```
 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     @Test
132     public final void testAddEmpty() {
133         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
134         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
135                 "green");
136         m.add("green");
137         assertEquals(mExpected, m);
138     }
139
140     // TODO - add test cases for add, changeToExtractionMode, removeFirst,
141     // isInInsertionMode, order, and size
142
143     //Test for the insertion mode, false
144     @Test
145     public final void testInsertionModeFalse() {
146         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "cow");
147         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
148                 "cow");
149
150         boolean modeexpected = mexpected.isInInsertionMode();
151         boolean mode = m.isInInsertionMode();
152
153         assertEquals(modeexpected, mode);
154         assertEquals(m, mexpected);
155     }
156
157     //Test for the insertion mode, true
158     @Test
159     public final void testInsertionModeTrue() {
160         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "cow");
161         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, true,
162                 "cow");
163
164         boolean modeexpected = mexpected.isInInsertionMode();
165         boolean mode = m.isInInsertionMode();
166
167         assertEquals(modeexpected, mode);
168
169         m.changeToExtractionMode();
170         mexpected.changeToExtractionMode();
171
172         assertEquals(m, mexpected);
173     }
174
175     //Test for insertion mode when the sorting machine is empty, true
176     @Test
177     public final void testInsertionModeTrueEmpty() {
```

```
178         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
179         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, true);
180
181         boolean modeexpected = mexpected.isInInsertionMode();
182         boolean mode = m.isInInsertionMode();
183
184         assertEquals(modeexpected, mode);
185         assertEquals(m, mexpected);
186     }
187
188     //Test changing from insertion to extraction mode with no elements
189     @Test
190     public final void testExtractionModeEmpty() {
191         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
192         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false);
193
194         m.changeToExtractionMode();
195
196         assertEquals(mexpected, m);
197     }
198
199     //Test for changing mode
200     @Test
201     public final void testExtractionModeNonEmpty() {
202         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "cow");
203         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
204                 "cow");
205
206         m.changeToExtractionMode();
207
208         assertEquals(mexpected, m);
209     }
210
211     //Test for changing mode when elements are out of order
212     @Test
213     public final void testExtractionModeOutOfOrder() {
214         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "2",
215                 "1");
216         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
217                 "1", "2");
218
219         m.changeToExtractionMode();
220
221         assertEquals(mexpected, m);
222     }
223
224     //Test for removing first and making the sorting machine empty
225     @Test
226     public final void testRemoveFirst() {
227         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1");
228         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
229                 "1");
230
231         String test = m.removeFirst();
232         String ref = mexpected.removeFirst();
233
234         assertEquals(test, ref);
```

```
235         assertEquals(mexpected, m);
236     }
237
238     //Test for removing first and leaving the sorting machine filled
239     @Test
240     public final void testRemoveFirstNonEmpty() {
241         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
242                 "2");
243         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
244                 "1", "2");
245
246         String test = m.removeFirst();
247         String ref = mexpected.removeFirst();
248
249         assertEquals(ref, test);
250         assertEquals(mexpected, m);
251     }
252
253     //Test for machine order on empty sorting machine
254     @Test
255     public final void testOrderEmpty() {
256         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
257         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, true);
258
259         Comparator<String> test = m.order();
260         Comparator<String> ref = mexpected.order();
261
262         assertEquals(test, ref);
263         assertEquals(mexpected, m);
264     }
265
266     //Test for machine order on full sorting machine
267     @Test
268     public final void testOrderNonEmpty() {
269         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
270                 "2");
271         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
272                 "1", "2");
273
274         Comparator<String> test = m.order();
275         Comparator<String> ref = mexpected.order();
276
277         assertEquals(test, ref);
278         assertEquals(mexpected, m);
279     }
280
281     //Test for size in extraction mode
282     @Test
283     public final void testSizeEmptyExtraction() {
284         SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
285         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false);
286
287         int test = m.size();
288         int ref = mexpected.size();
289
290         assertEquals(test, ref);
291         assertEquals(mexpected, m);
```

```
292     }
293
294     //Test for size in extraction mode
295     @Test
296     public final void testSizeNonEmptyExtraction() {
297         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
298                 "2");
299         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
300                 "1", "2");
301
302         int test = m.size();
303         int ref = mexpected.size();
304
305         assertEquals(test, ref);
306         assertEquals(mexpected, m);
307     }
308
309     //Test for size in insertion mode
310     @Test
311     public final void testSizeNonEmptyInsertion() {
312         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "cow",
313                 "chicken");
314         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, true,
315                 "cow", "chicken");
316
317         int test = m.size();
318         int ref = mexpected.size();
319
320         assertEquals(test, ref);
321         assertEquals(mexpected, m);
322     }
323
324     //Test for size in insertion mode
325     @Test
326     public final void testSizeEmptyInsertion() {
327         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
328         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, true);
329
330         int test = m.size();
331         int ref = mexpected.size();
332
333         assertEquals(test, ref);
334         assertEquals(mexpected, m);
335     }
336
337     //Test for removing the first
338     @Test
339     public final void testRemoveFirstLargeHeap() {
340         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
341                 "2", "3", "4", "5", "6", "7");
342         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
343                 "1", "2", "3", "4", "5", "6", "7");
344
345         String test = m.removeFirst();
346         String ref = mexpected.removeFirst();
347
348         assertEquals(ref, test);
```

```
349         assertEquals(mexpected, m);
350     }
351
352     //Test for removing the first
353     @Test
354     public final void testRemoveFirstLargeRandom() {
355         SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "3",
356                 "2", "5", "6", "4", "7", "1");
357         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
358                 "3", "2", "5", "6", "4", "7", "1");
359
360         String test = m.removeFirst();
361         String ref = mexpected.removeFirst();
362
363         assertEquals(ref, test);
364         assertEquals(mexpected, m);
365     }
366
367     //Test for adding one
368     @Test
369     public final void testAddNonEmpty() {
370         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "cow");
371         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
372                 "chicken", "cow");
373         m.add("chicken");
374         m.changeToExtractionMode();
375         assertEquals(mexpected, m);
376     }
377
378     //Test for adding three
379     @Test
380     public final void testAddThree() {
381         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
382         SortingMachine<String> mexpected = this.createFromArgsRef(ORDER, false,
383                 "cow", "chicken", "goat");
384         m.add("cow");
385         m.add("chicken");
386         m.add("goat");
387         m.changeToExtractionMode();
388         assertEquals(mexpected, m);
389     }
390
391     //Test for adding three
392     @Test
393     public final void testAddThreeNonEmpty() {
394         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
395                 "rabbit");
396         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
397                 "cow", "chicken", "rabbit", "goat");
398         m.add("cow");
399         m.add("chicken");
400         m.add("goat");
401         m.changeToExtractionMode();
402         assertEquals(mExpected, m);
403     }
404 }
405
```