

```

1 import java.util.Scanner;

2 /**This class defines an assistant (object) who works at the university, and can be used as an
   assistant on shift */

3 public class Assistant {

4

5     public String email;

6     public String name;

7     public String assistantsOnShifts;

8

9     public String getEmail(){

10         /** @return a unique email for each assistant */

11         Scanner sc= new Scanner(System.in); //System.in is a standard input stream.

12

13         if (assistantsOnShifts.contains(this.email)) {

14             System.out.print("This email already exists for an assistant, please try again");

15             String newEmail= sc.nextLine(); //reads string.

16             return newEmail;

17         }

18         else {

19             return this.email;

20         }

21     }

22     public String setName() {

23         /** Sets the name of the assistant if it is not a blank

24          * @return a string that is either the name, or a message saying that the name cannot be
25          blank

26          */

27         if (this.name != null){

28             return this.name;

29         }

30         else{return "Name cannot be blank";}

31     }

```

```
31
32 //System.out.print(getName + getEmail);
33
34 //connstructor
35 public Assistant(String email, String name) {
36     /**Constructs the assisstant */
37     this.email = email;
38     this.name = name;
39 }
40 }
41
42 //Print template: | <name> | <email> |
```

```

1 //import java.lang.reflect.Method;

2 import java.time.LocalDateTime;

3 import java.util.ArrayList;

4 //import java.util.function.Function;

5 /**This class defines an assistant on shift (object) who is has shifts and is present during the
booked covid tests*/

6 public class AssistantOnShift {

7     public LocalDateTime dateAndTime;

8     public String AssistantOnShiftstatus;

9     boolean available;

10    public Boolean removable;

11

12    public String FREE;

13    public String BUSY;

14    //constructor:

15    public AssistantOnShift(LocalDateTime dateAndTime, String AssistantOnShiftstatus) {

16        /**Constructs the assistant on shift class*/

17        this.dateAndTime = dateAndTime;

18        this.AssistantOnShiftstatus = AssistantOnShiftstatus;

19    }

20

21    //public string assistantEmail

22    public String getAssistantStatus(){

23        /** Calculates the status of the assistant of shift (based on its availability) and returns said
status

24        * @return the status of the assistant on shift

25        */

26        if (available == true) {

27            this.AssistantOnShiftstatus = FREE;

28        }

29        else if (available == false){

30            this.AssistantOnShiftstatus = BUSY;

```

```

31     }
32     return this.AssistantOnShiftstatus;
33 }
34 public boolean isRemovable(){
35     /** Returns whether or not the assistant on shift is removable based on its status
36      * @return the boolean value of removeable
37      */
38     if (this.AssistantOnShiftstatus == FREE){
39         removable = true;
40     }
41     else {
42         removable = false;
43     }
44     return removable;
45 }
46
47 // list assistant on shifts status:FREE
48 public ArrayList FREEassistantsOnShiftlist;
49 public ArrayList getFREEassistantsOnShiftlist(){
50     /** This function loops through all of the assitants on shift in the list, and selects the ones
51     with a FREE status and puts them in a list specific for that type of assitants on shift
52
53     *@return the list of assistants on shift with a FREE status*/
54
55     for(int i = 0; i < FREEassistantsOnShiftlist.size(); i++) {
56         if (this.AssistantOnShiftstatus == FREE){
57             this.dateAndTime = dateAndTime;
58             this.AssistantOnShiftstatus = AssistantOnShiftstatus;
59             FREEassistantsOnShiftlist.add(new AssistantOnShift(dateAndTime, AssistantOnShiftstatus));
60         }
61     }
62     return FREEassistantsOnShiftlist;

```

61 }

62

63 //assistant is registered to shifts for the entire day (7 AM to 10 AM). Given the current 60-minute duration of a

64 //time-slot, when selecting a date, the system will be creating three assistant on shifts.

65 //6. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> |

66 }

67

```

1 import java.lang.reflect.Method;

2 import java.time.LocalDateTime;

3 import java.util.ArrayList;

4 /** This class defines a bookable room (object) in which the tests will take place once they are
booked. */

5 public class BookableRoom {

6     public LocalDateTime dateAndTime;

7     public int occupancy;

8     public String BookableRoomStatus;

9     public String code;

10

11     public int Capacity;

12     //Room.getCapacity();

13     public String EMPTY;

14     public String AVAIAABLE;

15     public String FULL;

16

17     public ArrayList EMPTYBookableRoomslist;

18     public ArrayList getEMPTYBookableRoomslist(){

19         /** This function loops through all of the bookable rooms in the list, and selects the ones with
an EMPTY status and puts them in a list specific for that type of bookable rooms

20         *@return the list of bookablerooms with an EMPTY status*/

21         //FREEBookableRoomslist.size() = size

22         //for (bookableRooms in bookableRooms) {

23         for(int i = 0; i < BookableRoomslist.size(); i++) {

24             if (this.BookableRoomStatus == EMPTY){

25

26                 this.dateAndTime = dateAndTime;

27                 this.occupancy = occupancy;

28                 this.BookableRoomStatus = BookableRoomStatus;

29                 this.code = code;

```

```

30     EMPTYBookableRoomslist.add(new BookableRoom(dateAndTime, occupancy,
BookableRoomStatus, code));
31 }
32 }
33 return EMPTYBookableRoomslist;
34 }
35
36 public String removeEBR(String usersRoom){
37     /**This function removes the bookable room from the list of bookable rooms status:EMPTY)*/
38     EMPTYBookableRoomslist.remove(usersRoom);
39     //getEMPTYBookableRoomslist.remove(room);
40     String message;
41     message = "The room is no longer empty";
42     return message;
43 }
44 public String getremovedRoom(String usersRoom){
45     /**@returns the bookable room that was removed*/
46     return usersRoom;
47 }
48
49 public String getBookableRoomStatus(){
50     /** Calaculates and returns the status of the bookable room based on its occupancy
51      * @return the status of the bookable room as a string
52      */
53     if (occupancy == 0){
54         this.BookableRoomStatus = EMPTY;
55     }
56     else if (occupancy < Capacity) {
57         this.BookableRoomStatus = AVAIABLE;
58     }
59     else if (occupancy == Capacity) {

```

```

60     this.BookableRoomStatus = FULL;
61 }
62 else if (occupancy > Capacity){
63     //print error message
64     System.out.print("The occupancy of a room cannot be higher than its capacity, please
enter a lower occupancy");
65
66     // System.out.print("")
67 }
68 return this.BookableRoomStatus;
69 }
70 public boolean removable = false;
71 public boolean removable(){
72     /** Returns whether the bookable room is removable or not based on its status
73     * @return the boolean value of removable
74     */
75     if (this.BookableRoomStatus == "EMPTY"){
76         removable = true;
77     }
78     else {
79         removable = false;
80     }
81     return removable;
82 }
83 public int updateOccupancy() {
84     /** Adds one to the occupancy of the bookable room, this is called whenever a booking is
made i n a particular bookable room
85     * @return the updated occupancy as an integer
86     */
87     return occupancy += 1;
88 }
89 public int getOccupancy(){

```



```

90    /** @return the occupancy of the room */
91    return this.occupancy;
92 }
93
94 //constructor:
95 public BookableRoom(LocalDate dateAndTime, int occupancy, String BookableRoomStatus,
String code){
96 /**This function constructs the BookableRoom class*/
97 this.dateAndTime = dateAndTime;
98 this.occupancy = occupancy;
99 this.BookableRoomStatus = BookableRoomStatus;
100 this.code = code;
101 }
102 }
103 //if the occupancy changes, then the getBookableRoomStatus method is called to update the
BookableRoomStatus of the bookable room
104 //to do check
105
106 //2. A bookable room is a room allocated in a specific time-slot (dd/mm/yyyy HH:MM). Since
rooms are available
107 //from 7 AM - 10 AM, the system will offer at most three bookable rooms (time-slots) per room
per day.
108
109 //6. The BookableRoomStatus of a bookable room must be updated whenever its occupancy
changes.
110
111 //7. Print template: | <dd/mm/yyyy HH:MM> | <BookableRoomStatus> | <roomCode> |
occupancy: <occupancy> |
112
113 //@param
114 //@return
115

```

```

1 import java.util.ArrayList;
2 import java.util.Scanner;
3 import java.lang.reflect.Method;
4 import java.time.LocalDateTime;
5 /**This booking class defines what variables belong to the booking object */
6 public class Booking {
7
8     public LocalDateTime dateAndTime;
9     public String status;
10    public String studentEmail;
11    public int bookingIDcode;
12    public String roomCode;
13    public String assistantEmail;
14
15    public String COMPLETED;
16    public String FREE;
17    public String SCHEDULED;
18    public String FULL;
19
20    public String AssistantOnShiftstatus; //HOW CAN I GET THIS FROM THE ASSISTANT ON SHIFT
    CLASS? CHECK!
21    public String BookableRoomStatus; // HOW CAN I GET THIS FROM THE BOOKABLE ROOM
    CLASS? CHECK!
22
23    public String emailInput;
24
25    public String getEmail(){
26        /**This function is used to get the students' email
27        @return the students' email*/
28        System.out.print("What is the start of the students email?");
29        Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
30        String emailInput = sc.nextLine(); //reads string.

```

```

31     return studentEmail = (emailInput + "@uok.ac.uk");
32 }
33 public boolean happened = false;
34 public String getBookingStatus(){
35     /**This function calculates tje booking status based on whether or not the test has occured
36     @return the status of the booking*/
37     if (happened = true){
38         this.status = "SCHEDULED";
39     }
40     else if (happened = false) {
41         this.status = "COMPLETED";
42     }
43     return this.status;
44 }
45 public boolean available = false;
46 public boolean AvailableResources() {
47     /**This function calculates whether or not a resource is available based on its statuses
48     @return all of the available resources*/
49     if (this.BookableRoomStatus == "FULL"){
50         available = false;
51         System.out.print("This Bookable Room is full, choose another room");
52     }
53     else if (this.AssistantOnShiftstatus != "FREE"){
54         available = false;
55         System.out.print("This Assistant on shift isn't availabe right now, please choose another
assistant.");
56     }
57     //return AvailableResources = true;
58     else {
59         return available = true;}
60     //do i need to make this method a boolean? how do i do this

```

```

61     return available;
62 }
63
64 public boolean rb = false; //rb = removable Booking
65 public boolean rb() {
66     /**This function calculates whether or not a boking is removable based on whether it has
67     been completed or not which we can find out from its status
68     @return the boolean value of the removable booking variable */
69     if (status.equals("COMPLETED")) {
70         rb = true;
71     }
72     else {
73         rb = false;
74     }
75     return rb;
76 }
77 public Booking getBooking(){
78     return this.booking;
79 }
80 //constructor:
81 public Booking(LocalDate dateAndTime, String status, String assistantEmail, String
82 roomCode, String studentEmail, int bookingIDcode) {
83     /**This function constructs the booking class*/
84     this.dateAndTime = dateAndTime;
85     this.status = status;
86     this.assistantEmail = assistantEmail;
87     this.roomCode = roomCode;
88     this.studentEmail = studentEmail;
89     this.bookingIDcode = bookingIDcode;
90 }
91 }

```

91

92 //1. A booking consists of matching a bookable room and an assistant on shift at a specific time-slot to perform a

93 //COVID-19 test on a student. It is the main function of the system.

94

95 //2. A booking has a unique sequential number (identification code) and the email of the student being tested

96

97 //3. To create a booking in a time-slot, the system must certify the availab

98 //5. A booking not COMPLETED can be cancelled, i.e., deleted from the system.

99 //After cancellation, the resources (room and assistant) should be released for booking again, i.e., their statuses must be updated.

100

101

102

103 //1. A booking consists of matching a bookable room and an assistant on shift at a specific time-slot to perform a

104 //COVID-19 test on a student. It is the main function of the system.

105 //6. A booking SCHEDULED can become COMPLETED. Once completed, the booking cannot be deleted due to audit processes.

106 //7. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> | <roomCode> | <studentEmail> |

107

108

109

110 /\*public static void main (String args[]) {

111

112 Scanner input = new Scanner(System.in);

113 String newAssistant = Scanner.nextline();

114

115 // String newAssistant = input.next();

116 assistants.add(newAssistant);

117 //return assistants;

```
118
119  }
120  /*public function addRoom(){
121      rooms.add(userInput.next());
122      return rooms;
123  } */
124
```

```
1 import java.util.*;
2 //import java.util.Scanner;
3 /**This application class is where the covid test booking application is run, it's where things
   actually happen*/
4 public class BookingApp {
5     /**The main class for the program, where the menus are loaded and all of the functions are
   called. */
6     public static void someMethod() {
7         /*This function is used for outputting the original menu screen*/
8         System.out.println("University of Knowledge - COVID test");
9         System.out.println();
10        System.out.println("Manage Bookings");
11        System.out.println();
12
13        System.out.println("Please, enter the number to select your option:");
14        System.out.println();
15        System.out.println("To manage Bookable Rooms:");
16        System.out.println("1. List");
17        System.out.println("2. Add");
18        System.out.println("3. Remove");
19        System.out.println("To manage Assistants on Shift:");
20        System.out.println("4. List");
21        System.out.println("5. Add");
22        System.out.println("6. Remove");
23        System.out.println("To manage Bookings:");
24        System.out.println("7. List");
25        System.out.println("8. Add");
26        System.out.println("9. Remove");
27        System.out.println("10. Conclude");
28
29     }
30 }
```

```
31 public static final String QUIT = "-1. Quit application.";
32 public static final String BACK = "0. Back to main menu.";
33 public static final String ERROR = "Error!";
34
35 public static void main(String[] args) {
36     /** This is the main method where the application is run*/
37     someMethod();
38     System.out.println("Enter a string: ");
39
40     Scanner input = new Scanner(System.in);
41     int userOption = input.nextInt();
42     if (userOption == 1) {
43         //collection.clear(); //clear interface
44         System.out.println("University of Knowledge - COVID test");
45         System.out.println();
46         UniversityResources.getRooms();
47         System.out.println(BACK);
48         System.out.println(QUIT);
49         System.out.println();
50
51     }
52     else if (userOption == 2) {
53         //collection.clear(); //clear interface
54         Boolean Valid = true;
55         System.out.println("University of Knowledge - COVID test");
56         System.out.println();
57         System.out.println("Adding bookable room");
58         System.out.println();
59         System.out.println(UniversityResources.getRooms());
60         System.out.println("Please, enter one of the following:");
61         System.out.println();
```



```
62      System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time
(HH:MM), separated by a white space.");
63      System.out.println(BACK);
64      System.out.println(QUIT);
65      System.out.println();
66
67      //Adding bookable room
68      if (Valid = true) {
69          System.out.println("Bookable Room added successfully:");
70          System.out.println(BookingSystem.getBookableRoom());
71          System.out.println("Please, enter one of the following:");
72          System.out.println();
73          System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time
(HH:MM), separated by a white space.");
74          System.out.println(BACK);
75          System.out.println(QUIT);
76          System.out.println();
77      }
78      else if (Valid = false) {
79          System.out.println(ERROR);
80
81          System.out.println("There is no availability for this room at this particular time, please
try again rooom isn't available");
82          System.out.println("This room does not exist, please try again");
83          System.out.println("This time isn't available, you can only book for 07:00, 08:00 and
09:00");
84          /*If the entry is NOT valid, the system remains unchanged. In this way, you should
append to the screen the following message explaining the problem:
85          Error!
86          <message explaining the error>*/
87
88          System.out.println("Please, enter one of the following:");
89          System.out.println();
```

```

90      System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time
(HH:MM), separated by a white space.");
91      System.out.println(BACK);
92      System.out.println(QUIT);
93
94      System.out.println();
95  }
96  }
97  else if (userOption == 3) {
98      //collection.clear(); //clear interface
99      Boolean Valid = true;
100     System.out.println("University of Knowledge - COVID test");
101     System.out.println();
102     System.out.println(BookableRoom.getEMPTYBookableRoomslist()); //static error
103     System.out.println("Removing bookable room");
104     System.out.println();
105     System.out.println("Please, enter one of the following:");
106     System.out.println();
107     System.out.println("The sequential ID to select the bookable room to be removed.");
108     Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
109     String usersRoom= sc.nextLine(); //reads string.
110     System.out.println(BACK);
111     System.out.println(QUIT);
112     System.out.println();
113
114
115     if (Valid = true) {
116
117         BookableRoom.removeEBR(usersRoom); //static error //remove the bookable room
from the list of bookable rooms status:EMPTY)
118         System.out.println("Bookable Room removed successfully");

```

```
119         System.out.println(BookableRoom.getremovedRoom(usersRoom)); //that was
deleted);
120         System.out.println("Please, enter one of the following:");
121         System.out.println();
122         System.out.println("The sequential ID to select the bookable room to be removed.");
123         System.out.println(BACK);
124         System.out.println(QUIT);
125         System.out.println();
126     }
127     else if (Valid = false) {
128         System.out.println(ERROR);
129         System.out.println("This room is not an empty room that can be removed from the
system, please try again");
130         System.out.println("Please, enter one of the following:");
131         System.out.println();
132         System.out.println("The sequential ID to select the bookable room to be removed.");
133         System.out.println(BACK);
134         System.out.println(QUIT);
135
136
137     }
138 }
139 else if (userOption == 4) {
140     //collection.clear(); //clear interface
141     System.out.println("University of Knowledge - COVID test");
142     System.out.println();
143     //System.out.println(assistant on shifts List);
144     System.out.println(BACK);
145     System.out.println(QUIT);
146     System.out.println();
147 }
148 else if (userOption == 5) {
```

```
149      //collection.clear(); //clear interface
150      Boolean valid = true;
151      System.out.println("University of Knowledge - COVID test");
152      System.out.println();
153      System.out.println("Adding assistant on shift");
154      System.out.println();
155      //System.out.println(assistants);
156      System.out.println("Please, enter one of the following:");
157      System.out.println();
158      System.out.println("The sequential ID of an assistant, and date (dd/mm/yyyy), separated
by a white space.");
159      System.out.println(BACK);
160      System.out.println(QUIT);
161      System.out.println();
162
163      //Adding assistant on shift
164      if (valid = true) {
165          //add the assistant to the list of assistants
166          System.out.println("Assistant on Shift added successfully:");
167          //System.out.println(Assistant on Shift);
168          System.out.println("Please, enter one of the following:");
169          System.out.println();
170          System.out.println("The sequential ID listed of an assistant, a date (dd/mm/yyyy),
separated by a white space.");
171          System.out.println(BACK);
172          System.out.println(QUIT);
173          System.out.println();
174      }
175      else if (valid = false) {
176          System.out.println(ERROR);
177
178          //System.out.println(message explaining the error)
```

```

179
180         System.out.println("There is no availability for this room at this particular time, please
try again rooom isn't available");
181         System.out.println("This room does not exist, please try again");
182         System.out.println("This time isn't available, you can only book for 07:00, 08:00 and
09:00");
183         /*If the entry is NOT valid, the system remains unchanged.In this way, you should
append to the screen the following
184 message explaining the problem:
185 Error!
186 <message explaining the error>*/
187
188         System.out.println("Please, enter one of the following:");
189         System.out.println();
190         System.out.println("The sequential ID of an assistant and date (dd/mm/yyyy),
separated by a white space.");
191         System.out.println(BACK);
192         System.out.println(QUIT);
193         System.out.println();
194     }
195 }
196 else if (userOption == 6){
197     //collection.clear(); //clear interface
198     Boolean Valid = true;
199     System.out.println("University of Knowledge - COVID test");
200     System.out.println();
201     System.out.println(AssistantOnShift.getFREEassistantsOnShiftlist());
202     System.out.println("Removing assistant on shift");
203     System.out.println();
204     System.out.println("Please, enter one of the following:");
205     System.out.println();
206     System.out.println("The sequential ID to select the assistant on shift to be removed.");

```

```
207     System.out.println(BACK);
208     System.out.println(QUIT);
209     System.out.println();
210
211     //removing an assistants
212     if (Valid = true) {
213         //remove the assistant from the list of assistant on shifts status:FREE)
214         System.out.println("Assistant on Shift removed successfully:");
215         //System.out.println(Assistant on Shift); //that was deleted);
216         System.out.println("Please, enter one of the following:");
217         System.out.println();
218         System.out.println("The sequential ID to select the assistant on shift to be removed");
219
220         System.out.println(BACK);
221         System.out.println(QUIT);
222         System.out.println();
223     }
224     else if (Valid = false) {
225         /*If the entry is NOT valid, the system remains unchanged. In this way, you should
append to the screen the following
226         message explaining the problem:
227         Error!
228         <message explaining the error>*/
229         System.out.println(ERROR);
230         System.out.println("This is not an assistant on shift that is free to be removed from the
system, please try again");
231
232         System.out.println("Please, enter one of the following:");
233         System.out.println();
234         System.out.println("The sequential ID to select the bookable room to be removed.");
235
236         System.out.println(BACK);
```

```

237         System.out.println(QUIT);
238         System.out.println();
239     }
240
241 }
242 else if (userOption == 7){
243     //collection.clear(); //clear interface
244     System.out.println("University of Knowledge - COVID test");
245     System.out.println();
246     System.out.println("Select which booking to list:");
247     System.out.println("1. All");
248     System.out.println("2. Only bookings status:SCHEDULED");
249     System.out.println("3. Only bookings status:COMPLETED");
250     System.out.println(BACK);
251     System.out.println(QUIT);
252     System.out.println();
253     if (userOption == 1 || userOption == 2 || userOption == 3) {
254         append:
255         System.out.println(BACK);
256         System.out.println(QUIT);
257         System.out.println();
258     }
259     else {
260         /*If the entry is NOT valid, show by default ALL bookings. Append the screen with the
following to explain the problem:
261         <list bookings>
262         0. Back to main menu.
263         -1. Quit application.
264         <new line>*/
265         //System.out.println(list bookings);
266         System.out.println(BACK);

```

```

267         System.out.println(QUIT);
268         System.out.println();
269     }
270 }
271 else if (userOption == 8) {
272     //collection.clear(); //clear interface
273     Boolean valid = true;
274     System.out.println("University of Knowledge - COVID test");
275     System.out.println();
276     System.out.println("Adding booking (appointment for a COVID test) to the system");
277     System.out.println();
278     System.out.println("List of available time-slots:");
279     //system.out.println(List of available slots + 11, 12,...)
280     // 11. dd/mm/yyyy HH:MM
281     //12. dd/mm/yyyy HH:MM
282     //13. dd/mm/yyyy HH:MM
283     //...
284     System.out.println();
285     System.out.println("Please, enter one of the following:");
286     System.out.println();
287     System.out.println("The sequential ID of an available time-slot and the student email,
separated by a white space.");
288     System.out.println(BACK);
289     System.out.println(QUIT);
290     System.out.println();
291
292     if (valid = true){
293         //create a booking in the System
294         //If in the selected time-slot there are more than one combination of bookable rooms
and assistants on shift, you can implement any strategy to choose the resources for the booking. T
295         //just sequentially pick the first ones in the list
296         System.out.println("Booking added successfully:");

```



```
297         //System.out.println(print booking);
298         System.out.println();
299         System.out.println("List of available time-slots:");
300         //system.out.println(List of available slots + 11, 12,...)
301         //11. dd/mm/yyyy HH:MM;
302         //12. dd/mm/yyyy HH:MM;
303         //13. dd/mm/yyyy HH:MM;
304         // ...;
305         System.out.println();
306         System.out.println("Please, enter one of the following:");
307         System.out.println();
308         System.out.println("The sequential ID of an available time-slot and the student email,
separated by a white space.");
309         System.out.println(BACK);
310         System.out.println(QUIT);
311         System.out.println();
312     }
313     else if (valid = false) {
314         System.out.println(ERROR);
315
316         //add in some potential errors
317         //System.out.println(message explaining the error);
318
319         System.out.println("Please, enter one of the following:");
320         System.out.println();
321         System.out.println("The sequential ID of an available time-slot and the student email,
separated by a white space.");
322         System.out.println(BACK);
323         System.out.println(QUIT);
324         System.out.println();
325     }
326 }
```

```
327     else if (userOption == 9) {
328         //collection.clear(); //clear interface
329         Boolean valid = true;
330         System.out.println("University of Knowledge - COVID test");
331         System.out.println();
332         //System.out.println(list booking status:SCHEDULED); //list of bookings with the status
scheduled
333         System.out.println("Removing booking from the system");
334         System.out.println();
335         System.out.println("Please, enter one of the following:");
336         System.out.println();
337         System.out.println("The sequential ID to select the booking to be removed from the listed
bookings above.");
338         System.out.println(BACK);
339         System.out.println(QUIT);
340         System.out.println();
341         if (valid = true) {
342             //remove the booking from the system
343             System.out.println("Booking removed successfully:");
344             //System.out.println(print booking); //print the booking that was just removed
345             System.out.println("Please, enter one of the following:");
346             System.out.println();
347             System.out.println("The sequential ID to select the booking to be removed from the
listed bookings above.");
348             System.out.println(BACK);
349             System.out.println(QUIT);
350             System.out.println();
351         }
352         else if (valid = false) {
353             System.out.println(ERROR);
354
355             //add in some potential errors
```

```
356         //System.out.println(message explaining the error);
357
358
359
360         System.out.println("Please, enter one of the following:");
361         System.out.println();
362         System.out.println("The sequential ID to select the booking to be removed from the
listed bookings above.");
363         System.out.println(BACK);
364         System.out.println(QUIT);
365         System.out.println();
366     }
367 }
368 else if (userOption == 10) {
369     //collection.clear(); //clear interface
370     Boolean valid = true;
371     //Conclude Booking: If a user selects 10 from the Manage Booking menu, they can
conclude (finish) a booking.
372     //That is, the testing was performed as planned and the record can no longer be deleted
from the system.
373     //The screen shows:
374     System.out.println("University of Knowledge - COVID test");
375     System.out.println();
376
377     //System.out.println(list booking status:SCHEDULED) //print the list of bookings with the
status: SHEDULED
378
379     System.out.println("Conclude booking");
380     System.out.println();
381     System.out.println("Please, enter one of the following:");
382     System.out.println();
383     System.out.println("The sequential ID to select the booking to be completed.");
```

```

384     System.out.println(BACK);
385     System.out.println(QUIT);
386     System.out.println();
387
388
389     //If the sequential ID is valid, complete the respective booking in the system, and append
the screen with the following:
390     //complete booking, add to scheduled bookings
391
392     if (valid = true) {
393         System.out.println("Booking completed successfully:");
394         System.out.println(print Booking.getBooking()); //print the successful booking
395         System.out.println("Please, enter one of the following:");
396         System.out.println();
397         System.out.println("The sequential ID to select the booking to be completed.");
398         System.out.println(BACK);
399         System.out.println(QUIT);
400         System.out.println();
401     }
402     else if (valid = false) {
403
404         //If the entry is NOT valid, the system remains unchanged. Append the screen with the
following to explain the problem:
405         System.out.println(ERROR);
406         System.out.println("Your booking id is invalid, please try again");
407         System.out.println("Please, enter one of the following:");
408         System.out.println();
409         System.out.println("The sequential ID to select the booking to be completed.");
410         System.out.println(BACK);
411         System.out.println(QUIT);
412         System.out.println();
413     }

```

```
414
415     //public static final String BACK = "0. Back to main menu.";
416     else if (userOption == -1){ //" -1. Quit application."
417         System.exit(1);
418     }
419     else if (userOption == 0){ //"0. Back to main menu."
420         someMethod();
421     }
422 }
423
424 }
425
426 }
427
```

```
1
2 import java.util.ArrayList;
3 import java.util.Scanner;
4 import java.time.LocalDateTime;
5     /** This class is where bookings are made, added to the list of bookings, and also removed,
    thus it is also where assistants in shift and bookable rooms are created, added and removed */
6 public class BookingSystem {
7
8
9
10
11     private ArrayList<BookableRoom> bookableRooms=new ArrayList<BookableRoom>();
12     private ArrayList<AssistantOnShift> assistantsOnShifts=new ArrayList<AssistantOnShift>();
13     private ArrayList<Booking>bookings=new ArrayList<Booking>();
14     public static final String ENTERDAT = "Please enter a date and time";
15     public ArrayList createBookableRooms(){
16         /** This function creates the bookable rooms by adding them to the list of bookable rooms,
    with appropriate variables
17         * @return the list of bookable rooms after the new bookable rooms have been added
18         */
19         bookableRooms.add(new BookableRoom(2021-05-25 08:00:00, "EMPTY", BR12, 0)); //1
20         bookableRooms.add(new BookableRoom(2021-06-24 09:00:00, "AVAILABLE", BR23, 2)); //2
21         bookableRooms.add(new BookableRoom(2021-09-12 07:00:00, "EMPTY", BR34, 0)); //3
22         bookableRooms.add(new BookableRoom(2021-10-09 07:00:00, "FULL", BR43, 2)); //4
23         bookableRooms.add(new BookableRoom(2021-12-06 09:00:00, "AVAILABLE", BR49, 3)); //5
24         bookableRooms.add(new BookableRoom(2021-11-21 08:00:00, "FULL", BR51, 1)); //6
25         bookableRooms.add(new BookableRoom(2021-06-15 09:00:00, "AVAILABLE", BR60, 4)); //7
26         bookableRooms.add(new BookableRoom(2021-07-04 08:00:00, "AVAILABLE", BR73, 2)); //8
27         bookableRooms.add(new BookableRoom(2021-05-16 07:00:00, "FULL", BR98, 4)); //9
28
29         return bookableRooms;
30     }
```

```

31 //• 6 assistants on shift
32 public ArrayList createAssistantsOnShift(){
33     /** This function creates the assistants on shift by adding them to the list of assistants on
34     shift, with appropriate variables
35     * @return the list of assistantsOnShifts after the new assistantsOnShifts have been added
36     */
37     assistantsOnShifts.add(new AssistantOnShift(2021-05-10 07:00:00, "JA32@uok.ac.uk")); //1
38     assistantsOnShifts.add(new AssistantOnShift(2021-06-21 07:00:00, "Molly20@uok.ac.uk"));
39     //2
40     assistantsOnShifts.add(new AssistantOnShift(2021-09-09 07:00:00, "Sam43@uok.ac.uk"));
41     //3
42     assistantsOnShifts.add(new AssistantOnShift(2021-07-07 07:00:00, "Bryan90@uok.ac.uk"));
43     //4
44     assistantsOnShifts.add(new AssistantOnShift(2021-10-16 07:00:00, "Jose56@uok.ac.uk"));
45     //5
46     assistantsOnShifts.add(new AssistantOnShift(2021-11-05 07:00:00, "Amy72@uok.ac.uk"));
47     //6
48
49     return assistantsOnShifts;
50 }
51
52
53
54
55 public BookableRoom getBookableRoom(){
56     /**@return the bookable room*/
57     return this.BookableRoom;
58 }
59
60
61
62
63 //list of bookable rooms status:EMPTY
64 ArrayList EmptyBookableRooms;
65 String EMPTY;

```

```

58
59 public void addBookableRooms() {
60     /** add a new bookable room to the list*/
61
62     //bookableRooms.add(dateAndTime, status,roomCode, occupancy);
63     //bookableRooms.add(12/03/2022, AVAILABLE,A567, 1);
64     //Print template: | <dd/mm/yyyy HH:MM> | <status> | <roomCode> | occupancy:
    <occupancy> |
65     Scanner scanner = new Scanner(System.in);
66     System.out.print(ENTERDAT);
67     String dateAndTimeString = scanner.next();
68     LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
69     System.out.print("Please enter a room code");
70     String code = scanner.next();
71     int oocupancy = BookableRoom.updateOccupancy(); //increase the occupancy by 1 //STATIC
    ERRORR CHECK
72     int status = BookableRoom.getRoomBookableRoomStatus(); //STATIC ERRORR CHECK //get
    the statuis of the room
73     bookableRooms.add(new BookableRoom(dateAndTime, Status, code, occupancy));
74
75 }
76
77 public void addAssistantsOnShift() {
78     /** add a new assistanmt on shift to the list*/
79
80     // Print template: | <Odd/mm/yyyy HH:MM> | <status> | <assistantEmail> |
81     Scanner scanner = new Scanner(System.in);
82
83     System.out.print(ENTERDAT);
84     String dateAndTimeString = scanner.next();
85     LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
86     System.out.print("Please enter an assistant's email");

```



```

87     String email = scanner.next();
88     //AssistantOnShift.getAssistantStatus(); //get the status of the assistant
89     assistantsOnShifts.add(new AssistantOnShift(dateAndTime, email));
90 }
91
92 public void addBookings() {
93     /** add a new booking to the list*/
94
95     //7. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> | <roomCode>
    | <studentEmail> |
96     Scanner scanner = new Scanner(System.in);
97
98     System.out.print(ENTERDAT);
99     String dateAndTimeString = scanner.next();
100     LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
101     System.out.print("Please enter an assistant's email");
102     String email = scanner.next();
103     System.out.print("Please enter a room code");
104     String code = scanner.next();
105     System.out.print("Please enter a student's email");
106     String studentEmail = scanner.next();
107     //BookableRoom.updateOccupancy(); //increase the occupancy of the room by 1
    //CHECK
108
109     BookableRoom.getBookableRoomStatus(); //get the status of the room //CHECK
110     AssistantOnShift.getAssistantStatus(); //get the status of the assistant //CHECK
111     Status = Booking.getBookingStatus(); //get the status of the booking
112     //Booking b = new Booking();
113     bookings.add(new Booking(dateAndTime, Status, email, code, studentEmail));
114
115 }
116 public String removeBookableRooms() {

```

```

117     /** remove a bookable room from the list
118     * @return a string confirming to the user that the bookable room has been removed
119     */
120     Scanner scanner = new Scanner(System.in);
121     System.out.print("Please enter the code of the room you wish to delete");
122     String UserCode = scanner.next();
123     int deleteIndex1 = bookableRooms.indexOf(UserCode);
124     bookableRooms.remove(deleteIndex1);
125     return "The bookable room has been removed";
126 }
127
128 public String removeAssistantOnShift() {
129     /** remove an assitant on shift from the list
130     * @return a string confirming to the user that the assistant on shift has been removed
131     */
132     Scanner scanner = new Scanner(System.in);
133     System.out.print("Please enter the email address of the assistant on shift you would like
to remove");
134     String UserEmail1 = scanner.next();
135     int deleteIndex2 = assistantsOnShifts.indexOf(UserEmail1);
136     assistantsOnShifts.remove(deleteIndex2);
137     return "The assistant on shift has been removed";
138 }
139
140 public String removeBookings() {
141     /** remove a booking from the list
142     * @return a string confriming to the user that the booking jas been removed
143     */
144     Scanner scanner = new Scanner(System.in);
145     System.out.print("Please enter the students email of the booking you would like to
remove");
146     String UserEmail2 = scanner.next();

```

```
147     int deleteIndex3 = bookings.indexOf(UserEmail2);
148     bookings.remove(deleteIndex3);
149     return "The booking has been removed";
150 }
```

151

```
152 public ArrayList showAssistantsOnShift(){
153     /** @return the list of assistants on shift */
154     return assistantsOnShifts;
155 }
```

```
156 public ArrayList showBookableRooms(){
157     /** @return the list of bookable rooms */
158     return bookableRooms;
159 }
```

```
160 public ArrayList showBookings(){
161     /** @return the list bookings */
162     return bookings;
163 }
```

```
164 }//DO: check
```

165 //3. There is a time-slot concept that will guide the booking system. For instance, rooms will be available, and

166 //assistants will work at a specific time-slot, i.e., date, time and duration. Hence, tests should be booked at

167 //available slots.

168 //4. Every time-slot has a fixed duration – a positive number representing the duration of a test, in minutes. This

169 //quantity includes the time spent doing the test and the time to sanitize the room. The current policy establishes

170 //this duration to be 60 minutes.

171

172 /\*1. The booking System is responsible for most functionalities. It has a list of bookable rooms, a list of assistants on

173 shift, and a list of bookings.

174

175 3. There is a time-slot concept that will guide the booking System. For instance, rooms will be available, and

176 assistants will work at a specific time-slot, i.e., date, time and duration. Hence, tests should be booked at

177 available slots.

178

179 4. Every time-slot has a fixed duration – a positive number representing the duration of a test, in minutes. This

180 quantity includes the time spent doing the test and the time to sanitize the room. The current policy establishes

181 this duration to be 60 minutes.

182 \*/

183

```

1 import java.util.ArrayList;

2 /** This class defines a room (object) in the university that can be used as a bookable room for
   covid tests */

3 public class Room {

4     public String code;

5     public int capacity;

6

7

8     public String getCode() {

9         /** This function is a getter method to retrieve the unique room code

10         @return the unique room code */

11         this.code = code;

12         UniversityResources.getRooms();

13         ArrayList rooms = UniversityResources.getRooms();

14         if(rooms.contains(this.code)){

15             System.out.print("This room code is already being used for another room, please use
another room code.");

16         }

17         else {

18             return this.code;

19         }

20     }

21

22     public int getCapacity() {

23         /** This function is a getter method used to retrieve the capacity of a room if it is not < or = to
0

24         @return the capacity if it is legitimate, otherwise @return 1 as a default room capacity */

25         this.capacity = capacity;

26         if (this.capacity <= 0){

27             System.out.print("The capacity needs to be an integer value greater than 0, please try
again");

28             return 1;

```

```

29     }
30     else {
31         return this.capacity;
32     }
33 }
34
35 //constructor:
36 public Room(String code, int capacity) {
37     /**This function constructs the room class*/
38     this.code = code;
39     this.capacity = capacity;
40 }
41 }
42 //System.out.print(getCode() + getCapacity());
43
44 /*1.1.2 Room
45 public Module(int year, byte term, ModuleDescriptor module, StudentRecord[] records, double
finalAverageGrade) {
46     this.year = year;
47 1. The university has several rooms, and some of the rooms can be allocated to apply COVID
tests.
48 2. A room must have a string code (e.g., IC215) and a capacity.
49 3. The code is used to identify the room and, therefore, must be unique.
50 4. The capacity must be an integer value greater than zero. It represents the number of
concurrent assistants that
51 can be safely allocated in the room to perform tests.
52 5. Print template: | <code> | capacity: <capacity> | */
53

```

```
1 import java.lang.reflect.Method;
2 import java.util.ArrayList;
3 import java.util.Scanner;
4 /**This class is where all of the university resources are made, such s rooms and assitants. */
5 public class UniversityResources {
6     private ArrayList<Assistant> assistants = new ArrayList<Assistant>();
7     public static ArrayList<Room> rooms = new ArrayList<Room>();
8
9     public static ArrayList getRooms(){
10         /** @return the list of rooms */
11         return rooms;
12     }
13     public ArrayList getAssistants() {
14         /** @return the list of assistants*/
15         return assistants;
16     }
17
18     public ArrayList createRooms(){
19         /** This function creates the rooms by adding them to the list of rooms, with apporriate
20         variables for the room code and capcity
21         * @return the list of rooms after the new rooms have been added
22         */
23         //getcode();
24         rooms.add(new Room("A33",5));
25         rooms.add(new Room("B22", 1));
26         rooms.add(new Room("C39", 2));
27         rooms.add(new Room("D54", 3));
28         rooms.add(new Room("E28", 1));
29         return rooms;
30     }
31     public ArrayList createAssistants() {
```

```

31  /** This function creayes the assistants by adding them to the list of assistants, with
appropriate variables for the name and email
32      * @return the list of assistants after the new assistants have been added
33      */
34  assistants.add(new Assistant("Julie", "JA32@uok.ac.uk"));
35  assistants.add(new Assistant("Sandy", "MrsSandMan@uok.ac.uk"));
36  assistants.add(new Assistant("Louise", "LoulOU00@uok.ac.uk"));
37  assistants.add(new Assistant("Kathrin", "Kay7890@uok.ac.uk"));
38  assistants.add(new Assistant("Robbert", "Bobbles@uok.ac.uk"));
39  return assistants;
40  }

41  //private ArrayList<Assistant> assistants = new ArrayList<Assistant>();
42
43
44 public String addRoom() {
45  /** This function allows the user to add a room to the list of rooms (if a new room is available
for testing in the university for example)
46      * @return a message confirming to the user that the room has been added
47      */
48
49  Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
50  System.out.print("Please enter a room code");
51  String code = sc.nextLine(); //reads string
52  System.out.print("Please enter a capacity");
53  String cap = sc.nextLine();
54  int capInt=Integer.parseInt(cap);
55  rooms.add(new Room(code, capInt));
56  return "The room has been added";
57  }
58
59 public String addAssistant() {

```



```

60  /**This function allows the user to add an assistant to the list of assists (if for example a new
memeber of staff at the university now has the time to jelp with covid tests)
61  *@return a message confirming to the user that the assistant has been added
62  */
63  Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
64  System.out.print("Please enter an assistant name");
65  String name = sc.nextLine(); //reads string
66  System.out.print("Please enter an email");
67  String email = sc.nextLine(); //reads string
68  assistants.add(new Assistant(name, email));
69  return "The assistant has been added";
70 }
71 //constructor:
72 public UniversityResources(ArrayList assistants, ArrayList rooms) {
73  /**This function sonstructs the University Recourses class*/
74  this.assistants = assistants;
75  this.rooms = rooms;
76 }
77 }
78 /*University Resources
79 1. The University has a list of assistants and a list of rooms.
80 2. You should implement functions to add, both assistants and rooms.
81 3. Due to time constraints, you don't need to develop screen to manage the university resources,
but you need to
82 pre-load the system with instances of rooms and assistants. Please, check section 1.3 for more
details.
83 */
84

```