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

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; }
     */

    @Test
    // boundary, no overlap
    public void overlap2() {
        String str1 = "hello ";
        String str2 = "world";
        int over = StringReassembly.overlap(str1, str2);

        assertEquals(0, over);
    }

    @Test
    // boundary, one overlap
    public void overlap3() {
        String str1 = "hello w";
        String str2 = "world";
        int over = StringReassembly.overlap(str1, str2);

        assertEquals(1, over);
    }

    @Test
    // routine, large overlap
    public void overlap4() {
        /*
         * Set up variables and call method under test
         */
        String str1 = "hello world";
        String str2 = "world here I come";
        int over = StringReassembly.overlap(str1, str2);
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(5, over);
    }

    /**
     * Routine test of combination.
     */
    @Test
    public void testCombination1() {
        String str1 = "HelloWorld";
        String str2 = "World";
        int overlap = 5;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("HelloWorld", result);
    }

```

```
63      /**
64       * Routine test of combination.
65       */
66      @Test
67      public void testCombination2() {
68          String str1 = "icantwaituntil";
69          String str2 = "untiligraduate";
70          int overlap = 5;
71          String result = StringReassembly.combination(str1, str2, overlap);
72          assertEquals("icantwaituntiligraduate", result);
73      }
74
75      /**
76       * border test of combination. both strings are the same
77       */
78      @Test
79      public void testCombination3() {
80          String str1 = "food";
81          String str2 = "food";
82          int overlap = 4;
83          String result = StringReassembly.combination(str1, str2, overlap);
84          assertEquals("food", result);
85      }
86
87      /**
88       * border test of combination. both strings are empty.
89       */
90      @Test
91      public void testCombination4() {
92          String str1 = "";
93          String str2 = "";
94          int overlap = 0;
95          String result = StringReassembly.combination(str1, str2, overlap);
96          assertEquals("", result);
97      }
98
99      /**
100      * Routine test of addToSetAvoidingSubstrings.
101      */
102     @Test
103     public void testaddToSetAvoidingSubstrings1() {
104         Set<String> set = new Set1L<>();
105         String check = "";
106
107         String str = "";
108         StringReassembly.addToSetAvoidingSubstrings(set, str);
109         check = set.removeAny();
110
111         assertEquals(str, check);
112     }
113
114     @Test
115     // boundary, should do nothing
116     public void addToSetAvoidingSubstrings1() {
117         Set<String> set = new Set1L<>();
118         String check = "";
119
```

```
120         String str = "";
121         StringReassembly.addToSetAvoidingSubstrings(set, str);
122         check = set.removeAny();
123 
124         assertEquals(str, check);
125     }
126 
127     @Test
128     // routine, should add to the set
129     public void addToSetAvoidingSubstrings2() {
130         Set<String> set = new Set1L<>();
131         String check = "";
132 
133         String str = "hello world";
134         StringReassembly.addToSetAvoidingSubstrings(set, str);
135         check = set.removeAny();
136 
137         assertEquals(str, check);
138     }
139 }
```