```
    import java.time.LocalDateTime;

2. public class AssistantOnShift {
3.
4. // Used for displaying the availability of an assistant.
5. enum StatusValue{
6. FREE,
7. BUSY
8.
   // Formatted yyyy-mm-dd-HH-MM-ss-ns
10.
         private LocalDateTime timeSlot;
11.
         private Assistant assistant;
12.
         private StatusValue status = StatusValue.FREE;
13.
         private static AssistantOnShift[] assistantOnShifts =
  new
14.
        AssistantOnShift[100];
15.
         private static int numAssistantsOnShift = 0;
16.
         * The cconstrutor method for Assistants on Shift.
17.
18.
         * @param timeSlot a LocalDateTime used to store the
19.
  time the assistant on the
20.
        shift.
         * @param assistant this is the class representing the
21.
  actual assistant.
22.
         */
23.
         public AssistantOnShift(LocalDateTime timeSlot,
  Assistant assistant){
24.
         this.timeSlot = checkTimeSlot(timeSlot);
25.
         this.assistant = assistant;
26.
         addAssistantsOnShift(this);
27.
         iterateNumAssistantsOnShift();
28.
         }
29.
         * The getter method for time slots
30.
31.
32.
         * @return the timeslot of the assistant on shift.
33.
34.
         public LocalDateTime getTimeSlot(){
35.
         return timeSlot;
36.
         }
         /**
37.
         * The getter method for the assistant
38.
39.
         * @return the assistant
40.
41.
42.
         public Assistant getAssistant(){
43.
         return assistant;
44.
45.
         * The getter method for the status.
46.
47.
48.
         * @return the status from the enum, StatusValue.
49.
50.
         public StatusValue getStatus(){
```

```
51.
         return status;
52.
         }
53.
         * The getter method for the list of assistants on
54.
  shift.
55.
         * @return the list of assistants on shift.
56.
57.
58.
         public static AssistantOnShift[]
  getAssistantOnShifts(){
         return assistantOnShifts;
60.
61.
         * The getter method for the number of assistants on
62.
  shift
63.
64.
         * @return the number of assistants on shift.
65.
66.
         public static int getnumAssistantsOnShifts(){
67.
         return numAssistantsOnShift;
68.
         }
         /**
69.
         * The setter method for the time slot.
70.
71.
         * @param timeSlot a new time slot for the assistant
72.
  on shift.
73.
         public void setTimeSlot(LocalDateTime timeSlot){
74.
75.
         this.timeSlot = checkTimeSlot(timeSlot);
76.
         }
77.
78.
         * The setter method for the status of the assistant
  on shift.
79.
         * @param busy true - sets the status to busy, false -
80.
  sets the status to free.
81.
82.
         public void setStatus(boolean busy){
83.
         if (busy){
84.
         status = StatusValue.BUSY;
85.
         }else{
86.
         status = StatusValue.FREE;
87.
88.
         }
         /**
89.
         * The method to add an assistant on shift to the list
  of assistants on shift.
91.
         * @param assistantOnShift this is an assistant on
92.
  shift to be added to the
93.
        list.
94.
         */
         private static void
95.
  addAssistantsOnShift(AssistantOnShift assistantOnShift){
```

```
assistantOnShifts[numAssistantsOnShift] =
96.
  assistantOnShift:
97.
         }
98.
        * The method to remove an assistant on shift from the
99.
  list of assistants on
        shift.
100.
101.
           @param assistantOnShift this is an assistant on
102.
  shift to be removed from the
103.
        list.
104.
         */
         public static void
105.
  removeAssistantsOnShift(AssistantOnShift assistantOnShift){
         if (assistantOnShift.status != StatusValue.FREE){
106.
107.
         throw new IllegalArgumentException("To remove an
  assistant on shift it
108.
        cannot be busy.");
109.
110.
         // linear search to find booking in the array.
111.
         boolean found = false;
112.
         int index = -1;
113.
         while (!found){
         index += 1;
114.
         if (assistantOnShifts[index] == assistantOnShift){
115.
116.
         found = true;
117.
         }else if (index >= numAssistantsOnShift){
118.
         throw new IllegalArgumentException("The assistant on
  shift was not
119.
        found.");
120.
         }
121.
         // Shifting the last elements.
122.
         for (int i=index;i<=numAssistantsOnShift-1;i++){</pre>
123.
124.
         assistantOnShifts[i] = assistantOnShifts[i+1];
125.
126.
         assistantOnShifts[numAssistantsOnShift] = null;
127.
         numAssistantsOnShift -= 1;
128.
         }
129.
         * A method to increment the number of assistants
130.
         * Usually called in the constructor of
131.
  AssistantOnShift.
132.
         * /
         private static void iterateNumAssistantsOnShift(){
133.
134.
         numAssistantsOnShift += 1;
135.
         }
        /**
136.
         * The method to check if a time slot is valid.
137.
138.
         * @param timeSlot this is the timeslot being checked
139.
140.
         * @return this will return the timeslot if it is
  valid.
141.
```

```
private LocalDateTime checkTimeSlot(LocalDateTime
142.
  timeSlot){
         LocalDateTime currentTime = LocalDateTime.now();
143.
144.
         if (timeSlot.isBefore(currentTime)){
         throw new IllegalArgumentException("The time slot has
145.
  already
        passed.");
146.
         }else if (timeSlot.getHour() < 7 ||</pre>
147.
  timeSlot.getHour() > 10){
148.
         throw new IllegalArgumentException("The time slot
  must be between 7 and
149.
        10.");
         }else if (timeSlot.getMinute() != 0 ||
150.
  timeSlot.getSecond() != 0){
151.
         throw new IllegalArgumentException("The time slot
  must be at the start
152.
        of the hour.");
153.
         for (int i=0;i<numAssistantsOnShift;i++){</pre>
154.
         if (timeSlot == assistantOnShifts[i].getTimeSlot() &&
155.
156.
        assistantOnShifts[i].getAssistant().equals(assistant))
157.
         throw new IllegalArgumentException("Duplicate time
  slot.");
158.
159.
160.
         return timeSlot;
161.
         }
         162.
         * This is the static method to format the date
163.
  correctly
164.
         * @param time this is the time that needs to be
165.
  formatted.
166.
         * @return A string of the formatted date.
167.
168.
         public static String formatDate(LocalDateTime time){
         return time.getDayOfMonth()+"/"+time.getMonthValue()
169.
  +"/"+time.getYear()+"
        "+time.getHour()+":"+time.getMinute()+"0";
170.
171.
172.
173.
         * This is the method to convert the index and the
  sequential id
174.
         * when removing an assistant on shift.
175.
         * @param status this is the status to be filtering
  the assistants on shift by.
         * @return this is the list of indexes used to convert
  the id to the idex.
178.
         */
179.
         public static int[] convertIndex(StatusValue status){
180.
         int[] indexList = new int[numAssistantsOnShift+1];
181.
         int index = 11;
```

```
182.
         int i;
183.
         for (i=0;i<numAssistantsOnShift;++i){</pre>
184.
         if (assistantOnShifts[i].status == status){
185.
         indexList[i] = index++;
186.
187.
         indexList[numAssistantsOnShift] = index-11; // To
188.
  store the actual length
189.
        of the list.
190.
         return indexList;
191.
         }
192.
193.
         * This is the overloaded method to return the the
  string of all assistants on
194.
        shift
195.
         * @return a string of all assistants on shift.
196.
197.
198.
         public static String toStringAll(){
         String allAssistantsOnShift = "Assistants on
199.
  Shift-\n";
200.
         for (int i=0;i<numAssistantsOnShift;i++){</pre>
201.
         allAssistantsOnShift =
  allAssistantsOnShift.concat((i+11)+".
        "+assistantOnShifts[i].toString()+"\n");
202.
203.
204.
         return allAssistantsOnShift;
205.
         }
206.
         * This is the overloaded method to return the the
207.
  string of all assistants on
208.
        shift
209.
         * @param status this is the status to filter the
210.
  assistants on shift by.
         * @return a string of all assistants on shift that
211.
  are valid for the status.
212.
         public static String toStringAll(StatusValue status){
213.
         String allAssistantsOnShift = "Assistants on
214.
  Shift-\n";
215.
         int index = 0;
216.
         for (int i=0;i<numAssistantsOnShift;i++){</pre>
217.
         if (assistantOnShifts[i].status == status){
218.
         allAssistantsOnShift =
  allAssistantsOnShift.concat((index+11)+".
        "+assistantOnShifts[i].toString()+"\n");
219.
220.
         index++;
221.
222.
223.
         return allAssistantsOnShift:
224.
225.
226.
         * This is the method to return a string of an
```

```
assistant on shift.
227.  *
228.  * @return a string of an assistant on shift.
229.  */
230.  public String toString(){
231.  return "| "+formatDate(timeSlot)+" | "+status+" |
    "+assistant.getEmail()+"
232.  |";
233.  }
234. }
```