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

import org.junit.Test;

import components.set.Set;
import components.set.Set1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

public class StringReassemblyTest {

    private static Set<String> createFromArgs String... args) {
        Set<String> set = new Set1L<String>();
        for (String s : args) {
            set.add(s);
        }
        return set;
    }

    /**
     * Routine test of combination.
     */
    @Test
    public void Combination1() {
        String str1 = "HelloWorld";
        String str2 = "World";
        int overlap = 5;

        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("HelloWorld", result);
    }

    /**
     * challenging test of combination, long strings
     */
    @Test
    public void Combination2() {
        String str1 = "icantwaituntil";
        String str2 = "untiligraduate";
        int overlap = 5;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("icantwaituntiligraduate", result);
    }

    /**
     * challenge test of combination. both strings are the same
     */
    @Test
    public void Combination3() {
        String str1 = "food";
        String str2 = "food";
        int overlap = 4;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("food", result);
    }

    /**
```

```
58      * border test of combination. both strings are empty.
59      */
60     @Test
61     public void Combination4() {
62         String str1 = "";
63         String str2 = "";
64         int overlap = 0;
65         String result = StringReassembly.combination(str1, str2, overlap);
66         assertEquals("", result);
67     }
68
69     /**
70      * border test of combination. one string is empty.
71      */
72     @Test
73     public void Combination5() {
74         String str1 = "blob";
75         String str2 = "";
76         int overlap = 0;
77         String result = StringReassembly.combination(str1, str2, overlap);
78         assertEquals("blob", result);
79     }
80
81     /**
82      * border test of combination. one character strings.
83      */
84     @Test
85     public void Combination6() {
86         String str1 = "m";
87         String str2 = "e";
88         int overlap = 0;
89         String result = StringReassembly.combination(str1, str2, overlap);
90         assertEquals("me", result);
91     }
92
93     /**
94      * routine test of combination. one overlap
95      */
96     @Test
97     public void Combination7() {
98         String str1 = "mee";
99         String str2 = "ep";
100        int overlap = 1;
101        String result = StringReassembly.combination(str1, str2, overlap);
102        assertEquals("meep", result);
103    }
104
105    @Test
106    // boundary, empty strings
107    public void addToSetAvoidingSubstrings1() {
108        Set<String> set = createFromArgs();
109        String str = "";
110        Set<String> expected = createFromArgs("");
111
112        StringReassembly.addToSetAvoidingSubstrings(set, str);
113
114        assertEquals(expected, set);
```

```
115     }
116
117     @Test
118     // boundary, one empty string one with content
119     public void addToSetAvoidingSubstrings2() {
120         Set<String> set = createFromArgs();
121         String str = "hello world";
122         Set<String> expected = createFromArgs("hello world");
123
124         StringReassembly.addToSetAvoidingSubstrings(set, str);
125         //System.out.print(set);
126         assertEquals(expected, set);
127     }
128
129     @Test
130     // boundary, no overlap and small units
131     public void addToSetAvoidingSubstrings3() {
132         Set<String> set = createFromArgs("i");
133         String str = "s";
134         Set<String> expected = createFromArgs("s", "i");
135
136         StringReassembly.addToSetAvoidingSubstrings(set, str);
137         // System.out.print(set);
138         assertEquals(expected, set);
139     }
140
141     @Test
142     // routine, both have content and overlap
143     public void addToSetAvoidingSubstrings4() {
144         Set<String> set = createFromArgs("he");
145         String str = "ell";
146         Set<String> expected = createFromArgs("ell", "he");
147
148         StringReassembly.addToSetAvoidingSubstrings(set, str);
149         //System.out.print(set);
150         assertEquals(expected, set);
151     }
152
153     @Test
154     // routine, one is a substring
155     public void addToSetAvoidingSubstrings5() {
156         Set<String> set = createFromArgs("i can fly");
157         String str = "fly";
158         Set<String> expected = createFromArgs("i can fly");
159
160         StringReassembly.addToSetAvoidingSubstrings(set, str);
161         //System.out.print(set);
162         assertEquals(expected, set);
163     }
164
165     @Test
166     // boundary, string contains all "~", should end up empty
167     public void printWithLineSeparators1() {
168         SimpleWriter out = new SimpleWriter1L();
169
170         String str = "~~~~~~";
171         StringReassembly.printWithLineSeparators(str, out);
```

```
172
173         //if output is empty, test passed
174     }
175
176     @Test
177     // routine, normal functioning
178     public void printWithLineSeparators2() {
179         SimpleWriter out = new SimpleWriter1L();
180
181         String str = "~bob~";
182         StringReassembly.printWithLineSeparators(str, out);
183         String check = "bob";
184
185         //if output == check, test passed
186     }
187
188     @Test
189     // boundary, empty string
190     public void printWithLineSeparators3() {
191         SimpleWriter out = new SimpleWriter1L();
192
193         String str = "";
194         StringReassembly.printWithLineSeparators(str, out);
195
196         //if output is empty, test passed
197     }
198
199     @Test
200     // challenging, string contains all "~", long string, should end up empty
201     public void printWithLineSeparators4() {
202         SimpleWriter out = new SimpleWriter1L();
203
204         String str = "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~";
205         StringReassembly.printWithLineSeparators(str, out);
206
207         //if output is empty, test passed
208     }
209 }
```