```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 for StringReassembly with comprehensive test cases.
 *
 * @author Jeng Zhuang
 */
public class StringReassemblyTest {

    /*
     * Tests of overlap method
     */

    /**
     * Tests overlap with "qwere" and "erewq" which should have 3-character
     * overlap.
     */
    @Test
    public void testOverlapRacecar() {
        String str1 = "qwere";
        String str2 = "erewq";
        final int three = 3;
        int overlap = StringReassembly.overlap(str1, str2);
        assertEquals(three, overlap);
    }

    /**
     * Tests overlap with "columb" and "olumbus" which should have 5-character
     * overlap.
```

```
39      */
40     @Test
41     public void testOverlapWashington() {
42         String str1 = "columb";
43         String str2 = "olumbus";
44         final int five = 5;
45         int overlap = StringReassembly.overlap(str1, str2);
46         assertEquals(five, overlap);
47     }
48 
49     /**
50      * Tests overlap with non-overlapping strings "Hey" and "hi".
51      */
52     @Test
53     public void testOverlapHey() {
54         String str1 = "Hey";
55         String str2 = "hi";
56         int overlap = StringReassembly.overlap(str1, str2);
57         assertEquals(0, overlap);
58     }
59 
60     /*
61      * Tests of combination method
62      */
63 
64     /**
65      * Tests combination of "racec" and "cecar" with 3-character
   overlap.
66      */
67     @Test
68     public void testCombinationRacecar() {
69         String str1 = "qwere";
70         String str2 = "erewq";
71         final int three = 3;
72         int overlap = three;
73         String combine = StringReassembly.combination(str1, str2,
   overlap);
74         assertEquals("qwerewq", combine);
75     }
76 
```

```
77      /**
78       * Tests combination of "Colum" and "umbus" with 2-character
    overlap.
79       */
80      @Test
81      public void testCombinationWashington() {
82          String str1 = "Colum";
83          String str2 = "umbus";
84          int overlap = 2;
85          String combine = StringReassembly.combination(str1, str2,
    overlap);
86          assertEquals("Columbus", combine);
87      }
88
89      /*
90       * Tests of addToSetAvoidingSubstrings method
91       */
92
93      /**
94       * Tests adding "welcome" to set containing substrings.
95       */
96      @Test
97      public void testAddToSetAvoidingSubstrings1() {
98          Set<String> strSet = new Set1L<>();
99          strSet.add("hey");
100         strSet.add("hello");
101         strSet.add("come");
102         String str = "welcome";
103         Set<String> expect = new Set1L<>();
104         expect.add("hey");
105         expect.add("hello");
106         expect.add("welcome");
107         StringReassembly.addToSetAvoidingSubstrings(strSet, str);
108         assertEquals(expect, strSet);
109     }
110
111     /**
112      * Tests adding "fish" to set where it's a substring of
    existing element.
113      */
```

```
114     @Test
115     public void testAddToSetAvoidingSubstrings2() {
116         Set<String> strSet = new Set1L<>();
117         strSet.add("bear");
118         strSet.add("tiger");
119         strSet.add("fish");
120         String str = "fish";
121         Set<String> expect = new Set1L<>();
122         expect.add("bear");
123         expect.add("tiger");
124         expect.add("fish");
125         StringReassembly.addToSetAvoidingSubstrings(strSet, str);
126         assertEquals(expect, strSet);
127     }
128 
129     /*
130      * Tests of printWithLineSeparators method
131      */
132 
133     /**
134      * Tests printing with multiple tildes in middle of string.
135      */
136     @Test
137     public void testPrintWithLineSeparators1() {
138         SimpleWriter out = new SimpleWriter1L("cheer-8-2.txt");
139         SimpleReader in = new SimpleReader1L("cheer-8-2.txt");
140         String text = "Testing 1~2 3 4~hi";
141         String expect = "Testing 1" + "\n" + "2 3 4" + "\n" + "hi";
142         StringReassembly.printWithLineSeparators(text, out);
143         String test = in.nextLine();
144         String test2 = in.nextLine();
145         String test3 = in.nextLine();
146         in.close();
147         out.close();
148         assertEquals(expect, test + "\n" + test2 + "\n" + test3);
149     }
150 
151     /**
152      * Tests printing with single tilde at end of string.
153      */
```

```
154     @Test
155     public void testPrintWithLineSeparators2() {
156         SimpleWriter out = new SimpleWriter1L("cheer-8-2.txt");
157         SimpleReader in = new SimpleReader1L("cheer-8-2.txt");
158         String text = "Testing 1 2 3 4~hi";
159         String expect = "Testing 1 2 3 4" + "\n" + "hi";
160         StringReassembly.printWithLineSeparators(text, out);
161         String test = in.nextLine();
162         String test2 = in.nextLine();
163         in.close();
164         out.close();
165         assertEquals(expect, test + "\n" + test2);
166     }
167 
168     /**
169      * Tests printing with tildes separating complete words.
170      */
171     @Test
172     public void testPrintWithLineSeparators3() {
173         SimpleWriter out = new SimpleWriter1L("cheer-8-2.txt");
174         SimpleReader in = new SimpleReader1L("cheer-8-2.txt");
175         String text = "Here is~The Ohio~State University";
176         String expect = "Here is" + "\n" + "The Ohio" + "\n" +
    "State University";
177         StringReassembly.printWithLineSeparators(text, out);
178         String test = in.nextLine();
179         String test2 = in.nextLine();
180         String test3 = in.nextLine();
181         in.close();
182         out.close();
183         assertEquals(expect, test + "\n" + test2 + "\n" + test3);
184     }
185 
186     /*
187      * Tests of assemble method
188      */
189 
190     /**
191      * Tests assembling overlapping string fragments.
192      */
```

```
193     @Test
194     public void testAssemble1() {
195         Set<String> strSet = new Set1L<>();
196         strSet.add("Wow i");
197         strSet.add("ow it'");
198         strSet.add("it's g");
199         strSet.add("'s good!");
200         Set<String> expect = new Set1L<>();
201         expect.add("Wow it's good!");
202         StringReassembly.assemble(strSet);
203         assertEquals(expect, strSet);
204     }
205 
206     /**
207      * Tests assembling with non-overlapping fragments remaining.
208      */
209     @Test
210     public void testAssemble2() {
211         Set<String> strSet = new Set1L<>();
212         strSet.add("Wow i");
213         strSet.add("ow it'");
214         strSet.add("come");
215         strSet.add("it's g");
216         strSet.add("'s good!");
217         strSet.add("on");
218         Set<String> expect = new Set1L<>();
219         expect.add("Wow it's good!");
220         expect.add("come");
221         expect.add("on");
222         StringReassembly.assemble(strSet);
223         assertEquals(expect, strSet);
224     }
225 }
226 
```