```
1. import java.time.LocalDateTime;
2. public class BookableRoom {
3.  // Used for the displaying the occupancy of the roooms
4.  enum StatusValue{
5.  EMPTY,
6.  AVAILABLE,
7.  FULL
8.  }
9.  // Formatted yyyy-mm-dd-HH-MM-ss-ns
10.      private LocalDateTime timeSlot;
11.      private int occupancy = 0;
12.      private StatusValue status;
13.
14.      private Room room;
15.      private static BookableRoom[] bookableRooms = new
   BookableRoom[100];
16.      private static int numBookableRooms = 0;
17.      /**
18.      * The constructor for the class BookableRooms.
19.      *
20.      * @param room this is the room that will be booked.
21.      * @param timeSlot this is the timeslot that the room
   can be booked from.
22.      */
23.      public BookableRoom(Room room, LocalDateTime
   timeSlot){
24.      this.room = room;
25.      this.timeSlot = checkTimeSlot(timeSlot);
26.      status = createStatus();
27.      addBookableRoom(this);
28.      interateNumBookableRooms();
29.      }
30.      /**
31.      * The getter method for the time slot
32.      *
33.      * @return the time slot that the room can be booked
   from.
34.      */
35.      public LocalDateTime getTimeSlot(){
36.      return timeSlot;
37.      }
38.      /**
39.      * The getter method for the occupancy.
40.      *
41.      * @return the current occupancy at the time slot
42.      */
43.      public int getOccupancy(){
44.      return occupancy;
45.      }
46.      /**
47.      * The getter method for the status.
48.      *
49.      * @return the status of the bookable room.
50.      */
```

```
51.      public StatusValue getStatus(){
52.      return status;
53.      }
54.      /**
55.      * The getter method for the room.
56.      *
57.      * @return the room that will be booked.
58.      */
59.      public Room getRoom(){
60.      return room;
61.      }
62.      /**
63.      * The static getter method for the list of bookable
   rooms.
64.      *
65.      * @return the list of bookable rooms.
66.      */
67.      public static BookableRoom[] getBookableRooms(){
68.      return bookableRooms;
69.      }
70.      /**
71.      * The static getter method for the number of bookable
   rooms
72.      *
73.      * @return the number of bookable rooms.
74.      */
75.      public static int getNumBookableRooms(){
76.      return numBookableRooms;
77.      }
78.      /**
79.      * The setter method for the occupancy of the room.
80.      *
81.      * @param occupancy the number of tests currently
   taking place in the room.
82.      */
83.      public void setOccupancy(int occupancy){
84.      this.occupancy = checkOccupancy(occupancy);
85.      this.status = createStatus();
86.      }
87.      /**
88.      * The static method to incement the number of
   bookable rooms.
89.      */
90.      private static void interateNumBookableRooms(){
91.      numBookableRooms += 1;
92.      }
93.      /**
94.      * The static method to add a bookable room to the
   list of bookable rooms.
95.      *
96.      * @param bookableRoom a bookable room to be added to
   the list.
97.      */
98.      private static void addBookableRoom(BookableRoom
```

```
   bookableRoom){
99.      bookableRooms[numBookableRooms] = bookableRoom;
100.     }
101.     /**
102.     * The static method to remove a bookable room from
   the list of bookable rooms.
103.     *
104.     * @param bookableRoom a bookable room to be removed
   from the list.
105.     */
106.     public static void removeBookableRoom(int index){
107.     if (bookableRooms[index].status != StatusValue.EMPTY)
   {
108.     throw new IllegalArgumentException("To remove a
   bookable room it must
109.    be empty.");
110.     }
111.     // Shifting the last elements.
112.     for (int i=index;i<=numBookableRooms-1;i++){
113.     bookableRooms[i] = bookableRooms[i+1];
114.     }
115.     bookableRooms[numBookableRooms] = null;
116.     numBookableRooms -= 1;
117.     }
118.     /**
119.     * The method to check if the occupancy of the room is
   valid.
120.     *
121.     * @param occupancy the occupancy being checked.
122.     * @return the occupancy returned if it is valid.
123.     */
124.     private int checkOccupancy(int occupancy){
125.     if (occupancy < 0){
126.     throw new IllegalArgumentException("Occupancy is
   smaller then zero.");
127.     }
128.     return occupancy;
129.     }
130.     /**
131.     * The method to check what state the status should be
   in.
132.     *
133.     * @return the new status of the room
134.     */
135.     private StatusValue createStatus(){
136.     if (occupancy == 0){
137.     return StatusValue.EMPTY;
138.     }else if (occupancy < room.getCapacity()){
139.     return StatusValue.AVAILABLE;
140.     }else if (occupancy == room.getCapacity()){
141.     return StatusValue.FULL;
142.     }else{
143.     throw new IllegalArgumentException("Occupancy cannot
   be greater then
```

```
144.    capacity of a room.");
145.     }
146.     }
147.     /**
148.     * The method to check if a time-slot is valid.
149.     *
150.     * @param timeSlot a new time-slot.
151.     * @return the time-slot if it is valid.
152.     */
153.     private LocalDateTime checkTimeSlot(LocalDateTime
   timeSlot){
154.     LocalDateTime currentTime = LocalDateTime.now();
155.     if (timeSlot.isBefore(currentTime)){
156.     throw new IllegalArgumentException("The time slot has
   already
157.    passed.");
158.     }else if (timeSlot.getHour() < 7 ||
   timeSlot.getHour() > 10){
159.     throw new IllegalArgumentException("The time slot
   must be between 7 and
160.    10.");
161.     }else if (timeSlot.getMinute() != 0 ||
   timeSlot.getSecond() != 0){
162.     throw new IllegalArgumentException("The time slot
   must be at the start
163.    of the hour.");
164.     }
165.     for (int i=0;i<numBookableRooms;i++){
166.     if (timeSlot == bookableRooms[i].getTimeSlot() &&
167.    bookableRooms[i].getRoom().equals(room)){
168.     throw new IllegalArgumentException("Duplicate time
   slot.");
169.     }
170.     }
171.     return timeSlot;
172.     }
173.     /**
174.     * This is the static method to format the date
   correctly
175.     *
176.     * @param time this is the time that needs to be
   formatted.
177.     * @return A string of the formatted date.
178.     */
179.     public static String formatDate(LocalDateTime time){
180.     return time.getDayOfMonth()+"/"+time.getMonthValue()
   +"/"+time.getYear()+"
181.    "+time.getHour()+":"+time.getMinute()+"0";
182.     }
183.     /**
184.     * This is the method to convert the index and the
   sequential id
185.     * when removing an assistant on shift.
186.     *
```

```
187.     * @param status this is the status to be filtering
   the bookable room by.
188.     * @return this is the list of indexes used to convert
   the id to the idex.
189.     */
190.     public static int[] convertIndex(StatusValue status){
191.     int[] indexList = new int[numBookableRooms+1];
192.     int index = 11;
193.     int i;
194.     for (i=0;i<numBookableRooms;++i){
195.     if (bookableRooms[i].status == status){
196.     indexList[i] = index++;
197.     }
198.     }
199.     indexList[numBookableRooms] = index-11; // To store
   the actual length of
200.    the list.
201.     return indexList;
202.     }
203.
204.     /**
205.     * This is the overloaded method to return the the
   string of all bookable
206.    rooms.
207.     *
208.     * @return a string of all bookable rooms.
209.     */
210.     public static String toStringAll(){
211.     String allBookableRooms = "Bookable Rooms-\n";
212.     for (int i=0;i<numBookableRooms;i++){
213.     allBookableRooms = allBookableRooms.concat((i+11)+".
214.    "+bookableRooms[i].toString()+"\n");
215.     }
216.     return allBookableRooms;
217.     }
218.     /**
219.     * This is the overloaded method to return the the
   string of all bookable rooms
220.     *
221.     * @param status this is the status to filter the
   bookable rooms by.
222.     * @return a string of bookable rooms that are valid
   for the status.
223.     */
224.     public static String toStringAll(StatusValue status){
225.     String allBookableRooms = "Bookable Rooms-\n";
226.     int index = 0;
227.     for (int i=0;i<numBookableRooms;i++){
228.     if (bookableRooms[i].status == status){
229.     allBookableRooms =
   allBookableRooms.concat((index+11)+".
230.    "+bookableRooms[i].toString()+"\n");
231.     index++;
232.     }
```

```
233.     }
234.     return allBookableRooms;
235.     }
236.     /**
237.     * This is the method to return a string of a bookable
   room.
238.     *
239.     * @return a string of a bookable room.
240.     */
241.     public String toString(){
242.     return "| "+formatDate(timeSlot)+" | "+status+" |
   "+room.getCode()+" |
243.    occupancy: "+occupancy+" |";
244.     }
245.     }
```