```
 1 import static org.junit.Assert.assertEquals;
 8
 9 /**
10  * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
11  * kernel methods.
12  *
13  * @author Put your name here
14  *
15  */
16 public abstract class SortingMachineTest {
17
18     /**
19      * Invokes the appropriate {@code SortingMachine} constructor for the
20      * implementation under test and returns the result.
21      *
22      * @param order
23      *            the {@code Comparator} defining the order for {@code String}
24      * @return the new {@code SortingMachine}
25      * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
26      * @ensures constructorTest = (true, order, {})
27      */
28     protected abstract SortingMachine<String> constructorTest(
29             Comparator<String> order);
30
31     /**
32      * Invokes the appropriate {@code SortingMachine} constructor for the
33      * reference implementation and returns the result.
34      *
35      * @param order
36      *            the {@code Comparator} defining the order for {@code String}
37      * @return the new {@code SortingMachine}
38      * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
39      * @ensures constructorRef = (true, order, {})
40      */
41     protected abstract SortingMachine<String> constructorRef(
42             Comparator<String> order);
43
44     /**
45      *
46      * Creates and returns a {@code SortingMachine<String>} of the
47      * implementation under test type with the given entries and mode.
48      *
49      * @param order
50      *            the {@code Comparator} defining the order for {@code String}
51      * @param insertionMode
52      *            flag indicating the machine mode
53      * @param args
54      *            the entries for the {@code SortingMachine}
55      * @return the constructed {@code SortingMachine}
56      * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
57      * @ensures <pre>
58      * createFromArgsTest = (insertionMode, order, [multiset of entries in args])
59      * </pre>
60      */
61     private SortingMachine<String> createFromArgsTest(Comparator<String> order,
62             boolean insertionMode, String... args) {
63         SortingMachine<String> sm = this.constructorTest(order);
```

```
64          for (int i = 0; i < args.length; i++) {
65              sm.add(args[i]);
66          }
67          if (!insertionMode) {
68              sm.changeToExtractionMode();
69          }
70          return sm;
71      }
72
73      /**
74       *
75       * Creates and returns a {@code SortingMachine<String>} of the reference
76       * implementation type with the given entries and mode.
77       *
78       * @param order
79       *            the {@code Comparator} defining the order for {@code String}
80       * @param insertionMode
81       *            flag indicating the machine mode
82       * @param args
83       *            the entries for the {@code SortingMachine}
84       * @return the constructed {@code SortingMachine}
85       * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
86       * @ensures <pre>
87       * createFromArgsRef = (insertionMode, order, [multiset of entries in args])
88       * </pre>
89       */
90      private SortingMachine<String> createFromArgsRef(Comparator<String> order,
91              boolean insertionMode, String... args) {
92          SortingMachine<String> sm = this.constructorRef(order);
93          for (int i = 0; i < args.length; i++) {
94              sm.add(args[i]);
95          }
96          if (!insertionMode) {
97              sm.changeToExtractionMode();
98          }
99          return sm;
100     }
101
102     /**
103      * Comparator<String> implementation to be used in all test cases. Compare
104      * {@code String}s in lexicographic order.
105      */
106     private static class StringLT implements Comparator<String> {
107
108         @Override
109         public int compare(String s1, String s2) {
110             return s1.compareToIgnoreCase(s2);
111         }
112
113     }
114
115     /**
116      * Comparator instance to be used in all test cases.
117      */
118     private static final StringLT ORDER = new StringLT();
119
120     /*
```

```
121      * Sample test cases.
122      */
123
124     @Test
125     public final void testConstructor() {
126         SortingMachine<String> m = this.constructorTest(ORDER);
127         SortingMachine<String> mExpected = this.constructorRef(ORDER);
128         assertEquals(mExpected, m);
129     }
130
131     /*
132      * Add test cases
133      */
134
135     @Test
136     public final void testAddEmpty() {
137         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
138         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
139                 "green");
140         m.add("green");
141         assertEquals(mExpected, m);
142     }
143
144     @Test
145     public final void testAddNonempty() {
146         //setup
147         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "green");
148         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true, "green",
    "blue");
149         //call
150         m.add("blue");
151         //eval
152         assertEquals(mExpected, m);
153     }
154
155     /*
156      * Extraction mode test cases
157      */
158
159     @Test
160     public final void testExtractionMode() {
161         //setup
162         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
163         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
164         //call
165         m.changeToExtractionMode();
166         //eval
167         assertEquals(mExpected, m);
168     }
169
170     /*
171      * Remove first test cases
172      */
173
174     @Test
175     public final void testRemoveFirstEmpty() {
176         //setup
```

```
177         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "green");
178         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
179         //call
180         m.removeFirst();
181         //eval
182         assertEquals(mExpected, m);
183     }
184
185     @Test
186     public final void testRemoveFirst() {
187         //setup
188         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "green",
189                 "blue");
190         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
191                 "blue");
192         //call
193         m.removeFirst();
194         //eval
195         assertEquals(mExpected, m);
196     }
197
198     /*
199      * Insertion mode test cases
200      */
201
202     @Test
203     public final void testIsInsertTrue() {
204         //setup
205         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
206         boolean mExpected = true;
207         //call
208         boolean mActual = m.isInInsertionMode();
209         //eval
210         assertEquals(mExpected, mActual);
211     }
212
213     @Test
214     public final void testIsInsertFalse() {
215         //setup
216         SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
217         boolean mExpected = false;
218         //call
219         boolean mActual = m.isInInsertionMode();
220         //eval
221         assertEquals(mExpected, mActual);
222     }
223
224     /*
225      * Order test cases
226      */
227
228     @Test
229     public final void testOrder() {
230         //setup
231         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
232         Comparator<String> mExpected = ORDER;
233         //call
```

```
234         Comparator<String> mActual = m.order();
235         //eval
236         assertEquals(mExpected, mActual);
237     }
238 
239     /*
240      * Size test cases
241      */
242 
243     @Test
244     public final void testSizeInsertionMode() {
245         //setup
246         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "green");
247         int mExpected = 1;
248         //call
249         int mActual = m.size();
250         //eval
251         assertEquals(mExpected, mActual);
252     }
253 
254     @Test
255     public final void testSizeExtractionMode() {
256         //setup
257         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "green");
258         int mExpected = 1;
259         //call
260         int mActual = m.size();
261         //eval
262         assertEquals(mExpected, mActual);
263     }
264 
265 
266     // TODO - add test cases for, order, and size
267 
268 
269 
270 }
271 
```