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

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 David P & Zach
 *
 */
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])
     * @ensures constructorRef = (true, order, {})
     */
    protected abstract SortingMachine<String> constructorRef(
            Comparator<String> order);

    /**
     *
     * Creates and returns a {@code SortingMachine<String>} of the
     * implementation under test type with the given entries and mode.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @param insertionMode
     *            flag indicating the machine mode
     * @param args
     *            the entries for the {@code SortingMachine}
     * @return the constructed {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @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     /*
141      * Test cases for add
142      */
143
144     @Test
145     public final void testAddOneElement() {
146         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
147         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
148                 "blue");
149         m.add("blue");
150         assertEquals(mExpected, m);
151     }
152
153     @Test
154     public final void testAddMultipleElements() {
155         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
156         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
157                 "blue", "red", "yellow");
158         m.add("blue");
159         m.add("red");
160         m.add("yellow");
161         assertEquals(mExpected, m);
162     }
163
164     @Test
165     public final void testAddDuplicateElements() {
166         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
167         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
168                 "blue", "blue");
169         m.add("blue");
170         m.add("blue");
171         assertEquals(mExpected, m);
```

```
172     }
173
174     /*
175      * Test cases for changeToExtractionMode
176      */
177
178     @Test
179     public final void testChangeToExtractionModeEmpty() {
180         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
181         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
182         m.changeToExtractionMode();
183         assertEquals(mExpected, m);
184     }
185
186     @Test
187     public final void testChangeToExtractionModeNonEmpty() {
188         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "apple",
189                 "banana", "cherry");
190         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
191                 "apple", "banana", "cherry");
192         m.changeToExtractionMode();
193         assertEquals(mExpected, m);
194     }
195
196     @Test
197     public final void testChangeToExtractionModeAfterAdd() {
198         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
199         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
200                 "apple", "banana");
201         m.add("apple");
202         m.add("banana");
203         m.changeToExtractionMode();
204         assertEquals(mExpected, m);
205     }
206
207     /*
208      * Test cases for removeFirst
209      */
210
211     @Test
212     public final void testRemoveFirstSingleElement() {
213         SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
214                 "apple");
215         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
216         String first = m.removeFirst();
217         assertEquals("apple", first);
218         assertEquals(mExpected, m);
219     }
220
221     @Test
222     public final void testRemoveFirstMultipleElements() {
223         SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
224                 "apple", "banana", "cherry");
225         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
226                 "banana", "cherry");
227         String first = m.removeFirst();
228         assertEquals("apple", first);
```

```
229         assertEquals(mExpected, m);
230     }
231
232     /*
233      * Test cases for isInInsertionMode
234      */
235
236     @Test
237     public final void testIsInInsertionModeTrue() {
238         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
239         boolean result = m.isInInsertionMode();
240         assertEquals(true, result);
241     }
242
243     @Test
244     public final void testIsInInsertionModeFalse() {
245         SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
246                 "apple");
247         boolean result = m.isInInsertionMode();
248         assertEquals(false, result);
249     }
250
251     /*
252      * Test cases for order
253      */
254
255     @Test
256     public final void testOrder() {
257         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
258         Comparator<String> order = m.order();
259         assertEquals(ORDER, order);
260     }
261
262     /*
263      * Test cases for size
264      */
265
266     @Test
267     public final void testSizeInInsertionModeEmpty() {
268         SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
269         int size = m.size();
270         assertEquals(0, size);
271     }
272
273     @Test
274     public final void testSizeInInsertionModeNonEmpty() {
275         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "apple",
276                 "banana");
277         int size = m.size();
278         assertEquals(2, size);
279     }
280
281     @Test
282     public final void testSizeInExtractionMode() {
283         SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
284                 "apple", "banana", "cherry");
285         int size = m.size();
```

```
286         assertEquals(3, size);
287     }
288 }
289
```