

```

1 import java.lang.reflect.Constructor;
2
3
4
5
6
7
8
9
10 /**
11  * {@code SortingMachine} represented as a {@code Queue} (using an embedding of
12  * insertion sort), with implementations of primary methods.
13  *
14  * @param <T>
15  *     type of {@code SortingMachine} entries
16  * @mathdefinitions <pre>
17  * IS_TOTAL_PREORDER (
18  *   r: binary relation on T
19  * ) : boolean is
20  *   for all x, y, z: T
21  *     ((r(x, y) or r(y, x)) and
22  *      (if (r(x, y) and r(y, z)) then r(x, z)))
23  *
24  * IS_SORTED (
25  *   s: string of T,
26  *   r: binary relation on T
27  * ) : boolean is
28  *   for all x, y: T where (<x, y> is substring of s) (r(x, y))
29  * </pre>
30  * @convention <pre>
31  * IS_TOTAL_PREORDER([relation computed by $this.machineOrder.compare method]) and
32  * IS_SORTED($this.entries, [relation computed by $this.machineOrder.compare method])
33  * </pre>
34  * @correspondence <pre>
35  * this =
36  *   ($this.insertionMode, $this.machineOrder, multiset_entries($this.entries))
37  * </pre>
38  */
39 public class SortingMachine3<T> extends SortingMachineSecondary<T> {
40
41     /*
42     * Private members -----
43     */
44
45     /**
46     * Insertion mode.
47     */
48     private boolean insertionMode;
49
50     /**
51     * Order.
52     */
53     private Comparator<T> machineOrder;
54
55     /**
56     * Entries.
57     */
58     private Queue<T> entries;
59
60     /**
61     * Creator of initial representation.
62     *
63     * @param order
64     *     total preorder for sorting
65     */
66     private void createNewRep(Comparator<T> order) {

```

```

67         this.insertionMode = true;
68         this.machineOrder = order;
69         this.entries = new QueueLL<T>();
70     }
71
72     /**
73      * Inserts the given {@code T} in the {@code Queue<T>} sorted according to
74      * the given {@code Comparator<T>} and maintains the {@code Queue<T>}
75      * sorted.
76      *
77      * @param <T>
78      *         type of {@code Queue} entries
79      * @param q
80      *         the {@code Queue} to insert into
81      * @param x
82      *         the {@code T} to insert
83      * @param order
84      *         the {@code Comparator} defining the order for {@code T}
85      * @updates q
86      * @requires <pre>
87      * IS_TOTAL_PREORDER([relation computed by order.compare method]) and
88      * IS_SORTED(q, [relation computed by order.compare method])
89      * </pre>
90      * @ensures <pre>
91      * perms(q, #q * <x>) and
92      * IS_SORTED(q, [relation computed by order.compare method])
93      * </pre>
94      */
95     private static <T> void insertInOrder Queue<T> q, T x,
96         Comparator<T> order) {
97         assert q != null : "Violation of: q is not null";
98         assert x != null : "Violation of: x is not null";
99         assert order != null : "Violation of: order is not null";
100
101         boolean in = false;
102         int idx = 0;
103
104         // while index less than length and x not inserted into queue
105         while (idx < q.length() && !in) {
106             // if the front value is alphabetically less than or equal to x
107             if (order.compare(q.front(), x) <= 0) {
108                 q.enqueue(x);
109                 in = true;
110             }
111
112             idx++;
113             // move front to back
114             T temp = q.dequeue();
115             q.enqueue(temp);
116         }
117
118         // put queue back in order
119         while (idx < q.length()) {
120             T temp = q.dequeue();
121             q.enqueue(temp);
122             idx++;
123         }
124     }
125 }

```

```

126
127  /*
128   * Constructors -----
129   */
130
131  /**
132   * Constructor from order.
133   *
134   * @param order
135   *         total preorder for sorting
136   */
137  public SortingMachine3(Comparator<T> order) {
138      this.createNewRep(order);
139  }
140
141  /*
142   * Standard methods -----
143   */
144
145  @SuppressWarnings("unchecked")
146  @Override
147  public final SortingMachine<T> newInstance() {
148      try {
149          Constructor<?> c = this.getClass().getConstructor(Comparator.class);
150          return (SortingMachine<T>) c.newInstance(this.machineOrder);
151      } catch (ReflectiveOperationException e) {
152          throw new AssertionError(
153              "Cannot construct object of type " + this.getClass());
154      }
155  }
156
157  @Override
158  public final void clear() {
159      this.createNewRep(this.machineOrder);
160  }
161
162  @Override
163  public final void transferFrom(SortingMachine<T> source) {
164      assert source != null : "Violation of: source is not null";
165      assert source != this : "Violation of: source is not this";
166      assert source instanceof SortingMachine3<?> : ""
167          + "Violation of: source is of dynamic type SortingMachine3<?>";
168      /*
169       * This cast cannot fail since the assert above would have stopped
170       * execution in that case: source must be of dynamic type
171       * SortingMachine3<?>, and the ? must be T or the call would not have
172       * compiled.
173       */
174      SortingMachine3<T> localSource = (SortingMachine3<T>) source;
175      this.insertionMode = localSource.insertionMode;
176      this.machineOrder = localSource.machineOrder;
177      this.entries = localSource.entries;
178      localSource.createNewRep(localSource.machineOrder);
179  }
180
181  /*
182   * Kernel methods -----
183   */
184

```

```
185     @Override
186     public final void add T x () {
187         assert x != null : "Violation of: x is not null";
188         assert this.isInInsertionMode() : "Violation of: this.insertion_mode";
189
190         this.insertInOrder this.entries, x, this.machineOrder;
191     }
192
193
194     @Override
195     public final void changeToExtractionMode () {
196         assert this.isInInsertionMode() : "Violation of: this.insertion_mode";
197     }
198
199
200     @Override
201     public final T removeFirst () {
202         assert !this
203             .isInInsertionMode() : "Violation of: not this.insertion_mode";
204         assert this.size() > 0 : "Violation of: this.contents /= {}";
205
206         // TODO #4 - remove and return first entry in machine contents
207
208         // This line added just to make the component compilable.
209         return null;
210     }
211
212     @Override
213     public final boolean isInInsertionMode () {
214
215         // TODO #5 - report whether machine is in insertion mode
216
217         // This line added just to make the component compilable.
218         return false;
219     }
220
221     @Override
222     public final Comparator<T> order () {
223
224         // TODO #6 - report order used by machine
225
226         // This line added just to make the component compilable.
227         return null;
228     }
229
230     @Override
231     public final int size () {
232
233         return this.entries.length();
234     }
235
236
237     @Override
238     public final Iterator<T> iterator () {
239         return this.entries.iterator();
240     }
241
242
243
```