1 Assistant.java

```java
/**
 * The type Assistant.
 */
public class Assistant {

    private String name;

    private String email;

    /**
     * Print assistant template string.
     *
     * @return Template for assistant details.
     */
    public String printAssistantTemplate() {
        String assistantTemplate = " | " + name + " | " + email + " |";

        return assistantTemplate;
    }

    /**
     * Constructor to instantiate an assistant.
     *
     * @param name name of assistant.
     * @param email email of assistant.
     */
    public Assistant(String name, String email) {
        this.name = name;
        this.email = email;
    }

    /**
     * Gets email.
     *
     * @return email of assistant object.
     */
    public String getEmail() {
        return email;
    }
}
```

2 Room.java

```java
/**
 * The type Room.
 */
public class Room {

    private String code;

    private int capacity;

```

```
10      /**
11       * Print room template string.
12       *
13       * @return Template for room details.
14       */
15      public String printRoomTemplate() {
16          String RoomTemplate = " | " + code + " | capacity: " + capacity + " |";
17
18          return RoomTemplate;
19      }
20
21      /**
22       * Constructor to instantiate a room.
23       *
24       * @param code    code of room.
25       * @param capacity capacity of room.
26       */
27      public Room(String code, int capacity) {
28          this.code = code;
29          this.capacity = capacity;
30      }
31
32
33      /**
34       * getter for code.
35       *
36       * @return code of room object.
37       */
38      public String getCode() {
39          return code;
40      }
41
42      /**
43       * getter for capacity of room.
44       *
45       * @return capacity of room object.
46       */
47      public int getCapacity() {
48          return capacity;
49      }
50  }
```

3 UniversityResources.java

```
1   /**
2    * The type University resources.
3    */
4   public class UniversityResources {
5
6       private Assistant[] assistants;
7
8       private Room[] rooms;
9
10      /**
```

```
11      * Constructor to instantiate university resources.
12      *
13      * @param assistants list of assistants.
14      * @param rooms     list of rooms.
15      */
16     public UniversityResources(Assistant[] assistants, Room[] rooms) {
17         this.assistants = assistants;
18         this.rooms = rooms;
19     }
20
21     /**
22      * getter for rooms.
23      *
24      * @return list of rooms.
25      */
26     public Room[] getRooms(){
27         return rooms;
28     }
29
30     /**
31      * getter for assistants.
32      *
33      * @return list of assistants.
34      */
35     public Assistant[] getAssistants(){
36         return assistants;
37     }
38
39     /**
40      * String containing details of rooms.
41      *
42      * @return String with information needed for adding bookable room to system.
43      */
44     public String addBookableRoomsString() {
45         String addBookableRoomsString = "University of Knowledge - COVID test";
46         addBookableRoomsString += "\n\n" + "Adding bookable room" + "\n\n" + "List of Rooms:\n";
47         int a = 10;
48         for (int i = 0; i < rooms.length; i++) {
49             a++;
50             addBookableRoomsString += a + "." + this.rooms[i].printRoomTemplate() + "\n";
51         }
52         addBookableRoomsString += "Please, enter one of the following:\n\n";
53         addBookableRoomsString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and a time
               (HH:MM),\n" + "separated by a white space.\n";
54         addBookableRoomsString += "0. Back to main menu.\n" + "-1. Quit application.\n";
55         return addBookableRoomsString;
56     }
57
58     /**
59      * Add bookable room invalid length string string.
60      *
61      * @return String displayed if input for adding bookable room is invalid.
62      */
63     public String addBookableRoomInvalidLengthString() {
64         String addBookableRoomInvalidLengthString = "Error!\nAmount of inputs is invalid.\nPlease, enter one
```

```
            of the following:\n\n";
65          addBookableRoomInvalidLengthString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and
                a time (HH:MM), \nseparated by a white space.\n";
66          addBookableRoomInvalidLengthString += "0. Back to main menu.\n" + "-1. Quit application.\n";
67          return addBookableRoomInvalidLengthString;
68      }
69
70      /**
71       * Add bookable room invalid id string string.
72       *
73       * @return String displayed if input for adding bookable room is invalid.
74       */
75      public String addBookableRoomInvalidIDString() {
76          String addBookableRoomInvalidLengthString = "Error!\nInvalid sequential ID.\nPlease, enter one of
                the following:\n\n";
77          addBookableRoomInvalidLengthString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and
                a time (HH:MM), \nseparated by a white space.\n";
78          addBookableRoomInvalidLengthString += "0. Back to main menu.\n" + "-1. Quit application.\n";
79          return addBookableRoomInvalidLengthString;
80      }
81
82      /**
83       * Add bookable room invalid date string string.
84       *
85       * @return String displayed if input for adding bookable room is invalid.
86       */
87      public String addBookableRoomInvalidDateString(){
88          String addBookableRoomInvalidDateString = "Error!\nInvalid Date input.\nPlease, enter one of the
                following:\n\n";
89          addBookableRoomInvalidDateString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and a
                time (HH:MM), \nseparated by a white space.\n";
90          addBookableRoomInvalidDateString += "0. Back to main menu.\n" + "-1. Quit application.\n";
91          return addBookableRoomInvalidDateString;
92      }
93
94      /**
95       * Add bookable room invalid time string string.
96       *
97       * @return String displayed if input for adding bookable room is invalid.
98       */
99      public String addBookableRoomInvalidTimeString() {
100         String addBookableRoomInvalidTimeString = "Error!\nInvalid Time input.\nPlease, enter one of the
                following:\n\n";
101         addBookableRoomInvalidTimeString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and a
                time (HH:MM), \nseparated by a white space.\n";
102         addBookableRoomInvalidTimeString += "0. Back to main menu.\n" + "-1. Quit application.\n";
103         return addBookableRoomInvalidTimeString;
104     }
105
106     /**
107      * Bookable room already exists string string.
108      *
109      * @return String displayed if bookable room for the chosen room already exists for the chosen date and
            time.
110      */
```

```
111     public String bookableRoomAlreadyExistsString() {
112         String bookableRoomAlreadyExistsString = "Error!\nBookable room already exists.\nPlease, enter one
                of the following:\n\n";
113         bookableRoomAlreadyExistsString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and a
                time (HH:MM), \nseparated by a white space.\n";
114         bookableRoomAlreadyExistsString += "0. Back to main menu.\n" + "-1. Quit application.\n";
115         return bookableRoomAlreadyExistsString;
116     }
117
118     /**
119      * Assistant on shift already exists string string.
120      *
121      * @return String displayed if assistant on shift for chosen assistant already exists for the chosen
             date.
122      */
123     public String assistantOnShiftAlreadyExistsString() {
124         String assistantOnShiftAlreadyExistsString = "Error!\nAssistant already has a shift in that
                day.\nPlease, enter one of the following:\n\n";
125         assistantOnShiftAlreadyExistsString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and
                a time (HH:MM),\nseparated by a white space.\n";
126         assistantOnShiftAlreadyExistsString += "0. Back to main menu.\n-1. Quit application.\n";
127         return assistantOnShiftAlreadyExistsString;
128     }
129
130     /**
131      * Add assistant on shift invalid id string string.
132      *
133      * @return String displayed if input for adding assistant on shit is invalid.
134      */
135     public String addAssistantOnShiftInvalidIDString(){
136         String addAssistantOnShiftInvalidIDString = "Error!\nInvalid sequential ID.\nPlease, enter one of
                the following:\n\n";
137         addAssistantOnShiftInvalidIDString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and
                a time (HH:MM),\nseparated by a white space.\n";
138         addAssistantOnShiftInvalidIDString += "0. Back to main menu.\n-1. Quit application.\n";
139         return addAssistantOnShiftInvalidIDString;
140     }
141
142     /**
143      * Add assistant on shift invalid length string string.
144      *
145      * @return String displayed if input for adding assistant on shit is invalid.
146      */
147     public String addAssistantOnShiftInvalidLengthString(){
148         String addAssistantOnShiftInvalidLengthString = "Error!\nAmount of inputs is invalid.\nPlease, enter
                one of the following:\n\n";
149         addAssistantOnShiftInvalidLengthString += "The sequential ID listed to a room, a date (dd/mm/yyyy),
                and a time (HH:MM),\nseparated by a white space.\n";
150         addAssistantOnShiftInvalidLengthString += "0. Back to main menu.\n-1. Quit application.\n";
151         return addAssistantOnShiftInvalidLengthString;
152     }
153
154     /**
155      * Add assistant on shift invalid date string string.
156      *
```

```
157      * @return String displayed if input for adding assistant on shit is invalid.
158      */
159     public String addAssistantOnShiftInvalidDateString(){
160         String addAssistantOnShiftInvalidDateString = "Error!\nInvalid Date input.\nPlease, enter one of the
                following:\n\n";
161         addAssistantOnShiftInvalidDateString += "The sequential ID listed to a room, a date (dd/mm/yyyy),
                and a time (HH:MM),\nseparated by a white space.\n";
162         addAssistantOnShiftInvalidDateString += "0. Back to main menu.\n-1. Quit application.\n";
163         return addAssistantOnShiftInvalidDateString;
164     }
165
166     /**
167      * Add assistants on shift string.
168      *
169      * @return String with information for adding assistant on shift.
170      */
171     public String addAssistantsOnShift() {
172         String addAssistantsOnShiftString = "University of Knowledge - COVID test";
173         addAssistantsOnShiftString += "\n\n" + "List of Assistants:\n";
174         int a = 10;
175         for (int i = 0; i < assistants.length; i++) {
176             a++;
177             addAssistantsOnShiftString += a + "." + this.assistants[i].printAssistantTemplate() + "\n";
178         }
179         addAssistantsOnShiftString += "Please, enter one of the following:\n\n";
180         addAssistantsOnShiftString += "The sequential ID of an assistant and date (dd/mm/yyyy), separated by
                a white space.\n";
181         addAssistantsOnShiftString += "0. Back to main menu.\n" + "-1. Quit application.\n";
182         return addAssistantsOnShiftString;
183     }
184
185
186 }
```

4 BookingSystem.java

```
1  import java.io.IOException;
2  import java.util.ArrayList;
3
4  /**
5   * The type Booking system.
6   */
7  public class BookingSystem {
8
9      private ArrayList<BookableRoom> bookableRooms;
10
11     private ArrayList<AssistantOnShift> assistantsOnShift;
12
13     private ArrayList<Booking> bookings;
14
15
16     /**
17      * Constructor for instantiating the booking system.
18      *
```

```
19      * @param bookableRooms  list of bookable rooms.
20      * @param assistantsOnShift list of assistants on shift.
21      * @param bookings       list of bookings.
22      */
23     public BookingSystem(ArrayList<BookableRoom> bookableRooms, ArrayList<AssistantOnShift>
            assistantsOnShift, ArrayList<Booking> bookings) {
24         this.bookableRooms = bookableRooms;
25         this.assistantsOnShift = assistantsOnShift;
26         this.bookings = bookings;
27     }
28
29
30     /**
31      * getter for list of bookable rooms.
32      *
33      * @return array list of bookable rooms.
34      */
35     public ArrayList<BookableRoom> getBookableRooms() {
36         return bookableRooms;
37     }
38
39     /**
40      * getter for list of assistants on shift.
41      *
42      * @return array list of assistants on shift.
43      */
44     public ArrayList<AssistantOnShift> getAssistantsOnShift() {
45         return assistantsOnShift;
46     }
47
48     /**
49      * getter for list of bookings.
50      *
51      * @return array list of bookings.
52      */
53     public ArrayList<Booking> getBookings() {
54         return bookings;
55     }
56
57     /**
58      * method for checking the validity of date input.
59      *
60      * @param date date input from user.
61      * @return whether the date is valid or not in boolean.
62      */
63     public boolean checkDateValidity(String date) {
64         String datePattern = "\\d{2}/\\d{2}/\\d{4}";
65         return date.matches(datePattern);
66
67     }
68
69     /**
70      * method for checking the validity of time input.
71      *
72      * @param time time input from user.
```

```
     * @return whether the time input is valid or not in boolean.
     */
    public boolean checkTimeValidity(String time) {
        int a = 0;
        String[] times = {"07:00", "08:00", "09:00"};
        for (int i = 0; i < times.length; i++) {
            if (time.equals(times[i])) {
                a++;
            }
        }

        return a == 1;

    }

    /**
     * This method prints the String for main menu of the booking app.
     */
    public void loadMainMenu() {
        String mainMenu = "University of Knowledge - COVID test";
        mainMenu += "\n" + "\n" + "Manage Bookings" + "\n" + "\n";
        mainMenu += "Please, enter the number to select your option:" + "\n" + "\n";
        mainMenu += "To manage Bookable Rooms:" + "\n" + "1. List" + "\n" + "2. Add" + "\n" + "3. Remove" +
            "\n";
        mainMenu += "To manage Assistants on Shift:" + "\n" + "4. List" + "\n" + "5. Add" + "\n" + "6.
            Remove" + "\n";
        mainMenu += "To manage Bookings:" + "\n" + "7. List" + "\n" + "8. Add" + "\n" + "9. Remove" + "\n" +
            "10. Conclude" + "\n";
        mainMenu += "After selecting one the options above, you will be presented other screens." + "\n";
        mainMenu += "If you press 0, you will be able to return to this main menu." + "\n";
        mainMenu += "Press -1 (or ctrl+c) to quit this application." + "\n";
        System.out.println(mainMenu);
    }


    /**
     * This method shows the list of bookable rooms.
     */
    public void listBookableRooms() {
        String listBookableRoomsString = "University of Knowledge - COVID test";
        listBookableRoomsString = listBookableRoomsString + "\n" + "\n" + "List of Bookable Rooms:\n";
        int a = 10;
        for (int i = 0; i < bookableRooms.size(); i++) {
            a++;
            listBookableRoomsString += a + "." + this.bookableRooms.get(i).printBookableRoomTemplate() +
                "\n";
        }
        listBookableRoomsString += "\n" + "0. Back to main menu." + "\n" + "-1. Quit application." + "\n";
        System.out.println(listBookableRoomsString);
    }

    /**
     * Adds bookable room to the list of bookable rooms.
     *
     * @param date String representing date of bookable room.
```

```
124         * @param time String representing time of bookable room.
125         * @param room room associated with the bookable room.
126         */
127        public void addBookableRoom(String date, String time, Room room) {
128            BookableRoom bookableRoom1 = new BookableRoom(room, 0, "EMPTY", date, time);
129            bookableRooms.add(bookableRoom1);
130            System.out.println(bookableRoom1.addedBookableRoomString());
131        }
132
133        /**
134         * Method for removing a bookable room if input from user for removing bookable room is valid.
135         * Also prints information following the removal of the bookable room.
136         *
137         * @param bookableRoom bookable room that is being removed.
138         */
139        public void removeBookableRoom(BookableRoom bookableRoom){
140            BookableRoom bookableRoom1 = bookableRoom;
141            bookableRooms.remove(bookableRoom);
142            String latestRemovedBookableRoomString = "Bookable Room removed successfully:\n";
143            latestRemovedBookableRoomString += bookableRoom1.printBookableRoomTemplate();
144            latestRemovedBookableRoomString += "\nPlease, enter one of the following:\n\n";
145            latestRemovedBookableRoomString += "The sequential ID to select the bookable room to be removed.\n";
146            latestRemovedBookableRoomString += "0. Back to main menu.\n-1. Quit application.\n";
147            clearTerminal();
148            removeBookableRoomsString();
149            System.out.println(latestRemovedBookableRoomString);
150        }
151
152        /**
153         * Remove bookable room error string string.
154         *
155         * @return String for invalid input for removing bookable room.
156         */
157        public String removeBookableRoomErrorString(){
158            String removeBookableRoomErrorString = "Error!\nInvalid sequential ID.\n";
159            removeBookableRoomErrorString += "Please, enter one of the following:\n\n";
160            removeBookableRoomErrorString += "The sequential ID to select the bookable room to be removed.\n";
161            removeBookableRoomErrorString += "0. Back to main menu.\n -1. Quit application.\n";
162            return removeBookableRoomErrorString;
163        }
164
165        /**
166         * Prints String containing information for removal on bookable room.
167         */
168        public void removeBookableRoomsString() {
169            String removeBookableRoomsString = "University of Knowledge - COVID test\n\n" + "List of Empty
                   Bookable Rooms:\n";
170            int a = 10;
171            for (int i = 0; i < bookableRooms.size(); i++) {
172                if (bookableRooms.get(i).getBookableRoomStatus().equals("EMPTY")) {
173                    a++;
174                    removeBookableRoomsString += a + "." + this.bookableRooms.get(i).printBookableRoomTemplate()
                           + "\n";
175                }
176            }
```

```
177         removeBookableRoomsString += "Removing bookable room\n\n" + "Please, enter one of the
                following:\n\n";
178         removeBookableRoomsString += "The sequential ID to select the bookable room to be removed.\n";
179         removeBookableRoomsString += "0. Back to main menu.\n" + "-1. Quit application.\n";
180         System.out.println(removeBookableRoomsString);
181     }
182
183     /**
184      * Shows list of assistants on shift to user.
185      */
186     public void listAssistantsOnShift() {
187         String listAssistantsOnShiftString = "University of Knowledge - COVID test\n\n" + "List of Assistant
                on Shifts:\n";
188         int a = 10;
189         for (int i = 0; i < assistantsOnShift.size(); i++) {
190             a++;
191             listAssistantsOnShiftString += a + "." +
                    this.assistantsOnShift.get(i).printAssistantOnShiftTemplate() + "\n";
192         }
193         listAssistantsOnShiftString += "\n" + "0. Back to main menu." + "\n" + "-1. Quit application." +
                "\n";
194         System.out.println(listAssistantsOnShiftString);
195     }
196
197     /**
198      * Method for adding assistant on shift after valid input.
199      *
200      * @param assistant assistant for assistant on shift.
201      * @param date    date of assistant on shift.
202      */
203     public void addAssistantOnShift(Assistant assistant, String date) {
204         AssistantOnShift assistantOnShift1 = new AssistantOnShift(assistant, date, "FREE", "07:00");
205         AssistantOnShift assistantOnShift2 = new AssistantOnShift(assistant, date, "FREE", "08:00");
206         AssistantOnShift assistantOnShift3 = new AssistantOnShift(assistant, date, "FREE", "09:00");
207         assistantsOnShift.add(assistantOnShift1);
208         assistantsOnShift.add(assistantOnShift2);
209         assistantsOnShift.add(assistantOnShift3);
210         String addedAssistantOnShiftString = "Assistant on Shift added successfully:\n";
211         addedAssistantOnShiftString += assistantOnShift1.printAssistantOnShiftTemplate() + "\n";
212         addedAssistantOnShiftString += assistantOnShift2.printAssistantOnShiftTemplate() + "\n" +
                assistantOnShift3.printAssistantOnShiftTemplate();
213         addedAssistantOnShiftString += "\nPlease, enter one of the following:\n\n";
214         addedAssistantOnShiftString += "The sequential ID of an assistant and date (dd/mm/yyyy), separated
                by a white space.\n";
215         addedAssistantOnShiftString += "0. Back to main menu.\n-1. Quit application.\n";
216         System.out.println(addedAssistantOnShiftString);
217     }
218
219     /**
220      * Method for removing assistant on shift after valid input.
221      *
222      * @param assistantOnShift assistant on shift that is removed.
223      */
224     public void removeAssistantOnShift(AssistantOnShift assistantOnShift){
225         AssistantOnShift assistantOnShift1 = assistantOnShift;
```

```
226         assistantsOnShift.remove(assistantOnShift);
227         String latestRemovedAssistantOnShiftString = "Assistant on Shift removed successfully:\n\n";
228         latestRemovedAssistantOnShiftString += assistantOnShift1.printAssistantOnShiftTemplate();
229         latestRemovedAssistantOnShiftString += "\nPlease, enter one of the following:\n\n";
230         latestRemovedAssistantOnShiftString += "The sequential ID to select the assistant on shift to be
                removed.\n\n";
231         latestRemovedAssistantOnShiftString += "0. Back to main menu.\n-1. Quit application.\n";
232         clearTerminal();
233         System.out.println(removeAssistantsOnShiftString());
234         System.out.println(latestRemovedAssistantOnShiftString);
235     }
236 
237     /**
238      * Remove assistant on shift error string string.
239      *
240      * @return String for invalid input for removing assistant on shift.
241      */
242     public String removeAssistantOnShiftErrorString() {
243         String removeAssistantOnShiftErrorString = "Error!\nInvalid sequential ID.\n";
244         removeAssistantOnShiftErrorString += "Please, enter one of the following:\n\n";
245         removeAssistantOnShiftErrorString += "The sequential ID to select the assistant on shift to be
                removed.\n\n";
246         removeAssistantOnShiftErrorString += "0. Back to main menu.\n -1. Quit application.\n";
247         return removeAssistantOnShiftErrorString;
248     }
249 
250 
251     /**
252      * Remove assistants on shift string string.
253      *
254      * @return String containing information for removal of assistant on shift.
255      */
256     public String removeAssistantsOnShiftString() {
257         String removeAssistantsOnShiftString = "University of Knowledge - COVID test\n\n" + "List of Free
                Assistant on Shifts:\n";
258         int a = 10;
259         for (int i = 0; i < assistantsOnShift.size(); i++) {
260             if (assistantsOnShift.get(i).getAssistantOnShiftStatus().equals("FREE")) {
261                 a++;
262                 removeAssistantsOnShiftString = removeAssistantsOnShiftString + a + "." +
                        this.assistantsOnShift.get(i).printAssistantOnShiftTemplate() + "\n";
263             }
264         }
265         removeAssistantsOnShiftString += "Removing assistant on shift\n\n" + "Please, enter one of the
                following:\n\n";
266         removeAssistantsOnShiftString += "The sequential ID to select the assistant on shift to be
                removed.\n";
267         removeAssistantsOnShiftString += "0. Back to main menu.\n" + "-1. Quit application.\n";
268         return removeAssistantsOnShiftString;
269     }
270 
271     /**
272      * Shows information for accessing list of bookings to user.
273      */
274     public void listBookings() {
```

```
275         String listBookingsString = "University of Knowledge - COVID test\n\n";
276         listBookingsString += "Select which booking to list:\n";
277         listBookingsString += "1. All\n" + "2. Only bookings status:SCHEDULED\n" + "3. Only bookings
                status:COMPLETED\n";
278         listBookingsString += "0. Back to main menu.\n" + "-1. Quit application.\n";
279         System.out.println(listBookingsString);
280     }
281
282     /**
283      * Shows list of all bookings to user.
284      */
285     public void listAllBookings() {
286         String listAllBookings = "List of ALL Bookings:\n";
287         for (int i = 0; i < bookings.size(); i++) {
288             listAllBookings += this.bookings.get(i).printBookingTemplate() + "\n";
289         }
290         listAllBookings += "0. Back to main menu.\n" + "-1. Quit application.\n";
291         System.out.println(listAllBookings);
292     }
293
294     /**
295      * Shows list of scheduled bookings.
296      */
297     public void listScheduledBookings() {
298         String listScheduledBookings = "List of SCHEDULED Bookings:\n";
299         for (int i = 0; i < bookings.size(); i++) {
300             if (bookings.get(i).getBookingStatus().equals("SCHEDULED")) {
301                 listScheduledBookings += this.bookings.get(i).printBookingTemplate() + "\n";
302             }
303         }
304         listScheduledBookings += "0. Back to main menu.\n" + "-1. Quit application.\n";
305         System.out.println(listScheduledBookings);
306     }
307
308     /**
309      * Shows list of completed bookings.
310      */
311     public void listCompletedBookings() {
312         String listCompletedBookings = "List of COMPLETED Bookings:\n";
313         for (int i = 0; i < bookings.size(); i++) {
314             if (bookings.get(i).getBookingStatus().equals("COMPLETED")) {
315                 listCompletedBookings += this.bookings.get(i).printBookingTemplate() + "\n";
316             }
317         }
318         listCompletedBookings += "0. Back to main menu.\n" + "-1. Quit application.\n";
319         System.out.println(listCompletedBookings);
320     }
321
322     /**
323      * Here, a list of available dates and times are made using list of bookable rooms and list of
            assistants on shift.
324      * List of dates and times are made from free assistant on shifts and empty or available bookable rooms.
325      *
326      * @return String with list of available dates and times for bookings.
327      */
```

```
328     public String addBookingString() {
329         ArrayList<BookableRoom> matchedDateTimes = new ArrayList<>();
330         for (int i = 0; i < bookableRooms.size(); i++) {
331             if (!bookableRooms.get(i).getBookableRoomStatus().equals("FULL")) {
332                 for (int j = 0; j < assistantsOnShift.size(); j++) {
333                     if (assistantsOnShift.get(j).getAssistantOnShiftStatus().equals("FREE")) {
334                         if
                                (bookableRooms.get(i).getBookableRoomTimeSlot().equals(assistantsOnShift.get(j).getAssistant
335                                    &&
                                        bookableRooms.get(i).getBookableRoomDate().equals(assistantsOnShift.get(j).getAssista
                                        {
336                             BookableRoom bookableRoom1 = new
                                    BookableRoom(bookableRooms.get(i).getBookableRoomDate(),
                                    bookableRooms.get(i).getBookableRoomTimeSlot());
337                             matchedDateTimes.add(bookableRoom1);
338                         }
339                     }
340                 }   // above, list containing matching dates and timeslots are made using free assistants of
                        shift and empty or available bookable rooms.
341             }
342         }
343 
344         String addBookingString = "University of Knowledge - COVID test\n\nAdding booking (appointment for a
                COVID test) to the system\n\n";
345         addBookingString += "List of available time-slots:\n";
346         int a = 10;
347         int n = matchedDateTimes.size();
348         BookableRoom temp = new BookableRoom(null, null);
349         for (int i = 0; i < chronologicalOrder(matchedDateTimes, temp).size(); i++) { // In this line,
                function for sorting list of dates and times in chronological order is called.
350             a++;
351             addBookingString += a + "." + matchedDateTimes.get(i).printDateTimeTemplate() + "\n";
352         }
353         addBookingString += "\nPlease, enter one of the following:\n";
354         addBookingString += "The sequential ID of an available time-slot and the student email, separated by
                a white space.\n";
355         addBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
356         return addBookingString;
357     }
358 
359 
360     /**
361      * Add booking string 2 string.
362      *
363      * @return same string as above without the title in string.
364      */
365     public String addBookingString2() {
366         ArrayList<BookableRoom> matchedDateTimes = new ArrayList<>();
367         for (int i = 0; i < bookableRooms.size(); i++) {
368             if (!bookableRooms.get(i).getBookableRoomStatus().equals("FULL")) {
369                 for (int j = 0; j < assistantsOnShift.size(); j++) {
370                     if (assistantsOnShift.get(j).getAssistantOnShiftStatus().equals("FREE")) {
371                         if
                                (bookableRooms.get(i).getBookableRoomTimeSlot().equals(assistantsOnShift.get(j).getAssistant
372                                    &&
```

```
                                bookableRooms.get(i).getBookableRoomDate().equals(assistantsOnShift.get(j).getAssista
                                {
373                         BookableRoom bookableRoom1 = new
                                BookableRoom(bookableRooms.get(i).getBookableRoomDate(),
                                bookableRooms.get(i).getBookableRoomTimeSlot());
374                         matchedDateTimes.add(bookableRoom1);
375                     }
376                 }
377             }
378         }
379     }
380     String addBookingString = "List of available time-slots:\n";
381     int a = 10;
382     BookableRoom temp = new BookableRoom(null, null);
383     for (int i = 0; i < matchedDateTimes.size(); i++) {
384         a++;
385         addBookingString += a + "." + chronologicalOrder(matchedDateTimes,
                temp).get(i).printDateTimeTemplate() + "\n";
386     }
387     addBookingString += "\nPlease, enter one of the following:\n";
388     addBookingString += "The sequential ID of an available time-slot and the student email, separated by
            a white space.\n";
389     addBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
390
391     return addBookingString;
392 }
393
394 /**
395  * Method for sorting list of matching dates and times in chronological order.
396  *
397  * @param matchedDateTimes list of matching dates and times of free assistant on shifts and empty or
        available bookable rooms.
398  * @param temp           temporary date and time needed for the method.
399  * @return matched dates and times in chronological order.
400  */
401 public ArrayList<BookableRoom> chronologicalOrder(ArrayList<BookableRoom> matchedDateTimes,
        BookableRoom temp) {
402     int n = matchedDateTimes.size();
403     for (int i = 0; i < n - 1; i++) {
404         for (int j = 0; j < n - i - 1; j++) {
405             if (matchedDateTimes.get(j).getBookableRoomDate().equals(matchedDateTimes.get(j +
                    1).getBookableRoomDate()) &&
                    Integer.parseInt(matchedDateTimes.get(j).getBookableRoomTimeSlot().substring(0, 1)) >
                    Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomTimeSlot().substring(0, 1)))
                    {
406                 temp.setBookableRoomTimeSlot(matchedDateTimes.get(j).getBookableRoomTimeSlot());
407                 temp.setBookableRoomDate(matchedDateTimes.get(j).getBookableRoomDate());
408                 matchedDateTimes.get(j).setBookableRoomDate(matchedDateTimes.get(j +
                        1).getBookableRoomDate());
409                 matchedDateTimes.get(j).setBookableRoomTimeSlot(matchedDateTimes.get(j +
                        1).getBookableRoomTimeSlot());
410                 matchedDateTimes.get(j + 1).setBookableRoomDate(temp.getBookableRoomDate());
411                 matchedDateTimes.get(j + 1).setBookableRoomTimeSlot(temp.getBookableRoomTimeSlot());
412
413             } else if (Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(6, 10)) >
```

```
                    Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(6, 10))) {
414                 temp.setBookableRoomTimeSlot(matchedDateTimes.get(j).getBookableRoomTimeSlot());
415                 temp.setBookableRoomDate(matchedDateTimes.get(j).getBookableRoomDate());
416                 matchedDateTimes.get(j).setBookableRoomDate(matchedDateTimes.get(j +
                        1).getBookableRoomDate());
417                 matchedDateTimes.get(j).setBookableRoomTimeSlot(matchedDateTimes.get(j +
                        1).getBookableRoomTimeSlot());
418                 matchedDateTimes.get(j + 1).setBookableRoomDate(temp.getBookableRoomDate());
419                 matchedDateTimes.get(j + 1).setBookableRoomTimeSlot(temp.getBookableRoomTimeSlot());
420
421             } else if (Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(6, 10))
                    == Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(6, 10))
                    &&
422                     Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(3, 5)) ==
                            Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(3,
                            5))) {
423                 if (Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(0, 2)) >
                        Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(0, 2))) {
424                     temp.setBookableRoomTimeSlot(matchedDateTimes.get(j).getBookableRoomTimeSlot());
425                     temp.setBookableRoomDate(matchedDateTimes.get(j).getBookableRoomDate());
426                     matchedDateTimes.get(j).setBookableRoomDate(matchedDateTimes.get(j +
                            1).getBookableRoomDate());
427                     matchedDateTimes.get(j).setBookableRoomTimeSlot(matchedDateTimes.get(j +
                            1).getBookableRoomTimeSlot());
428                     matchedDateTimes.get(j + 1).setBookableRoomDate(temp.getBookableRoomDate());
429                     matchedDateTimes.get(j + 1).setBookableRoomTimeSlot(temp.getBookableRoomTimeSlot());
430                 }
431             } else if (Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(6, 10))
                    == Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(6, 10)))
                    {
432                 if (Integer.parseInt(matchedDateTimes.get(j).getBookableRoomDate().substring(3, 5)) >
                        Integer.parseInt(matchedDateTimes.get(j + 1).getBookableRoomDate().substring(3, 5))) {
433                     temp.setBookableRoomTimeSlot(matchedDateTimes.get(j).getBookableRoomTimeSlot());
434                     temp.setBookableRoomDate(matchedDateTimes.get(j).getBookableRoomDate());
435                     matchedDateTimes.get(j).setBookableRoomDate(matchedDateTimes.get(j +
                            1).getBookableRoomDate());
436                     matchedDateTimes.get(j).setBookableRoomTimeSlot(matchedDateTimes.get(j +
                            1).getBookableRoomTimeSlot());
437                     matchedDateTimes.get(j + 1).setBookableRoomDate(temp.getBookableRoomDate());
438                     matchedDateTimes.get(j + 1).setBookableRoomTimeSlot(temp.getBookableRoomTimeSlot());
439                 }
440             }
441         }
442     }
443     return matchedDateTimes;
444
445
446 }
447
448 /**
449  * Adding booking after valid inputs.
450  *
451  * @param booking1   booking that is added to list of bookings.
452  * @param studentEmail student email needed for the booking.
453  */
```

```
454     public void addBooking(Booking booking1, String studentEmail){
455         booking1.setBookingStatus("SCHEDULED");
456         booking1.setStudentEmail(studentEmail);
457         bookings.add(booking1);
458         String successfulBookingString = "Booking added successfully:\n";
459         successfulBookingString += booking1.printBookingTemplate() + "\n\n";
460         successfulBookingString += addBookingString2();
461         System.out.println(successfulBookingString);
462 
463     }
464 
465 
466     /**
467      * Add booking error string string.
468      *
469      * @return String for invalid input when adding booking.
470      */
471     public String addBookingErrorString() {
472         String addBookingString = "Error!\n";
473         addBookingString += "The sequential ID provided does not match a sequential ID in the list.";
474         addBookingString += "\nPlease, enter one of the following:\n";
475         addBookingString += "The sequential ID of an available time-slot and the student email, separated by
                a white space.\n";
476         addBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
477 
478         return addBookingString;
479     }
480 
481     /**
482      * Add booking invalid length string string.
483      *
484      * @return String for invalid input when adding booking.
485      */
486     public String addBookingInvalidLengthString() {
487         String addBookingString = "Error!\n";
488         addBookingString += "Invalid amount of inputs.";
489         addBookingString += "\nPlease, enter one of the following:\n";
490         addBookingString += "The sequential ID of an available time-slot and the student email, separated by
                a white space.\n";
491         addBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
492 
493         return addBookingString;
494     }
495 
496     /**
497      * Add booking invalid email string string.
498      *
499      * @return String for invalid input when adding booking.
500      */
501     public String addBookingInvalidEmailString() {
502         String addBookingString = "Error!\n";
503         addBookingString += "The email provided is not appropriate.";
504         addBookingString += "\nPlease, enter one of the following:\n";
505         addBookingString += "The sequential ID of an available time-slot and the student email, separated by
                a white space.\n";
```

```
506         addBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
507 
508         return addBookingString;
509     }
510 
511     /**
512      * Removing a booking.
513      *
514      * @param booking booking that is removed from list of bookings after valid input from user.
515      */
516     public void removeBooking(Booking booking){
517         Booking booking1 = booking;
518         bookings.remove(booking);
519         String latestRemovedBookingString = "Booking removed successfully:\n\n";
520         latestRemovedBookingString += booking1.printBookingTemplate();
521         latestRemovedBookingString += "\nPlease, enter one of the following:\n\n";
522         latestRemovedBookingString += "The sequential ID to select the booking to be removed from the listed
                bookings above.\n\n";
523         latestRemovedBookingString += "0. Back to main menu.\n-1. Quit application.\n";
524         clearTerminal();
525         System.out.println(removeAssistantsOnShiftString());
526         System.out.println(latestRemovedBookingString);
527     }
528 
529     /**
530      * String contains list of scheduled bookings.
531      *
532      * @return String with information about removal of a booking.
533      */
534     public String removeBookingString() {
535         String removeBookingString = "University of Knowledge - COVID test\n\n" + "List of SCHEDULED
                Bookings:\n";
536         int a = 10;
537         for (int i = 0; i < bookings.size(); i++) {
538             if (bookings.get(i).getBookingStatus().equals("SCHEDULED")) {
539                 a++;
540                 removeBookingString += a + "." + this.bookings.get(i).printBookingTemplate() + "\n";
541             }
542         }
543         removeBookingString = removeBookingString + "Removing booking from the system\n\n" + "Please, enter
                one of the following:\n\n";
544         removeBookingString = removeBookingString + "The sequential ID to select the booking to be removed
                from the listed bookings above.\n";
545         removeBookingString = removeBookingString + "0. Back to main menu.\n" + "-1. Quit application.\n";
546         return removeBookingString;
547     }
548 
549     /**
550      * Remove booking error string string.
551      *
552      * @return String for invalid input when removing booking.
553      */
554     public String removeBookingErrorString() {
555         String error = "Error!\nThe sequential ID provided does not match a sequential ID in the
                list.\nPlease, enter one of the following:\n\n";
```

```
556         error += "The sequential ID to select the booking to be removed from the listed bookings above\n";
557         error += "0. Back to main menu.\n-1. Quit application.\n";
558         return error;
559 
560     }
561 
562     /**
563      * Remove booking invalid length string.
564      *
565      * @return String for invalid input when removing booking.
566      */
567     public String removeBookingInvalidLength() {
568         String error = "Error!\nAmount of inputs is invalid.\nPlease, enter one of the following:\n\n";
569         error += "The sequential ID to select the booking to be removed from the listed bookings above\n";
570         error += "0. Back to main menu.\n-1. Quit application.\n";
571         return error;
572 
573     }
574 
575     /**
576      * Method for concluding a booking.
577      *
578      * @param booking booking that will be completed.
579      */
580     public void concludeBooking(Booking booking){
581         booking.setBookingStatus("COMPLETED");
582         Booking booking1 = booking;
583         String latestCompletedBookingString = "Booking completed successfully:\n\n";
584         latestCompletedBookingString += booking1.printBookingTemplate();
585         latestCompletedBookingString += "\nPlease, enter one of the following:\n\n";
586         latestCompletedBookingString += "The sequential ID to select the booking to be completed.\n\n";
587         latestCompletedBookingString += "0. Back to main menu.\n-1. Quit application.\n";
588         clearTerminal();
589         System.out.println(concludeBookingString());
590         System.out.println(latestCompletedBookingString);
591     }
592 
593 
594     /**
595      * Conclude booking string string.
596      *
597      * @return String with information for concluding a booking.
598      */
599     public String concludeBookingString() {
600         String concludeBookingString = "University of Knowledge - COVID test\n\n" + "List of SCHEDULED
                Bookings:\n";
601         int a = 10;
602         for (int i = 0; i < bookings.size(); i++) {
603             if (bookings.get(i).getBookingStatus().equals("SCHEDULED")) {
604                 a++;
605                 concludeBookingString += a + "." + this.bookings.get(i).printBookingTemplate() + "\n";
606             }
607         }
608         concludeBookingString += "Conclude booking\n\n" + "Please, enter one of the following:\n\n";
609         concludeBookingString += "The sequential ID to select the booking to be completed.\n\n";
```

```
610         concludeBookingString += "0. Back to main menu.\n" + "-1. Quit application.\n";
611         return concludeBookingString;
612     }
613 
614     /**
615      * Conclude booking error string string.
616      *
617      * @return String for invalid input from user when concluding booking.
618      */
619     public String concludeBookingErrorString() {
620         String error = "Error!\nThe sequential ID provided does not match a sequential ID in the
                list.\nPlease, enter one of the following:\n\n";
621         error += "The sequential ID to select the booking to be completed.";
622         error += "0. Back to main menu.\\n\" + \"-1. Quit application.\n";
623         return error;
624     }
625 
626     /**
627      * Conclude booking invalid length string.
628      *
629      * @return String for invalid input from user when concluding booking.
630      */
631     public String concludeBookingInvalidLength() {
632         String error = "Error!\nInvalid amount of inputs.\nPlease, enter one of the following:\n\n";
633         error += "The sequential ID to select the booking to be completed.";
634         error += "0. Back to main menu.\\n\" + \"-1. Quit application.\n";
635         return error;
636     }
637 
638     /**
639      * Method for clearing the terminal.
640      */
641     public final void clearTerminal() {
642         try {
643             if (System.getProperty("os.name").contains("Windows")) {
644                 new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
645             } else {
646                 System.out.print("\033\143");
647             }
648         } catch (IOException | InterruptedException e) {}
649 
650     }
651 }
```

5 BookableRoom.java

```
1  /**
2   * The type Bookable room.
3   */
4  public class BookableRoom {
5 
6      private Room room;
7 
8      private int occupancy;
```

```
9
10      private String bookableRoomStatus;
11
12      private String bookableRoomDate;
13
14      private String bookableRoomTimeSlot;
15
16
17      /**
18       * Print bookable room template string.
19       *
20       * @return Template for a bookable room.
21       */
22      public String printBookableRoomTemplate() {
23          String bookableRoomTemplate = " | " + bookableRoomDate + " " + bookableRoomTimeSlot + " | " +
                bookableRoomStatus +
24                  " | " + this.room.getCode() + " | occupancy: " + occupancy + " |";
25          return bookableRoomTemplate;
26      }
27
28      /**
29       * Constructor with date and timeslot used when matching date and time during booking.
30       *
31       * @param bookableRoomDate  date
32       * @param bookableRoomTimeSlot timeslot
33       */
34      public BookableRoom(String bookableRoomDate, String bookableRoomTimeSlot) {
35          this.bookableRoomDate = bookableRoomDate;
36          this.bookableRoomTimeSlot = bookableRoomTimeSlot;
37      }
38
39      /**
40       * Constructor for instantiating a bookable room.
41       *
42       * @param room              room of bookable room.
43       * @param occupancy         occupancy of bookable room.
44       * @param bookableRoomStatus status of bookable room.
45       * @param bookableRoomDate  date of bookable room.
46       * @param bookableRoomTimeSlot timeslot for bookable room.
47       */
48      public BookableRoom(Room room, int occupancy, String bookableRoomStatus, String bookableRoomDate,
            String bookableRoomTimeSlot) {
49          this.room = room;
50          this.occupancy = occupancy;
51          this.bookableRoomStatus = bookableRoomStatus;
52          this.bookableRoomDate = bookableRoomDate;
53          this.bookableRoomTimeSlot = bookableRoomTimeSlot;
54      }
55
56      /**
57       * Gets room.
58       *
59       * @return room of bookable room.
60       */
61      public Room getRoom() {
```

```
62          return room;
63      }
64
65      /**
66       * Gets bookable room time slot.
67       *
68       * @return timeslot for bookable room.
69       */
70      public String getBookableRoomTimeSlot() {
71          return bookableRoomTimeSlot;
72      }
73
74      /**
75       * Gets bookable room status.
76       *
77       * @return status of bookable room.
78       */
79      public String getBookableRoomStatus() {
80          return bookableRoomStatus;
81      }
82
83      /**
84       * Gets bookable room date.
85       *
86       * @return date of bookable room.
87       */
88      public String getBookableRoomDate() {
89          return bookableRoomDate;
90      }
91
92      /**
93       * Gets occupancy.
94       *
95       * @return occupancy of bookable room.
96       */
97      public int getOccupancy() {
98          return occupancy;
99      }
100
101     /**
102      * setter for occpancy.
103      *
104      * @param occupancy occupancy of bookable room.
105      */
106     public void setOccupancy(int occupancy) {
107         this.occupancy = occupancy;
108     }
109
110     /**
111      * setter for status.
112      *
113      * @param bookableRoomStatus status of bookable room.
114      */
115     public void setBookableRoomStatus(String bookableRoomStatus) {
116         this.bookableRoomStatus = bookableRoomStatus;
```

```
117     }
118
119     /**
120      * setter for date.
121      *
122      * @param bookableRoomDate date of bookable room.
123      */
124     public void setBookableRoomDate(String bookableRoomDate) {
125         this.bookableRoomDate = bookableRoomDate;
126     }
127
128     /**
129      * setter for timeslot.
130      *
131      * @param bookableRoomTimeSlot timeslot of bookable room.
132      */
133     public void setBookableRoomTimeSlot(String bookableRoomTimeSlot) {
134         this.bookableRoomTimeSlot = bookableRoomTimeSlot;
135     }
136
137
138     /**
139      * Added bookable room string string.
140      *
141      * @return String for successful booking.
142      */
143     public String addedBookableRoomString() {
144         String addedBookableRoomString = "Bookable Room added successfully:\n";
145         addedBookableRoomString += this.printBookableRoomTemplate();
146         addedBookableRoomString += "\nPlease, enter one of the following:\n\n";
147         addedBookableRoomString += "The sequential ID listed to a room, a date (dd/mm/yyyy), and a time
                (HH:MM),\nseparated by a white space.\n";
148         addedBookableRoomString += "0. Back to main menu.\n-1. Quit application.\n";
149         return addedBookableRoomString;
150     }
151
152     /**
153      * Print date time template string.
154      *
155      * @return Template for date and timeslot.
156      */
157     public String printDateTimeTemplate() {
158         String dateTimeTemplate = bookableRoomDate + " " + bookableRoomTimeSlot;
159         return dateTimeTemplate;
160     }
161
162
163 }
```

6 AssistantOnShift.java

```
1  /**
2   * The type Assistant on shift.
3   */
```

```java
public class AssistantOnShift {

    private Assistant assistant;

    private String assistantOnShiftDate;

    private String assistantOnShiftStatus;

    private String assistantTimeSlot;

    /**
     * Print assistant on shift template string.
     *
     * @return template for assistant on shift.
     */
    public String printAssistantOnShiftTemplate() {
        String AssistantOnShiftTemplate = " | " + assistantOnShiftDate + " " + assistantTimeSlot + " | " +
            assistantOnShiftStatus + " | " +
               this.assistant.getEmail() + " |";
        return AssistantOnShiftTemplate;
    }


    /**
     * Constructor for instantiating a assistant on shift.
     *
     * @param assistant           assistant of assistant on shift.
     * @param assistantOnShiftDate date of assistant on shift.
     * @param assistantOnShiftStatus status of assistant on shift.
     * @param assistantTimeSlot   timeslot of assistant on shift.
     */
    public AssistantOnShift(Assistant assistant, String assistantOnShiftDate, String
         assistantOnShiftStatus, String assistantTimeSlot) {
        this.assistant = assistant;
        this.assistantOnShiftDate = assistantOnShiftDate;
        this.assistantOnShiftStatus = assistantOnShiftStatus;
        this.assistantTimeSlot = assistantTimeSlot;

    }

    /**
     * getter.
     *
     * @return assistant of assistant on shift.
     */
    public Assistant getAssistant() {
        return assistant;
    }

    /**
     * getter for date of assistant on shift.
     *
     * @return date of assistant on shift.
     */
    public String getAssistantOnShiftDate() {
```

```
        return assistantOnShiftDate;
    }

    /**
     * getter for status of assistant.
     *
     * @return status of assistant on shift.
     */
    public String getAssistantOnShiftStatus() {
        return assistantOnShiftStatus;
    }

    /**
     * getter for timeslot of assistant on shift.
     *
     * @return timeslot of assistant on shift.
     */
    public String getAssistantOnShiftTimeSlot() {
        return assistantTimeSlot;
    }

    /**
     * setter for status of assistant on shift.
     *
     * @param assistantOnShiftStatus status of assistant on shift.
     */
    public void setAssistantOnShiftStatus(String assistantOnShiftStatus) {
        this.assistantOnShiftStatus = assistantOnShiftStatus;
    }









}
```

7 Booking.java

```
/**
 * The type Booking.
 */
public class Booking {

    private BookableRoom bookableRoom;

    private AssistantOnShift assistantOnShift;

    private String bookingDate;

    private String bookingTimeSlot;

    private String bookingStatus;

```

```
16      private String roomCode;
17
18      private String studentEmail;
19
20      /**
21       * Print booking template string.
22       *
23       * @return String of template for booking details.
24       */
25      public String printBookingTemplate() {
26          String bookingTemplate = " | " + bookingDate + " " + bookingTimeSlot + " | " +
                this.assistantOnShift.getAssistant().getEmail() + " | " +
27                 this.bookableRoom.getRoom().getCode() + " | " + studentEmail + " |";
28          return bookingTemplate;
29      }
30
31      /**
32       * Constructor for instantiating a booking.
33       *
34       * @param bookableRoom  bookable room of booking.
35       * @param assistantOnShift assistant on shift of booking.
36       * @param bookingDate   date of booking.
37       * @param bookingTimeSlot timeslot of booking.
38       * @param bookingStatus  status of bookings.
39       * @param roomCode      room code of booking.
40       * @param studentEmail  student email of booking.
41       */
42      public Booking(BookableRoom bookableRoom, AssistantOnShift assistantOnShift, String bookingDate, String
             bookingTimeSlot,
43                     String bookingStatus, String roomCode, String studentEmail) {
44          this.bookableRoom = bookableRoom;
45          this.assistantOnShift = assistantOnShift;
46          this.bookingDate = bookingDate;
47          this.bookingTimeSlot = bookingTimeSlot;
48          this.bookingStatus = bookingStatus;
49          this.roomCode = roomCode;
50          this.studentEmail = studentEmail;
51      }
52
53      /**
54       * getter for status of booking.
55       *
56       * @return status of booking.
57       */
58      public String getBookingStatus(){
59          return bookingStatus;
60      }
61
62      /**
63       * setter for bookable room.
64       *
65       * @param bookableRoom bookable room of booking.
66       */
67      public void setBookableRoom(BookableRoom bookableRoom){
68          this.bookableRoom = bookableRoom;
```

```
69      }
70
71      /**
72       * setter for assistant on shift.
73       *
74       * @param assistantOnShift assistant on shift of booking.
75       */
76      public void setAssistantOnShift(AssistantOnShift assistantOnShift){
77          this.assistantOnShift = assistantOnShift;
78      }
79
80      /**
81       * setter for booking date.
82       *
83       * @param bookingDate of booking.
84       */
85      public void setBookingDate(String bookingDate){
86          this.bookingDate = bookingDate;
87      }
88
89      /**
90       * setter for booking timeslot.
91       *
92       * @param bookingTimeSlot timeslot of booking.
93       */
94      public void setBookingTimeSlot(String bookingTimeSlot){
95          this.bookingTimeSlot = bookingTimeSlot;
96      }
97
98      /**
99       * setter for booking status.
100      *
101      * @param bookingStatus status of booking.
102      */
103     public void setBookingStatus(String bookingStatus){
104         this.bookingStatus = bookingStatus;
105     }
106
107     /**
108      * setter for room code.
109      *
110      * @param roomCode room code of booking.
111      */
112     public void setRoomCode(String roomCode){
113         this.roomCode = roomCode;
114     }
115
116     /**
117      * setter for student email.
118      *
119      * @param studentEmail student email of booking.
120      */
121     public void setStudentEmail(String studentEmail){
122         this.studentEmail = studentEmail;
123     }
```

```
124
125 }
```

8 BookingApp.java

```
1  import java.util.Scanner;
2  import java.util.ArrayList;
3
4
5  /**
6   * BookingApp
7   *
8   * @author Jason Gurung 25/02/2021
9   */
10 public class BookingApp {
11
12     /**
13      * The Booking system.
14      */
15     public BookingSystem bookingSystem;
16     /**
17      * The University resources.
18      */
19     public UniversityResources universityResources;
20
21
22     /**
23      * Checks input from user to display the screen they want to see.
24      */
25     public void mainMenuInput() {
26         Scanner in = new Scanner(System.in);
27                 String input = in.next();
28                 switch (input) {
29                     case "-1":
30                         System.exit(0);
31                         break;
32                     case "0":
33                         bookingSystem.clearTerminal();  //clear terminal
34                         bookingSystem.loadMainMenu();
35                         mainMenuInput();       //calls input checker.
36                         break;
37                     case "1":
38                         bookingSystem.clearTerminal();
39                         bookingSystem.listBookableRooms();
40                         listBookableRoomsInput();
41
42                         break;
43                     case "2":
44                         bookingSystem.clearTerminal();
45                         System.out.println(universityResources.addBookableRoomsString());
46                         addBookableRoomsInput();
47
48                         break;
49                     case "3":
```

```
                    bookingSystem.clearTerminal();
                    bookingSystem.removeBookableRoomsString();
                    removeBookableRoomsInput();
                    break;
                case "4":
                    bookingSystem.clearTerminal();
                    bookingSystem.listAssistantsOnShift();
                    listAssistantsOnShiftInput();
                    break;
                case "5":
                    bookingSystem.clearTerminal();
                    System.out.println(universityResources.addAssistantsOnShift());
                    addAssistantsOnShiftInput();
                    break;
                case "6":
                    bookingSystem.clearTerminal();
                    System.out.println(bookingSystem.removeAssistantsOnShiftString());
                    removeAssistantsOnShiftInput();
                    break;
                case "7":
                    bookingSystem.clearTerminal();
                    bookingSystem.listBookings();
                    listBookingsInput();
                    break;
                case "8":
                    bookingSystem.clearTerminal();
                    System.out.println(bookingSystem.addBookingString());
                    addBookingInput();
                    break;
                case "9":
                    bookingSystem.clearTerminal();
                    System.out.println(bookingSystem.removeBookingString());
                    removeBookingInput();
                    break;
                case "10":
                    bookingSystem.clearTerminal();
                    System.out.println(bookingSystem.concludeBookingString());
                    concludeBookingInput();
                default:
                    System.out.println("Invalid input.");
                    mainMenuInput();
            }

            // Above, these are different cases depending on the number user inputs.
    }

    /**
     * Checks input for list bookable rooms to output the option they choose.
     */
    public void listBookableRoomsInput(){
        Scanner in = new Scanner(System.in);
        String input = in.next();
        switch (input) {
            case "0":
                bookingSystem.clearTerminal();
```

```
105                 bookingSystem.loadMainMenu();
106                 mainMenuInput();
107                 break;
108             case "-1":
109                 System.exit(0);
110                 break;
111             default:
112                 System.out.println(" Invalid input.");
113                 listBookableRoomsInput();
114         }
115 
116     }
117 
118     /**
119      * Checks input for adding bookable room.
120      */
121     public void addBookableRoomsInput() {
122         Scanner scan = new Scanner(System.in);
123         String addBookableRoomInput = scan.nextLine();
124         String[] inputs = addBookableRoomInput.split(" "); // stores inputs separated by white space in an
                array.
125         if (addBookableRoomInput.equals("0")) {
126             bookingSystem.clearTerminal();
127             bookingSystem.loadMainMenu();
128             mainMenuInput();
129         } else if (addBookableRoomInput.equals("-1")) {
130             System.exit(0);
131         } else if (inputs.length == 3) {
132             String iD = inputs[0];
133             String date = inputs[1]; // inputs assigned to variables.
134             String time = inputs[2];
135             try {
136                 if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <=
                        (universityResources.getRooms().length + 10)) { //Checks if the ID entered by user is
                        valid.
137                     if (bookingSystem.checkDateValidity(date)) {
138                         if (bookingSystem.checkTimeValidity(time)) {
139                             boolean bookableRoomAlreadyExists = false;
140                             Room room1 = universityResources.getRooms()[Integer.parseInt(iD) - 11];
141                             for (int i = 0; i < bookingSystem.getBookableRooms().size(); i++) {
142                                 if (bookingSystem.getBookableRooms().get(i).getRoom().equals(room1)) {
143                                     if
                                            (bookingSystem.getBookableRooms().get(i).getBookableRoomDate().equals(date))
                                            {
144                                         if
                                                (bookingSystem.getBookableRooms().get(i).getBookableRoomTimeSlot().equals(time
                                                { // if date and time matches a bookable room with same room then
                                                bookable room already exists for that date and time.
145                                             bookableRoomAlreadyExists = true;
146                                             System.out.println(universityResources.bookableRoomAlreadyExistsString());
147                                             addBookableRoomsInput();
148                                         }
149                                     }
150                                 }
151                             }
```

```
152                         if (!bookableRoomAlreadyExists) {
153                             bookingSystem.addBookableRoom(date, time, room1); // Adds bookable room if
                                    bookable room doesn't already exist.
154                             addBookableRoomsInput();
155                         }
156
157
158                     } else {
159                         System.out.println(universityResources.addBookableRoomInvalidTimeString());
160                         addBookableRoomsInput();
161                     }
162
163                 } else {
164                     System.out.println(universityResources.addBookableRoomInvalidDateString());
165                     addBookableRoomsInput();
166                 }
167             } else {
168                 System.out.println(universityResources.addBookableRoomInvalidIDString());   //Strings for
                        invalid inputs.
169                 addBookableRoomsInput();
170             }
171         } catch (NumberFormatException e) {
172             System.out.println(universityResources.addBookableRoomInvalidIDString());
173             addBookableRoomsInput();
174         }
175     } else {
176         System.out.println(universityResources.addBookableRoomInvalidLengthString());
177         addBookableRoomsInput();
178     }
179
180 }
181
182
183 /**
184  * Checks input for removing bookable room.
185  */
186 public void removeBookableRoomsInput() {
187     Scanner scan = new Scanner(System.in);
188     String removeBookableRoomInput = scan.nextLine();
189     String[] inputs = removeBookableRoomInput.split(" ");
190     if (removeBookableRoomInput.equals("0")){
191         bookingSystem.loadMainMenu();
192         mainMenuInput();
193     }
194     else if (removeBookableRoomInput.equals("-1")){
195         System.exit(0);
196     }
197     else if (inputs.length == 1) {  //Checks amount of inputs from user separated by white space.
198         String iD = inputs[0];
199         ArrayList<Integer> numbers = new ArrayList<>();
200         for (int i = 0; i < bookingSystem.getBookableRooms().size(); i++) {
201             if (bookingSystem.getBookableRooms().get(i).getBookableRoomStatus().equals("EMPTY")) {
202                 System.out.println("number = " + i);
203                 numbers.add(i);
204             }
```

```
205             }
206             try{
207             if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <= (numbers.size() + 10)) { //Checks ID
                    entered by user.
208                 bookingSystem.removeBookableRoom(bookingSystem.getBookableRooms().get(numbers.get(Integer.parseInt(iD)
                        - 11)));
209             }else {
210                 System.out.println(bookingSystem.removeBookableRoomErrorString());
211             }
212             removeBookableRoomsInput();
213         } catch(NumberFormatException e){       //Catch exception when ID is not entered in number format
                and therefore
214                 System.out.println(bookingSystem.removeBookableRoomErrorString());
215                 removeBookableRoomsInput();
216             }
217         }
218             else {
219             System.out.println(bookingSystem.removeBookableRoomErrorString());
220             removeBookableRoomsInput(); }
221
222
223
224     }
225
226     /**
227      * Checks input for list of assistants on shift.
228      */
229     public void listAssistantsOnShiftInput() {
230         Scanner in = new Scanner(System.in);
231         String input = in.next();
232         switch (input) {
233             case "0":
234                 bookingSystem.clearTerminal();
235                 bookingSystem.loadMainMenu();
236                 mainMenuInput();
237                 break;
238             case "-1":
239                 System.exit(0);
240                 break;
241             default:
242                 System.out.println(" Invalid input.");
243                 listAssistantsOnShiftInput();
244
245         }
246     }
247
248     /**
249      * Checks input for adding assistants on shift.
250      */
251     public void addAssistantsOnShiftInput() {
252         Scanner scan = new Scanner(System.in);
253         String addAssistantsOnShiftInput = scan.nextLine();
254         String[] inputs = addAssistantsOnShiftInput.split(" ");
255         if (addAssistantsOnShiftInput.equals("0")){
256             bookingSystem.loadMainMenu();
```

```
257             mainMenuInput();
258         }
259         else if (addAssistantsOnShiftInput.equals("-1")){
260             System.exit(0);
261         }
262         else if (inputs.length == 2) {
263             String iD = inputs[0];
264             String date = inputs[1];
265             try {
266                 if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <=
                        (universityResources.getAssistants().length + 10)) {
267                     if (bookingSystem.checkDateValidity(date)) {
268                         boolean assistantOnShiftAlreadyExists = false;
269                         Assistant assistant1 = universityResources.getAssistants()[Integer.parseInt(iD) - 11];
270                         for (int i = 0; i < bookingSystem.getAssistantsOnShift().size(); i++) {
271                             if (bookingSystem.getAssistantsOnShift().get(i).getAssistant().equals(assistant1))
                                    {
272                                 if
                                        (bookingSystem.getAssistantsOnShift().get(i).getAssistantOnShiftDate().equals(date))
                                        { //Checks if assistant on shift with same date already exists.
273                                     assistantOnShiftAlreadyExists = true;
274                                     System.out.println(universityResources.assistantOnShiftAlreadyExistsString());
275                                     addAssistantsOnShiftInput();
276                                 }
277                             }
278                         }
279                         if (!assistantOnShiftAlreadyExists) {
280                             bookingSystem.addAssistantOnShift(assistant1, date);  //If assistant on shift
                                    already exists with same date then assistant on shift is not added to system.
281                             addAssistantsOnShiftInput();
282                         }
283                     } else {
284                         System.out.println(universityResources.addAssistantOnShiftInvalidDateString());
285                         addAssistantsOnShiftInput();
286                     }
287                 } else {
288                     System.out.println(universityResources.addAssistantOnShiftInvalidIDString());
289                     addAssistantsOnShiftInput();
290                 }
291 
292             } catch (NumberFormatException e) {
293                 System.out.println(universityResources.addAssistantOnShiftInvalidIDString());
294                 addAssistantsOnShiftInput();
295             }
296 
297         }else {
298             System.out.println(universityResources.addAssistantOnShiftInvalidLengthString());
299             addAssistantsOnShiftInput();
300         }
301 
302     }
303 
304     /**
305      * checks input for removing assistant on shift.
306      */
```

```
307     public void removeAssistantsOnShiftInput() {
308         Scanner scan = new Scanner(System.in);
309         String removeAssistantOnShiftInput = scan.nextLine();
310         String[] inputs = removeAssistantOnShiftInput.split(" ");
311         if (removeAssistantOnShiftInput.equals("0")){
312             bookingSystem.clearTerminal();
313             bookingSystem.loadMainMenu();
314             mainMenuInput();
315         }
316         else if (removeAssistantOnShiftInput.equals("-1")){
317             System.exit(0);
318         }
319         else if (inputs.length == 1) {
320             String iD = inputs[0];
321             ArrayList<Integer> numbers = new ArrayList<>();
322 
323             for (int i = 0; i < bookingSystem.getAssistantsOnShift().size(); i++) {
324                 if (bookingSystem.getAssistantsOnShift().get(i).getAssistantOnShiftStatus().equals("FREE")) {
325                     numbers.add(i);       // using array to get the assistant on shift chosen by user for it
                            to be removed.
326                 }
327             }
328             try {
329                 if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <= (numbers.size() + 10)) {
330                     bookingSystem.removeAssistantOnShift(bookingSystem.getAssistantsOnShift().get(numbers.get(Integer.pa
                            - 11)));
331 
332                 } else {
333                     System.out.println(bookingSystem.removeAssistantOnShiftErrorString());
334                 }
335                 removeAssistantsOnShiftInput();
336             } catch (NumberFormatException e) {
337                 System.out.println(bookingSystem.removeAssistantOnShiftErrorString());
338                 removeAssistantsOnShiftInput();
339             }
340         }else{
341             System.out.println(bookingSystem.removeAssistantOnShiftErrorString());
342             removeAssistantsOnShiftInput();
343         }
344 
345     }
346 
347     /**
348      * Checks input for list of bookings.
349      */
350     public void listBookingsScanner(){
351         Scanner in = new Scanner(System.in);
352         String input = in.next();
353         switch (input) {
354             case "0":
355                 bookingSystem.clearTerminal();
356                 bookingSystem.loadMainMenu();
357                 mainMenuInput();
358                 break;
359             case "-1":
```

```
360                 System.exit(0);
361                 break;
362             default:
363                 System.out.println(" Invalid input.");
364                 listBookingsScanner();
365         }
366     }
367 
368 
369     /**
370      * Checks input for list of bookings.
371      */
372     public void listBookingsInput() {
373         Scanner in = new Scanner(System.in);
374         String input = in.next();
375         switch (input) {
376             case "0":
377                 bookingSystem.clearTerminal();
378                 bookingSystem.loadMainMenu();
379                 mainMenuInput();
380                 break;
381             case "-1":
382                 System.exit(0);
383                 break;
384             case "1":
385                 bookingSystem.listAllBookings();
386                 listBookingsScanner();
387                 break;
388             case "2":
389                 bookingSystem.listScheduledBookings();  //Loading different cases for viewing bookings.
390                 listBookingsScanner();
391                 break;
392             case "3":
393                 bookingSystem.listCompletedBookings();
394                 listBookingsScanner();
395                 break;
396             default:
397                 System.out.println(" Invalid input.");
398                 bookingSystem.listAllBookings();
399                 listBookingsScanner();
400 
401         }
402 
403     }
404 
405 
406     /**
407      * Checks input for adding bookings.
408      */
409     public void addBookingInput() {
410         Scanner scan = new Scanner(System.in);
411         String addBookingInput = scan.nextLine();
412         String[] inputs = addBookingInput.split(" ");
413         if (addBookingInput.equals("0")) {
414             bookingSystem.clearTerminal();
```

```
415             bookingSystem.loadMainMenu();
416             mainMenuInput();
417         } else if (addBookingInput.equals("-1")) {
418             System.exit(0);
419         } else if(inputs.length == 2) {
420             String iD = inputs[0];
421             String studentEmail = inputs[1];
422 
423             ArrayList<BookableRoom> matchedDateTimes = new ArrayList<>();
424             for (int i = 0; i < bookingSystem.getBookableRooms().size(); i++) {
425                 if (!bookingSystem.getBookableRooms().get(i).getBookableRoomStatus().equals("FULL")) {
426                     for (int j = 0; j < bookingSystem.getAssistantsOnShift().size(); j++) {
427                         if
                                (bookingSystem.getAssistantsOnShift().get(j).getAssistantOnShiftStatus().equals("FREE"))
                                {
428                             if
                                    (bookingSystem.getBookableRooms().get(i).getBookableRoomTimeSlot().equals(bookingSystem.
429                                        &&
                                            bookingSystem.getBookableRooms().get(i).getBookableRoomDate().equals(bookingSyste
                                            {
430                                 BookableRoom bookableRoom1 = new
                                        BookableRoom(bookingSystem.getBookableRooms().get(i).getBookableRoomDate(),
                                        bookingSystem.getBookableRooms().get(i).getBookableRoomTimeSlot());
431                                 matchedDateTimes.add(bookableRoom1);
432                             }
433                         }
434                     }
435                 }
436             }
437             BookableRoom temp = new BookableRoom(null, null);
438             try {
439                 if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <=
                        (bookingSystem.chronologicalOrder(matchedDateTimes, temp).size() + 10)) {
440                     if (studentEmail.endsWith("@uok.ac.uk")) { //Student email checker.
441                         Booking booking1 = new Booking(null, null, null, null, null, null, null);
442                         for (int i = 0; i < bookingSystem.getAssistantsOnShift().size(); i++) {
443                             AssistantOnShift assistantOnShift1 = bookingSystem.getAssistantsOnShift().get(i);
444                             if
                                    (assistantOnShift1.getAssistantOnShiftDate().equals(matchedDateTimes.get(Integer.parseIn
                                    - 11).getBookableRoomDate()) &&
445                                        assistantOnShift1.getAssistantOnShiftTimeSlot().equals(matchedDateTimes.get(Integer.p
                                            - 11).getBookableRoomTimeSlot())) {
446                                 bookingSystem.getAssistantsOnShift().get(i).setAssistantOnShiftStatus("BUSY");
447                                 booking1.setAssistantOnShift(bookingSystem.getAssistantsOnShift().get(i));
448                                 booking1.setBookingDate(bookingSystem.getAssistantsOnShift().get(i).getAssistantOnShiftDa
449                                 booking1.setBookingTimeSlot(bookingSystem.getAssistantsOnShift().get(i).getAssistantOnShi
450                             }
451                         }
452                         for (int i = 0; i < bookingSystem.getBookableRooms().size(); i++) {
453                             BookableRoom bookableRoom1 = bookingSystem.getBookableRooms().get(i);
454                             if
                                    (bookableRoom1.getBookableRoomDate().equals(matchedDateTimes.get(Integer.parseInt(iD)
                                    - 11).getBookableRoomDate()) &&
455                                        bookableRoom1.getBookableRoomTimeSlot().equals(matchedDateTimes.get(Integer.parseInt(
                                            - 11).getBookableRoomTimeSlot())) {
```

```
456                             bookingSystem.getBookableRooms().get(i).setOccupancy(bookingSystem.getBookableRooms().get
                                    + 1);
457                             if (bookingSystem.getBookableRooms().get(i).getOccupancy() ==
                                    bookingSystem.getBookableRooms().get(i).getRoom().getCapacity()) {
458                                 bookingSystem.getBookableRooms().get(i).setBookableRoomStatus("FULL");
459                                 booking1.setBookableRoom(bookingSystem.getBookableRooms().get(i));
460                                 booking1.setRoomCode(bookingSystem.getBookableRooms().get(i).getRoom().getCode());
461                             } else if (bookingSystem.getBookableRooms().get(i).getOccupancy() <
                                    bookingSystem.getBookableRooms().get(i).getRoom().getCapacity()) {
462                                 bookingSystem.getBookableRooms().get(i).setBookableRoomStatus("AVAILABLE");
463                                 booking1.setBookableRoom(bookingSystem.getBookableRooms().get(i));
464                                 booking1.setRoomCode(bookingSystem.getBookableRooms().get(i).getRoom().getCode());
                                        //Using setters to set the details for booking before it is added
                                        using the method to add booking.
465                             }
466                         }
467                     }
468                     bookingSystem.addBooking(booking1, studentEmail); //The booking is used as parameter.
469                 } else {
470                     System.out.println(bookingSystem.addBookingInvalidEmailString());
471                     addBookingInput();
472 
473                 }
474 
475             } else {
476                 System.out.println(bookingSystem.addBookingErrorString());
477             }
478             addBookingInput();
479 
480         }catch(NumberFormatException e){
481             System.out.println(bookingSystem.addBookingErrorString());
482             addBookingInput();
483         }
484     }else{
485         System.out.println(bookingSystem.addBookingInvalidLengthString());
486         addBookingInput();
487     }
488 
489 
490 
491 }
492 
493 /**
494  * checks input for removing bookings.
495  */
496 public void removeBookingInput() {
497     Scanner scan = new Scanner(System.in);
498     String removeBookingInput = scan.nextLine();
499     String[] inputs = removeBookingInput.split(" ");
500     if (removeBookingInput.equals("0")){
501         bookingSystem.clearTerminal();
502         bookingSystem.loadMainMenu();
503         mainMenuInput();
504     }
505     else if (removeBookingInput.equals("-1")){
```

```
506             System.exit(0);
507         }
508         else if (inputs.length == 1) {
509             String iD = inputs[0];
510             ArrayList<Integer> numbers = new ArrayList<>();
511 
512             for (int i = 0; i < bookingSystem.getBookings().size(); i++) {
513                 if (bookingSystem.getBookings().get(i).getBookingStatus().equals("SCHEDULED")) {
514                     numbers.add(i);
515                 }
516             }
517             try{
518             if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <= (numbers.size() + 10)) { //Checking if
                     ID entered by user is valid.
519                 System.out.println(numbers.size());
520 
521                 bookingSystem.removeBooking(bookingSystem.getBookings().get(numbers.get(Integer.parseInt(iD)
                         - 11)));
522                 System.out.println("hello");
523 
524             } else {
525                 System.out.println(bookingSystem.removeBookingErrorString());
526             }
527             removeBookingInput();
528         } catch (NumberFormatException e) {
529                 System.out.println(bookingSystem.removeBookingErrorString());
530                 removeBookingInput();
531             }
532         }else{
533             System.out.println(bookingSystem.removeBookingInvalidLength());
534             removeBookingInput();
535         }
536 
537     }
538 
539     /**
540      * checks input for concluding booking.
541      */
542     public void concludeBookingInput() {
543         Scanner scan = new Scanner(System.in);
544         String concludeBookingInput = scan.nextLine();
545         String[] inputs = concludeBookingInput.split(" ");
546         if (concludeBookingInput.equals("0")){
547             bookingSystem.clearTerminal();
548             bookingSystem.loadMainMenu();
549             mainMenuInput();
550         }
551         else if (concludeBookingInput.equals("-1")){
552             System.exit(0);       //Exits Booking app when "-1" entered.
553         }
554         else if (inputs.length == 1) {
555             String iD = inputs[0];
556             ArrayList<Integer> numbers = new ArrayList<>();
557 
558             for (int i = 0; i < bookingSystem.getBookings().size(); i++) {
```

```
559                 if (bookingSystem.getBookings().get(i).getBookingStatus().equals("SCHEDULED")) {
560                     numbers.add(i);
561                 }
562             }
563             try{
564             if (Integer.parseInt(iD) > 10 && Integer.parseInt(iD) <= (numbers.size() + 10)) {
565                 bookingSystem.concludeBooking(bookingSystem.getBookings().get(numbers.get(Integer.parseInt(iD)
                        - 11)));
566             } else {
567                 System.out.println(bookingSystem.concludeBookingErrorString());
568             }
569             concludeBookingInput();
570         } catch (NumberFormatException e) {
571                 System.out.println(bookingSystem.concludeBookingErrorString());
572                 concludeBookingInput();
573             }
574         }else{
575             System.out.println(bookingSystem.concludeBookingInvalidLength());
576             concludeBookingInput();
577         }
578 
579     }
580 
581     /**
582      * Booking App is started here.
583      *
584      * @param args command line arguments.
585      */
586     public static void main(String[] args) {
587         Room room1 = new Room("201", 1);
588         Room room2 = new Room("202", 2);
589         Room room3 = new Room("203", 3);
590         // Creation of rooms.
591 
592 
593         Room[] rooms = {room1, room2, room3};
594         // List of rooms.
595 
596         Assistant assistant1 = new Assistant("Bob", "bob@uok.ac.uk");
597         Assistant assistant2 = new Assistant("Rob", "rob@uok.ac.uk");
598         Assistant assistant3 = new Assistant("Tom", "tom@uok.ac.uk");
599         // Creation of assistants.
600 
601         Assistant[] assistants = {assistant1, assistant2, assistant3};
602         // List of assistants.
603 
604 
605 
606         BookableRoom bookableRoom1 = new BookableRoom(room1, 0, "EMPTY", "21/03/2021", "07:00");
607         BookableRoom bookableRoom2 = new BookableRoom(room1, 1, "FULL", "21/03/2021", "08:00");
608         BookableRoom bookableRoom3 = new BookableRoom(room1, 1, "FULL", "21/03/2021", "09:00");
609         BookableRoom bookableRoom4 = new BookableRoom(room2, 0, "EMPTY", "22/03/2021", "07:00");
610         BookableRoom bookableRoom5 = new BookableRoom(room2, 1, "AVAILABLE", "21/03/2021", "08:00");
611         BookableRoom bookableRoom6 = new BookableRoom(room2, 2, "FULL", "21/03/2021", "09:00");
612         BookableRoom bookableRoom7 = new BookableRoom(room3, 0, "EMPTY", "22/03/2021", "07:00");
```

```
613         BookableRoom bookableRoom8 = new BookableRoom(room3, 1, "AVAILABLE", "21/03/2021", "08:00");
614         BookableRoom bookableRoom9 = new BookableRoom(room3, 2, "AVAILABLE", "23/03/2021", "09:00");
615         // Creation of bookable rooms.
616
617
618         ArrayList<BookableRoom> bookableRooms = new ArrayList<>();
619         bookableRooms.add(bookableRoom1);
620         bookableRooms.add(bookableRoom2);
621         bookableRooms.add(bookableRoom3);
622         bookableRooms.add(bookableRoom4);
623         bookableRooms.add(bookableRoom5);
624         bookableRooms.add(bookableRoom6);
625         bookableRooms.add(bookableRoom7);
626         bookableRooms.add(bookableRoom8);
627         bookableRooms.add(bookableRoom9);
628         // Adding bookable rooms to an array list.
629
630
631         AssistantOnShift assistantOnShift1 = new AssistantOnShift(assistant1, "21/03/2021", "FREE", "07:00");
632         AssistantOnShift assistantOnShift2 = new AssistantOnShift(assistant1, "22/03/2021", "FREE", "08:00");
633         AssistantOnShift assistantOnShift3 = new AssistantOnShift(assistant1, "21/03/2021", "BUSY", "09:00");
634         AssistantOnShift assistantOnShift4 = new AssistantOnShift(assistant2, "22/03/2021", "FREE", "07:00");
635         AssistantOnShift assistantOnShift5 = new AssistantOnShift(assistant2, "21/03/2021", "FREE", "08:00");
636         AssistantOnShift assistantOnShift6 = new AssistantOnShift(assistant3, "23/03/2021", "FREE", "09:00");
637         // Creation of assistants on shift.
638
639         ArrayList<AssistantOnShift> assistantsOnShift = new ArrayList<>();
640         assistantsOnShift.add(assistantOnShift1);
641         assistantsOnShift.add(assistantOnShift2);
642         assistantsOnShift.add(assistantOnShift3);
643         assistantsOnShift.add(assistantOnShift4);
644         assistantsOnShift.add(assistantOnShift5);
645         assistantsOnShift.add(assistantOnShift6);
646         // Adding assistants on shift to an array list.
647
648         Booking booking1 = new Booking(bookableRoom3, assistantOnShift3, "21/03/2021", "09:00", "SCHEDULED",
                "201", "jason@uok.ac.uk");
649         Booking booking2 = new Booking(bookableRoom9, assistantOnShift1, "21/02/2021", "07:00", "COMPLETED",
                "203", "kenny@uok.ac.uk");
650         // Creation of bookings.
651
652         ArrayList<Booking> bookings = new ArrayList<>();
653         bookings.add(booking1);
654         bookings.add(booking2);
655         // Adding bookings to an array list.
656
657
658
659         BookingApp app = new BookingApp();
660         app.bookingSystem = new BookingSystem(bookableRooms, assistantsOnShift, bookings);
661         app.universityResources = new UniversityResources(assistants, rooms);
662
663
664         app.bookingSystem.loadMainMenu();
665         app.mainMenuInput();
```

```
666         // Calling load main menu and main menu input checker methods.
667
668         }
669     }
```