

```
1 import static org.junit.Assert.assertEquals;
2
3 /**
4  * JUnit test fixture for {@code Stack<String>}s constructor and kernel
5  * methods.
6  *
7  * @author Put your name here
8  *
9  */
10 public abstract class StackTest {
11     /**
12      * Invokes the appropriate {@code Stack} constructor for the implementation
13      * under test and returns the result.
14      *
15      * @return the new stack
16      * @ensures constructorTest = <>
17      */
18     protected abstract Stack<String> constructorTest();
19
20     /**
21      * Invokes the appropriate {@code Stack} constructor for the reference
22      * implementation and returns the result.
23      *
24      * @return the new stack
25      * @ensures constructorRef = <>
26      */
27     protected abstract Stack<String> constructorRef();
28
29     /**
30      * Creates and returns a {@code Stack<String>} of the implementation under
31      * test type with the given entries.
32      *
33      * @param args
34      *         the entries for the stack
35      * @return the constructed stack
36      * @ensures createFromArgsTest = [entries in args]
37      */
38     private Stack<String> createFromArgsTest(String... args) {
39         Stack<String> stack = this.constructorTest();
40         for (String s : args) {
41             stack.push(s);
42         }
43         stack.flip();
44         return stack;
45     }
46
47     /**
48      * Creates and returns a {@code Stack<String>} of the reference
49      * implementation type with the given entries.
50      *
51      * @param args
52      *         the entries for the stack
53      * @return the constructed stack
54      * @ensures createFromArgsRef = [entries in args]
55     */
56 }
```

```
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     // TODO - add test cases for constructor, push, pop, and length
73
74     @Test
75     public void pushTest1() {
76         Stack<Object> test = new Stack2<>();
77         Stack<Object> ref = new Stack2<>();
78
79         test.push(4);
80         test.push(5);
81         test.push(6);
82         ref.push(4);
83         ref.push(5);
84         ref.push(6);
85
86         assertEquals(ref.top(), test.top());
87     }
88
89     public void pushTest2() {
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111     @Test
112     public void pushTest3() {
113         Stack<Object> test = new Stack2<>();
114         Stack<Object> ref = new Stack2<>();
115
116         test.push(9);
117         ref.push(9);
118
119         assertEquals(ref.top(), test.top());
120     }
121
122     public void popTest1() {
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143     @Test
144     public void popTest2() {
145         Stack<Object> test = new Stack2<>();
146         Stack<Object> ref = new Stack2<>();
147
148         test.push(4);
149         test.push(5);
150         test.push(6);
151         test.push(4);
152         test.push(5);
153         test.push(6);
154         ref.push(4);
155         ref.push(5);
156         ref.push(6);
157         ref.push(4);
```

```
158         ref.push(5);
159         ref.push(6);
160
161         assertEquals(ref.pop(), test.pop());
162         assertEquals(ref.top(), test.top());
163
164         assertEquals(ref.pop(), test.pop());
165         assertEquals(ref.top(), test.top());
166
167         assertEquals(ref.pop(), test.pop());
168         assertEquals(ref.top(), test.top());
169
170         assertEquals(ref.pop(), test.pop());
171         assertEquals(ref.top(), test.top());
172
173         assertEquals(ref.pop(), test.pop());
174         assertEquals(ref.top(), test.top());
175
176         assertEquals(ref.pop(), test.pop());
177     }
178
179     @Test
180     public void pushPopTest1() {
181         Stack<Object> test = new Stack2<>();
182         Stack<Object> ref = new Stack2<>();
183
184         test.push(1);
185         ref.push(1);
186
187         assertEquals(ref.top(), test.top());
188         assertEquals(ref.pop(), test.pop());
189
190         test.push(2);
191         ref.push(2);
192         test.push(3);
193         ref.push(3);
194
195         assertEquals(ref.top(), test.top());
196         assertEquals(ref.pop(), test.pop());
197
198         assertEquals(ref.top(), test.top());
199         assertEquals(ref.pop(), test.pop());
200
201         test.push(0);
202         ref.push(0);
203
204         assertEquals(ref.top(), test.top());
205         assertEquals(ref.pop(), test.pop());
206     }
207
208
209
210     @Test
211     public void lengthTest() {
212         Stack<Object> test = new Stack2<>();
213         Stack<Object> ref = new Stack2<>();
214
```

```
215         test.push(4);
216         test.push(5);
217         test.push(6);
218         ref.push(4);
219         ref.push(5);
220         ref.push(6);
221
222         assertEquals(ref.length(), test.length());
223     }
224 }
225
```