

```

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 Chloe Feller and Krish Patel
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     * Comparator instance to be used in all test cases.
116     */
117
118     private static final StringLT ORDER = new StringLT();

```

```
119
120  /*
121   * Sample test cases.
122   */
123
124  /**
125   * Test COnstructor.
126   */
127  @Test
128  public final void testConstructor() {
129      SortingMachine<String> m = this.constructorTest(ORDER);
130      SortingMachine<String> mExpected = this.constructorRef(ORDER);
131      assertEquals(mExpected, m);
132  }
133
134  /**
135   * Test add thats empty.
136   */
137  @Test
138  public final void testAddEmpty() {
139      SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
140      SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
141          "green");
142      m.add("green");
143      assertEquals(mExpected, m);
144  }
145
146  /**
147   * Test add that has a variable.
148   */
149  @Test
150  public final void testAddTwo() {
151      SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a");
152      SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
153          "a", "b");
154      m.add("b");
155      assertEquals(mExpected, m);
156  }
157
158  /**
159   * Test add that has multiple variable.
160   */
161  @Test
162  public final void testAddThree() {
163      SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a",
164          "b", "c", "d", "e");
165      SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
166          "a", "b", "c", "d", "e", "f");
167      m.add("f");
168      assertEquals(mExpected, m);
169  }
170
171  // TODO - add test cases for add, changeToExtractionMode, removeFirst,
172  // isInInsertionMode, order, and size
173
174  /**
175   * Test cases for changeToExtractionMode.
176   */
177  /**
```

```
178     * Test for changeToExtractionMode with an empty SortingMachine.
179     */
180     @Test
181     public final void changeToExtractionModeOne() {
182         // Initialize the variables
183         SortingMachine<String> sort = this.createFromArgsTest(ORDER, true);
184         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
185             false);
186
187         // Call the method
188         sort.changeToExtractionMode();
189
190         // Assert variables are equal
191         assertEquals(sortExpected, sort);
192     }
193
194     /**
195     * Test for changeToExtractionMode with a non-empty SortingMachine.
196     */
197     @Test
198     public final void changeToExtractionModeTwo() {
199         // Initialize the variables
200         SortingMachine<String> sort = this.createFromArgsTest(ORDER, true,
201             "Boolean", "Double", "String");
202         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
203             false, "Boolean", "Double", "String");
204
205         // Call the method
206         sort.changeToExtractionMode();
207
208         // Assert variables are equal
209         assertEquals(sortExpected, sort);
210     }
211
212     /**
213     * Test cases for removeFirst.
214     */
215     /**
216     * Test for removeFirst resulting in an empty SortingMachine.
217     */
218     @Test
219     public final void removeFirstOne() {
220         // Initialize the variables
221         SortingMachine<String> sort = this.createFromArgsTest(ORDER, false,
222             "Double");
223         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
224             false);
225
226         // Call the method
227         String removed = sort.removeFirst();
228
229         // Assert variables are equal
230         assertEquals(sortExpected, sort);
231         assertEquals("Double", removed);
232     }
233
234     /**
235     * Test for removeFirst resulting in a non-empty SortingMachine.
236     */
```

```
237     @Test
238     public final void removeFirstTwo() {
239         // Initialize the variables
240         SortingMachine<String> sort = this.createFromArgsTest(ORDER, false,
241             "Boolean", "Array", "Double", "Ai");
242         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
243             false, "Boolean", "Array", "Double");
244
245         // Call the method
246         String removed = sort.removeFirst();
247
248         // Assert variables are equal
249         assertEquals(sortExpected, sort);
250         assertEquals("Ai", removed);
251     }
252
253     /**
254     * Test cases for isInInsertionMode.
255     */
256     /**
257     * Test for isInInsertionMode with an empty SortingMachine.
258     */
259     @Test
260     public final void isInInsertionModeOne() {
261         // Initialize the variables
262         SortingMachine<String> sort = this.createFromArgsTest(ORDER, true);
263         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
264             true);
265
266         // Call the method
267         boolean insert = sort.isInInsertionMode();
268
269         // Assert variables are equal
270         assertEquals(sortExpected, sort);
271         assertEquals(true, insert);
272     }
273
274     /**
275     * Test for isInInsertionMode with a non-empty SortingMachine.
276     */
277     @Test
278     public final void isInInsertionModeTwo() {
279         // Initialize the variables
280         SortingMachine<String> sort = this.createFromArgsTest(ORDER, true,
281             "Double", "Boolean");
282         SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
283             true, "Double", "Boolean");
284
285         // Call the method
286         boolean insert = sort.isInInsertionMode();
287
288         // Assert variables are equal
289         assertEquals(sortExpected, sort);
290         assertEquals(true, insert);
291     }
292
293     /**
294     * Test order through insertion.
295     */
```

```
296     @Test
297     public final void orderTestOne() {
298         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
299         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true);
300
301         Comparator<String> o = m.order();
302         Comparator<String> oExpected = mExpected.order();
303
304         assertEquals(oExpected, o);
305         assertEquals(mExpected, m);
306     }
307
308     /**
309     * Test order through extraction.
310     */
311     @Test
312     public final void orderTestTwo() {
313         SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
314         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
315
316         Comparator<String> o = m.order();
317         Comparator<String> oExpected = mExpected.order();
318
319         assertEquals(oExpected, o);
320         assertEquals(mExpected, m);
321     }
322
323     /**
324     * Test for size with empty variables through insertion.
325     */
326     @Test
327     public final void sizeTestInsertionOne() {
328         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
329
330         int i = m.size();
331
332         assertEquals(0, i);
333     }
334
335     /**
336     * Test for size with empty variables through extraction.
337     */
338     @Test
339     public final void sizeTestExtractionOne() {
340         SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
341
342         int i = m.size();
343
344         assertEquals(0, i);
345     }
346
347     /**
348     * Test for size with a variable through insertion.
349     */
350     @Test
351     public final void sizeTestInsertionTwo() {
352         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a");
353
354         int i = m.size();
```

```
355
356     assertEquals(1, i);
357 }
358
359 /**
360  * Test for size with a variable through extraction.
361  */
362 @Test
363 public final void sizeTestExtractionTwo() {
364     SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "a");
365
366     int i = m.size();
367
368     assertEquals(1, i);
369 }
370
371 /**
372  * Test for size with multiple variables through insertion.
373  */
374 @Test
375 public final void sizeTestInsertionThree() {
376     SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a",
377         "b", "c", "d", "e", "f");
378
379     int i = m.size();
380
381     final int expectedI = 6;
382
383     assertEquals(expectedI, i);
384 }
385
386 }
387
```