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

import org.junit.AfterClass;
import org.junit.Test;

import components.set.Set;
import components.set.Set2;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

public class StringReassemblyTest {

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    private static final int Olap = 3;
    private static final int Olap2 = 5;
    private static final int FormyLoop = 100000;
    private static final int LowerBound = 500;
    private static final int UpperBound = 2000;

    @Test
    public void testCombinationTypicalCase() { // routine
        String str1 = "concat";
        String str2 = "cater";
        int overlap = Olap;
        String expected = "concater";
        String result = StringReassembly.combination(str1,
  str2, overlap);
        assertEquals(expected, result);
    }

    @Test
    public void testCombinationEmptyStrings() { //edge
        String str1 = "";
```

```
39          String str2 = "";
40          int overlap = 0;
41          String expected = "";
42          String result = StringReassembly.combination(str1,
    str2, overlap);
43          assertEquals(expected, result);
44      }
45
46      @Test
47      public void testCombinationLargeOverlap() { //challenge
48          String str1 = "concater";
49          String str2 = "cater";
50          int overlap = Olap2; // overlap is the length of
    "cater"
51          String expected = "concater"; // Expecting str1 since
    it contains str2
52          String result = StringReassembly.combination(str1,
    str2, overlap);
53          assertEquals(expected, result);
54      }
55
56      @Test
57      public void testAddToSetRoutine() { //routine
58          Set<String> strSet = new Set2<>(); // Use the custom
    Set implementation
59          String str = "hello";
60          String expectedSet = "{hello}"; // Adjusted expected
    result
61
62          StringReassembly.addToSetAvoidingSubstrings(strSet,
    str);
63
64          assertEquals(expectedSet, strSet.toString());
65      }
66
67      @Test
68      public void testAddToSetEdge() { //edge
69          Set<String> strSet = new Set2<>();
70          strSet.add("hello");
```

```
 71         String str = "hell";
 72         String expectedSet = "{hello}";
 73
 74         StringReassembly.addToSetAvoidingSubstrings(strSet,
    str);
 75
 76         assertEquals(expectedSet, strSet.toString());
 77     }
 78
 79     @Test
 80     public void testAddToSetChallenge() { //challenge
 81         Set<String> strSet = new Set2<>();
 82         strSet.add("apple");
 83         strSet.add("banana");
 84
 85         String str = "pineapple";
 86         Set<String> expectedSet = new Set2<>();
 87         expectedSet.add("apple");
 88         expectedSet.add("banana");
 89         expectedSet.add("pineapple");
 90
 91         StringReassembly.addToSetAvoidingSubstrings(strSet,
    str);
 92
 93         assertEquals(expectedSet, strSet);
 94     }
 95
 96     @Test
 97     public void testLinesFromInputRoutine() {
 98         // Routine case: Test with two-line input
 99         SimpleReader input = new SimpleReader1L("data/
    cheer-8-2.txt");
100         Set<String> result =
    StringReassembly.linesFromInput(input);
101         input.close();
102
103         Set<String> expected = new Set2<>();
104         expected.add("Bucks -- Beat");
105         expected.add("Go Bucks");
```

```
106
107         assertEquals(expected, result);
108     }
109
110     @Test
111     public void testLinesFromInputEdge() { //edge
112         // Edge case: Test with an empty input file
113         SimpleReader in = new SimpleReader1L("data/
    gettysburg-30-4.txt");
114         Set<String> result =
    StringReassembly.linesFromInput(in);
115         in.close();
116         assertTrue("Expected an empty set for empty input
    file",
117                 result.size() == 0);
118     }
119
120     @Test
121     public void testLinesFromInputLargeFile() { //challenge
122         // Challenge test: Test with a large input file
123         SimpleReader in = new SimpleReader1L("data/
    declaration-50-8.txt");
124         Set<String> result =
    StringReassembly.linesFromInput(in);
125         in.close();
126
127         // check if within acceptable range
128         assertTrue(
129                 "Expected number of unique lines to be within
    range of large file",
130                 result.size() >= LowerBound && result.size() <=
    UpperBound);
131     }
132
133     @Test
134     public void testPrintWithLineSeparatorsRoutine() {
135         // Routine: Test with a string containing only '~'
    characters
136         String allTildes = "~~~~~";
```

```
137         SimpleWriter outRoutine = new SimpleWriter1L();
138         StringReassembly.printWithLineSeparators(allTildes,
    outRoutine);
139         outRoutine.close();
140     }
141 
142     @Test
143     public void testPrintWithLineSeparatorsEdgeCase() {
144         // Edge case: Test with an empty string
145         String emptyInput = "";
146         SimpleWriter outEdge = new SimpleWriter1L();
147         StringReassembly.printWithLineSeparators(emptyInput,
    outEdge);
148         outEdge.close();
149     }
150 
151     @Test
152     public void testPrintWithLineSeparatorsChallenge() {
153         // Challenge: Test with a very large input string
154         StringBuilder largeInput = new StringBuilder();
155         for (int i = 0; i < FormyLoop; i++) {
156             largeInput.append("a~");
157         }
158         SimpleWriter outChallenge = new SimpleWriter1L();
159 
    StringReassembly.printWithLineSeparators(largeInput.toString(),
160                 outChallenge);
161         outChallenge.close();
162     }
163 
164 }
165 
```