```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);
    }

    /**
```

```java
     * border test of combination. both strings are empty.
     */
    @Test
    public void Combination4() {
        String str1 = "";
        String str2 = "";
        int overlap = 0;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("", result);
    }

    /**
     * border test of combination. one string is empty.
     */
    @Test
    public void Combination5() {
        String str1 = "blob";
        String str2 = "";
        int overlap = 0;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("blob", result);
    }

    /**
     * border test of combination. one character strings.
     */
    @Test
    public void Combination6() {
        String str1 = "m";
        String str2 = "e";
        int overlap = 0;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("me", result);
    }

    /**
     * routine test of combination. one overlap
     */
    @Test
    public void Combination7() {
        String str1 = "mee";
        String str2 = "ep";
        int overlap = 1;
        String result = StringReassembly.combination(str1, str2, overlap);
        assertEquals("meep", result);
    }

    @Test
    // boundary, empty strings
    public void addToSetAvoidingSubstrings1() {
        Set<String> set = createFromArgs();
        String str = "";
        Set<String> expected = createFromArgs("");

        StringReassembly.addToSetAvoidingSubstrings(set, str);

        assertEquals(expected, set);
    }

    @Test
    // boundary, one empty string one with content
    public void addToSetAvoidingSubstrings2() {
        Set<String> set = createFromArgs();
        String str = "hello world";
        Set<String> expected = createFromArgs("hello world");

        StringReassembly.addToSetAvoidingSubstrings(set, str);
        //System.out.print(set);
        assertEquals(expected, set);
    }

    @Test
    // boundary, no overlap and small units
    public void addToSetAvoidingSubstrings3() {
        Set<String> set = createFromArgs("i");
        String str = "s";
        Set<String> expected = createFromArgs("s", "i");

        StringReassembly.addToSetAvoidingSubstrings(set, str);
        // System.out.print(set);
        assertEquals(expected, set);
    }

    @Test
    // routine, both have content and overlap
    public void addToSetAvoidingSubstrings4() {
        Set<String> set = createFromArgs("he");
        String str = "ell";
        Set<String> expected = createFromArgs("ell", "he");

        StringReassembly.addToSetAvoidingSubstrings(set, str);
        //System.out.print(set);
        assertEquals(expected, set);
    }

    @Test
    // routine, one is a substring
    public void addToSetAvoidingSubstrings5() {
        Set<String> set = createFromArgs("i can fly");
        String str = "fly";
        Set<String> expected = createFromArgs("i can fly");

        StringReassembly.addToSetAvoidingSubstrings(set, str);
        //System.out.print(set);
        assertEquals(expected, set);
    }

    @Test
    // boundary, string contains all "~", should end up empty
    public void printWithLineSeparators1() {
        SimpleWriter out = new SimpleWriter1L();

        String str = "~~~~~~";
        StringReassembly.printWithLineSeparators(str, out);
```

```java

        //if output is empty, test passed
    }

    @Test
    // routine, normal functioning
    public void printWithLineSeparators2() {
        SimpleWriter out = new SimpleWriter1L();

        String str = "~bob~";
        StringReassembly.printWithLineSeparators(str, out);
        String check = "bob";

        //if output == check, test passed
    }

    @Test
    // boundary, empty string
    public void printWithLineSeparators3() {
        SimpleWriter out = new SimpleWriter1L();

        String str = "";
        StringReassembly.printWithLineSeparators(str, out);

        //if output is empty, test passed
    }

    @Test
    // challenging, string contains all "~", long string, should end up empty
    public void printWithLineSeparators4() {
        SimpleWriter out = new SimpleWriter1L();

        String str = "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~";
        StringReassembly.printWithLineSeparators(str, out);

        //if output is empty, test passed
    }
}
```