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

/**
 * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
 * kernel methods.
 *
 * @author Put your name here
 *
 */
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>
     * createFromArgsTest = (insertionMode, order, [multiset of entries in args])
     * </pre>
     */
    private SortingMachine<String> createFromArgsTest(Comparator<String> order,
            boolean insertionMode, String... args) {
        SortingMachine<String> sm = this.constructorTest(order);
        for (int i = 0; i < args.length; i++) {
            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 for default constructor
```

```
125     @Test
126     public final void testConstructor() {
127         SortingMachine<String> m = this.constructorTest(ORDER);
128         SortingMachine<String> mExpected = this.constructorRef(ORDER);
129
130         assertEquals(mExpected, m);
131     }
132
133     // Test for constructor with one arg
134     @Test
135     public final void testForOneArg() {
136         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1");
137         SortingMachine<String> mExp = this.createFromArgsRef(ORDER, true, "1");
138
139         assertEquals(mExp, m);
140     }
141
142     // Test for constructor with multiple args
143     @Test
144     public final void testForMultipleArg() {
145         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
146                 "apple,", "banana", "cherry", "date", "elderberry");
147         SortingMachine<String> mExp = this.createFromArgsRef(ORDER, true,
148                 "apple,", "banana", "cherry", "date", "elderberry");
149
150         assertEquals(mExp, m);
151     }
152
153     // Test for add to empty sorting machine
154     @Test
155     public final void testAddEmpty() {
156         SortingMachine<String> m = this.constructorTest(ORDER);
157         SortingMachine<String> mExp = this.createFromArgsRef(ORDER, true, "r");
158         m.add("r");
159         assertEquals(mExp, m);
160     }
161
162     // Test for multiple add calls
163     @Test
164     public final void add() {
165         SortingMachine<String> m = this.constructorTest(ORDER);
166         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
167                 "green", "blue", "yellow");
168         m.add("blue");
169         m.add("yellow");
170         m.add("green");
171
172         assertEquals(mExpected, m);
173     }
174
175     // Test for changing to extraction mode
176     @Test
177     public final void testChangeToExtractionModeEmpty() {
178         // Default constructor creates sorting machine with insertion mode
179         SortingMachine<String> m = this.constructorTest(ORDER);
180         SortingMachine<String> mExpected = this.constructorRef(ORDER);
181
182         m.changeToExtractionMode();
183         mExpected.changeToExtractionMode();
```

```
184
185         assertEquals(mExpected, m);
186     }
187
188     // Test for changing to extraction mode
189     @Test
190     public final void testChangeToExtractionModeNonEmpty() {
191         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
192                 "apple,", "banana", "cherry", "date", "elderberry");
193         SortingMachine<String> mExp = this.createFromArgsRef(ORDER, true,
194                 "apple,", "banana", "cherry", "date", "elderberry");
195
196         m.changeToExtractionMode();
197         mExp.changeToExtractionMode();
198
199         assertEquals(mExp.isInInsertionMode(), m.isInInsertionMode());
200         assertEquals(mExp, m);
201     }
202
203     // Test for removeFirst one item
204     @Test
205     public final void removeFirstOneItem() {
206         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "6");
207         SortingMachine<String> mExpected = this.constructorRef(ORDER);
208
209         m.changeToExtractionMode();
210         mExpected.changeToExtractionMode();
211
212         String str = m.removeFirst();
213
214         assertEquals(mExpected, m);
215         assertEquals(str, "6");
216     }
217
218     // Test for removeFirst multiple items
219     @Test
220     public final void removeFirst() {
221         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
222                 "4", "3", "7", "6");
223         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
224                 "4", "6", "7");
225
226         m.changeToExtractionMode();
227         // need to change to extraction mode so assertEquals works
228         mExpected.changeToExtractionMode();
229
230         String str1 = m.removeFirst();
231         String str2 = m.removeFirst();
232
233         assertEquals(mExpected, m);
234         assertEquals(str1, "1");
235         assertEquals(str2, "3");
236     }
237
238     // Test for IsInsertionModeTrue
239     @Test
240     public final void testIsInsertionModeTrue() {
241         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
242                 "green");
```

```
243         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
244                 "green");
245
246         assertEquals(m.isInInsertionMode(), mExpected.isInInsertionMode());
247         assertEquals(m.isInInsertionMode(), true);
248         assertEquals(mExpected, m);
249     }
250
251     // Test for IsInsertionModeTrue
252     @Test
253     public final void testIsInsertionModeFalse() {
254         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
255                 "green");
256         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
257                 "green");
258
259         m.changeToExtractionMode();
260         mExpected.changeToExtractionMode();
261
262         assertEquals(m.isInInsertionMode(), mExpected.isInInsertionMode());
263         assertEquals(m.isInInsertionMode(), false);
264         assertEquals(mExpected, m);
265     }
266
267     // Test for order
268     @Test
269     public final void testOrderEmpty() {
270         SortingMachine<String> m = this.constructorTest(ORDER);
271         SortingMachine<String> mExpected = this.constructorRef(ORDER);
272
273         assertEquals(m.order(), mExpected.order());
274         assertEquals(m.order(), ORDER);
275         assertEquals(mExpected, m);
276     }
277
278     // Test for order on non empty sorting machine
279     @Test
280     public final void testOrderNonEmpty() {
281         SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
282                 "2", "3", "4");
283         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
284                 "1", "2", "3", "4");
285
286         assertEquals(m.order(), mExpected.order());
287         assertEquals(m.order(), ORDER);
288         assertEquals(mExpected, m);
289     }
290
291     // Test for size Zero when insertion mode
292     @Test
293     public final void testSizeZeroInsertionMode() {
294         SortingMachine<String> m = this.constructorTest(ORDER);
295         SortingMachine<String> mExpected = this.constructorRef(ORDER);
296
297         assertEquals(m.size(), 0);
298         assertEquals(mExpected, m);
299     }
300
301     // Test for size Zero when extraction mode
```

```
302     @Test
303     public final void testSizeZeroExtractionMode() {
304         SortingMachine<String> m = this.constructorTest(ORDER);
305         SortingMachine<String> mExpected = this.constructorRef(ORDER);
306
307         m.changeToExtractionMode();
308         mExpected.changeToExtractionMode();
309
310         assertEquals(m.size(), 0);
311         assertEquals(mExpected, m);
312     }
313
314     // Test for size multiple when insertion mode
315     @Test
316     public final void testSizeMultipleInsertionMode() {
317         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
318                 "purple", "green", "yellow", "red", "orange", "blue");
319         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
320                 "purple", "green", "yellow", "red", "orange", "blue");
321
322         final int six = 6;
323
324         assertEquals(m.size(), six);
325         assertEquals(mExpected, m);
326     }
327
328     // Test for size multiple when extraction mode
329     @Test
330     public final void testSizeMultipleExtractionMode() {
331         SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
332                 "purple", "green", "yellow", "red", "orange", "blue");
333         SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
334                 "purple", "green", "yellow", "red", "orange", "blue");
335
336         m.changeToExtractionMode();
337         mExpected.changeToExtractionMode();
338
339         final int six = 6;
340
341         assertEquals(m.size(), six);
342         assertEquals(mExpected, m);
343     }
344 }
345
```