```java
/**
 * The type Room.
 */
public class Room {
    private String code;
    private int capacity;

    /**
     * Instantiates a new Room.
     *
     * @param code     the code
     * @param capacity the capacity
     */
    public Room(String code,  int capacity){
        this.code = code;
        this.capacity = capacity;
    }

    public String toString() {
        return "| " + code + " | Capacity: " + capacity + " |";
    }

    /**
     * Gets code.
     *
     * @return the code
     */
    public String getCode() {
        return code;
    }

    /**
     * Gets capacity.
     *
     * @return the capacity
     */
    public int getCapacity() {
        return capacity;
    }
}

```

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * The type Booking.
 */
public class Booking {
    private BookableRoom room;
    private AssistantOnShift assistant;
    private LocalDateTime startTime;
    private BookingStatus status;
    private DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
    private String studentEmail;

    /**
     * Instantiates a new Booking.
     *
     * @param room         the room
     * @param assistant    the assistant
     * @param studentEmail the student email
     * @param startTime    the start time
     */
    public Booking(BookableRoom room, AssistantOnShift assistant, String studentEmail, LocalDateTime startTime) {
        this.room = room;
        room.addOccupant();
        this.assistant = assistant;
        assistant.setStatus(StaffStatus.BUSY);
        this.studentEmail = studentEmail;
        this.startTime = startTime;
        status = BookingStatus.SCHEDULED;
    }

    /**
     * Gets assistant.
     *
     * @return the assistant
     */
    public AssistantOnShift getAssistant() {
        return assistant;
    }

```

```
42      /**
43       * Gets room.
44       *
45       * @return the room
46       */
47      public BookableRoom getRoom() {
48          return room;
49      }
50
51      /**
52       * Gets start time.
53       *
54       * @return the start time
55       */
56      public LocalDateTime getStartTime() {
57          return startTime;
58      }
59
60      /**
61       * Gets status.
62       *
63       * @return the status
64       */
65      public BookingStatus getStatus() {
66          return status;
67      }
68
69      /**
70       * Sets status.
71       *
72       * @param status the status
73       */
74      public void setStatus(BookingStatus status) {
75          this.status = status;
76      }
77
78      public String toString() {
79          return "| " + startTime.format(dateFormat) +
    " | " + status + " | " + assistant.getAssistant().
    getEmail() + " | " + room.getRoom().getCode() + " | "
     + studentEmail + " |";
80      }
81  }
82
```

```java
/**
 * The type Assistant.
 */
public class Assistant {
    private String name;
    private String email;

    /**
     * Instantiates a new Assistant.
     *
     * @param name  the name
     * @param email the email
     */
    public Assistant(String name, String email) {
        this.name = name;
        this.email = email;
    }

    public String toString(){
        return "| " + name + " | " + email + " |";
    }

    /**
     * Gets email.
     *
     * @return the email
     */
    public String getEmail() {
        return email;
    }
}

```

```java
import java.util.Scanner;

/**
 * The type Booking app.
 */
public class BookingApp {

    /**
     * The entry point of application.
     *
     * @param args the input arguments
     */
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        BookingSystem bookingSystem = new BookingSystem(true);

        //main system loop
        while(bookingSystem.isLive()) {
            BookingSystem.clearScreen();
            mainMenu();
            switch (scan.nextLine()) {
                case "-1" -> bookingSystem.setLive(false);
                case "1" -> bookingSystem.showBookableRooms();
                case "2" -> bookingSystem.addBookableRoom();
                case "3" -> bookingSystem.removeBookableRoom();
                case "4" -> bookingSystem.showAssistantOnShift();
                case "5" -> bookingSystem.addAssistantOnShift();
                case "6" -> bookingSystem.removeAssistantOnShift();
                case "7" -> bookingSystem.showBookings();
                case "8" -> bookingSystem.addBooking();
                case "9" -> bookingSystem.removeBooking();
                case "10" -> bookingSystem.concludeBooking();
```

```
33                  default -> System.out.println("
   invalid input, try again");
34              }
35          }
36      }
37
38      /**
39       * Main menu.
40       */
41      public static void mainMenu() {
42          String menu = """
43                  University of Knowledge - COVID test
44                  \s
45                  Manage Bookings
46                  \s
47                  Please, enter the number to select
   your option:
48                  To manage Bookable Rooms:
49                  1. List
50                  2. Add
51                  3. Remove
52                  To manage Assistants on Shift:
53                  4. List
54                  5. Add
55                  6. Remove
56                  To manage Bookings:
57                  7. List
58                  8. Add
59                  9. Remove
60                  10. Conclude
61                  After selecting one the options above
   , you will be presented other screens.
62                  If you press 0, you will be able to
   return to this main menu.
63                  Press -1 (or ctrl+c) to quit this
   application.
64                  """;
65
66          System.out.println(menu);
67      }
68  }
```

```java
/**
 * The enum Room status.
 */
enum RoomStatus {
    /**
     * Empty room status.
     */
    EMPTY,
    /**
     * Available room status.
     */
    AVAILABLE,
    /**
     * Full room status.
     */
    FULL}

```

```java
/**
 * The enum Staff status.
 */
enum StaffStatus {
    /**
     * Free staff status.
     */
    FREE,
    /**
     * Busy staff status.
     */
    BUSY}

```

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * The type Bookable room.
 */
public class BookableRoom {

    private Room room;
    private LocalDateTime startTime;
    private RoomStatus status;
    private int occupancy;
    private DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

    /**
     * Instantiates a new Bookable room.
     *
     * @param room      the room
     * @param startTime the start time
     */
    public BookableRoom(Room room, LocalDateTime startTime) {
        this.room = room;
        this.startTime = startTime;
        status = RoomStatus.EMPTY;
        occupancy = 0;
    }

    /**
     * Gets status.
     *
     * @return the status
     */
    public RoomStatus getStatus() {
        return status;
    }

    /**
     * Sets status.
     *
     * @param status the status
     */
    public void setStatus(RoomStatus status) {
```

```
43          this.status = status;
44      }
45
46      /**
47       * Gets room.
48       *
49       * @return the room
50       */
51      public Room getRoom() {
52          return room;
53      }
54
55      /**
56       * Gets start time.
57       *
58       * @return the start time
59       */
60      public LocalDateTime getStartTime() {
61          return startTime;
62      }
63
64      /**
65       * Add occupant.
66       */
67      public void addOccupant(){
68          occupancy++;
69          if(room.getCapacity()==occupancy)
70              status=RoomStatus.FULL;
71          else
72              status=RoomStatus.AVAILABLE;
73      }
74
75      /**
76       * Remove occupant.
77       */
78      public void removeOccupant(){
79          occupancy--;
80          if(occupancy==0)
81              status=RoomStatus.EMPTY;
82          else
83              status=RoomStatus.AVAILABLE;
84      }
85
86      public String toString() {
```

```
87          return "| " + startTime.format(dateFormat
    ) + " | " + status + " | " + room.getCode() + " |
    occupancy: " + occupancy +" |";
88      }
89  }
90
```

```java
import java.util.Comparator;

/**
 * The type Booking sorter.
 */
public class BookingSorter implements Comparator<Booking>
{
    @Override
    public int compare(Booking o1, Booking o2) {
        return o1.getStartTime().compareTo(o2.getStartTime());
    }
}
```

```java
/**
 * The enum Booking status.
 */
enum BookingStatus {
    /**
     * Scheduled booking status.
     */
    SCHEDULED,
    /**
     * Completed booking status.
     */
    COMPLETED}

```

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.regex.Pattern;

/**
 * The type Booking system.
 */
public class BookingSystem {
    //arraylists
    private UniversityRecourses recourses = new UniversityRecourses();
    private ArrayList<Room> rooms = recourses.getRooms();
    private ArrayList<BookableRoom> bookableRooms = new ArrayList<>();
    private ArrayList<Assistant> assistants = recourses.getAssistants();
    private ArrayList<AssistantOnShift> assistantsOnShift = new ArrayList<>();
    private ArrayList<Booking> bookings = new ArrayList<>();
    private ArrayList<LocalDateTime> timeSlots = new ArrayList<>();

    //attributes and such
    private boolean live;
    /**
     * The Scan.
     */
    Scanner scan = new Scanner(System.in);
    private DateTimeFormatter dateTimeFormat = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");


    /**
     * Instantiates a new Booking system.
     *
     * @param live the live
     */
    public BookingSystem(boolean live) {
        this.live = live;
```

```
37          initialiseData();
38      }
39
40      /**
41       * Initialise data.
42       */
43      public void initialiseData() {
44          bookableRooms.add(new BookableRoom(rooms.get(
    0), LocalDateTime.parse("04/05/2021 07:00",
    dateTimeFormat)));//0
45          bookableRooms.add(new BookableRoom(rooms.get(
    0), LocalDateTime.parse("04/05/2021 08:00",
    dateTimeFormat)));//1
46          bookableRooms.add(new BookableRoom(rooms.get(
    0), LocalDateTime.parse("04/05/2021 09:00",
    dateTimeFormat)));//2
47          bookableRooms.add(new BookableRoom(rooms.get(
    1), LocalDateTime.parse("04/05/2021 07:00",
    dateTimeFormat)));//3
48          bookableRooms.add(new BookableRoom(rooms.get(
    1), LocalDateTime.parse("04/05/2021 08:00",
    dateTimeFormat)));//4
49          bookableRooms.add(new BookableRoom(rooms.get(
    1), LocalDateTime.parse("04/05/2021 09:00",
    dateTimeFormat)));//5
50          bookableRooms.add(new BookableRoom(rooms.get(
    2), LocalDateTime.parse("04/05/2021 07:00",
    dateTimeFormat)));//6
51          bookableRooms.add(new BookableRoom(rooms.get(
    2), LocalDateTime.parse("04/05/2021 08:00",
    dateTimeFormat)));//7
52          bookableRooms.add(new BookableRoom(rooms.get(
    2), LocalDateTime.parse("04/05/2021 09:00",
    dateTimeFormat)));//8
53
54          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(0), LocalDateTime.parse("04/05/2021 07
    :00", dateTimeFormat)));//0
55          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(0), LocalDateTime.parse("04/05/2021 08
    :00", dateTimeFormat)));//1
56          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(1), LocalDateTime.parse("04/05/2021 09
    :00", dateTimeFormat)));//2
```

```
57          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(1), LocalDateTime.parse("04/05/2021
    07:00", dateTimeFormat)));//3
58          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(2), LocalDateTime.parse("04/05/2021
    08:00", dateTimeFormat)));//4
59          assistantsOnShift.add(new AssistantOnShift(
    assistants.get(2), LocalDateTime.parse("04/05/2021
    09:00", dateTimeFormat)));//5
60
61          bookings.add(new Booking(bookableRooms.get(8
    ), assistantsOnShift.get(5), "mijevils@uok.ac.uk",
    assistantsOnShift.get(5).getStartTime()));
62          bookings.add(new Booking(bookableRooms.get(0
    ), assistantsOnShift.get(0), "frol@uok.ac.uk",
    assistantsOnShift.get(0).getStartTime()));
63          bookings.add(new Booking(bookableRooms.get(4
    ), assistantsOnShift.get(4), "inshibob@uok.ac.uk",
    assistantsOnShift.get(4).getStartTime()));
64          bookings.get(2).setStatus(BookingStatus.
    COMPLETED); bookings.get(2).getRoom().removeOccupant
    (); bookings.get(2).getAssistant().setStatus(
    StaffStatus.FREE);
65      }
66
67      /**
68       * List rooms.
69       */
70  /*
71      Display all the items of the different lists
    with certain status'
72       */
73      public void listRooms() {
74          System.out.println("List of rooms:");
75          for(int n = 0; n < rooms.size(); n++) {
76              System.out.println(n+11 + ". " + rooms.
    get(n).toString());
77          }
78      }
79
80      /**
81       * List bookable rooms.
82       *
83       * @param empty the empty
```

```
 84      */
 85     public void listBookableRooms(Boolean empty) {
 86         bookableRooms.sort(new BookableRoomSorter
    ());
 87 
 88         if(empty){
 89             System.out.println("List of empty
    bookable rooms:");
 90             for(int n = 0; n < bookableRooms.size
    (); n++) {
 91                 if(bookableRooms.get(n).getStatus().
    equals(RoomStatus.EMPTY))
 92                     System.out.println(n+11 + ". "
     + bookableRooms.get(n).toString());
 93             }
 94         }
 95         else{
 96             System.out.println("List of bookable
    rooms:");
 97             for(int n = 0; n < bookableRooms.size
    (); n++)
 98                 System.out.println(n+11 + ". " +
    bookableRooms.get(n).toString());
 99         }
100     }
101 
102     /**
103      * List assistants.
104      */
105     public void listAssistants() {
106         System.out.println("List of assistants:");
107         for(int n = 0; n < assistants.size(); n++) {
108             System.out.println(n+11 + ". " +
    assistants.get(n).toString());
109         }
110     }
111 
112     /**
113      * List assistants on shift.
114      *
115      * @param free the free
116      */
117     public void listAssistantsOnShift(Boolean free
    ) {
```

```
118         assistantsOnShift.sort(new
    AssistantOnShiftSorter());
119
120         if(free){
121             System.out.println("List of free
    assistants on shift:");
122             for(int n = 0; n < assistantsOnShift.
    size(); n++) {
123                 if(assistantsOnShift.get(n).
    getStatus().equals(StaffStatus.FREE))
124                     System.out.println(n+11 + ". "
     + assistantsOnShift.get(n).toString());
125             }
126         }
127         else {
128             System.out.println("List of assistants
    on shift:");
129             for(int n = 0; n < assistantsOnShift.
    size(); n++)
130                 System.out.println(n+11 + ". " +
    assistantsOnShift.get(n).toString());
131         }
132     }
133
134     /**
135      * List bookings.
136      *
137      * @param Status the status
138      */
139     public void listBookings(String Status) {
140         bookings.sort(new BookingSorter());
141         System.out.println("List of bookings: " +
    Status);
142         switch (Status){
143             case "ALL":
144                 for(int n = 0; n< bookings.size(); n
    ++)
145                     System.out.println(n+11 + ". "
     + bookings.get(n).toString());
146                 break;
147             case "SCHEDULED":
148                 for(int n = 0; n< bookings.size(); n
    ++) {
149                     if(bookings.get(n).getStatus().
```

```
149 equals(BookingStatus.SCHEDULED))
150                         System.out.println(n + 11 +
    ". " + bookings.get(n).toString());
151                 }
152                 break;
153             case "COMPLETED":
154                 for(int n = 0; n< bookings.size(); n
    ++) {
155                     if(bookings.get(n).getStatus().
    equals(BookingStatus.COMPLETED))
156                         System.out.println(n + 11 +
    ". " + bookings.get(n).toString());
157                 }
158                 break;
159         }
160     }
161 
162     /**
163      * List time slots.
164      */
165     public void listTimeSlots() {
166         timeSlots.sort(new TimeSlotSorter());
167         System.out.println("List of timeslots");
168         for(int n = 0; n < timeSlots.size(); n++) {
169             System.out.println(n+11 + ". " +
    timeSlots.get(n).format(dateTimeFormat).toString());
170         }
171         System.out.println();
172     }
173 
174     /**
175      * Show bookable rooms.
176      */
177 /*
178     bookable Rooms
179      */
180     public void showBookableRooms() {
181         String input = "pokemon lets show";
182         while(!(input.equals("0") || input.equals("-
    1"))) {
183             clearScreen();
184             System.out.println("University of
    Knowledge - COVID test \n ");
185             listBookableRooms(false);
```

```
186             System.out.println("""
187                     0. Back to main menu.\s
188                     -1. Quit application.\s
189                     \s
190                     """);
191             input = scan.nextLine();
192             if (input.equals("-1"))
193                 live = false;
194         }
195     }
196 
197     /**
198      * Add bookable room.
199      */
200     public void addBookableRoom() {
201         boolean successfulAdd = false;
202         boolean validID = false;
203         boolean validDate = false;
204         int id;
205         String input;
206         String[] roomInfo;
207         LocalDateTime dateTime = LocalDateTime.now
    ();
208         String errorMessage = "we had an
    unidentified whoopsie";
209 
210         clearScreen();
211         System.out.println("""
212                 University of Knowledge - COVID test
    \s
213                 \s
214                 Adding bookable room\s
215                 """);
216 
217         listRooms();
218 
219         System.out.println("""
220                 \s
221                 Please, enter one of the following:\
    s
222                 \s
223                 The sequential ID listed to a room,
    a date (dd/mm/yyyy), and a time (HH:MM),separated by
     a white space.
```

```
224                 0. Back to main menu.\s
225                 -1. Quit application.\s
226                 """);
227
228         input = scan.nextLine();
229
230         while(!(input.equals("0") || input.equals("-
    1"))) {
231             successfulAdd = false;
232             validID = false;
233             validDate = false;
234
235             if(Pattern.matches("[0-9][0-9][ ][0-9][0
    -9][/][0-9][0-9][/][0-9][0-9][0-9][0-9][ ][0-9][0-9
    ][:][0-9][0-9]", input)) {
236                 roomInfo = input.split(" ", 2);
237                 id = Integer.parseInt(roomInfo[0
    ]) - 11;
238
239                 if (id < rooms.size() && id >= 0)
240                     validID = true;
241                 else {
242                     errorMessage = "ID invalid";
243                     validID = false;
244                 }
245
246                 try {
247                     dateTime = LocalDateTime.parse(
    roomInfo[1], dateTimeFormat);
248                     validDate = true;
249                 } catch (Exception e) {
250                     errorMessage = "Date and time
    are invalid";
251                     validDate = false;
252                 }
253
254                 if (validDate && validID) {
255                     for (BookableRoom bookableRoom
     : bookableRooms) {
256                         //if time and room id are
    the same, room already assigned to time
257                         if (bookableRoom.
    getStartTime().equals(dateTime) && bookableRoom.
    getRoom().getCode().equals(rooms.get(id).getCode
```

```
257 ())) {
258                             validDate = false;
259                             errorMessage = "This
    room is already assigned to this timeslot";
260                         }
261                         //if time is 7am, 8am or 9am
     then its in a valid timeslot
262                         else if ((dateTime.getHour
    () == LocalDateTime.parse("01/01/2021 07:00",
    dateTimeFormat).getHour()) || (dateTime.getHour
    () == LocalDateTime.parse("01/01/2021 08:00",
    dateTimeFormat).getHour()) || (dateTime.getHour
    () == LocalDateTime.parse("01/01/2021 09:00",
    dateTimeFormat).getHour())) {
263                             validDate = true;
264                         }
265                         //if not in timeslot or in
    use then bad
266                         else {
267                             validDate = false;
268                             errorMessage = "Not a
    valid timeslot, please make rooms bookable for 7am,
    8am or 9am";
269                         }
270                     }
271                 }
272
273                 if (validDate && validID) {
274                     successfulAdd = true;
275                     bookableRooms.add(new
    BookableRoom(rooms.get(id), dateTime));
276                 }
277             }
278             else {
279                 errorMessage = "Input in wrong
    format, try again";
280                 successfulAdd = false;
281             }
282
283             clearScreen();
284             if (successfulAdd) {
285                 System.out.println("Bookable Room
    added successfully:");
286                 System.out.println(bookableRooms.get
```

```
286 (bookableRooms.toArray().length - 1).toString());
287             } else {
288                 System.out.println("Error!");
289                 System.out.println(errorMessage);
290             }
291             listRooms();
292             System.out.println("""
293                     Please, enter one of the
    following:
294                     \s
295                     The sequential ID listed to a
    room, a date (dd/mm/yyyy), and a time (HH:MM),
296                     separated by a white space.
297                     0. Back to main menu.
298                     -1. Quit application.
299                     """);
300
301             input = scan.nextLine();
302             if (input.equals("-1"))
303                 live = false;
304         }
305     }
306
307     /**
308      * Remove bookable room.
309      */
310     public void removeBookableRoom() {
311         String input;
312         String errorMessage = "We had an
    unidentified whoopsie";
313         int id = 0;
314         boolean succesfulRemove;
315
316         clearScreen();
317         System.out.println("University of Knowledge
     - COVID test");
318         listBookableRooms(true);
319         System.out.println("""
320                 Removing bookable room\s
321                 \s
322                 Please, enter one of the following:\
    s
323                 \s
324                 The sequential ID to select the
```

```
bookable room to be removed.\s
                0. Back to main menu.\s
                -1. Quit application.\s
                """);

        input = scan.nextLine();

        while(!(input.equals("0") || input.equals("-1"))) {
            if(Pattern.matches("[0-9][0-9]", input)){
                id = Integer.parseInt(input)-11;
                if(id < bookableRooms.size() && id >= 0 && bookableRooms.get(id).getStatus().equals(RoomStatus.EMPTY))
                    succesfulRemove = true;
                else{
                    succesfulRemove = false;
                    errorMessage = "ID out of range, try a valid ID";
                }
            }
            else{
                succesfulRemove = false;
                errorMessage = "ID not valid, try a valid ID";
            }

            clearScreen();
            if(succesfulRemove) {
                System.out.println("Bookable Room removed successfully");
                System.out.println(bookableRooms.get(id).toString()+"\n");
                bookableRooms.remove(id);
            }
            else{
                System.out.println("Error!");
                System.out.println(errorMessage);
            }

            System.out.println("Please, enter one of the following:\n");
            listBookableRooms(true);
```

```
359             System.out.println("""
360                     \s
361                     The sequential ID to select the
    bookable room to be removed.\s
362                     0. Back to main menu.\s
363                     -1. Quit application.\s
364                     """);
365
366             input = scan.nextLine();
367             if (input.equals("-1"))
368                 live = false;
369         }
370     }
371
372     /**
373      * Show assistant on shift.
374      */
375 /*
376     Assistants on shift
377      */
378     public void showAssistantOnShift() {
379         String input = "ultimate show down";
380         while(!(input.equals("0") || input.equals("-
    1"))) {
381             clearScreen();
382             System.out.println("University of
    Knowledge - COVID test \n \n");
383             listAssistantsOnShift(false);
384             System.out.println("""
385                     0. Back to main menu.\s
386                     -1. Quit application.\s
387                     \s
388                     """);
389             input = scan.nextLine();
390             if (input.equals("-1"))
391                 live = false;
392         }
393     }
394
395     /**
396      * Add assistant on shift.
397      */
398     public void addAssistantOnShift() {
399         String input;
```

```
400         String[] assistantInfo;
401         int id;
402         boolean validID = false;
403         String errorMessage = "we had an
    unidentified whoopsie";
404         LocalDateTime dateTime = LocalDateTime.now
    ();
405         boolean validDateTime = false;
406         boolean successfulAdd = false;
407         int numAdded = 0;
408 
409         clearScreen();
410         System.out.println("""
411                 University of Knowledge - COVID test
    \s
412                 \s
413                 Adding assistant on shift\s
414                 \s
415                 """);
416 
417         listAssistants();
418 
419         System.out.println("""
420                 Please, enter one of the following:\
    s
421                 \s
422                 The sequential ID of an assistant
    and date (dd/mm/yyyy), separated by a white space.\s
423                 0. Back to main menu.\s
424                 -1. Quit application.\s
425                 """);
426 
427         input = scan.nextLine();
428 
429         while(!(input.equals("0") || input.equals("-
    1"))) {
430             validID = false;
431             validDateTime = false;
432             successfulAdd = true;
433             numAdded = 0;
434 
435             if(Pattern.matches("[0-9][0-9][ ][0-9][0
    -9][/][0-9][0-9][/][0-9][0-9][0-9][0-9]", input)){
436                 assistantInfo = input.split(" ", 2);
```

```
437                 id = Integer.parseInt(assistantInfo[
    0]) - 11;
438
439                 if (id < rooms.size() && id >= 0)
440                     validID = true;
441                 else {
442                     errorMessage = "ID invalid";
443                     validID = false;
444                 }
445
446                 try {
447                     dateTime = LocalDateTime.parse(
    assistantInfo[1] + " 07:00", dateTimeFormat);
448                     validDateTime = true;
449                 } catch (Exception e) {
450                     errorMessage = "Date and time
    are invalid";
451                     validDateTime = false;
452                 }
453
454                 clearScreen();
455                 if (validDateTime && validID) {
456                     //for time slots h: 0=7am, 1=8am
    , 2=9am
457                     for (int h = 0; h<3; h++) {
458                         //for each assistant in the
    timeslot
459                         for(AssistantOnShift
    assistantOnShift: assistantsOnShift) {
460                             //if assistant and
    timeslot are the same, assistant already on shift
461                             if(assistantOnShift.
    getAssistant().equals(assistants.get(id))&&
    assistantOnShift.getStartTime().equals(dateTime.
    plusHours(h))){
462                                 successfulAdd =
    false;
463                                 errorMessage = "
    Assistant already on shift this timeslot";
464                                 System.out.println("
    Error!");
465                                 System.out.println(
    errorMessage);
466                                 break;
```

```
467                             }
468                             else
469                                 successfulAdd = true
    ;
470                         }
471                         if(successfulAdd){
472                             assistantsOnShift.add(
    new AssistantOnShift(assistants.get(id), dateTime.
    plusHours(h)));
473                             System.out.println("
    Assistant on Shift added successfully:");
474                             System.out.println(
    assistantsOnShift.get(assistantsOnShift.size()-1).
    toString());
475                         }
476                         else {
477                             System.out.println("
    Error!");
478                             System.out.println(
    errorMessage);
479                         }
480                     }
481                 }
482             }
483             else{
484                 errorMessage = "Input in wrong
    format, try again";
485                 successfulAdd = false;
486             }
487
488             if (!successfulAdd) {
489                 clearScreen();
490                 System.out.println("Error!");
491                 System.out.println(errorMessage);
492             }
493             listAssistants();
494             System.out.println("""
495                     Please, enter one of the
    following:\s
496                     \s
497                     The sequential ID of an
    assistant and date (dd/mm/yyyy),\s
498                     separated by a white space.\s
499                     0. Back to main menu.\s
```

```
500                     -1. Quit app\s
501                     """);
502
503             input = scan.nextLine();
504             if (input.equals("-1"))
505                 live = false;
506         }
507     }
508
509     /**
510      * Remove assistant on shift.
511      */
512     public void removeAssistantOnShift() {
513         String input;
514         String errorMessage = "We had an
    unidentified whoopsie";
515         int id = 0;
516         boolean succesfulRemove;
517
518         clearScreen();
519         System.out.println("University of Knowledge
     - COVID test");
520         listAssistantsOnShift(true);
521         System.out.println("""
522                 Removing assistant on shift\s
523                 \s
524                 Please, enter one of the following:\
    s
525                 \s
526                 The sequential ID to select the
    bookable room to be removed.\s
527                 0. Back to main menu.\s
528                 -1. Quit application.\s
529                 """);
530
531         input = scan.nextLine();
532
533         while(!(input.equals("0") || input.equals("-
    1"))) {
534             if(Pattern.matches("[0-9][0-9]", input
    )){
535                 id = Integer.parseInt(input)-11;
536                 if(id < assistantsOnShift.size() &&
    id >= 0 && assistantsOnShift.get(id).getStatus().
```

```
equals(StaffStatus.FREE))
                    succesfulRemove = true;
                else{
                    succesfulRemove = false;
                    errorMessage = "ID out of range
, try a valid ID";
                }
            }
            else{
                succesfulRemove = false;
                errorMessage = "ID not valid, try a
valid ID";
            }

            clearScreen();
            if(succesfulRemove) {
                System.out.println("Assistant on
Shift removed successfully");
                System.out.println(assistantsOnShift
.get(id).toString()+"\n");
                assistantsOnShift.remove(id);
            }
            else{
                System.out.println("Error!");
                System.out.println(errorMessage);
            }

            System.out.println("Please, enter one of
 the following:\n");
            listAssistantsOnShift(true);
            System.out.println("""
                    \s
                    The sequential ID to select the
assistant on shift to be removed.\s
                    0. Back to main menu.\s
                    -1. Quit application.\s
                    """);

            input = scan.nextLine();
            if (input.equals("-1"))
                live = false;
        }
    }

```

```
574     /**
575      * Show bookings.
576      */
577 /*
578     Bookings
579      */
580     public void showBookings() {
581         String input = "the showlar express";
582         boolean oneOrZero = false;
583         clearScreen();
584         System.out.println("""
585                 University of Knowledge - COVID test\s
586                 \s
587                 Select which booking to list:\s
588                 1. All\s
589                 2. Only bookings status: SCHEDULED\s
590                 3. Only bookings status: COMPLETED\s
591                 0. Back to main menu.\s
592                 -1. Quit application\s
593                 """);
594         input = scan.nextLine();
595         while (!(input.equals("0") || input.equals("-1"))) {
596             //list all bookings
597             switch (input) {
598                 case "2" -> listBookings("SCHEDULED");
599                 case "3" -> listBookings("COMPLETED");
600                 default -> listBookings("ALL");
601             }
602 
603             System.out.println("""
604                     0. Back to main menu.
605                     -1. Quit application.
606                     """);
607             input = scan.nextLine();
608         }
609         if (input.equals("-1"))
610             live = false;
611     }
```

```
612
613     /**
614      * Add booking.
615      */
616     public void addBooking() {
617         String input;
618         String errorMessage = "we had an
    unidentified whoopsie";
619         String email;
620         String[] bookingInfo;
621         boolean timeSlotAvailable = updateTimeSlots
    ();
622         boolean validID;
623         boolean successfulAdd;
624         int id;
625         AssistantOnShift theAssistant;
626         BookableRoom theRoom;
627         LocalDateTime theTimeSlot = LocalDateTime.
    now();
628
629         clearScreen();
630         System.out.println("""
631                 University of Knowledge - COVID test
632                 \s
633                 Adding booking (appointment for a
    COVID test) to the system
634                 """);
635
636         listTimeSlots();
637
638         System.out.println("""
639                 Please, enter one of the following:
640                 \s
641                 The sequential ID of an available
    time-slot and the student email, separated by a
    white space.
642                 0. Back to main menu.
643                 -1. Quit application.
644                 """);
645
646         input = scan.nextLine();
647
648         while(!(input.equals("0") || input.equals("-
    1"))) {
```

```
649             validID = false;
650             successfulAdd = true;
651 
652             if (Pattern.matches("[0-9][0-9][ ](.*)@
    uok.ac.uk", input)) {
653                 bookingInfo = input.split(" ", 2);
654                 id = Integer.parseInt(bookingInfo[0
    ]) - 11;
655                 email = bookingInfo[1];
656 
657                 if (id < timeSlots.size() && id >= 0
    ) {
658                     validID = true;
659                     theTimeSlot = timeSlots.get(id);
660                 }
661                 else {
662                     errorMessage = "ID invalid";
663                     validID = false;
664                 }
665 
666                 if(validID && timeSlotAvailable){
667                     //for each bookable room
668                     for(BookableRoom bookableRoom:
    bookableRooms) {
669                         //if the room isn't full and
     is in timeslot
670                         if ((bookableRoom.getStatus
    ().equals(RoomStatus.EMPTY) || bookableRoom.
    getStatus().equals(RoomStatus.AVAILABLE)) &&
    bookableRoom.getStartTime().equals(theTimeSlot)) {
671                             //for each assistant on
    shift
672                             for(AssistantOnShift
    assistantOnShift: assistantsOnShift) {
673                                 //if the assistant
    is free and is in timeslot
674                                 if ((
    assistantOnShift.getStatus().equals(StaffStatus.FREE
    )) && assistantOnShift.getStartTime().equals(
    theTimeSlot)) {
675                                     //get first
    available room
676                                     theRoom =
    bookableRoom;
```

```
677
678                                     //get first
    available assistant
679                                     theAssistant =
    assistantOnShift;
680
681                                     //add the
    booking to the system
682                                     bookings.add(new
     Booking(theRoom,theAssistant,email,theTimeSlot));
683                                     successfulAdd =
    true;
684                                     break;
685                                 }
686                                 else {
687                                     errorMessage = "
    No assistants available in timeslot";
688                                     successfulAdd =
    false;
689                                 }
690                             }
691                             //if booking added, stop
     looking for rooms/assistants
692                             if(successfulAdd)
693                                 break;
694                         }
695                         else {
696                             errorMessage = "No rooms
     available in timeslot";
697                             successfulAdd = false;
698                         }
699                     }
700                 }
701                 else {
702                     errorMessage = "ID invalid or no
     timeslots available";
703                     successfulAdd = false;
704                 }
705             }
706             else{
707                 successfulAdd = false;
708                 errorMessage = "Input format
    incorrect";
709             }
```

```
710
711         if(successfulAdd){
712             System.out.println("Booking added
    successfully:");
713             System.out.println(bookings.get(bookings
    .size()-1).toString());
714         }
715         else {
716             System.out.println("Error!");
717             System.out.println(errorMessage);
718         }
719
720         timeSlotAvailable = updateTimeSlots();
721         listTimeSlots();
722
723         System.out.println("""
724                         Please, enter one of the
    following:
725                         \s
726                         The sequential ID of an
    available time-slot and the student email, separated
     by a white space.
727                         0. Back to main menu.
728                         -1. Quit application.
729                         """);
730
731         input = scan.nextLine();
732         if (input.equals("-1"))
733             live = false;
734         }
735     }
736
737     /**
738      * Remove booking.
739      */
740     public void removeBooking() {
741         String input;
742         String errorMessage = "We had an
    unidentified whoopsie";
743         int id = 0;
744         boolean succesfulRemove;
745
746         clearScreen();
747         System.out.println("University of Knowledge
```

```
747  - COVID test");
748         listBookings("SCHEDULED");
749         System.out.println("""
750                 Removing Booking from system
751                 \s
752                 Please, enter one of the following:
753                 \s
754                 The sequential ID to select the
    booking to be removed from the listed bookings above
    .
755                 0. Back to main menu.
756                 -1. Quit application.
757                 """);
758 
759         input = scan.nextLine();
760 
761         while(!(input.equals("0") || input.equals("-
    1"))) {
762             if(Pattern.matches("[0-9][0-9]", input
    )){
763                 id = Integer.parseInt(input)-11;
764                 if(id < bookings.size() && id >= 0
     && bookings.get(id).getStatus().equals(
    BookingStatus.SCHEDULED))
765                     succesfulRemove = true;
766                 else{
767                     succesfulRemove = false;
768                     errorMessage = "ID out of range
    , try a valid ID";
769                 }
770             }
771             else{
772                 succesfulRemove = false;
773                 errorMessage = "ID not valid, try a
    valid ID";
774             }
775 
776             clearScreen();
777             if(succesfulRemove) {
778                 System.out.println("Booking removed
    successfully");
779                 System.out.println(bookings.get(id).
    toString()+"\n");
780                 bookings.get(id).getAssistant().
```

```
780 setStatus(StaffStatus.FREE);
781                 bookings.get(id).getRoom().
    removeOccupant();
782                 bookings.remove(id);
783             }
784             else{
785                 System.out.println("Error!");
786                 System.out.println(errorMessage);
787             }
788 
789             System.out.println("Please, enter one of
     the following:\n");
790             listBookings("SCHEDULED");
791             System.out.println("""
792                     \s
793                     The sequential ID to select the
    booking to be removed from the listed bookings above
    .
794                     0. Back to main menu.
795                     -1. Quit application.
796                     """);
797 
798             input = scan.nextLine();
799             if (input.equals("-1"))
800                 live = false;
801         }
802     }
803 
804     /**
805      * Conclude booking.
806      */
807     public void concludeBooking() {
808         String input;
809         String errorMessage = "We had an
    unidentified whoopsie";
810         int id = 0;
811         boolean successfulConclude;
812 
813         clearScreen();
814         System.out.println("University of Knowledge
     - COVID test");
815         listBookings("SCHEDULED");
816         System.out.println("""
817                 Conclude booking
```

```
818                 \s
819                 Please, enter one of the following:
820                 \s
821                 The sequential ID to select the
    booking to be completed.
822                 0. Back to main menu.
823                 -1. Quit application.
824                 """);
825 
826         input = scan.nextLine();
827 
828         while(!(input.equals("0") || input.equals("-
    1"))) {
829             if (Pattern.matches("[0-9][0-9]", input
    )) {
830                 id = Integer.parseInt(input) - 11;
831                 if (id < bookings.size() && id >= 0
     && bookings.get(id).getStatus().equals(
    BookingStatus.SCHEDULED))
832                     successfulConclude = true;
833                 else {
834                     successfulConclude = false;
835                     errorMessage = "ID out of range
    , try a valid ID";
836                 }
837             } else {
838                 successfulConclude = false;
839                 errorMessage = "ID not valid, try a
    valid ID";
840             }
841 
842             clearScreen();
843             if(successfulConclude) {
844                 System.out.println("Booking
    completed successfully");
845                 System.out.println(bookings.get(id).
    toString()+"\n");
846                 bookings.get(id).setStatus(
    BookingStatus.COMPLETED);
847                 bookings.get(id).getRoom().
    removeOccupant();
848                 bookings.get(id).getAssistant().
    setStatus(StaffStatus.FREE);
849             }
```

```
850             else{
851                 System.out.println("Error!");
852                 System.out.println(errorMessage);
853             }
854 
855             System.out.println("Please, enter one of
     the following:\n");
856             listBookings("SCHEDULED");
857             System.out.println("""
858                     \s
859                     The sequential ID to select the
    booking to be completed.
860                     0. Back to main menu.
861                     -1. Quit application.
862                     """);
863 
864             input = scan.nextLine();
865             if (input.equals("-1"))
866                 live = false;
867         }
868     }
869 
870     /**
871      * Clear screen.
872      */
873 /*
874     other methods
875      */
876     public static void clearScreen() {
877         for(int i = 0;i<20;i++)
878             System.out.println();
879         //print 20 lines for testing in intellij
880 
881         try {
882             if(System.getProperty("os.name").
    contains("Windows"))
883                 new ProcessBuilder("cmd", "/c", "cls
    ").inheritIO().start().waitFor();
884             else
885                 Runtime.getRuntime().exec("clear");
886         } catch (IOException | InterruptedException
    ex){ }
887         //clear terminal
888     }
```

```
889
890     /**
891      * Update time slots boolean.
892      *
893      * @return the boolean
894      */
895     public boolean updateTimeSlots() {
896         boolean timeSlotAvailable = false;
897         //empty timeslots so existing timeslots can
    be added
898         timeSlots.clear();
899
900         //for each room
901         for(BookableRoom bookableRoom: bookableRooms
    ){
902             //if the room isn't full
903             if(bookableRoom.getStatus().equals(
    RoomStatus.EMPTY) || bookableRoom.getStatus().equals
    (RoomStatus.AVAILABLE)){
904                 //for each assistant
905                 for(AssistantOnShift
    assistantOnShift: assistantsOnShift){
906                     //if the assistant is free
907                     if(assistantOnShift.getStatus().
    equals(StaffStatus.FREE)){
908                         //if the room and assistant
    have the same timeslot
909                         if(bookableRoom.getStartTime
    ().equals(assistantOnShift.getStartTime())){
910                             timeSlots.add(
    bookableRoom.getStartTime());
911                             //add available timeslot
     to timeslot list
912                             //there is at least 1
    timeslot available
913                             timeSlotAvailable = true
    ;
914                             break;
915                         }
916                     }
917                 }
918             }
919         }
920         return timeSlotAvailable;
```

```
921     }
922
923     /*
924     getters and setters
925      */
926
927     /**
928      * Is live boolean.
929      *
930      * @return the boolean
931      */
932     public boolean isLive() {
933         return live;
934     }
935
936     /**
937      * Sets live.
938      *
939      * @param live the live
940      */
941     public void setLive(boolean live) {
942         this.live = live;
943     }
944
945 }
946
```

```java
import java.time.LocalDateTime;
import java.util.Comparator;

/**
 * The type Time slot sorter.
 */
public class TimeSlotSorter  implements Comparator<LocalDateTime>
{
    @Override
    public int compare(LocalDateTime o1, LocalDateTime o2) {
        return o1.compareTo(o2);
    }
}

```

```java
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * The type Assistant on shift.
 */
public class AssistantOnShift {

    private Assistant assistant;
    private LocalDateTime startTime;
    private StaffStatus status;
    private DateTimeFormatter dateFormat = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

    /**
     * Instantiates a new Assistant on shift.
     *
     * @param assistant the assistant
     * @param startTime the start time
     */
    public AssistantOnShift(Assistant assistant, LocalDateTime startTime) {
        this.assistant = assistant;
        this.startTime = startTime;
        status = StaffStatus.FREE;
    }

    /**
     * Gets status.
     *
     * @return the status
     */
    public StaffStatus getStatus() {
        return status;
    }

    /**
     * Sets status.
     *
     * @param status the status
     */
    public void setStatus(StaffStatus status) {
        this.status = status;
    }
```

```
43
44      /**
45       * Gets assistant.
46       *
47       * @return the assistant
48       */
49      public Assistant getAssistant() {
50          return assistant;
51      }
52
53      /**
54       * Gets start time.
55       *
56       * @return the start time
57       */
58      public LocalDateTime getStartTime() {
59          return startTime;
60      }
61
62      public String toString() {
63          return "| " + startTime.format(dateFormat) +
    " | " + status + " | " + assistant.getEmail() + " |";
64      }
65  }
66
```

```java
import java.util.Comparator;

/**
 * The type Bookable room sorter.
 */
public class BookableRoomSorter implements Comparator<BookableRoom> {
    @Override
    public int compare(BookableRoom o1, BookableRoom o2) {
        return o1.getStartTime().compareTo(o2.getStartTime());
    }
}

```

```java
import java.util.ArrayList;
import java.util.regex.*;

/**
 * The type University recourses.
 */
public class UniversityRecourses {
    private ArrayList<Assistant> assistants = new ArrayList<Assistant>();
    private ArrayList<Room> rooms = new ArrayList<Room>();

    /**
     * Instantiates a new University recourses.
     */
    public UniversityRecourses() {
        assistants.add(new Assistant("Matthew Auger", "mauger@uok.ac.uk"));
        assistants.add(new Assistant("Simran Sahota", "simba@uok.ac.uk"));
        assistants.add(new Assistant("Nishil Amin", "nish_nish@uok.ac.uk"));

        rooms.add(new Room("rm100", 3));
        rooms.add(new Room("rm101", 2));
        rooms.add(new Room("rm102", 1));
    }

    /**
     * Add assistants.
     *
     * @param name  the name
     * @param email the email
     */
    public void addAssistants(String name, String email){
        if(name != null && isEmailUniqueAndValid(email))
            assistants.add(new Assistant(name,email));
        else
            System.out.println("Assistant not valid, please use correct format");
    }
```

```
36
37      /**
38       * Add rooms.
39       *
40       * @param code     the code
41       * @param capacity the capacity
42       */
43      public void addRooms(String code, int capacity){
44          if(isCodeUnique(code) && capacity > 0)
45              rooms.add(new Room(code,capacity));
46          else
47              System.out.println("Room not valid,
    please use correct format");
48      }
49
50      /**
51       * Is email unique and valid boolean.
52       *
53       * @param email the email
54       * @return the boolean
55       */
56      public boolean isEmailUniqueAndValid(String email
    ) {
57          if(Pattern.matches("(.*)@uok.ac.uk", email
    )) {
58              for (Assistant assistant : assistants) {
59                  if (email.equals(assistant.getEmail
    ())) {
60                      System.out.println("Email already
     in use, please add a new assistant");
61                      return false;
62                  }
63              }
64          }
65          return true;
66      }
67
68      /**
69       * Is code unique boolean.
70       *
71       * @param code the code
72       * @return the boolean
73       */
74      public boolean isCodeUnique(String code) {
```

```
 75          for(Room room:rooms) {
 76              if (code.equals(room.getCode())) {
 77                  System.out.println("Code already in
    use, please add a new room");
 78                  return false;
 79              }
 80          }
 81          return true;
 82      }
 83
 84      /**
 85       * Gets assistants.
 86       *
 87       * @return the assistants
 88       */
 89      public ArrayList<Assistant> getAssistants() {
 90          return assistants;
 91      }
 92
 93      /**
 94       * Gets rooms.
 95       *
 96       * @return the rooms
 97       */
 98      public ArrayList<Room> getRooms() {
 99          return rooms;
100      }
101 }
102
```

```java
import java.util.Comparator;

/**
 * The type Assistant on shift sorter.
 */
public class AssistantOnShiftSorter implements Comparator<AssistantOnShift> {
    @Override
    public int compare(AssistantOnShift o1, AssistantOnShift o2) {
        return o1.getStartTime().compareTo(o2.getStartTime());
    }
}

```