

```

1 import static org.junit.Assert.assertEquals;
2
3 /**
4  * JUnit test fixture for {@code
5  *   SortingMachine<String>}s constructor and
6  * kernel methods.
7  *
8  * @author Zhuoyang Li + Xinci Ma
9  */
10 public abstract class SortingMachineTest {
11
12     /**
13      * Invokes the appropriate {@code
14      *   SortingMachine} constructor for the
15      * implementation under test and returns the
16      * result.
17      *
18      * @param order
19      *           the {@code Comparator}
20      *           defining the order for {@code String}
21      * @return the new {@code SortingMachine}
22      * @requires IS_TOTAL_PREORDER([relation
23      *   computed by order.compare method])
24      * @ensures constructorTest = (true, order,
25      *   {})
26      */
27     protected abstract SortingMachine<String>
28     constructorTest(
29         Comparator<String> order);
30
31     /**
32      * Invokes the appropriate {@code
33      *   SortingMachine} constructor for the
34      * 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 /**
116  * Comparator instance to be used in all
117  * test cases.
118  */
119 private static final StringLT ORDER = new
120 StringLT();
121
122 /**
123  * Sample test cases.
124  */
125 @Test
126 public final void testConstructor() {
127     SortingMachine<String> m =
128     this.constructorTest(ORDER);
129     SortingMachine<String> mExpected =
130     this.constructorRef(ORDER);
131     assertEquals(mExpected, m);
132 }
133
134 @Test
135 public final void testAddEmpty() {
136     SortingMachine<String> m =
137     this.createFromArgsTest(ORDER, true);
138     SortingMachine<String> mExpected =
139     this.createFromArgsRef(ORDER, true,
140         "green");
141     m.add("green");
142     assertEquals(mExpected, m);

```

```

138     }
139
140     // TODO - add test cases for add,
changeToExtractionMode, removeFirst,
141     // isInInsertionMode, order, and size
142     @Test
143     public final void testAdd1() {
144         SortingMachine<String> m =
this.createFromArgsTest(ORDER, true,
145             "green");
146         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
147             "green", "red");
148         m.add("red");
149         assertEquals(mExpected, m);
150     }
151
152     // space + String
153     @Test
154     public final void testAdd2() {
155         SortingMachine<String> m =
this.createFromArgsTest(ORDER, true, "1",
156             "7", "5", "5", "5", "8", "7",
"6", "1", "0");
157         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
158             "1", "7", "5", "5", "5", "8",
"7", "6", "1", "0", " 9");
159         m.add(" 9");
160         assertEquals(mExpected, m);
161     }
162
163
164     // space only
165     @Test

```

```

166     public final void testAdd3() {
167         SortingMachine<String> m =
168             this.createFromArgsTest(ORDER, true, "1",
169                                     "7", "5", "5", "5", "8", "7",
170                                     "6", "1", "0");
169         SortingMachine<String> mExpected =
170             this.createFromArgsRef(ORDER, true,
171                                   "1", "7", "5", "5", "5", "8",
172                                   "7", "6", "1", "0", " ");
171         m.add(" ");
172         assertEquals(mExpected, m);
173     }
174 }
175
176 @Test
177 public final void
178 testChangeToExtractionMode1() {
179     SortingMachine<String> m =
180         this.createFromArgsTest(ORDER, true, "1",
181                                 "7", "5", "5", "5", "8", "7",
182                                 "6", "1", "0");
180     SortingMachine<String> mExpected =
181         this.createFromArgsRef(ORDER, false,
182                                "1", "7", "5", "5", "5", "8",
183                                "7", "6", "1", "0");
182     m.changeToExtractionMode();
183     assertEquals(mExpected, m);
184 }
185
186 // test empty
187 @Test
188 public final void
189 testChangeToExtractionMode2() {
189     SortingMachine<String> m =
190         this.createFromArgsTest(ORDER, true);

```

```

190         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, false);
191         m.changeToExtractionMode();
192         assertEquals(mExpected, m);
193     }
194
195     @Test
196     public final void testRemoveFirst1() {
197         SortingMachine<String> m =
    this.createFromArgsTest(ORDER, false, "1",
198             "7", "5", "5", "5", "8", "7",
    "6", "1", "0");
199         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, false,
200             "1", "1", "5", "5", "5", "6",
    "7", "7", "8");
201         String s = m.removeFirst();
202
203         assertEquals(mExpected, m);
204         assertEquals("0", s);
205     }
206
207     // test one element
208     @Test
209     public final void testRemoveFirst2() {
210         SortingMachine<String> m =
    this.createFromArgsTest(ORDER, false,
211             "Real");
212         SortingMachine<String> mExpected =
    this.createFromArgsRef(ORDER, false);
213         String s = m.removeFirst();
214         assertEquals(mExpected, m);
215         assertEquals("Real", s);
216     }
217

```



```

218     //test same elements twice
219     @Test
220     public final void testRemoveFirst3() {
221         SortingMachine<String> m =
222         this.createFromArgsTest(ORDER, false, "1",
223         "1", "1", "1", "1");
224         SortingMachine<String> mExpected =
225         this.createFromArgsRef(ORDER, false,
226         "1", "1", "1", "1", "1");
227         String s = m.removeFirst();
228         String s1 = mExpected.removeFirst();
229
230         assertEquals(mExpected, m);
231         assertEquals(s1, s);
232     }
233
234     @Test
235     public final void testRemoveFirst4() {
236         SortingMachine<String> m =
237         this.createFromArgsTest(ORDER, false, "1",
238         "7", "5", "5", "5", "8", "7",
239         "6", "1", "0");
240         SortingMachine<String> mExpected =
241         this.createFromArgsRef(ORDER, false,
242         "1", "7", "5", "5", "5", "8",
243         "7", "6", "1", "0");
244         String s = m.removeFirst();
245         String s1 = mExpected.removeFirst();
246
247         assertEquals(mExpected, m);
248         assertEquals(s1, s);
249     }
250
251     @Test
252     public final void testIsInInsertionMode1() {

```

```

247         SortingMachine<String> m =
this.createFromArgsTest(ORDER, true, "1",
248             "7", "5", "5", "5", "8", "7",
"6", "1", "0");
249         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
250             "1", "7", "5", "5", "5", "8",
"7", "6", "1", "0");
251         boolean b = m.isInInsertionMode();
252
253         assertEquals(mExpected, m);
254         assertEquals(true, b);
255     }
256
257     //test empty false
258     @Test
259     public final void testIsInInsertionMode2() {
260         SortingMachine<String> m =
this.createFromArgsTest(ORDER, false);
261         SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false);
262         boolean b = m.isInInsertionMode();
263         boolean b1 =
mExpected.isInInsertionMode();
264
265         assertEquals(mExpected, m);
266         assertEquals(b1, b);
267     }
268
269     //test empty true
270     @Test
271     public final void testIsInInsertionMode3() {
272         SortingMachine<String> m =
this.createFromArgsTest(ORDER, true);
273         SortingMachine<String> mExpected =

```

```

        this.createFromArgsRef(ORDER, true);
274         boolean b = m.isInInsertionMode();
275         boolean b1 =
mExpected.isInInsertionMode();
276
277         assertEquals(mExpected, m);
278         assertEquals(b1, b);
279     }
280
281     //test nonempty false
282     @Test
283     public final void testIsInInsertionMode4() {
284         SortingMachine<String> m =
        this.createFromArgsTest(ORDER, false, "1",
285                                "7", "5", "5", "5", "8", "7",
        "6", "1", "0");
286         SortingMachine<String> mExpected =
        this.createFromArgsRef(ORDER, false,
287                               "1", "7", "5", "5", "5", "8",
        "7", "6", "1", "0");
288         boolean b = m.isInInsertionMode();
289         boolean b1 =
mExpected.isInInsertionMode();
290
291         assertEquals(mExpected, m);
292         assertEquals(b1, b);
293     }
294
295     @Test
296     public final void testOrder1() {
297         SortingMachine<String> m =
        this.createFromArgsTest(ORDER, true, "1",
298                                "7", "5", "5", "5", "8", "7",
        "6", "1", "0");
299         SortingMachine<String> mExpected =

```

```

    this.createFromArgsRef(ORDER, true,
300         "1", "7", "5", "5", "5", "8",
        "7", "6", "1", "0");
301         Comparator<String> c = m.order();
302
303         assertEquals(mExpected, m);
304         assertEquals(ORDER, c);
305     }
306
307     //test empty
308     @Test
309     public final void testOrder2() {
310         SortingMachine<String> m =
        this.createFromArgsTest(ORDER, true);
311         SortingMachine<String> mExpected =
        this.createFromArgsRef(ORDER, true);
312         Comparator<String> c = m.order();
313         Comparator<String> c1 =
        mExpected.order();
314
315         assertEquals(mExpected, m);
316         assertEquals(c1, c);
317     }
318
319     //test nonempty with same elements
320     @Test
321     public final void testOrder3() {
322         SortingMachine<String> m =
        this.createFromArgsTest(ORDER, true, "1",
323         "1", "1", "1", "1");
324         SortingMachine<String> mExpected =
        this.createFromArgsRef(ORDER, true,
325         "1", "1", "1", "1", "1");
326         Comparator<String> c = m.order();
327

```

```

328         assertEquals(mExpected, m);
329         assertEquals(ORDER, c);
330     }
331
332     //test empty with extraction mode
333     @Test
334     public final void testOrder4() {
335         SortingMachine<String> m =
336         this.createFromArgsTest(ORDER, false);
337         SortingMachine<String> mExpected =
338         this.createFromArgsRef(ORDER, false);
339         Comparator<String> c = m.order();
340
341         assertEquals(mExpected, m);
342         assertEquals(ORDER, c);
343     }
344
345     //test nonempty with extraction mode
346     @Test
347     public final void testOrder5() {
348         SortingMachine<String> m =
349         this.createFromArgsTest(ORDER, false, "1",
350         "7", "5", "5", "5", "8", "7",
351         "6", "1", "0");
352         SortingMachine<String> mExpected =
353         this.createFromArgsRef(ORDER, false,
354         "1", "7", "5", "5", "5", "8",
355         "7", "6", "1", "0");
356         Comparator<String> c = m.order();
357
358         assertEquals(mExpected, m);
359         assertEquals(ORDER, c);
360     }
361
362     @Test

```

```

357     public final void testSize1() {
358         SortingMachine<String> m =
359             this.createFromArgsTest(ORDER, true, "1",
360                                     "7", "5", "5", "5", "8", "7",
361                                     "6", "1", "0");
362         SortingMachine<String> mExpected =
363             this.createFromArgsRef(ORDER, true,
364                                   "1", "7", "5", "5", "5", "8",
365                                   "7", "6", "1", "0");
366         int s = m.size();
367         int s1 = mExpected.size();
368
369         assertEquals(mExpected, m);
370         assertEquals(s1, s);
371         assertEquals(10, s);
372     }
373
374     //test empty with insertion mode
375     @Test
376     public final void testSize2() {
377         SortingMachine<String> m =
378             this.createFromArgsTest(ORDER, true);
379         SortingMachine<String> mExpected =
380             this.createFromArgsRef(ORDER, true);
381         int s = m.size();
382         int s1 = mExpected.size();
383
384         assertEquals(mExpected, m);
385         assertEquals(s1, s);
386         assertEquals(0, s);
387     }
388
389     //test empty with extraction mode
390     @Test
391     public final void testSize3() {

```

```

386         SortingMachine<String> m =
            this.createFromArgsTest(ORDER, false);
387         SortingMachine<String> mExpected =
            this.createFromArgsRef(ORDER, false);
388         int s = m.size();
389         int s1 = 0;
390
391         assertEquals(mExpected, m);
392         assertEquals(s1, s);
393     }
394
395     //test nonempty with extraction mode
396     @Test
397     public final void testSize4() {
398         SortingMachine<String> m =
            this.createFromArgsTest(ORDER, false, "1",
399                                   "7", "5", "5", "5", "8", "7",
            "6", "1", "0");
400         SortingMachine<String> mExpected =
            this.createFromArgsRef(ORDER, false,
401                                 "1", "7", "5", "5", "5", "8",
            "7", "6", "1", "0");
402         int s = m.size();
403         int s1 = 10;
404
405         assertEquals(mExpected, m);
406         assertEquals(s1, s);
407     }
408
409 }
410

```