```java
import static org.junit.Assert.assertEquals;

import org.junit.Test;

import components.set.Set;
import components.set.Set1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Test class to test methods from StringReassembly utility class.
 *
 * @author Shyam Sai Bethina
 *
 */
public class StringReassemblyTest {

    /*
     * Tests of combination method
     */

    /**
     * Boundary test case with one character one character.
     */
    @Test
    public void testCombination_l() {
        String str1 = "l";
        String str1Expected = "l";
        String str2 = "l";
        String str2Expected = "l";
        String result = StringReassembly.combination(str1, str2,
1);
        assertEquals(str1Expected, str1);
        assertEquals(str2Expected, str2);
        assertEquals("l", result);
    }

    /**
     * Challenging test case with the same characters.
     */
    @Test
    public void testCombination_ab() {
```

```
44          String str1 = "ab";
45          String str1Expected = "ab";
46          String str2 = "ab";
47          String str2Expected = "ab";
48          String result = StringReassembly.combination(str1, str2,
    2);
49          assertEquals(str1Expected, str1);
50          assertEquals(str2Expected, str2);
51          assertEquals("ab", result);
52      }
53
54      /**
55       * Routine test case with overlap of 3 characters.
56       */
57      @Test
58      public void testCombination_toastAst3() {
59          String str1 = "toast";
60          String str1Expected = "toast";
61          String str2 = "ast";
62          String str2Expected = "ast";
63          String result = StringReassembly.combination(str1, str2,
    3);
64          assertEquals(str1Expected, str1);
65          assertEquals(str2Expected, str2);
66          assertEquals("toast", result);
67      }
68
69      /**
70       * Challenging test with not overlap.
71       */
72      @Test
73      public void testCombination_overLap() {
74          String str1 = "over";
75          String str1Expected = "over";
76          String str2 = "lap";
77          String str2Expected = "lap";
78          String result = StringReassembly.combination(str1, str2,
    0);
79          assertEquals(str1Expected, str1);
80          assertEquals(str2Expected, str2);
81          assertEquals("overlap", result);
82      }
83
84      /**
```

```
 85      * Challenging test case with both strings being the same
    character.
 86      */
 87     @Test
 88     public void testCombination_jjjj() {
 89         String str1 = "jjjj";
 90         String str1Expected = "jjjj";
 91         String str2 = "jjjj";
 92         String str2Expected = "jjjj";
 93         String result = StringReassembly.combination(str1, str2,
    2);
 94         assertEquals(str1Expected, str1);
 95         assertEquals(str2Expected, str2);
 96         assertEquals("jjjjjj", result);
 97     }
 98
 99     /*
100      * Tests of addToSetAvoidingSubstrings method
101      */
102
103     /**
104      * Boundary test case with only one element.
105      */
106     @Test
107     public void testAddToSetAvoidingSubstrings_1Element() {
108         Set<String> strSet = new Set1L<>();
109         strSet.add("b");
110         Set<String> strSetExpected = new Set1L<>();
111         strSetExpected.add("b");
112         String str1 = "b";
113         String str1Expected = "b";
114         StringReassembly.addToSetAvoidingSubstrings(strSet, str1);
115         assertEquals(str1Expected, str1);
116         assertEquals(strSetExpected, strSet);
117     }
118
119     /**
120      * Routine test case with three elements.
121      */
122     @Test
123     public void testAddToSetAvoidingSubstrings_3Elements() {
124         Set<String> strSet = new Set1L<>();
125         strSet.add("abcd");
126         strSet.add("abc");
```

```
127         strSet.add("bc");
128         Set<String> strSetExpected = new Set1L<>();
129         strSetExpected.add("abcd");
130         strSetExpected.add("abc");
131         strSetExpected.add("bc");
132         String str1 = "b";
133         String str1Expected = "b";
134         StringReassembly.addToSetAvoidingSubstrings(strSet, str1);
135         assertEquals(str1Expected, str1);
136         assertEquals(strSetExpected, strSet);
137     }
138 
139     /**
140      * Challenging test case as the str to be added is the superstring of all
141      * the elements in the set.
142      */
143     @Test
144     public void testAddToSetAvoidingSubstrings_3ElementsSuper() {
145         Set<String> strSet = new Set1L<>();
146         strSet.add("abc");
147         strSet.add("ab");
148         strSet.add("b");
149         Set<String> strSetExpected = new Set1L<>();
150         strSetExpected.add("abcd");
151         String str1 = "abcd";
152         String str1Expected = "abcd";
153         StringReassembly.addToSetAvoidingSubstrings(strSet, str1);
154         assertEquals(str1Expected, str1);
155         assertEquals(strSetExpected, strSet);
156     }
157 
158     /**
159      * Routine test case with three elements but the third is not a substring of
160      * any of them.
161      */
162     @Test
163     public void testAddToSetAvoidingSubstrings_3elementNoSubstring() {
164         Set<String> strSet = new Set1L<>();
165         strSet.add("abc");
166         strSet.add("ab");
167         strSet.add("b");
```

```
168         Set<String> strSetExpected = new Set1L<>();
169         strSetExpected.add("abc");
170         strSetExpected.add("ab");
171         strSetExpected.add("b");
172         strSetExpected.add("xyz");
173         String str1 = "xyz";
174         String str1Expected = "xyz";
175         StringReassembly.addToSetAvoidingSubstrings(strSet, str1);
176         assertEquals(str1Expected, str1);
177         assertEquals(strSetExpected, strSet);
178     }
179 
180     /*
181      * Tests of linesFromInput method
182      */
183 
184     /**
185      * Routine test case with two lines of input.
186      */
187     @Test
188     public void testLinesFromInput_helloBye() {
189         SimpleReader input = new SimpleReader1L("data/
    testLineFromInput1");
190         Set<String> result =
    StringReassembly.linesFromInput(input);
191         Set<String> resultExpected = new Set1L<>();
192         resultExpected.add("hello");
193         resultExpected.add("bye");
194         assertEquals(resultExpected, result);
195 
196     }
197 
198     /**
199      * Challenging test case with one string being the superstring
    of the
200      * others.
201      */
202     @Test
203     public void testLinesFromInput_superstring() {
204         SimpleReader input = new SimpleReader1L("data/
    testLineFromInput2");
205         Set<String> result =
    StringReassembly.linesFromInput(input);
206         Set<String> resultExpected = new Set1L<>();
```

```
207         resultExpected.add("abcd");
208         assertEquals(resultExpected, result);
209     }
210 
211     /**
212      * Boundary test case with one character.
213      */
214     @Test
215     public void testLinesFromInput_oneCharacter() {
216         SimpleReader input = new SimpleReader1L("data/
    testLineFromInput3");
217         Set<String> result =
    StringReassembly.linesFromInput(input);
218         Set<String> resultExpected = new Set1L<>();
219         resultExpected.add("l");
220         assertEquals(resultExpected, result);
221     }
222 
223     /**
224      * Challenging test case with one string not being a substring
    of the
225      * others.
226      */
227     @Test
228     public void testLinesFromInput_noSubstring() {
229         SimpleReader input = new SimpleReader1L("data/
    testLineFromInput4");
230         Set<String> result =
    StringReassembly.linesFromInput(input);
231         Set<String> resultExpected = new Set1L<>();
232         resultExpected.add("abcd");
233         resultExpected.add("xyz");
234         assertEquals(resultExpected, result);
235     }
236 
237     /*
238      * Tests of printWithLineSeparators method
239      */
240 
241     /**
242      * Routine test case with just one tilde.
243      */
244     @Test
245     public void testPrintWithLinesSeparators_oneTilde() {
```

```
246         SimpleWriter out = new SimpleWriter1L("data/
    testPrintWithSep1.txt");
247         String input = "hello~bye";
248         StringReassembly.printWithLineSeparators(input, out);
249         SimpleReader in = new SimpleReader1L("data/
    testPrintWithSep1.txt");
250
251         String line1 = "hello";
252         String line2 = "bye";
253         assertEquals(line1, in.nextLine());
254         assertEquals(line2, in.nextLine());
255
256         out.close();
257         in.close();
258     }
259
260     /**
261      * Challenging test case with multiple tildes.
262      */
263     @Test
264     public void testPrintWithLinesSeparators_multipleTildes() {
265         SimpleWriter out = new SimpleWriter1L("data/
    testPrintWithSep2.txt");
266         String input = "hello~~~bye";
267         StringReassembly.printWithLineSeparators(input, out);
268         SimpleReader in = new SimpleReader1L("data/
    testPrintWithSep2.txt");
269
270         String line1 = "hello";
271         String line2 = "";
272         String line3 = "";
273         String line4 = "bye";
274         assertEquals(line1, in.nextLine());
275         assertEquals(line2, in.nextLine());
276         assertEquals(line3, in.nextLine());
277         assertEquals(line4, in.nextLine());
278
279         out.close();
280         in.close();
281     }
282
283     /**
284      * Boundary test case with one tilde at the end.
285      */
```

```
286     @Test
287     public void testPrintWithLinesSeparators_hello() {
288         SimpleWriter out = new SimpleWriter1L("data/
    testPrintWithSep3.txt");
289         String input = "hello~";
290         StringReassembly.printWithLineSeparators(input, out);
291         SimpleReader in = new SimpleReader1L("data/
    testPrintWithSep3.txt");
292 
293         String line1 = "hello";
294         assertEquals(line1, in.nextLine());
295 
296         out.close();
297         in.close();
298     }
299 
300     /**
301      * Routine test case with a sentence.
302      */
303     @Test
304     public void testPrintWithLinesSeparators_sentence() {
305         SimpleWriter out = new SimpleWriter1L("data/
    testPrintWithSep1.txt");
306         String input = "hello~my~name~is~Sai";
307         StringReassembly.printWithLineSeparators(input, out);
308         SimpleReader in = new SimpleReader1L("data/
    testPrintWithSep1.txt");
309 
310         String line1 = "hello";
311         String line2 = "my";
312         String line3 = "name";
313         String line4 = "is";
314         String line5 = "Sai";
315         assertEquals(line1, in.nextLine());
316         assertEquals(line2, in.nextLine());
317         assertEquals(line3, in.nextLine());
318         assertEquals(line4, in.nextLine());
319         assertEquals(line5, in.nextLine());
320 
321         out.close();
322         in.close();
323     }
324 
325     /**
```

```
326      * Boundary test case with just tilde and nothing else.
327      */
328     @Test
329     public void testPrintWithLinesSeparators_justTilde() {
330         SimpleWriter out = new SimpleWriter1L("data/
    testPrintWithSep5.txt");
331         String input = "~";
332         StringReassembly.printWithLineSeparators(input, out);
333         SimpleReader in = new SimpleReader1L("data/
    testPrintWithSep5.txt");
334 
335         String line1 = "";
336         assertEquals(line1, in.nextLine());
337 
338         out.close();
339         in.close();
340     }
341 
342 }
343 
```