -> {
            stack.top();
        });
    }

    @Test
    public void test_whenStackIsNotUsed_givenNewStack_thenPopMethodThrowException() {
        //given
        Stack stack = new Stack();

        //when
        //stack not used

        //then
        assertThrows(Exception.class, () -> {
            stack.pop();
        });
    }

    //Punkt 2; 2 testy
    @Test
    public void test_whenPushValueOnStack_givenStack_thenShouldReturnPushedValue() throws Exception {
        //given
        Stack stack = new Stack();
        String[] testValues = new String[]{"1", "2", "*", "3", "+"};

        for(String str : testValues){
            //when
            stack.push(str);
            //then
            assertEquals(stack.top(), str);
        }
    }

    @Test
    public void test_whenPushValueOnStack_givenStack_thenShouldNotBeEmpty() throws Exception {
        //given
        Stack stack = new Stack();
        String[] testValues = new String[]{"1", "2", "*", "3", "+"};

        //when
        for(String str : testValues){
            stack.push(str);
        }

        //then
        assertFalse(stack.isEmpty());
    }
}
```

```

//Punkt 3; 3 test
@Test
public void test_whenPushValueOnStack_givenStack_thenShouldReturnSameValue() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);

    //then
    assertEquals(stack.pop(), value);
}

@Test
public void test_whenPushAndPopValueOnStack_givenStack_thenShouldBeEmpty() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);
    stack.pop();

    //then
    assertTrue(stack.isEmpty());
}

@Test
public void test_whenPushAndPopValueOnStack_givenStack_thenPopShouldThrowException() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);
    stack.pop();

    //then
    assertThrows(Exception.class, () -> {
        stack.pop();
    });
}

//Punkt 4; 1 test
@Test
public void test_whenPushValuesOnStack_givenStack_thenShouldReturnValuesReversOrderNext() throws Exception {
    //given
    Stack stack = new Stack();

    //when
    String[] testValues = new String[]{"1", "2", "*", "3", "+"};
    for(String str : testValues){
        stack.push(str);
    }

    //then
    for(int i = testValues.length - 1; i>=0; --i){
        assertEquals(stack.pop(), testValues[i]);
    }
}

//PUNKT 5; 1 test
@Test
public void test_whenPushNullValue_givenStack_thenShouldReturnNullValue() throws Exception {
    //given
    Stack stack = new Stack();

    //when
    stack.push(null);

    //then
    assertNull(stack.top());
}

//PUNKT 6; 1 test
@Test
public void test_whenStackThrowException_gienStack_thenStackShouldWorkCorrectly() throws Exception{
    //given
    Stack stack = new Stack();
    //when
    try {

```

```

        stack.top();
    } catch (Exception e) {
        e.printStackTrace();
    }
    stack.push("1");
    //then
    assertEquals(stack.top(), "1");
}
}

```

## Działanie programu

### Działanie programu bez testów jednostkowych

Outputem programu jest wynik działania  $2+2*3.3$  przedstawiony w odwrotnej notacji polskiej. Dla czytelnego zapisu każda liczba rozpoczyna się oraz kończy znakiem `_`.

```

_2__2__3.3_ *+

Process finished with exit code 0

```

### Działanie programu z testami jednostkowymi

Poniżej został przedstawiony opis oraz działanie testów jednostkowych.

#### StackTest.java

*test\_whenStackIsNotUsed\_givenNewStack\_thenIsEmptyReturnTrue*

```

@Test
public void test_whenStackIsNotUsed_givenNewStack_thenIsEmptyReturnTrue() {
    //given
    Stack stack = new Stack();

    //when
    //stack not used

    //then
    assertTrue(stack.isEmpty());
}

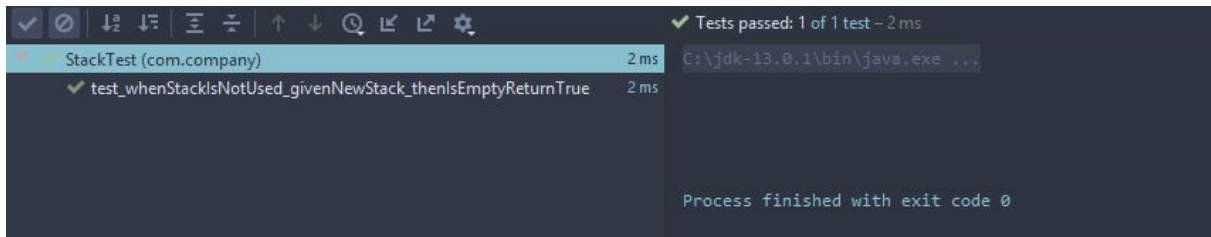
```

#### Opis testu

Przedstawiony powyżej test sprawdza, czy stos po utworzeniu jest pusty.

#### Wynik testu

Stos po utworzeniu jest pusty.



### *test\_whenStackIsNotUsed\_givenNewStack\_thenTopMethodThrowException*

```
@Test
public void test_whenStackIsNotUsed_givenNewStack_thenTopMethodThrowException() {
    //given
    Stack stack = new Stack();

    //when
    //stack not used

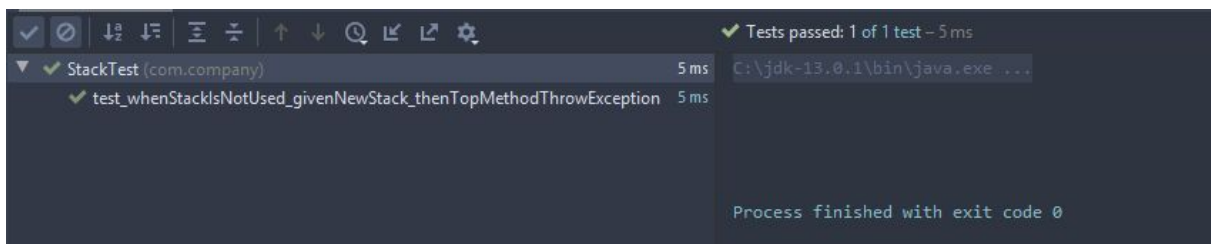
    //then
    assertThrows(Exception.class, () -> {
        stack.top();
    });
}
```

#### Opis testu

Test sprawdza, czy po utworzeniu stosu metoda *top()* zwraca wyjątek.

#### Wynik testu

Metoda *top()* zwraca wyjątek(ponieważ nie ma nic na szczycie stosu).



### *test\_whenStackIsNotUsed\_givenNewStack\_thenPopMethodThrowException*

```
@Test
public void test_whenStackIsNotUsed_givenNewStack_thenPopMethodThrowException() {
    //given
    Stack stack = new Stack();

    //when
    //stack not used

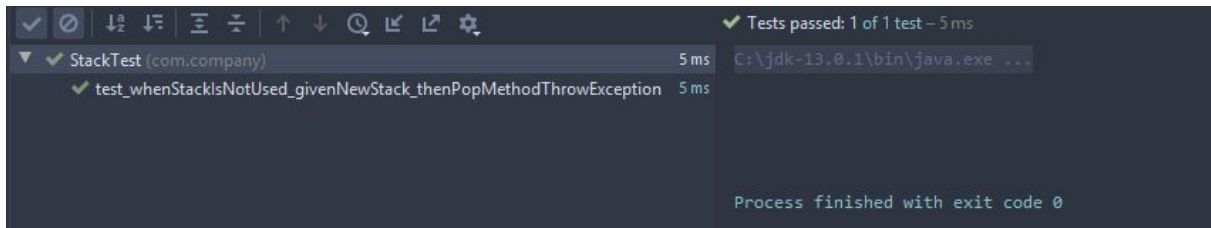
    //then
    assertThrows(Exception.class, () -> {
        stack.pop();
    });
}
```

#### Opis testu

Test sprawdza, czy po utworzeniu stosu metoda *pop()* zwraca wyjątek.

#### Wynik testu

Metoda *pop()* zwraca wyjątek(ponieważ nie ma co pobrać ze stosu).



### *test\_whenPushValueOnStack\_givenStack\_thenShouldReturnPushedValue*

```
@Test
public void test_whenPushValueOnStack_givenStack_thenShouldReturnPushedValue() throws Exception {
    //given
    Stack stack = new Stack();
    String[] testValues = new String[]{"1", "2", "*", "3", "+"};

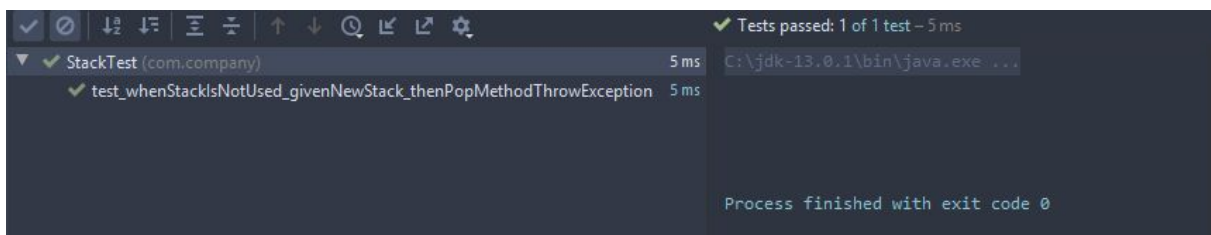
    for(String str : testValues){
        //when
        stack.push(str);
        //then
        assertEquals(stack.top(), str);
    }
}
```

#### Opis testu

Test sprawdza, czy po dodaniu wartości na stos jest ona wyświetlana.

#### Wynik testu

Po dodaniu elementu znajduje się on na szczycie stosu.



### *test\_whenPushValueOnStack\_givenStack\_thenShouldNotBeEmpty*

```
@Test
public void test_whenPushValueOnStack_givenStack_thenShouldNotBeEmpty() throws Exception {
    //given
    Stack stack = new Stack();
    String[] testValues = new String[]{"1", "2", "*", "3", "+"};

    //when
    for(String str : testValues){
        stack.push(str);
    }

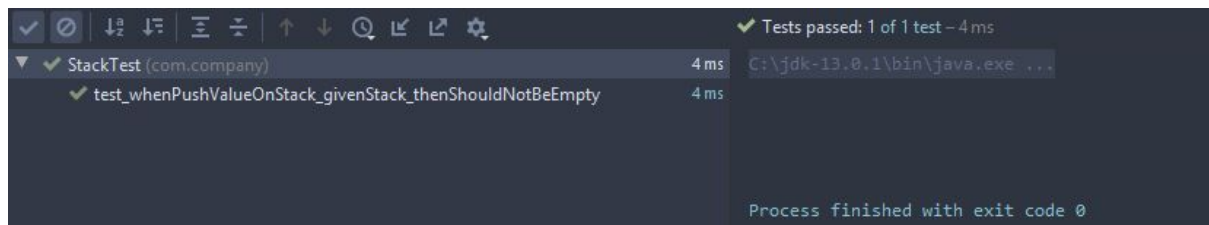
    //then
    assertFalse(stack.isEmpty());
}
```

#### Opis testu

Test sprawdza, czy po dodaniu wartości na stos fałszywe jest twierdzenie, jakoby był on pusty.

Wynik testu

Po przeprowadzeniu testu uzyskujemy wynik, że stos jest niepusty.



*test\_whenPushValueOnStack\_givenStack\_thenShouldReturnSameValue*

```
@Test
public void test_whenPushValueOnStack_givenStack_thenShouldReturnSameValue() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);

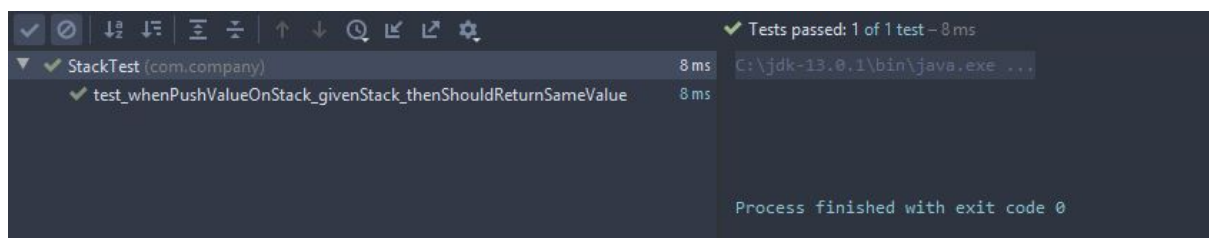
    //then
    assertEquals(stack.pop(), value);
}
```

Opis testu

Test sprawdza, czy po dodaniu wartości na stos, zdejmowana jest ta sama wartość.

Wynik testu

Element zdejmowany jest tym samym, który był dodany.



*test\_whenPushAndPopValueOnStack\_givenStack\_thenShouldBeEmpty*

```
@Test
public void test_whenPushAndPopValueOnStack_givenStack_thenShouldBeEmpty() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);
    stack.pop();

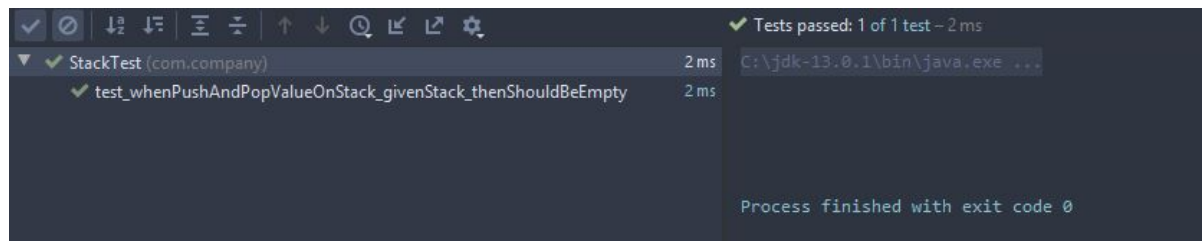
    //then
    assertTrue(stack.isEmpty());
}
```

Opis testu

Test sprawdza, czy po dodaniu i zabraniu wartości ze stosu jest on pusty.

Wynik testu

Stos po dodaniu i zabraniu jednej wartości jest pusty.



*test\_whenPushAndPopValueOnStack\_givenStack\_thenPopShouldThrowException*

```
@Test
public void test_whenPushAndPopValueOnStack_givenStack_thenPopShouldThrowException() throws Exception {
    //given
    Stack stack = new Stack();
    String value = "0";

    //when
    stack.push(value);
    stack.pop();

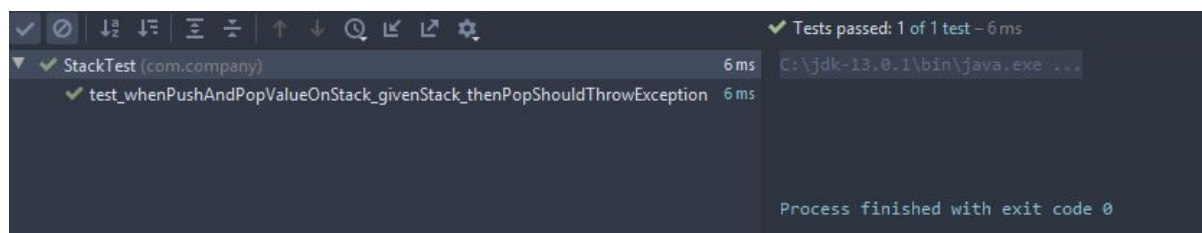
    //then
    assertThrows(Exception.class, () -> {
        stack.pop();
    });
}
```

Opis testu

Test sprawdza, czy zostanie zwrócony wyjątek przy pobraniu dwóch wartości, pomimo dodania tylko jednej.

Wynik testu

Zostanie wyrzucony wyjątek, nie można pobrać dwóch wartości, jeśli doda się tylko jedną.



*test\_whenPushValuesOnStack\_givenStack\_thenShouldReturnValuesReversOrderNext*

```
@Test
public void test_whenPushValuesOnStack_givenStack_thenShouldReturnValuesReversOrderNext() throws Exception {
    //given
    Stack stack = new Stack();

    //when
    String[] testValues = new String[]{"1", "2", "*", "3", "+"};
    for(String str : testValues){
        stack.push(str);
    }

    //then
    for(int i = testValues.length - 1; i>=0; --i){
        assertSame(stack.pop(), testValues[i]);
    }
}
```

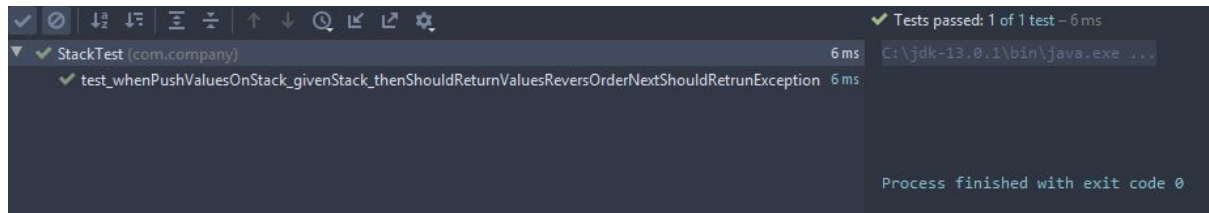


## Opis testu

Test sprawdza, czy wartości po dodaniu na stos, są zdejmowane w odwrotnej kolejności.

## Wynik testu

Test zwraca oczekiwane wyniki, elementy są w odwrotnej kolejności.



## test\_whenPushNullValue\_givenStack\_thenShouldReturnNullValue

```
@Test
public void test_whenPushNullValue_givenStack_thenShouldReturnNullValue() throws Exception {
    //given
    Stack stack = new Stack();

    //when
    stack.push(null);

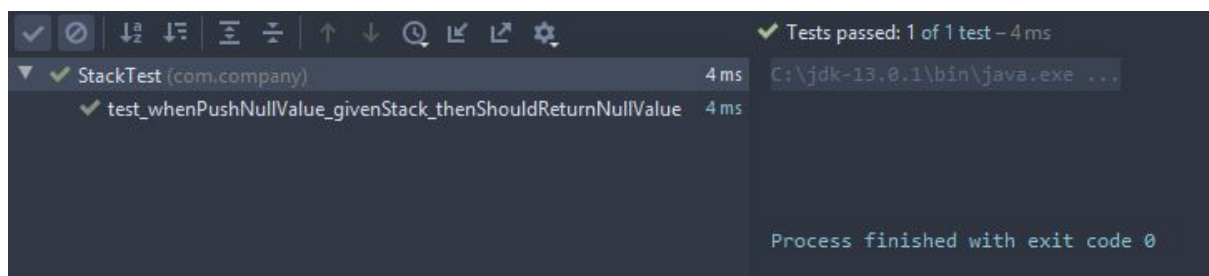
    //then
    assertNull(stack.top());
}
```

## Opis testu

Test sprawdza, czy po dodaniu wartości null, na szczycie stosu jest wartość null.

## Wynik testu

Po dodaniu wartości null, na szczycie stosu jest wartość null.



## test\_whenStackThrowEception\_gienStack\_thenStackShouldWorkCorrectly

```
@Test
public void test_whenStackThrowEception_gienStack_thenStackShouldWorkCorrectly() throws Exception{
    //given
    Stack stack = new Stack();

    //when
    try {
        stack.top();
    } catch (Exception e) {
        e.printStackTrace();
    }
    stack.push("1");

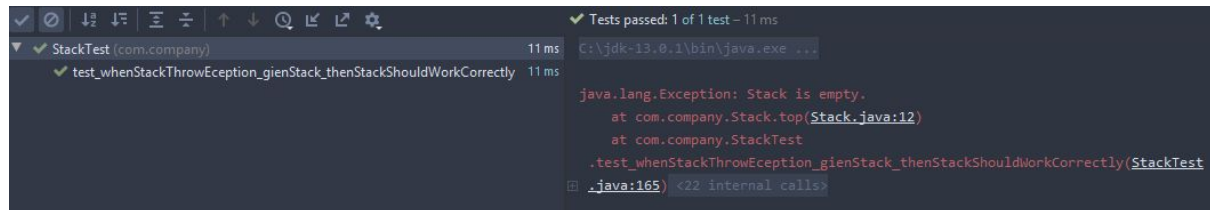
    //then
    assertEquals(stack.top(), "1");
}
```

## Opis testu

Test sprawdza, czy po zwróceniu wyjątku stos działa poprawnie.

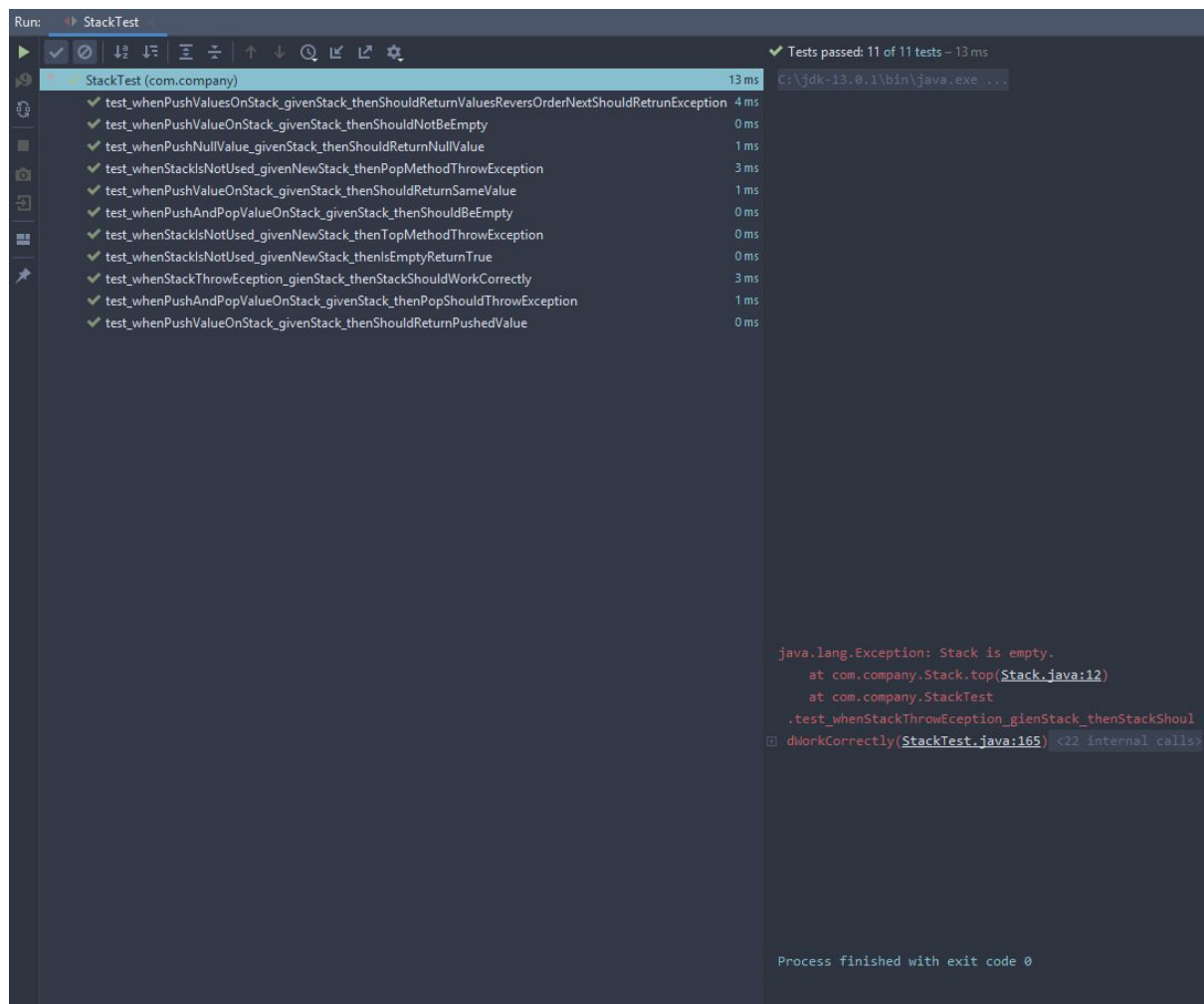
## Wynik testu

Pomimo zwrócenia wyjątku, stos działa poprawnie.



```
✓ Tests passed: 1 of 1 test - 11 ms
StackTest (com.company) 11 ms
  ✓ test_whenStackThrowException_gienStack_thenStackShouldWorkCorrectly 11 ms
    java.lang.Exception: Stack is empty.
      at com.company.Stack.top(Stack.java:12)
      at com.company.StackTest
        .test_whenStackThrowException_gienStack_thenStackShouldWorkCorrectly(StackTest
          .java:165) <22 internal calls>
```

## Wszystkie testy StackTest



```
Run: StackTest
StackTest (com.company) 13 ms
  ✓ test_whenPushValuesOnStack_givenStack_thenShouldReturnValuesReversOrderNextShouldRetrunException 4 ms
  ✓ test_whenPushValueOnStack_givenStack_thenShouldNotBeEmpty 0 ms
  ✓ test_whenPushNullValue_givenStack_thenShouldReturnNullValue 1 ms
  ✓ test_whenStackIsNotUsed_givenNewStack_thenPopMethodThrowException 3 ms
  ✓ test_whenPushValueOnStack_givenStack_thenShouldReturnSameValue 1 ms
  ✓ test_whenPushAndPopValueOnStack_givenStack_thenShouldBeEmpty 0 ms
  ✓ test_whenStackIsNotUsed_givenNewStack_thenTopMethodThrowException 0 ms
  ✓ test_whenStackIsNotUsed_givenNewStack_thenIsEmptyReturnTrue 0 ms
  ✓ test_whenStackThrowException_gienStack_thenStackShouldWorkCorrectly 3 ms
  ✓ test_whenPushAndPopValueOnStack_givenStack_thenPopShouldThrowException 1 ms
  ✓ test_whenPushValueOnStack_givenStack_thenShouldReturnPushedValue 0 ms
  ✓ Tests passed: 11 of 11 tests - 13 ms
    java.lang.Exception: Stack is empty.
      at com.company.Stack.top(Stack.java:12)
      at com.company.StackTest
        .test_whenStackThrowException_gienStack_thenStackShoul
          dWorkCorrectly(StackTest.java:165) <22 internal calls>
    Process finished with exit code 0
```

## OnpCalcTest.java

*test\_whenMakeTranslateInfixToOnp\_givenInfixFormulasWithSameOperatorPriority\_thenShouldReturnExpectedValue*

```
@ParameterizedTest()
@CsvSource({
    "0+0, _0_0_+",
    "1.1+7, _1.1_7_+",
    "2+3-1, _2_3_+_1_-",
    "2+3-4, _2_3_+_4_-",
    "2*3*4, _2_3_*_4_*",
    "2/3*4, _2_3/_4_*",
    "2^4^6, _2_4_^_6_^"
})
void
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithSameOperatorPriority_thenShouldReturnExpectedValue(String
expression, String expected) throws Exception {
    //given
    //given from CsvSource

    //when
    String result = OnpCalc.infixToOnp(expression);

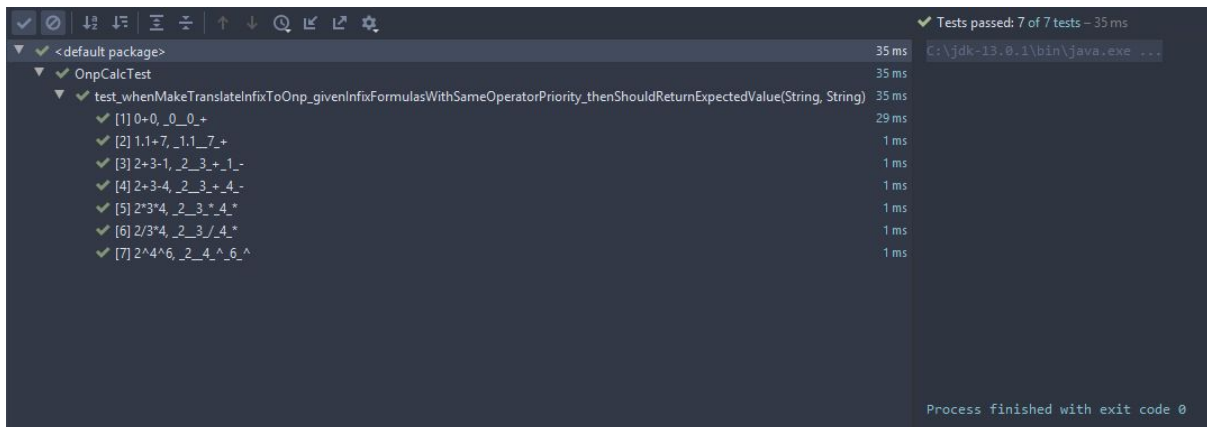
    //then
    assertEquals(expected, result); }
```

### Opis testu

Test sprawdza, czy funkcja *infixToOnp()* działa w zamierzony sposób, gdy operatory mają te same priorytety.

### Wynik testu

Funkcja poprawnie konwertuje dane testowe dla tych samych priorytetów.



*test\_whenMakeTranslateInfixToOnp\_givenInfixFormulasWithDifferentOperatorPriority\_thenShouldReturnExpectedValue*

```
@ParameterizedTest()
@CsvSource({
    "0+0*2, _0_0_2_*+",
    "1.1+7/4, _1.1_7_4_/+",
    "2+3^1, _2_3_1_^+",
    "2^3+4, _2_3_^_4_+",
    "2*3-4, _2_3_*_4_-",
    "2/3^4, _2_3_4_^/"
})
void
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDifferentOperatorPriority_thenShouldReturnExpectedValue(Str
```

```
ing expression, String expected) throws Exception {
    //given
    //given from CsvSource

    //when
    String result = OnpCalc.infixToOnp(expression);

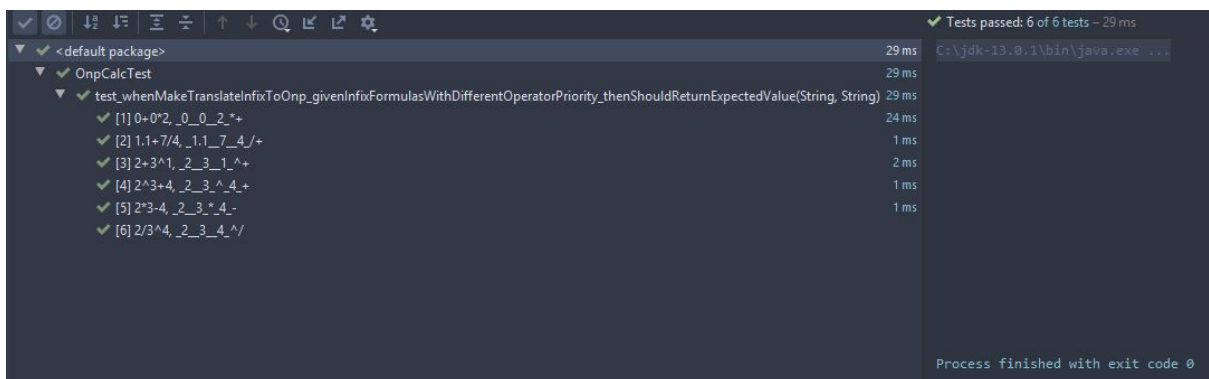
    //then
    assertEquals(expected, result);
}
```

## Opis testu

Test sprawdza, czy funkcja *infixToOnp()* z operatorami o innych priorytetach zwróci spodziewane wyniki.

## Wynik testu

Po zmianie priorytetów, funkcja zwraca spodziewane wyniki.



*test\_whenMakeTranslateInfixToOnp\_givenInfixFormulasWithDivideByZero\_thenShouldThrowException*

```
@ParameterizedTest()
@CsvSource({
    "2+2/0.000",
    "2/0.000+1",
    "2*3/0.0",
    "3+3+3+3+-4^2/0.0*6",
    "2^0/0.00",
})
void test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByZero_thenShouldThrowException(String
expression) throws Exception {
    //given
    //given from CsvSource

    //when
    //no result

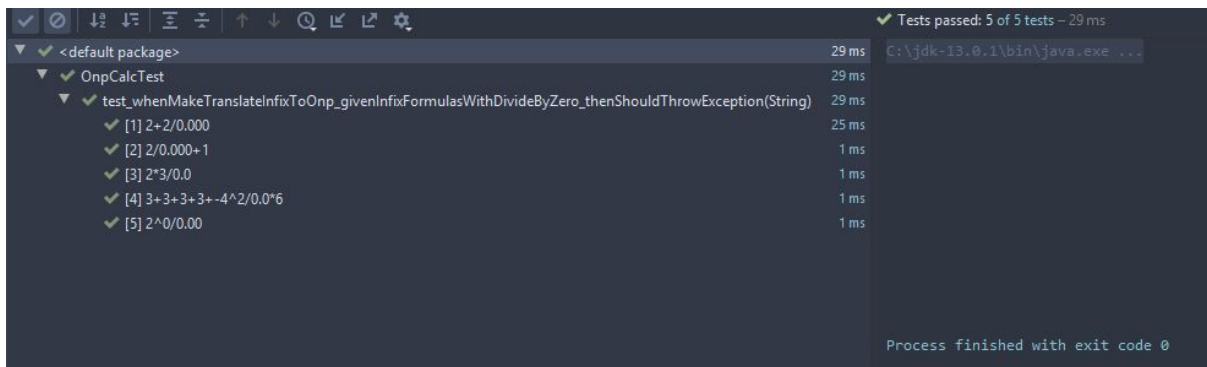
    //then
    assertThrows(Exception.class, () -> OnpCalc.infixToOnp(expression));
}
```

## Opis testu

Test sprawdza, czy przy dzieleniu przez zero zostanie zwrócony wyjątek.

## Wynik testu

Zostaje zwrócony wyjątek.



*test\_whenMakeTranslateInfixToOnp\_givenInfixFormulasWithDivideByAnythingExceptZero\_thenShouldNotThrowException*

```
@ParameterizedTest()
@CsvSource({
    "2+2/0.0001",
    "2/0.0010+1",
    "2*3/0.02",
    "3+3+3+3+-4^2/3*6",
    "2^0/1",
})
void
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByAnythingExceptZero_thenShouldNotThrowException(String expression) throws Exception {
    //given
    //given from CsvSource

    //when
    //no result

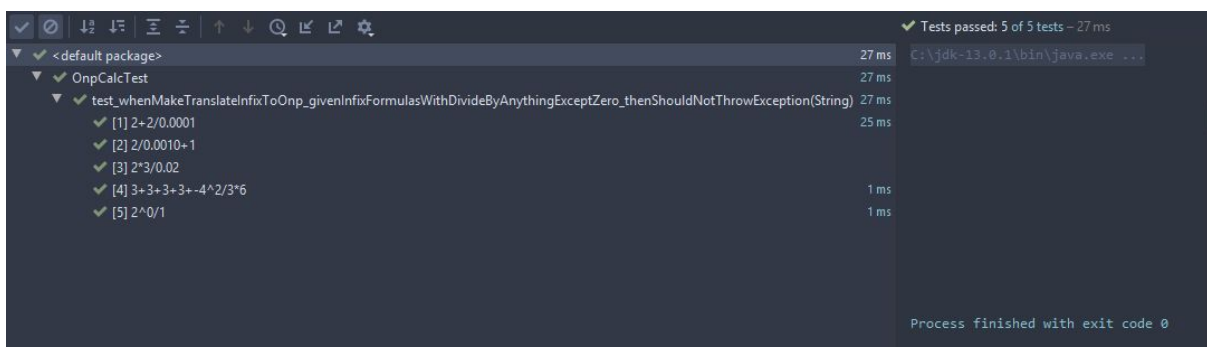
    //then
    assertDoesNotThrow(() -> OnpCalc.infixToOnp(expression));
}
```

Opis testu

Test sprawdza, czy działania niezawierające zera nie zwrócą wyjątku.

Wynik testu

Liczby inne od zera nie zwracają wyjątku.



*test\_whenMakeTranslateInfixToOnp\_givenInfixFormulasWithBracket\_thenShouldGiveExpectedValue*

```
@ParameterizedTest()
@CsvSource({
    "(2+2), _2_2_+",
    "(2+2)*3, _2_2_+_3_+",
    "3*(2+2), _3_2_2_+",
})
```

```

        "6^(2+2)*(9+9), _6_2_2_+^_9_9_+*",
        "(2+2)/((4*4)-3), _2_2_+_4_4_*_3_-/",
    })

    void test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithBracket_thenShouldGiveExpectedValue(String expression,
String expected) throws Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.infixToOnp(expression);

        //then
        assertEquals(expected, result);
    }
}

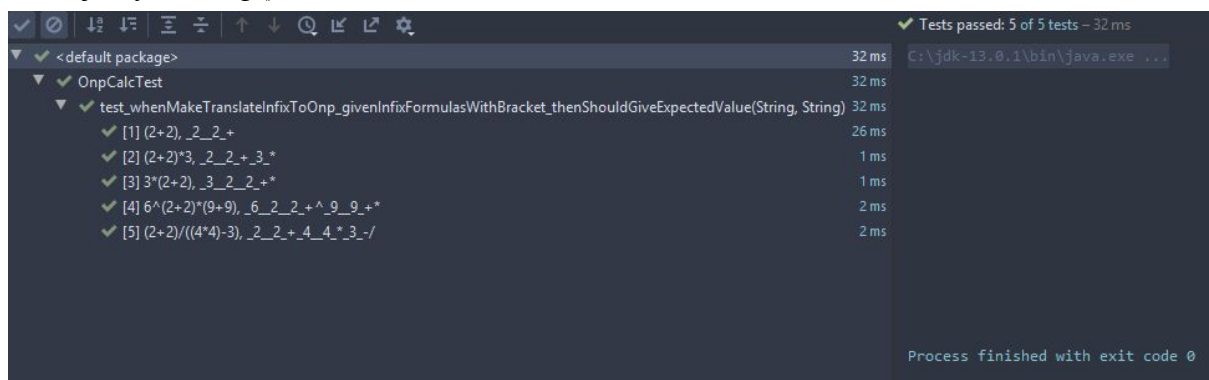
```

## Opis testu

Test sprawdza, czy transformacja z infix na ONP działa z nawiasami.

## Wynik testu

Funkcja *infixToOnp()* działa również z nawiasami.



## test\_whenCalculateOnp\_givenOnpFormula\_thenShouldGiveExpectedValueNaturalNumber

```

@ParameterizedTest()
@CsvSource({
    "_2_2_+, 4.0",
    "_2_2_+3_*, 12.0",
    "_3_2_2_+*, 12.0",
    "_6_2_2_+^_9_9_+*, 23328.0",
    "_3_3_*_9/_4_^, 1.0",
})
void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNaturalNumber(String expression, String
expected) throws Exception {
    //given
    //given from CsvSource

    //when
    String result = OnpCalc.calculateOnp(expression);

    //then
    assertEquals(expected, result);
}

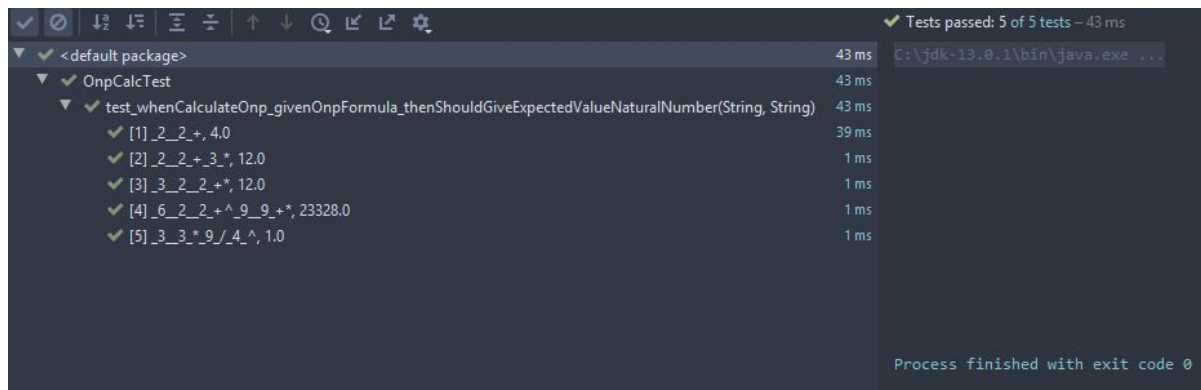
```

## Opis testu

Test sprawdza, czy zostanie zwrócona spodziewana liczba naturalna po wykonaniu kalkulacji ONP.

Wynik testu

Zostaje zwrócona spodziewana liczba naturalna.



*test\_whenCalculateOnp\_givenOnpFormula\_thenShouldGiveExpectedValueNotNaturalNumber*

```
@ParameterizedTest()
@CsvSource({
    "_2_2_+8_/, 0.5",
    "_2_2_+3_*_5_/, 2.4",
    "_3_10_/, 0.3",
    "_0.2_2_*, 0.4",
})
void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNotNaturalNumber(String expression, String
expected) throws Exception {
    //given
    //given from CsvSource

    //when
    String result = OnpCalc.calculateOnp(expression);

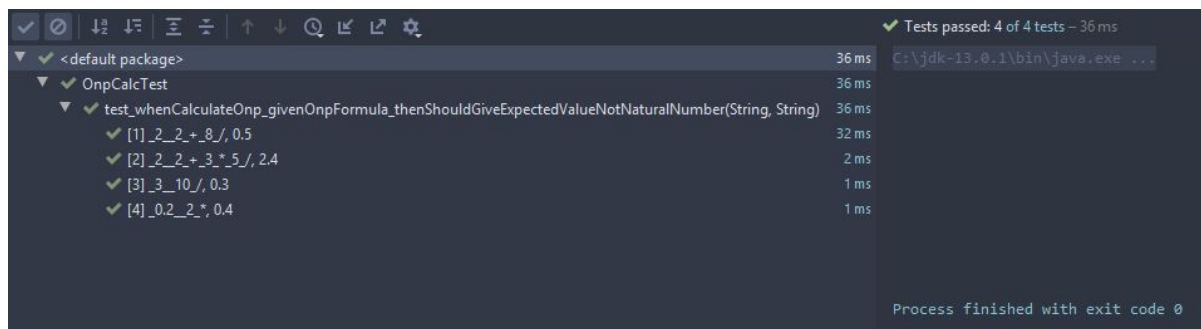
    //then
    assertEquals(expected, result);
}
```

Opis testu

Test sprawdza, czy w kalkulacji ONP zostanie zwrócona spodziewana liczba niecałkowita.

Wynik testu

Została zwrócona spodziewana liczba niecałkowita.



*test\_whenCalculateOnp\_givenOnpFormula\_thenShouldGiveExpectedValueEqualZero*

```
@ParameterizedTest()
@CsvSource({
```

```

        "_2_2_-",
        "_2_2/_1_-",
        "_3_9*_27_-",
        "_8_4*_2/_16_-",
        "_6_6*_4/_9/_1_-",
    })
    void test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueEqualZero(String expression) throws
    Exception {
        //given
        //given from CsvSource

        //when
        String result = OnpCalc.calculateOnp(expression);

        //then
        assertEquals("0.0", result);
    }
}

```

## Opis testu

Test sprawdza, czy w kalkulacji ONP zostanie zwrócona spodziewana liczba 0.

## Wynik testu

Zostaje zwrócona liczba 0.

Tests passed: 5 of 5 tests – 45 ms

Test Name	Duration
<default package>	45 ms
OnpCalcTest	45 ms
test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueEqualZero(String)	45 ms
[1] _2_2_-	40 ms
[2] _2_2/_1_-	2 ms
[3] _3_9*_27_-	1 ms
[4] _8_4*_2/_16_-	1 ms
[5] _6_6*_4/_9/_1_-	1 ms

Process finished with exit code 0

## Wszystkie testy OnpCalcTest

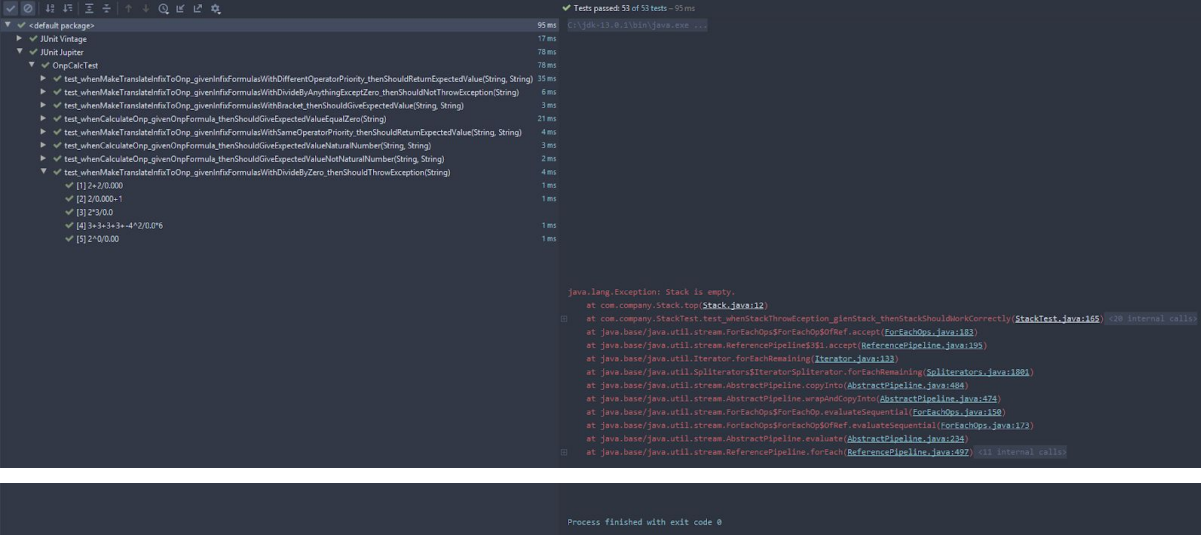
Tests passed: 42 of 42 tests – 68 ms

Test Name	Duration
<default package>	68 ms
OnpCalcTest	68 ms
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDifferentOperatorPriority_thenShouldReturnExpectedValue(String, String)	31 ms
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByAnythingExceptZero_thenShouldNotThrowException(String)	5 ms
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithBracket_thenShouldGiveExpectedValue(String, String)	3 ms
test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueEqualZero(String)	13 ms
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithSameOperatorPriority_thenShouldReturnExpectedValue(String, String)	4 ms
test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNaturalNumber(String, String)	3 ms
test_whenCalculateOnp_givenOnpFormula_thenShouldGiveExpectedValueNotNaturalNumber(String, String)	2 ms
test_whenMakeTranslateInfixToOnp_givenInfixFormulasWithDivideByZero_thenShouldThrowException(String)	7 ms
[1] 2+2/0.000	3 ms
[2] 2/0.000+1	1 ms
[3] 2^3/0.0	1 ms
[4] 3+3+3+3+ -4^2/0.0*6	1 ms
[5] 2^0/0.00	1 ms

Process finished with exit code 0



## Działanie programu ze wszystkimi testami



The screenshot shows an IDE window with a test runner on the left and a stack trace on the right. The test runner shows a list of tests, all of which passed. The stack trace on the right shows a `java.lang.Exception: Stack is empty.` error, which is a common issue in Java when a thread finishes its execution and the stack is not properly managed. The stack trace includes the following lines:

```
java.lang.Exception: Stack is empty.  
    at com.company.StackTest.test_whenStackThreadException_glenStack_thenStackShouldWorkCorrectly(StackTest.java:165) [20 internal calls]  
    at com.company.StackTest.test_whenStackThreadException_glenStack_thenStackShouldWorkCorrectly(StackTest.java:165) [20 internal calls]  
    at java.base/java.util.stream.ForEachOps$ForEachOp$OfRef.accept(ForEachOps.java:183)  
    at java.base/java.util.stream.ReferencePipeline$3$1.accept(ReferencePipeline.java:195)  
    at java.base/java.util.Iterator.forEachRemaining(Iterator.java:133)  
    at java.base/java.util.Spliterators$IteratorSpliterator.forEachRemaining(Spliterators.java:1801)  
    at java.base/java.util.stream.AbstractPipeline.copyInto(AbstractPipeline.java:484)  
    at java.base/java.util.stream.AbstractPipeline.wrapAndCopyInto(AbstractPipeline.java:474)  
    at java.base/java.util.stream.ForEachOps$ForEachOp.evaluateSequential(ForEachOps.java:158)  
    at java.base/java.util.stream.ForEachOps$ForEachOp$OfRef.evaluateSequential(ForEachOps.java:173)  
    at java.base/java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:254)  
    at java.base/java.util.stream.ReferencePipeline.forEach(ReferencePipeline.java:492) [20 internal calls]
```

Na powyższym zrzucie ekranu możemy zauważyć, że program przeszedł wszystkie wcześniej opisane testy.

## Wnioski

Testy jednostkowe to szybkie rozwiązanie, które pozwala zagwarantować, że dane funkcjonalności w kodzie działają poprawnie. Aby były miarodajne, powinny zawierać tylko po jednym assercie logicznym w każdym teście. Assert powinien być jednoznaczny, dawać zero-jedynkową odpowiedź i za każdym wykonaniem dawać tę samą odpowiedź, nawet na różnych maszynach (nie można uwzględniać testów od czasu i liczb pseudolosowych). Ponadto dobrą praktyką jest opisanie całego tekstu w nazwie, aby nie trzeba było się domyślać czego dotyczy dany test. Testy powinny być izolowane od siebie, aby nie wpływać nawzajem na swoje wykonanie, test nie może bazować na konfiguracji z innego testu. Dobrze napisane testy jednostkowe nie są jedynie formą sprawdzenia poprawności kodu ale również pełnią rolę dokumentacji. Do tego celu jednak powinny być napisane w jednaki, intuicyjny i przejrzysty sposób. Jednym ze schematów stosowanych przy tworzeniu testów jednostkowych jest użyty przez nas *Given-When-Then*.