Testowanie i weryfikacja oprogramowania
Projekt 2
Autor: Paweł Polański
Klasa StringCalculator

Celem projektu było utworzenie wg podejścia TDD klasy StringCalculator o metodzie int add (String numbers), dodającej liczby oddzielone przecinkami, podane jako parametr (w ciągu znaków). Podana była lista wymagań, które miały być sprawdzane po kolei. Najpierw utworzono szkielet klasy, potem dla każdego kroku testy i kod, który te testy przechodzi (nowe i wszystkie poprzenie).

SUT v. 0:
```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    return 0;
  }
}
```

W powyższej, zerowej wersji systemu testowanego znajduje się tylko deklaracja metody `add`, zgodna ze specyfikacją. Ponieważ musi ona zwracać jakąś wartość całkowitą, arbitralnie przyjąłem 0.
Stworzymy teraz pierwszy zestaw testów. Chcemy w nim sprawdzić, czy metoda odpowiednio reaguje na zbyt dużą liczbę danych (dopuszczalne są 0, 1 lub 2 liczby całkowite, rozdzielone przecinkami) i na napotkany ciąg znaków niebędący liczbą (zarówno pojedynczy, jak i występujący na liście obok liczby).
Zestaw testów nr 1:
```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  @Test(expected = RuntimeException.class)
  public void moreThanTwoNumbersShouldResultInException () {
    StringCalculator.add ("1,2,3");
  }

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }
}
```

Zgodnie z naszymi oczekiwaniami zerowa wersja klasy nie przechodzi testów.
Napiszemy teraz pierwszą wersję klasy, która przejdzie powyższe testy.
SUT v. 1

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    String [] numbersArray = numbers.split (",");
    if (numbersArray.length > 2) {
      throw new RuntimeException ("Up to 2 comma-separated
numbers are allowed.");
    } else {
      for (String number : numbersArray) {
        Integer.parseInt (number); // jeśli to nie liczba -
zgłoszony wyjątek
      }
    }
    return 0;
  }
}
```

Testy z zestawu nr 1 przechodzą.
Stworzymy teraz drugi zestaw. Tym razem sprawdzimy, czy metoda zwraca 0 dla pustego łańcucha znaków, jednak zgłasza wyjątek, jeśli któraś z liczb jest pustym ciągiem znaków, a cały ciąg nie jest pusty.

Zestaw testów nr 2

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  @Test(expected = RuntimeException.class)
  public void moreThanTwoNumbersShouldResultInException () {
    StringCalculator.add ("1,2,3");
  }

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }
  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }
}
```

Testy nie przechodzą.

SUT v. 2

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String [] numbersArray = numbers.split (",");
    if (numbersArray.length > 2) {
      throw new RuntimeException ("Up to 2 comma-separated
numbers are allowed.");
    } else {
      for (String number : numbersArray) {
        Integer.parseInt (number); // jeśli to nie liczba -
zgłoszony wyjątek
      }
    }
    return 0;
  }
}
```

Teraz testy przechodzą. Na początku metoda sprawdza, czy podany łańcuch jest pusty i jeśli tak, zwraca 0. Puste liczby, jako nie-liczby, powodują zgłoszenie wyjątku.

W kolejnym zestawie testowym upewnimy się, że dla danej jednej liczby metoda zwraca tę właśnie liczbę, a dla dwóch — ich sumę.

Zestaw testów nr 3

```
package stringcalculator;
```

```
import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  @Test(expected = RuntimeException.class)
  public void moreThanTwoNumbersShouldResultInException () {
    StringCalculator.add ("1,2,3");
  }

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }
}
```

Wersja 2. oczywiście nie przechodzi tych testów.

SUT v. 3

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    int returnValue = 0;
    String [] numbersArray = numbers.split (",");
    if (numbersArray.length > 2) {
      throw new RuntimeException ("Up to 2 comma-separated
numbers are allowed.");
    } else {
      for (String number : numbersArray) {
        returnValue += Integer.parseInt (number.trim()); // jeśli
to nie liczba - zgłoszony wyjątek
      }
    }
    return returnValue;
  }
}
```

Testy przechodzą. Po sprawdzeniu, czy podano jakieś liczby tworzymy zmienną i nadajemy jej wartość 0, po czym dodajemy do niej kolejno podane liczby i uzyskaną w ten sposób wartość zwracamy pod koniec.

W czwartym zestawie testowym sprawdzamy, czy metoda obsługuje nieokreśloną liczbę liczb. Musimy więc usunąć test sprawdzający, czy liczb jest nie więcej niż dwie.

Zestaw testów nr 4

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }
  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }
}
```

3. wersja klasy nie przechodzi testów.

SUT v. 4

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    int returnValue = 0;
    String [] numbersArray = numbers.split (",");
    for (String number : numbersArray) {
      returnValue += Integer.parseInt (number.trim()); // jeśli
to nie liczba - zgłoszony wyjątek
    }
    return returnValue;
  }
}
```
Teraz testy przechodzą.

W kolejnym teście sprawdzimy, czy znak nowej linii jest traktowany jako separator, tak samo jak przecinek.

Zestaw testów nr 5
```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-5
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }
}
```

4. wersja klasy nie przechodzi testów.

SUT v. 5

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.empty ()) {
      return 0;
    }
    int returnValue = 0;
    String [] numbersArray = numbers.split (",|\n");
    for (String number : numbersArray) {
      returnValue += Integer.parseInt (number.trim()); // jeśli
to nie liczba - zgłoszony wyjątek
    }
    return returnValue;
  }
}
```

To było proste. Wystarczyło dodać znak nowej linii do łańcucha będącego argumentem metody `split`. Teraz wszystkie testy przechodzą.

Następnie sprawdzimy, czy metoda obsługuje separatory ustawione przez użytkownika w pierwszej linii danych (po znakach `//`).

Zestaw testów nr 6.

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-6
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }
}
```

Na razie testy nie przechodzą.

SUT v. 6

```
package stringcalculator;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String delimiter = ",|\n";
    String numbersWithoutDelimiter = numbers;
    if (numbers.startsWith("//")) {
      int delimiterIndex = numbers.indexOf ("//") + 2;
      delimiter = numbers.substring(delimiterIndex,
delimiterIndex + 1);
      numbersWithoutDelimiter = numbers.substring
(numbers.indexOf("\n") + 1);
    }
    return add (numbersWithoutDelimiter, delimiter);
  }

  private static int add (String numbers, String delimiter) {
    int returnValue = 0;
    String [] numbersArray = numbers.split (delimiter);
    for (String number : numbersArray) {
      returnValue += Integer.parseInt (number.trim()); // jeśli
to nie liczba - zgłoszony wyjątek
    }
    return returnValue;
  }
}
```

Sprawdzamy, czy pierwsza linia zaczyna się od dwóch ukośników, a wtedy przyjmujemy jako separator znak po nich następujących. W przeciwnym razie stosujemy domyślne separatory. Testy przechodzą.

Kolejnym krokiem jest sprawdzenie, czy metoda zgłasza wyjątek, jeśli wśród podanych liczb jest jakaś ujemna. Jeśli jest, powinna zostać wyświetlona odpowiednia wiadomość mówiąca o przyczynie wyjątku i podająca listę wszystkich użytych liczb ujemnych.

Zestaw testów nr 7.

```
package stringcalculator;
```

```
import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-7
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForNegativeNumber () {
    StringCalculator.add ("1,1,2,3,-5,8");
  }

  @Test
  public void
shouldProvideExceptionReasonWithAllUsedNegativeNumbers () {
    RuntimeException exception = null;
    try {
      StringCalculator.add ("1,1,-2,3,-5,8");
    } catch (RuntimeException e) {
      exception = e;
    }
    assertNotNull (exception);
    assertEquals ("Negatives not allowed: [-2, -5]",
exception.getMessage());
  }
}
```
Testy na razie nie przechodzą.

SUT v. 7
```
package stringcalculator;

import java.util.ArrayList;
import java.util.List;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String delimiter = ",|\n";
    String numbersWithoutDelimiter = numbers;
    if (numbers.startsWith("//")) {
      int delimiterIndex = numbers.indexOf ("//") + 2;
      delimiter = numbers.substring(delimiterIndex,
delimiterIndex + 1);
      numbersWithoutDelimiter = numbers.substring
(numbers.indexOf("\n") + 1);
    }
    return add (numbersWithoutDelimiter, delimiter);
  }

  private static int add (String numbers, String delimiter) {
    int returnValue = 0;
    String [] numbersArray = numbers.split (delimiter);
    List negativeNumbers = new ArrayList ();
    for (String number : numbersArray) {
      int numberInt = Integer.parseInt (number.trim());
      if (numberInt < 0) {
        negativeNumbers.add (numberInt);
      }
      returnValue += numberInt;
    }
    if (negativeNumbers.size () > 0) {
      throw new RuntimeException ("Negatives not allowed: " +
negativeNumbers.toString ());
    }
    return returnValue;
  }
}
```

Dodaliśmy 2 importy potrzebne do utworzenia listy przechowującej wartości ujemne. Każdą liczbę ujemną dodajemy do listy i jeśli pod koniec nie jest ona pusta, zgłaszamy wyjątek wyszczególniający wszystkie te liczby. Testy przechodzą.

Następnie sprawdzamy, czy liczby większe od 1000 są ignorowane przy sumowaniu. Sprawdzamy warunek brzegowy (1000 ma być sumowane, 1001 już nie).

Zestaw testów nr 8

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-8
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }
  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForNegativeNumber () {
    StringCalculator.add ("1,1,2,3,-5,8");
  }

  @Test
  public void
shouldProvideExceptionReasonWithAllUsedNegativeNumbers () {
    RuntimeException exception = null;
    try {
      StringCalculator.add ("1,1,-2,3,-5,8");
    } catch (RuntimeException e) {
      exception = e;
    }
    assertNotNull (exception);
    assertEquals ("Negatives not allowed: [-2, -5]",
exception.getMessage());
  }

  @Test
  public void numbersGreaterThan1000ShouldBeIgnored () {
    assertEquals (17+9+1000, StringCalculator.add
("17,1001,9,1000"));
  }
}
```

Dotychczasowa wersja nie przechodzi tego zestawu testów.

SUT v. 8

```
package stringcalculator;

import java.util.ArrayList;
import java.util.List;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String delimiter = ",|\n";
    String numbersWithoutDelimiter = numbers;
    if (numbers.startsWith("//")) {
      int delimiterIndex = numbers.indexOf ("//") + 2;
      delimiter = numbers.substring(delimiterIndex,
delimiterIndex + 1);
      numbersWithoutDelimiter = numbers.substring
(numbers.indexOf("\n") + 1);
    }
    return add (numbersWithoutDelimiter, delimiter);
  }

  private static int add (String numbers, String delimiter) {
    int returnValue = 0;
    String [] numbersArray = numbers.split (delimiter);
    List negativeNumbers = new ArrayList ();
    for (String number : numbersArray) {
      int numberInt = Integer.parseInt (number.trim());
      if (numberInt < 0) {
        negativeNumbers.add (numberInt);
      } else if (numberInt <= 1000) {
        returnValue += numberInt;
      }
    }
    if (negativeNumbers.size () > 0) {
      throw new RuntimeException ("Negatives not allowed: " +
negativeNumbers.toString ());
    }
    return returnValue;
  }
}
```

Dodaliśmy kolejny warunek do `if`-a. Sumujemy tylko liczby nie większe od 1000. Testy przechodzą.

W kolejnym zestawie sprawdzamy, czy metoda pozwala na ustawienie separatora dowolnej długości i poprawnie go obsługuje.

Zestaw testów nr 9

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForNegativeNumber () {
    StringCalculator.add ("1,1,2,3,-5,8");
  }

  @Test
  public void
shouldProvideExceptionReasonWithAllUsedNegativeNumbers () {
    RuntimeException exception = null;
    try {
      StringCalculator.add ("1,1,-2,3,-5,8");
    } catch (RuntimeException e) {
      exception = e;
    }
    assertNotNull (exception);
    assertEquals ("Negatives not allowed: [-2, -5]",
exception.getMessage());
  }

  @Test
  public void numbersGreaterThan1000ShouldBeIgnored () {
    assertEquals (17+9+1000, StringCalculator.add
("17,1001,9,1000"));
  }

  @Test
  public void shouldSupportCustomDelimitersOfAnyLength () {
    //assertEquals (1+2+3, StringCalculator.add ("//[\n1—2—3"));
    assertEquals (1+2+3, StringCalculator.add ("//[—]\n1—2—3"));
    assertEquals (2+4+9, StringCalculator.add ("//
[;;]\n2;;4;;9"));
  }
}
```

Testy na razie nie przechodzą.

SUT v. 9

```
package stringcalculator;

import java.util.ArrayList;
import java.util.List;

public class StringCalculator {
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String delimiter = ",|\n";
    String numbersWithoutDelimiter = numbers;
    if (numbers.startsWith("//")) {
      int delimiterStartIndex = numbers.indexOf ("//") + 2;
      int delimiterEndIndex = numbers.indexOf("]");
      if (delimiterEndIndex == -1 || delimiterEndIndex >=
numbers.indexOf("\n")
          || !numbers.substring (delimiterStartIndex,
                              delimiterStartIndex +
1).equals("[")) {
        delimiter = numbers.substring(delimiterStartIndex,
delimiterStartIndex + 1);
      } else {
        delimiter = numbers.substring(delimiterStartIndex + 1,
            delimiterEndIndex);
      }
      numbersWithoutDelimiter = numbers.substring
(numbers.indexOf("\n") + 1);
    }
    return add (numbersWithoutDelimiter, delimiter);
  }

  private static int add (String numbers, String delimiter) {
    int returnValue = 0;
    String [] numbersArray = numbers.split (delimiter);
    List negativeNumbers = new ArrayList ();
    for (String number : numbersArray) {
      int numberInt = Integer.parseInt (number.trim());
      if (numberInt < 0) {
        negativeNumbers.add (numberInt);
      } else if (numberInt <= 1000) {
        returnValue += numberInt;
      }
    }
    if (negativeNumbers.size () > 0) {
      throw new RuntimeException ("Negatives not allowed: " +
negativeNumbers.toString ());
    }
    return returnValue;
  }
}
```

Klasa w tej wersji przechodzi testy.

Teraz chcemy umożliwić ustawienie kilku separatorów (w pierwszej linii w formacie

`//[sep1][sep2][...]`

Zestaw testów nr 10

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-10
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }

  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForNegativeNumber () {
    StringCalculator.add ("1,1,2,3,-5,8");
  }

  @Test
  public void
shouldProvideExceptionReasonWithAllUsedNegativeNumbers () {
    RuntimeException exception = null;
    try {
      StringCalculator.add ("1,1,-2,3,-5,8");
    } catch (RuntimeException e) {
      exception = e;
    }
    assertNotNull (exception);
    assertEquals ("Negatives not allowed: [-2, -5]",
exception.getMessage());
  }

  @Test
  public void numbersGreaterThan1000ShouldBeIgnored () {
    assertEquals (17+9+1000, StringCalculator.add
("17,1001,9,1000"));
  }

  @Test
  public void shouldSupportCustomDelimitersOfAnyLength () {
    //assertEquals (1+2+3, StringCalculator.add ("//[\n1—2—3"));
    assertEquals (1+2+3, StringCalculator.add ("//[—]\n1—2—3"));
    assertEquals (2+4+9, StringCalculator.add ("//
[;;]\n2;;4;;9"));
  }

  @Test
  public void shouldSupportSeveralDelimiters () {
    assertEquals (1+2+3, StringCalculator.add ("//[-][%]\n1-
2%3"));
  }
}
```

Testy oczywiście nie przechodzą.

SUT v. 10

```
package stringcalculator;

import java.util.ArrayList;
import java.util.List;

public class StringCalculator {
  // SUT 10
  public static int add (String numbers) {
    if (numbers.isEmpty ()) {
      return 0;
    }
    String delimiter = ",|\n";
    String numbersWithoutDelimiter = numbers;
    if (numbers.startsWith("//")) {
      delimiter = "";
      int delimiterStartIndex = numbers.indexOf ("//") + 2;
      int delimiterEndIndex = numbers.indexOf("]");
      if (delimiterEndIndex == -1 || delimiterEndIndex >=
numbers.indexOf("\n")
          || !numbers.substring (delimiterStartIndex,
                              delimiterStartIndex +
1).equals("[")) {
        delimiter = numbers.substring(delimiterStartIndex,
delimiterStartIndex + 1);
      } else {
        delimiter = numbers.substring(delimiterStartIndex + 1,
            delimiterEndIndex);
        delimiterStartIndex = numbers.indexOf ("[",
delimiterEndIndex);
        delimiterEndIndex = numbers.indexOf ("]",
delimiterStartIndex);
        while (delimiterStartIndex != -1 && delimiterEndIndex !=
-1 &&
               delimiterEndIndex < numbers.indexOf ("\n")) {
          delimiterStartIndex++;
          delimiter += "|" + numbers.substring
(delimiterStartIndex, delimiterEndIndex);
          delimiterStartIndex = numbers.indexOf ("[",
delimiterEndIndex);
        delimiterEndIndex = numbers.indexOf ("]",
delimiterStartIndex);
        }
      }
      numbersWithoutDelimiter = numbers.substring
(numbers.indexOf("\n") + 1);
    }
    return add (numbersWithoutDelimiter, delimiter);
  }

  private static int add (String numbers, String delimiter) {
    int returnValue = 0;
    String [] numbersArray = numbers.split (delimiter);
    List negativeNumbers = new ArrayList ();
    for (String number : numbersArray) {
      int numberInt = Integer.parseInt (number.trim());
      if (numberInt < 0) {
        negativeNumbers.add (numberInt);
      } else if (numberInt <= 1000) {
        returnValue += numberInt;
      }
    }
    if (negativeNumbers.size () > 0) {
      throw new RuntimeException ("Negatives not allowed: " +
negativeNumbers.toString ());
    }
    return returnValue;
  }
}
```

Metoda szuka po dwóch ukośnikach otwierającego nawiasu kwadratowego, i jeśli potem znajdzie nawias zamykający w tej samej linii, dodaje do listy separatorów ciąg znaków między nimi. Jest to powtarzane, aż nie będzie można znaleźć kolejnego nawiasu otwierającego. Testy teraz przechodzą.

W następnym kroku sprawdzamy, czy metoda dopuszcza kilka separatorów o długości dłuższej niż jeden znak.

Zestaw testów nr 11

```
package stringcalculator;

import org.junit.Test;
import static org.junit.Assert.*;

public class StringCalculatorTest {
  // Test set 1-11
  // commented - method is to support unknown amount of numbers
  //@Test(expected = RuntimeException.class)
  //public void moreThanTwoNumbersShouldResultInException () {
  //  StringCalculator.add ("1,2,3");
  //}

  @Test
  public void twoNumbersAreOK () {
    StringCalculator.add ("1,2");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberShouldResultInException () {
    StringCalculator.add ("@");
  }

  @Test(expected = RuntimeException.class)
  public void NonNumberInListShouldResultInException () {
    StringCalculator.add ("1,$");
  }

  @Test
  public void shouldReturnZeroForEmptyString () {
    assertEquals (0, StringCalculator.add(""));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForStringWithEmptyNumbers () {
    assertEquals (0, StringCalculator.add(","));
    assertEquals (0, StringCalculator.add(",,"));
    assertEquals (0, StringCalculator.add(",5"));
  }

  @Test
  public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber
() {
    assertEquals (5, StringCalculator.add ("5"));
  }
```

```java
  @Test
  public void shouldReturnSumOfGivenTwoNumbers () {
    assertEquals (3+4, StringCalculator.add ("3,4"));
  }

  @Test
  public void shouldReturnSumOfAnyAmountOfGivenNumbers () {
    assertEquals (3+6+1+7+8+8, StringCalculator.add
("3,6,1,7,8,8"));
  }

  @Test
  public void newLineShouldBeTreatedAsNumberSeparator () {
    assertEquals (1+2+3, StringCalculator.add ("1,2\n3"));
  }

  @Test
  public void shouldSupportCustomDelimiters () {
    assertEquals (4+9+16, StringCalculator.add("//;\n4;9;16"));
  }

  @Test(expected = RuntimeException.class)
  public void shouldRaiseExceptionForNegativeNumber () {
    StringCalculator.add ("1,1,2,3,-5,8");
  }

  @Test
  public void
shouldProvideExceptionReasonWithAllUsedNegativeNumbers () {
    RuntimeException exception = null;
    try {
      StringCalculator.add ("1,1,-2,3,-5,8");
    } catch (RuntimeException e) {
      exception = e;
    }
    assertNotNull (exception);
    assertEquals ("Negatives not allowed: [-2, -5]",
exception.getMessage());
  }

  @Test
  public void numbersGreaterThan1000ShouldBeIgnored () {
    assertEquals (17+9+1000, StringCalculator.add
("17,1001,9,1000"));
  }

  @Test
  public void shouldSupportCustomDelimitersOfAnyLength () {
    //assertEquals (1+2+3, StringCalculator.add ("//[\n1–2–3"));
    assertEquals (1+2+3, StringCalculator.add ("//[–]\n1–2–3"));
    assertEquals (2+4+9, StringCalculator.add ("//
[;;]\n2;;4;;9"));
```

```
  }

  @Test
  public void shouldSupportSeveralDelimiters () {
    assertEquals (1+2+3, StringCalculator.add ("//[-][%]\n1-
2%3"));
  }

  @Test
  public void shouldSupportSeveralDelimitersOfAnyLength () {
    assertEquals (1+1+2+3+5+8+13, StringCalculator.add ("//[–][%]
[--][;;]\n1–1%2%3;;5--8–13"));
    assertEquals (StringCalculator.add ("//[–][„][%]\n1–2„3%4"),
1+2+3+4);
  }
}
```

Tym razem poprzednia wersja klasy była napisana tak dobrze, że przeszła te testy. Po prostu uznałem, że łatwiej jest to zaimplementować to w takiej formie, niż za każdym razem dodawać jako separator jeden znak pomiędzy nawiasami.