```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;

import java.util.Comparator;

import org.junit.Test;

import components.sortingmachine.SortingMachine;

/**
 * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
 * kernel methods.
 *
 * @author Charan Nanduri and Evan Frisbie
 *
 */
public abstract class SortingMachineTest {

    /**
     * Invokes the appropriate {@code SortingMachine} constructor for the
     * implementation under test and returns the result.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @return the new {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @ensures constructorTest = (true, order, {})
     */
    protected abstract SortingMachine<String> constructorTest(
            Comparator<String> order);

    /**
     * Invokes the appropriate {@code SortingMachine} constructor for the
     * reference implementation and returns the result.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @return the new {@code SortingMachine}
     * @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     * Sample test cases.
123     */
124
125    @Test
126    public final void testConstructor() {
127        SortingMachine<String> m = this.constructorTest(ORDER);
128        SortingMachine<String> mExpected =
   this.constructorRef(ORDER);
129        assertEquals(mExpected, m);
130    }
131
132    @Test
133    public final void testAddEmpty() {
134        SortingMachine<String> m = this.createFromArgsTest(ORDER,
   true);
135        SortingMachine<String> mExpected =
   this.createFromArgsRef(ORDER, true,
136                "green");
137        m.add("green");
138        assertEquals(mExpected, m);
139    }
140
141    // TODO - add test cases for add, changeToExtractionMode,
   removeFirst,
142    // isInInsertionMode, order, and size
143
144    //add tests
145    @Test
146    public final void testAddNonEmpty() {
147        SortingMachine<String> m = this.createFromArgsTest(ORDER,
   true, "one",
148                "two", "three");
149        SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
   true, "one",
150                "two", "three");
151        m.add("four");
152        mExp.add("four");
153        assertEquals(mExp, m);
154    }
155
156    @Test
```

```
157     public final void testAddMultipleEntries() {
158
159         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
160                 "two");
161         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
162                 "two");
163         m.add("three");
164         mExp.add("three");
165         m.add("four");
166         mExp.add("four");
167         assertEquals(mExp, m);
168     }
169
170     //change mode tests
171     @Test
172     public final void testChangeToExtractionMode() {
173         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
174                 "two", "three");
175         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
176                 "two", "three");
177         m.changeToExtractionMode();
178         mExp.changeToExtractionMode();
179         assertFalse(m.isInInsertionMode());
180         assertEquals(mExp, m);
181     }
182
183     @Test
184     public final void testChangeToExtractionModeEmpty() {
185         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true);
186         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true);
187         m.changeToExtractionMode();
188         mExp.changeToExtractionMode();
189         assertFalse(m.isInInsertionMode());
190         assertEquals(mExp, m);
191     }
192
193     //removefirst tests
194     //this one is failing for no clear reason
195     @Test
```

```
196     public final void testRemoveFirst() {
197         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "one",
198                 "two", "three");
199         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false,
200                 "one", "two", "three");
201         String obj = m.removeFirst();
202         String objExp = mExp.removeFirst();
203         assertEquals(objExp, obj);
204         assertEquals(mExp, m);
205     }
206 
207     //this one is also failing for no clear reason
208     @Test
209     public final void testRemoveFirstNoOrder() {
210 
211         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "one",
212                 "two", "three");
213         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false,
214                 "one", "two", "three");
215         String obj = m.removeFirst();
216         String objExp = mExp.removeFirst();
217         assertEquals(obj, objExp);
218         assertEquals(mExp, m);
219 
220     }
221 
222     @Test
223     public final void testRemoveFirstSingleObject() {
224 
225         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "one");
226         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false,
227                 "one");
228         String obj = m.removeFirst();
229         String objExp = mExp.removeFirst();
230         assertEquals(obj, objExp);
231         assertEquals(mExp, m);
232 
233     }
234     //insertion mode tests
```

```
235
236     @Test
237     public final void testIsInInsertionModeTrue() {
238         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
239                 "two");
240         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
241                 "two");
242
243         assertEquals(mExp.isInInsertionMode(),
    m.isInInsertionMode());
244         assertEquals(mExp, m);
245     }
246
247     @Test
248     public final void testIsInInsertionModeFalse() {
249
250         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
251                 "two");
252         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
253                 "two");
254
255         assertEquals(mExp.isInInsertionMode(),
    m.isInInsertionMode());
256         assertEquals(mExp, m);
257
258     }
259
260     //order tests
261     @Test
262     public final void testOrderInInsertionMode() {
263
264         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
265                 "two", "three");
266         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
267                 "two", "three");
268
269         assertEquals(mExp.order(), m.order());
270         assertEquals(mExp, m);
271     }
```

```
272
273     @Test
274     public final void testOrderInExtractionMode() {
275
276         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "one",
277                 "two");
278         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false,
279                 "one", "two");
280
281         assertEquals(mExp.order(), m.order());
282         assertEquals(mExp, m);
283     }
284     //size tests
285
286     @Test
287     public final void testSizeInInsertionMode() {
288
289         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true, "one",
290                 "two");
291         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true, "one",
292                 "two");
293
294         assertEquals(mExp.size(), m.size());
295         assertEquals(mExp, m);
296     }
297
298     @Test
299
300     public final void testSizeInExtractionMode() {
301
302         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false, "one",
303                 "two");
304         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false,
305                 "one", "two");
306
307         assertEquals(mExp.size(), m.size());
308         assertEquals(mExp, m);
309     }
310
```

```
311     @Test
312     public final void testSizeInInsertionModeEmpty() {
313
314         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    true);
315         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    true);
316
317         assertEquals(mExp.size(), m.size());
318         assertEquals(mExp, m);
319     }
320
321     @Test
322
323     public final void testSizeInExtractionModeEmpty() {
324
325         SortingMachine<String> m = this.createFromArgsTest(ORDER,
    false);
326         SortingMachine<String> mExp = this.createFromArgsRef(ORDER,
    false);
327         assertEquals(mExp.size(), m.size());
328         assertEquals(mExp, m);
329     }
330
331 }
332
```