

```
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 Shyam Sai Bethina and Yihone Chu
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
48      mode.
49      *
50      * @param order
51      *           the {@code Comparator} defining the order for
52      {@code String}
53      * @param insertionMode
54      *           flag indicating the machine mode
55      * @param args
56      *           the entries for the {@code SortingMachine}
57      * @return the constructed {@code SortingMachine}
58      * @requires IS_TOTAL_PREORDER([relation computed by
59      order.compare method])
60      * @ensures <pre>
61      * createFromArgsTest = (insertionMode, order, [multiset of
62      entries in args])
63      * </pre>
64      */
65     private SortingMachine<String>
66     createFromArgsTest(Comparator<String> order,
67         boolean insertionMode, String... args) {
68         SortingMachine<String> sm = this.constructorTest(order);
69         for (int i = 0; i < args.length; i++) {
70             sm.add(args[i]);
71         }
72         if (!insertionMode) {
73             sm.changeToExtractionMode();
74         }
75         return sm;
76     }
77
78     /**
79      *
80      * Creates and returns a {@code SortingMachine<String>} of the
81      reference
82      * implementation type with the given entries and mode.
83      *
84      * @param order
85      *           the {@code Comparator} defining the order for
86      {@code String}
87      * @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     /**
125     * Routine test case for constructor.
126     */
127     @Test
128     public final void testConstructor() {
129         SortingMachine<String> m = this.constructorTest(ORDER);
130         SortingMachine<String> mExpected =
131         this.constructorRef(ORDER);
132         assertEquals(mExpected, m);
133     }
134
135     /**
136     * Edge case for add method.
137     */
138     @Test
139     public final void testAdd1() {
140         SortingMachine<String> m = this.createFromArgsTest(ORDER,
141         true);
142         SortingMachine<String> mExpected =
143         this.createFromArgsRef(ORDER, true,
144         "green");
145         m.add("green");
146         assertEquals(mExpected, m);
147     }
148
149     /**
150     * Challenging case for add method.
151     */
152     @Test
153     public final void testAdd2() {
154         SortingMachine<String> m = this.createFromArgsTest(ORDER,
155         true,
156         "green");
157         SortingMachine<String> mExpected =
158         this.createFromArgsRef(ORDER, true,
159         "green", "green");
160         m.add("green");
161         assertEquals(mExpected, m);
162     }
163
164     /**
```

```
160     * Routine case for add method.
161     */
162     @Test
163     public final void testAdd3() {
164         SortingMachine<String> m = this.createFromArgsTest(ORDER,
165 true,
166         "green");
167         SortingMachine<String> mExpected =
168 this.createFromArgsRef(ORDER, true,
169         "blue", "green");
170
171         m.add("blue");
172
173         assertEquals(mExpected, m);
174     }
175
176     /**
177     * Edge case for changeToExtractionMode method.
178     */
179     @Test
180     public final void testChangeMode1() {
181         SortingMachine<String> m = this.createFromArgsTest(ORDER,
182 true);
183         SortingMachine<String> mExpected =
184 this.createFromArgsRef(ORDER, true);
185
186         m.changeToExtractionMode();
187         mExpected.changeToExtractionMode();
188
189         assertEquals(mExpected, m);
190     }
191
192     /**
193     * Challenging case for changeToExtractionMode method.
194     */
195     @Test
196     public final void testChangeMode2() {
197         SortingMachine<String> m = this.createFromArgsTest(ORDER,
198 true, "");
199         SortingMachine<String> mExpected =
200 this.createFromArgsRef(ORDER, true,
201         "");
202
203         m.changeToExtractionMode();
```

```
198         mExpected.changeToExtractionMode();
199
200         assertEquals(mExpected, m);
201     }
202
203     /**
204      * Routine case for changeToExtractionMode method.
205      */
206     @Test
207     public final void testChangeMode3() {
208         SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "green",
209             "blue");
210         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
211             "green", "blue");
212
213         m.changeToExtractionMode();
214         mExpected.changeToExtractionMode();
215
216         assertEquals(mExpected, m);
217     }
218
219     /**
220      * Edge case for removeFirst method.
221      */
222     @Test
223     public final void testemoveFirst1() {
224         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false,
225             "green");
226         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
227             "green");
228
229         String removed = m.removeFirst();
230         String expected = mExpected.removeFirst();
231
232         assertEquals(mExpected, m);
233         assertEquals(expected, removed);
234     }
235
236     /**
237      * Challenging case for removeFirst method.
```

```
238     */
239     @Test
240     public final void testRemoveFirst2() {
241         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false, "");
242         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
243             "");
244
245         String removed = m.removeFirst();
246         String expected = mExpected.removeFirst();
247
248         assertEquals(mExpected, m);
249         assertEquals(expected, removed);
250     }
251
252     /**
253     * Routine case for removeFirst method.
254     */
255     @Test
256     public void testRemoveFirst3() {
257         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false,
258             "hello", "there", "professor");
259         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
260             "hello", "there", "professor");
261
262         String removed = m.removeFirst();
263         String expected = mExpected.removeFirst();
264
265         assertEquals(mExpected, m);
266         assertEquals(expected, removed);
267     }
268
269     /**
270     * Edge case for isInInsertionMode method.
271     */
272     @Test
273     public void testInsertionMode1() {
274         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false);
275         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false);
```

```
276
277     Boolean test = m.isInInsertionMode();
278     Boolean expected = mExpected.isInInsertionMode();
279
280     assertEquals(mExpected, m);
281     assertEquals(expected, test);
282
283 }
284
285 /**
286  * Challenging case for isInInsertionMode method.
287  */
288 @Test
289 public void testInsertionMode2() {
290     SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "");
291     SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
292         "");
293
294     Boolean test = m.isInInsertionMode();
295     Boolean expected = mExpected.isInInsertionMode();
296
297     assertEquals(mExpected, m);
298     assertEquals(expected, test);
299 }
300
301 /**
302  * Routine case for isInInsertionMode method.
303  */
304 @Test
305 public void testInsertionMode3() {
306     SortingMachine<String> m = this.createFromArgsTest(ORDER,
false, "blue",
307         "green");
308     SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
309         "blue", "green");
310
311     Boolean test = m.isInInsertionMode();
312     Boolean expected = mExpected.isInInsertionMode();
313
314     assertEquals(mExpected, m);
315     assertEquals(expected, test);
```



```
316     }
317
318     /**
319      * Edge case for order method.
320      */
321     @Test
322     public void testOrder1() {
323         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false);
324         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false);
325
326         Comparator<String> test = m.order();
327         Comparator<String> expected = mExpected.order();
328
329         assertEquals(mExpected, m);
330         assertEquals(expected, test);
331     }
332
333     /**
334      * Challenging case for order method.
335      */
336     @Test
337     public void testOrder2() {
338         SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "");
339         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
340             "");
341
342         Comparator<String> test = m.order();
343         Comparator<String> expected = mExpected.order();
344
345         assertEquals(mExpected, m);
346         assertEquals(expected, test);
347     }
348
349     /**
350      * Routine case for order method.
351      */
352     @Test
353     public void testOrder3() {
354         SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "blue",
```

```
355         "green");
356         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, true,
357         "blue", "green");
358
359         Comparator<String> test = m.order();
360         Comparator<String> expected = mExpected.order();
361
362         assertEquals(mExpected, m);
363         assertEquals(expected, test);
364     }
365
366     /**
367      * Edge case for size method.
368      */
369     @Test
370     public void testSize1() {
371         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true);
372         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, true);
373
374         int test = m.size();
375         int expected = mExpected.size();
376
377         assertEquals(mExpected, m);
378         assertEquals(expected, test);
379     }
380
381     /**
382      * Challenging case for size method.
383      */
384     @Test
385     public void testSize2() {
386         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "");
387         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, false,
388         "");
389
390         int test = m.size();
391         int expected = mExpected.size();
392
393         assertEquals(mExpected, m);
```

```
394         assertEquals(expected, test);
395     }
396
397     /**
398      * Routine case for size method.
399      */
400     @Test
401     public void testSize3() {
402         SortingMachine<String> m = this.createFromArgsTest(ORDER,
false, "blue",
403         "green");
404         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
405         "blue", "green");
406
407         int test = m.size();
408         int expected = mExpected.size();
409
410         assertEquals(mExpected, m);
411         assertEquals(expected, test);
412     }
413
414     // TODO - add test cases for add, changeToExtractionMode,
removeFirst,
415     // isInInsertionMode, order, and size
416
417 }
418
```