Kata : String Calculator

L'objectif est de développer une calculatrice en Java en utilisant le principe du développement dirigé par les tests.

I. Les fonctionnalités à développer

Il s'agit de créer une calculatrice qui dispose d'une méthode « add(String numbers) ». Les fonctionnalités sont données dans l'ordre de priorité du développement :

1. La méthode peut prendre 0, 1 ou 2 nombres séparés par une virgule, et doit retourner leur somme. Par exemple "" ou "1" ou "1,2" (pour une chaîne vide elle devra retourner 0).
2. La méthode doit pouvoir prendre un nombre quelconque de nombres.
3. La méthode doit pouvoir gérer une fin de ligne entre les nombres à la place de la virgule. L'entrée suivante est acceptable et retournera 6 : "1\n2,3".
4. Elle doit supporter différents délimiteurs. Pour changer le délimiteur, le début de la chaîne contiendra une ligne à part comme celle-ci : "//[delimiter]\n[numbers...]". Par exemple "//;n1;2" devra retourner 3 où le délimiteur est ';'. Cette première ligne est optionnelle.
5. Appeler la méthode avec un nombre négatif devra retourner une exception avec le message "negatives not allowed : " et la valeur négative passée. S'il y a plusieurs nombres négatifs dans la chaîne, il faut afficher tous ces nombres dans le message d'exception message.
6. Les nombres supérieurs à 1000 doivent être ignorés, donc add("2 + 1001") retournera 2.
7. Les délimiteurs peuvent être de n'importe quelle longueur avec le format suivant : "//[delimiter]\n" par exemple : "//[***]\n1***2***3" devra retourner 6.
8. Accepter plusieurs délimiteurs comme ceci : "//[delim1][delim2]\n". Par exemple

l'appel de la méthode avec la chaîne suivante “//[*][%]\n1*2%3” doit retourner 6.

9. Assurez-vous que la méthode peut accepter plusieurs délimiteurs avec une longueur de plus d'un caractère.

II. Solution possible

II.1 Fonctionnalité 1 : La méthode add() peut pendre une chaîne contenant 0, 1 ou 2 nombres séparés par des virgules

Par exemple : "", "1" ou "1,2"

Java Test

Ajouter la classe de test `StringCalculatorTest` :

```
1   public final class StringCalculatorTest {
2      @Test(expected = RuntimeException.class)
3      public void whenMoreThan2NumbersAreUsedThenExceptionIsThrown(){
4         StringCalculator.add("1,2,3");
5      }
6      @Test
7      public void when2NumbersAreUsedThenNoExceptionIsThrown() {
8         StringCalculator.add("1,2");
9         assertTrue(true);
10     }
11     @Test(expected = RuntimeException.class)
12     public void whenNonNumberIsUsedThenExceptionIsThrown() {
13        StringCalculator.add("1,X");
14     }
15  }
```

Java Implémentation

```
1   public final class StringCalculator1 {
2      public static void add(final String numbers) {
3         String[] numbersArray = numbers.split(",");
4         if (numbersArray.length > 2) {
5            throw new RuntimeException("Up to 2 numbers separated by"
6                                       + "comma (,) are allowed");
7         } else {
8            for (String number : numbersArray) {
9               // If it is not a number, parseInt will throw an exception
10              Integer.parseInt(number);
11           }
12        }
13     }
14  }
```

II.2 Fonctionnalité 2 : Pour une chaîne vide la méthode doit retourner 0

Java Test

```
1   @Test
2   public void whenEmptyStringIsUsedThenReturnValueIs0() {
3      assertEquals(0, StringCalculator.add(""));
4   }
```

Java Implémentation

```
1   public static int add(final String numbers) { // Changed void to int
2      String[] numbersArray = numbers.split(",");
3      if (numbersArray.length > 2) {
4         throw new RuntimeException("Up to 2 numbers separated by"
5                                    + "comma (,) are allowed");
6      } else {
7         for (String number : numbersArray) {
8            if (!number.isEmpty()) {
9               Integer.parseInt(number);
10           }
11        }
12     }
13     return 0; // Added return
14  }
```

II.3 Fonctionnalité 3 : La méthode doit retourner la somme des nombres

Java Test

```
1   @Test
2   public void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
3      assertEquals(3, StringCalculator.add("3"));
4   }
5
6   @Test
7   public void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
8      assertEquals(3+6, StringCalculator.add("3,6"));
9   }
```

Java Implémentation

```
1   public static int add(final String numbers) {
2      int returnValue = 0;
3      String[] numbersArray = numbers.split(",");
4      if (numbersArray.length > 2) {
5         throw new RuntimeException("Up to 2 numbers separated by "
6                                    + " comma (,) are allowed");
7      }
8      for (String number : numbersArray) {
9         if (!number.trim().isEmpty()) { // After refactoring
10           returnValue += Integer.parseInt(number);
11        }
12     }
13     return returnValue;
14  }
```

II.4 Fonctionnalité 4 : La méthode doit accepter un nombre quelconque de nombres

Java Test

```
1   //    @Test(expected = RuntimeException.class)
2   //    public void whenMoreThan2NumbersAreUsedThenExceptionIsThrown() {
3   //      StringCalculator.add("1,2,3");
4   //    }
5   @Test
6   public void whenAnyNumberOfNumbersIsUsedThenReturnTheirSums(){
7      assertEquals(3+6+15+18+46+33,
8                   StringCalculator.add("3,6,15,18,46,33"));
9   }
```

Java Implémentation

```
1   public static int add(final String numbers) {
2      int returnValue = 0;
3      String[] numbersArray = numbers.split(",");
4      // Removed after exception
5      // if (numbersArray.length > 2) {
6      // throw new RuntimeException("Up to 2 numbers separated by "
7      //                            + " comma (,) are al lowed");
8      // }
9      for (String number : numbersArray) {
10        if (!number.trim().isEmpty()) { // After refactoring
11          returnValue += Integer.parseInt(number);
12       }
13     }
14     return returnValue;
15  }
```

II.5 Fonctionnalité 5 : La méthode doit accepter une fin de ligne entre les nombres (à la place des virgules).

Java Test

```
1   @Test
2   public void whenNewLineIsUsedBetweenNumbersThenReturnTheirSums() {
3      assertEquals(3+6+15, StringCalculator.add("3,6n15"));
4   }
```

Java Implémentation

```
1   public static int add(final String numbers) {
2      int returnValue = 0;
3      String[] numbersArray = numbers.split(",|n"); // Add |n to the split regex
4      for (String number : numbersArray) {
5      if(!number.trim().isEmpty()) {
6         returnValue += Integer.parseInt(number.trim());
7      }
8   }
9   return returnValue;
10  }
```

II.6 Fonctionnalité 6 : La méthode doit accepter différents délimitateur

Pour changer un délimiteur, le début de la chaîne de caractères devra

contenir une ligne indépendante de la forme : “//[delimiter]\n[numbers...]”. Par exemple “//;\n1;2″ devra prendre 1 et 2 comme paramètres et retourner 3 avec un délimiteur par défaut ‘;’.

Java Test

```
1   @Test
2   public void whenDelimiterIsSpecifiedThenItIsUsedToSeparateNumbers(){
3      assertEquals(3+6+15, StringCalculator.add("//;n3;6;15"));
4   }
```

Java Implémentation

```
1   public static int add(final String numbers) {
2      String delimiter = ",|n";
3      String numbersWithoutDelimiter = numbers;
4      if (numbers.startsWith("//")) {
5         int delimiterIndex = numbers.indexOf("//") + 2;
6         delimiter = numbers.substring(delimiterIndex, delimiterIndex + 1);
7         numbersWithoutDelimiter = numbers.substring(numbers.indexOf("n") + 1);
8      }
9      return add(numbersWithoutDelimiter, delimiter);
10  }
11
12  private static int add(final String numbers, final String delimiter) {
13     int returnValue = 0;
14     String[] numbersArray = numbers.split(delimiter);
15     for (String number : numbersArray) {
16        if (!number.trim().isEmpty()) {
17           returnValue += Integer.parseInt(number.trim());
18        }
19     }
20     return returnValue;
21  }
```

II.7 Fonctionnalité 7 : Les nombres négatifs doivent générer une exception

Appeler add avec un nombre négatif devra lancer l'exception “negatives not allowed” – et la valeur négative passée. S'il y a plusieurs valeurs négatives, il faut les donner toutes dans le message d'exception.

Java Test

```
1   @Test(expected = RuntimeException.class)
2   public void whenNegativeNumberIsUsedThenRuntimeExceptionIsThrown() {
3      StringCalculator.add("3,-6,15,18,46,33");
4   }
5   @Test
6   public void whenNegativeNumbersAreUsedThenRuntimeExceptionIsThrown() {
7      RuntimeException exception = null;
8      try {
9         StringCalculator.add("3,-6,15,-18,46,33");
10     } catch (RuntimeException e) {
11        exception = e;
12     }
13     assertNotNull(exception);
14     assertEquals("Negatives not allowed: [-6, -18]", exception.getMessage());
15     }
```

Java Implémentation

```
1   private static int add(final String numbers, final String delimiter) {
2      int returnValue = 0;
3      String[] numbersArray = numbers.split(delimiter);
4      List negativeNumbers = new ArrayList();
5      for (String number : numbersArray) {
6         if (!number.trim().isEmpty()) {
7            int numberInt = Integer.parseInt(number.trim());
8            if (numberInt < 0) {
9               negativeNumbers.add(numberInt);
10           }
11           returnValue += numberInt;
12        }
13     }
14     if (negativeNumbers.size() > 0) {
15        throw new RuntimeException("Negatives not allowed: "
16                        + negativeNumbers.toString());
17     }
18     return returnValue;
19  }
```

II.8 Fonctionnalité 8 : Les nombres supérieurs à 1000 doivent être ignorés

Exemple : ajouter 2 + 1001 = 2

Java Test

```
1   @Test
2   public void
whenOneOrMoreNumbersAreGreaterThan1000IsUsedThenItIsNotIncludedInSum() {
3      assertEquals(3+1000+6, StringCalculator8.add("3,1000,1001,6,1234"));
4   }
```

Java Implémentation

```
1   private static int add(final String numbers, final String delimiter) {
2      int returnValue = 0;
3      String[] numbersArray = numbers.split(delimiter);
4      List negativeNumbers = new ArrayList();
5      for (String number : numbersArray) {
6         if (!number.trim().isEmpty()) {
7            int numberInt = Integer.parseInt(number.trim());
8            if (numberInt < 0) {
9               negativeNumbers.add(numberInt);
10           } else if (numberInt <= 1000) {
11              returnValue += numberInt;
12           }
13        }
14     }
15     if (negativeNumbers.size() > 0) {
16        throw new RuntimeException("Negatives not allowed: "
17                                    + negativeNumbers.toString());
18     }
19     return returnValue;
20  }
```

II.9 Fonctionnalité 9 : les délimiteurs peuvent être de n'importe quel taille

Le format suivant devra être utilisé : “//[delimiter]\n”. L'exemple suivant : “//[—]\n1—2—3″ devra retourner 6.

...

II.10 Fonctionnalité 10 : Permettre plusieurs délimiteurs

Le format suivant devra être utilisé : “//[delim1][delim2]\n”. L'exemple suivant “//[-][%]\n1-2%3″ devra retourner 6.

...

II.11 Fonctionnalité 11 : Assurer que l'on peut utiliser des délimiteurs de taille supérieur à un caractère

...