```
 1 import static org.junit.Assert.assertEquals;
 2
 3 import java.util.Comparator;
 4
 5 import org.junit.Test;
 6
 7 import components.sortingmachine.SortingMachine;
 8
 9 /**
10  * JUnit test fixture for {@code SortingMachine<String>}'s
   constructor and
11  * kernel methods.
12  *
13  * @author Daniil Gofman, Ansh Pachauri
14  *
15  */
16 public abstract class SortingMachineTest {
17
18     /**
19      * Invokes the appropriate {@code SortingMachine}
   constructor for the
20      * implesEntation under test and returns the result.
21      *
22      *
23      * @param order
24      *            the {@code Comparator} defining the order for
   {@code String}
25      * @return the new {@code SortingMachine}
26      * @requires IS_TOTAL_PREORDER([relation computed by
   order.compare method])
27      * @ensures constructorTest = (true, order, {})
28      */
29     protected abstract SortingMachine<String> constructorTest(
30             Comparator<String> order);
31
32     /**
33      * Invokes the appropriate {@code SortingMachine}
   constructor for the
34      * reference implementation and returns the result.
```

```
35      *
36      * @param order
37      *            the {@code Comparator} defining the order for
    {@code String}
38      * @return the new {@code SortingMachine}
39      * @requires IS_TOTAL_PREORDER([relation computed by
    order.compare method])
40      * @ensures constructorRef = (true, order, {})
41      */
42     protected abstract SortingMachine<String> constructorRef(
43             Comparator<String> order);
44
45     /**
46      *
47      * Creates and returns a {@code SortingMachine<String>} of
    the
48      * implementation under test type with the given entries
    and mode.
49      *
50      * @param order
51      *            the {@code Comparator} defining the order for
    {@code String}
52      * @param insertionMode
53      *            flag indicating the machine mode
54      * @param args
55      *            the entries for the {@code SortingMachine}
56      * @return the constructed {@code SortingMachine}
57      * @requires IS_TOTAL_PREORDER([relation computed by
    order.compare method])
58      * @ensures <pre>
59      * createFromArgsTest = (insertionMode, order, [multiset of
    entries in args])
60      * </pre>
61      */
62     private SortingMachine<String>
    createFromArgsTest(Comparator<String> order,
63             boolean insertionMode, String... args) {
64         SortingMachine<String> sm =
    this.constructorTest(order);
```

```
65          for (int i = 0; i < args.length; i++) {
66              sm.add(args[i]);
67          }
68          if (!insertionMode) {
69              sm.changeToExtractionMode();
70          }
71          return sm;
72      }
73
74      /**
75       *
76       * Creates and returns a {@code SortingMachine<String>} of
   the reference
77       * implementation type with the given entries and mode.
78       *
79       * @param order
80       *            the {@code Comparator} defining the order for
   {@code String}
81       * @param insertionMode
82       *            flag indicating the machine mode
83       * @param args
84       *            the entries for the {@code SortingMachine}
85       * @return the constructed {@code SortingMachine}
86       * @requires IS_TOTAL_PREORDER([relation computed by
   order.compare method])
87       * @ensures <pre>
88       * createFromArgsRef = (insertionMode, order, [multiset of
   entries in args])
89       * </pre>
90       */
91      private SortingMachine<String>
   createFromArgsRef(Comparator<String> order,
92              boolean insertionMode, String... args) {
93          SortingMachine<String> sm = this.constructorRef(order);
94          for (int i = 0; i < args.length; i++) {
95              sm.add(args[i]);
96          }
97          if (!insertionMode) {
98              sm.changeToExtractionMode();
```

```
 99         }
100         return sm;
101     }
102 
103     /**
104      * Comparator<String> implementation to be used in all test
    cases. Compare
105      * {@code String}s in lexicographic order.
106      */
107     private static class StringLT implements Comparator<String>
    {
108 
109         @Override
110         public int compare(String s1, String s2) {
111             return s1.compareToIgnoreCase(s2);
112         }
113 
114     }
115 
116     /**
117      * Comparator instance to be used in all test cases.
118      */
119     private static final StringLT ORDER = new StringLT();
120 
121     /**
122      * Test empty constructor.
123      */
124     @Test
125     public final void testConstructor() {
126         SortingMachine<String> s = this.constructorTest(ORDER);
127         SortingMachine<String> sExpected =
    this.constructorRef(ORDER);
128         assertEquals(sExpected, s);
129     }
130 
131     /**
132      * Test add boundary case.
133      */
134     @Test
```

```
135     public final void testAddEmpty() {
136         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true);
137         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, true,
138                 "apple");
139         s.add("apple");
140         assertEquals(sExpected, s);
141     }
142 
143     /**
144      * Test add routine case.
145      */
146     @Test
147     public final void testAddRoutine1() {
148         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true,
149                 "apples");
150         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, true,
151                 "apples", "avacado");
152         s.add("avacado");
153         assertEquals(sExpected, s);
154     }
155 
156     /**
157      * Test add routine case.
158      */
159     @Test
160     public final void testAddRoutine2() {
161         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true,
162                 "Apples", "Avacado");
163         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, true,
164                 "Apples", "Avacado", "Orange");
165         s.add("Orange");
166         assertEquals(sExpected, s);
167     }
```

```
168
169     /**
170      * Test changeToExtractionMode Empty case.
171      */
172     @Test
173     public final void testchangeToExtractionModeEmpty() {
174         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true);
175         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false);
176         s.changeToExtractionMode();
177         assertEquals(sExpected, s);
178     }
179
180     /**
181      * Test changeToExtractionMode routine case.
182      */
183     @Test
184     public final void testchangeToExtractionModeRoutine1() {
185         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true,
186                 "Apples", "Avacado");
187         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false,
188                 "Apples", "Avacado");
189         s.changeToExtractionMode();
190         assertEquals(sExpected, s);
191     }
192
193     /**
194      * Test changeToExtractionMode routine case.
195      */
196     @Test
197     public final void testchangeToExtractionModeRoutine2() {
198         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true,
199                 "Apples", "Avacado", "Orange");
200         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false,
```

```
201                 "Apples", "Avacado", "Orange");
202         s.changeToExtractionMode();
203         assertEquals(sExpected, s);
204     }
205 
206     /**
207      * Test removeFirst Empty test case.
208      */
209     @Test
210     public final void testRemoveFirstEmpty() {
211         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, false,
212                 "Apples");
213         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false);
214         String expectedVal = "Apples";
215         String val = s.removeFirst();
216         assertEquals(expectedVal, val);
217         assertEquals(sExpected, s);
218     }
219 
220     /**
221      * Test removeFirst routine test case.
222      */
223     @Test
224     public final void testRemoveFirstRoutine() {
225         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, false,
226                 "Apples", "Avacado", "Orange");
227         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false,
228                 "Avacado", "Orange");
229         String expectedVal = "Apples";
230         String val = s.removeFirst();
231         assertEquals(expectedVal, val);
232         assertEquals(sExpected, s);
233     }
234 
235     /**
```

```
236     * Test isInInsertionMode routine case.
237     */
238    @Test
239    public final void testIsInInsertionModeRoutine1() {
240        SortingMachine<String> m =
   this.createFromArgsTest(ORDER, true, "apple",
241                "banana");
242
243        boolean insertionMode = m.isInInsertionMode();
244        assertEquals(true, insertionMode);
245    }
246
247    /**
248     * Test isInInsertionMode routine case.
249     */
250    @Test
251    public final void testIsInInsertionModeRotuine2() {
252        SortingMachine<String> m =
   this.createFromArgsTest(ORDER, false,
253                "apple", "banana");
254
255        boolean insertionMode = m.isInInsertionMode();
256        assertEquals(false, insertionMode);
257    }
258
259    /**
260     * Test isInInsertionMode Empty case.
261     */
262    @Test
263    public final void testIsInInsertionModeEmpty1() {
264        SortingMachine<String> m =
   this.createFromArgsTest(ORDER, true);
265
266        boolean insertionMode = m.isInInsertionMode();
267        assertEquals(true, insertionMode);
268    }
269
270    /**
271     * Test isInInsertionMode Empty case.
```

```
272     */
273    @Test
274    public final void testIsInInsertionModeEmpty2() {
275        SortingMachine<String> m =
   this.createFromArgsTest(ORDER, false);
276
277        boolean insertionMode = m.isInInsertionMode();
278        assertEquals(false, insertionMode);
279    }
280
281    /**
282     * Test order routine test case.
283     */
284    @Test
285    public final void testOrderRoutine() {
286        SortingMachine<String> s =
   this.createFromArgsTest(ORDER, false,
287                "Apples", "Avacado", "Orange");
288        SortingMachine<String> sExpected =
   this.createFromArgsRef(ORDER, false,
289                "Apples", "Avacado", "Orange");
290        Comparator<String> orderTest = s.order();
291        Comparator<String> expectedOrderTest = ORDER;
292        assertEquals(expectedOrderTest, orderTest);
293        assertEquals(sExpected, s);
294    }
295
296    /**
297     * Test order empty test case.
298     */
299    @Test
300    public final void testOrderEmpty() {
301        SortingMachine<String> s =
   this.createFromArgsTest(ORDER, false);
302        SortingMachine<String> sExpected =
   this.createFromArgsRef(ORDER, false);
303        Comparator<String> orderTest = s.order();
304        Comparator<String> expectedOrderTest = ORDER;
305        assertEquals(expectedOrderTest, orderTest);
```

```
306         assertEquals(sExpected, s);
307     }
308 
309     /**
310      * Test size empty case in extraction mode.
311      */
312     @Test
313     public final void testSizeEmpty1() {
314         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, false);
315         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false);
316         assertEquals(0, s.size());
317         assertEquals(sExpected, s);
318     }
319 
320     /**
321      * Test size empty case in insertion mode.
322      */
323     @Test
324     public final void testSizeEmpty2() {
325         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true);
326         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, true);
327         assertEquals(0, s.size());
328         assertEquals(sExpected, s);
329     }
330 
331     /**
332      * Test size routine test case in extraction mode.
333      */
334     @Test
335     public final void testSizeNotEmpty1() {
336         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, false,
337                 "Apples", "Avacado", "Orange");
338         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, false,
```

```
339                 "Apples", "Avacado", "Orange");
340         assertEquals(3, s.size());
341         assertEquals(sExpected, s);
342     }
343 
344     /**
345      * Test size routine test case in extraction mode.
346      */
347     @Test
348     public final void testSizeNotEmpty2() {
349         SortingMachine<String> s =
    this.createFromArgsTest(ORDER, true,
350                 "Apples", "Avacado", "Orange");
351         SortingMachine<String> sExpected =
    this.createFromArgsRef(ORDER, true,
352                 "Apples", "Avacado", "Orange");
353         assertEquals(3, s.size());
354         assertEquals(sExpected, s);
355     }
356 }
357 
```