```java
import java.util.Comparator;
import java.util.Iterator;

/**
 * Layered implementations of secondary methods for {@code WaitingLine}.
 *
 * <p>
 * Assuming execution-time performance of O(1) for method {@code iterator} and
 * its return value's method {@code next}, execution-time performance of
 * {@code front} as implemented in this class is O(1). Execution-time
 * performance of {@code replaceFront} and {@code flip} as implemented in this
 * class is O(|{@code this}|). Execution-time performance of {@code append} as
 * implemented in this class is O(|{@code q}|). Execution-time performance of
 * {@code sort} as implemented in this class is O(|{@code this}| log
 * |{@code this}|) expected, O(|{@code this}|^2) worst case. Execution-time
 * performance of {@code rotate} as implemented in this class is
 * O({@code distance} mod |{@code this}|).
 *
 * @param <T>
 *            type of {@code WaitingLine} entries
 */
public abstract class WaitingLineSecondary<T> implements WaitingLine<T> {

    /*
     * Private members ---------------------------------------------------------
     */

    /*
     * 2221/2231 assignment code deleted.
     */

    /*
     * Public members ----------------------------------------------------------
     */

    /*
     * Common methods (from Object) --------------------------------------------
     */

    @Override
    public final boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (!(obj instanceof WaitingLine<?>)) {
            return false;
        }
        WaitingLine<?> q = (WaitingLine<?>) obj;
        if (this.length() != q.length()) {
            return false;
        }
        Iterator<T> it1 = this.iterator();
        Iterator<?> it2 = q.iterator();
        while (it1.hasNext()) {
```

```
58              T x1 = it1.next();
59              Object x2 = it2.next();
60              if (!x1.equals(x2)) {
61                  return false;
62              }
63          }
64          return true;
65      }
66
67      // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
68      @Override
69      public int hashCode() {
70          final int samples = 2;
71          final int a = 37;
72          final int b = 17;
73          int result = 0;
74          /*
75           * This code makes hashCode run in O(1) time. It works because of the
76           * iterator order string specification, which guarantees that the (at
77           * most) samples entries returned by the it.next() calls are the same
78           * when the two WaitingLines are equal.
79           */
80          int n = 0;
81          Iterator<T> it = this.iterator();
82          while (n < samples && it.hasNext()) {
83              n++;
84              T x = it.next();
85              result = a * result + b * x.hashCode();
86          }
87          return result;
88      }
89
90      // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
91      @Override
92      public String toString() {
93          StringBuilder result = new StringBuilder("<");
94          Iterator<T> it = this.iterator();
95          while (it.hasNext()) {
96              result.append(it.next());
97              if (it.hasNext()) {
98                  result.append(",");
99              }
100         }
101         result.append(">");
102         return result.toString();
103     }
104
105     /*
106      * Other non-kernel methods ------------------------------------------------
107      */
108
109     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
110     @Override
111     public T front() {
112         assert this.length() > 0 : "Violation of: this /= <>";
113         // return statement line to avoid error
114         return null;
```

```
115
116         /*
117          * 2221/2231 assignment code deleted.
118          */
119     }
120
121     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
122     @Override
123     public T replaceFront(T x) {
124         assert this.length() > 0 : "Violation of: this /= <>";
125         // return statement line to avoid error
126         return x;
127
128         /*
129          * 2221/2231 assignment code deleted.
130          */
131     }
132
133     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
134     @Override
135     public void append(WaitingLine<T> q) {
136         assert q != null : "Violation of: q is not null";
137         assert q != this : "Violation of: q is not this";
138
139         /*
140          * 2221/2231 assignment code deleted.
141          */
142     }
143
144     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
145     @Override
146     public void flip() {
147
148         /*
149          * 2221/2231 assignment code deleted.
150          */
151     }
152
153     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
154     @Override
155     public void sort(Comparator<T> order) {
156         assert order != null : "Violation of: order is not null";
157
158         /*
159          * 2221/2231 assignment code deleted.
160          */
161     }
162
163     // CHECKSTYLE: ALLOW THIS METHOD TO BE OVERRIDDEN
164     @Override
165     public void rotate(int distance) {
166
167         /*
168          * 2221/2231 assignment code deleted.
169          */
170     }
171
```

```
172 }
173
```