

```
1 import static org.junit.Assert.assertEquals;
2
3 import org.junit.Test;
4
5 import components.stack.Stack;
6
7 /**
8  * JUnit test fixture for {@code Stack<String>}s constructor and
9  * kernel
10 * methods.
11 *
12 * @author Put your name here
13 */
14 public abstract class StackTest {
15
16     /**
17      * Invokes the appropriate {@code Stack} constructor for the
18      * implementation
19      * under test and returns the result.
20      *
21      * @return the new stack
22      * @ensures constructorTest = <>
23      */
24     protected abstract Stack<String> constructorTest();
25
26     /**
27      * Invokes the appropriate {@code Stack} constructor for the
28      * reference
29      * implementation and returns the result.
30      *
31      * @return the new stack
32      * @ensures constructorRef = <>
33      */
34     protected abstract Stack<String> constructorRef();
35
36     /**
37      * Creates and returns a {@code Stack<String>} of the
38      * implementation under
39      * test type with the given entries.
40      *
41      * @param args
42      *         the entries for the stack
```

```
41     * @return the constructed stack
42     * @ensures createFromArgsTest = [entries in args]
43     */
44     private Stack<String> createFromArgsTest(String... args) {
45         Stack<String> stack = this.constructorTest();
46         for (String s : args) {
47             stack.push(s);
48         }
49         stack.flip();
50         return stack;
51     }
52
53     /**
54     *
55     * Creates and returns a {@code Stack<String>} of the
reference
56     * implementation type with the given entries.
57     *
58     * @param args
59     *         the entries for the stack
60     * @return the constructed stack
61     * @ensures createFromArgsRef = [entries in args]
62     */
63     private Stack<String> createFromArgsRef(String... args) {
64         Stack<String> stack = this.constructorRef();
65         for (String s : args) {
66             stack.push(s);
67         }
68         stack.flip();
69         return stack;
70     }
71
72     @Test
73     public final void testDefaultConstructor() {
74         Stack<String> s = this.constructorTest();
75         Stack<String> sExpected = this.constructorRef();
76         assertEquals(sExpected, s);
77     }
78
79     @Test
80     public final void push1() {
81         Stack<String> s = this.createFromArgsTest();
82         Stack<String> sExpected = this.createFromArgsRef("Hello");
83     }
```

```
84         s.push("Hello");
85
86         assertEquals(sExpected, s);
87     }
88
89     @Test
90     public final void push2() {
91         Stack<String> s = this.createFromArgsTest();
92         Stack<String> sExpected = this.createFromArgsRef("Hello",
93 "Bye");
94         s.push("Hello");
95         s.push("Bye");
96
97         assertEquals(sExpected, s);
98     }
99
100    @Test
101    public final void push3() {
102        Stack<String> s = this.createFromArgsTest();
103        Stack<String> sExpected = this.createFromArgsRef("Hello",
104 "Hello");
105        s.push("Hello");
106        s.push("Hello");
107
108        assertEquals(sExpected, s);
109    }
110
111    @Test
112    public final void pop1() {
113        Stack<String> s = this.createFromArgsTest("Hello");
114        Stack<String> sExpected = this.createFromArgsRef();
115
116        String ans = s.pop();
117
118        assertEquals(sExpected, s);
119        assertEquals("Hello", ans);
120    }
121
122    @Test
123    public final void pop2() {
124        Stack<String> s = this.createFromArgsTest("Hello", "Bye");
125        Stack<String> sExpected = this.createFromArgsRef();
```

```
126
127     String ans = s.pop();
128
129     assertEquals(sExpected, s);
130     assertEquals("Hello", ans);
131 }
132
133 @Test
134 public final void length() {
135     Stack<String> s = this.createFromArgsTest("Hello");
136     Stack<String> sExpected = this.createFromArgsRef("Hello");
137
138     assertEquals(sExpected, s);
139     assertEquals(sExpected.length(), s.length());
140 }
141
142 }
```