1 Assistant.java

```java
// The Assistant class:
public class Assistant{

   // A COVID-19 test assistant is someone related to the university (staff or student) who is volunteering
       to perform COVID tests.

   // 2 instance attributes:
   private String email;
   private String name;

   // Methods:

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

   // Setters -

   public void setEmail(String newEmail) {
      // Checks email ends with '@uok.ac.uk'.
      boolean endCorrect;
      endCorrect = newEmail.endsWith("@uok.ac.uk");
      if (endCorrect == true) {
         this.email = newEmail;
      } else {
         System.out.println("Could not set this email address as it does not end with '@uok.ac.uk'.");
      }
   }

   public void setName(String name) {
      if (name != "") {
         // Check name isn't null.
         this.name = name;
      }
   }

   // Getters -
   // These are needed to return the values of the attributes as they are private...

   public String getEmail() {
      return email;
   }

   public String getName() {
      return name;
   }

   // Constructor:
   public Assistant(String name, String email) {
      setName(name);
      setEmail(email);
```

```
    }

}
```

2 AssistantOnShift.java

```
// The AssistantOnShift class:
public class AssistantOnShift{

    // An assistant on shift is a volunteer already registered within the university that can be effectively
        allocated to a bookable room to perform a test.


    // 3 instance attributes:
    private String status;
    private String timeSlot;
    private Assistant assistant;

    // Methods:

    // toString Method -
    public String toString(){
      return "| "+timeSlot+" | "+status+" | "+assistant.getEmail()+" |";
    }

    //returns true if status is "BUSY" and false if not.
    public boolean isBusy() {
       if (status == "BUSY") {
          return true;
       } else {
          return false;
       }
    }

    // allows you to change the status of the AssistantOnShift
    public void isBooked(boolean booked) {
       if (booked == true) {
          this.status = "BUSY";
       } else {
          this.status = "FREE";
       }
    }

    // Getters -
    // These are needed to return the values of the attributes as they are private...

    public String getStatus() {
       return status;
    }

    public String getTimeSlot() {
       return timeSlot;
    }

```

```
48    public Assistant getAssistant() {
49       return assistant;
50    }
51
52    public String getEmail() {
53       return assistant.getEmail();
54    }
55
56    // Constructor:
57    public AssistantOnShift(String timeSlot, Assistant assistant) {
58       this.assistant = assistant;
59       this.timeSlot = timeSlot;
60       this.status = "FREE";
61    }
62
63 }
```

3 BookableRoom.java

```
1  // The BookableRoom class:
2  public class BookableRoom{
3
4     // A bookable room is a room registered by the university that can be effectively used for tests. As the
          name suggests, it is a room available for booking.
5
6     // 4 instance attributes:
7     private String status;
8     private String timeSlot;
9     private Room room;
10    private int occupancy = 0;
11
12    // Methods:
13
14    // toString Method -
15    public String toString(){
16      return "| "+timeSlot+" | "+status+" | "+room.getCode()+" | occupancy: "+occupancy+" |";
17    }
18
19    // returns true if status is "FULL" and false if not.
20    public boolean isFull() {
21       if (status == "FULL") {
22          return true;
23       } else {
24          return false;
25       }
26    }
27
28    //returns true if status is "EMPTY" and false if not.
29    public boolean isEmpty() {
30       if (status == "EMPTY") {
31          return true;
32       } else {
33          return false;
34       }
```

```
35      }
36
37      // used when a booking is added.
38      public void addBooking() {
39         occupancy += 1;
40         setStatus();
41      }
42
43      // used when a booking is completed.
44      public void completeBooking() {
45         occupancy -= 1;
46         setStatus();
47      }
48
49      // allows the user to change the status.
50      public void setStatus() {
51         if (occupancy == 0) {
52            this.status = "EMPTY";
53         } else if (room.getCapacity() - occupancy > 0) {
54            this.status = "AVAILABLE";
55         } else {
56            this.status = "FULL";
57         }
58      }
59
60      // Getters -
61      // These are needed to return the values of the attributes as they are private...
62
63      public String getStatus() {
64         return status;
65      }
66
67      public String getTimeSlot() {
68         return timeSlot;
69      }
70
71      public Room getRoom() {
72         return room;
73      }
74
75      public int getOccupancy() {
76         return occupancy;
77      }
78
79      public String getCode() {
80         return room.getCode();
81      }
82
83      // Constructor:
84      public BookableRoom(String timeSlot, Room room) {
85         this.room = room;
86         this.timeSlot = timeSlot;
87         this.occupancy = 0;
88         setStatus();
89
```

```
    }

}
```

4 Booking.java

```
// The Booking class:
public class Booking{

    // A booking consists of matching a bookable room and an assistant on shift at a specific time-slot to
        perform a COVID-19 test on a student. It is the main function of the system.

    // 4 instance attributes:
    private BookableRoom room;
    private AssistantOnShift assistant;
    private String studentEmail;
    private String status;

    // Methods:

    // toString Method -
    public String toString(){
      return "| "+room.getTimeSlot()+" | "+status+" | " +assistant.getEmail()+" | "+room.getCode()+" |
          "+studentEmail+" |";
    }

    // updates the status of the booking to "COMPLETED".
    public void testCompleted() {
          this.status = "COMPLETED";
    }

    //returns true if status is "COMPLETED" and false if not.
    public boolean isComplete() {
       if (status == "COMPLETED") {
          return true;
       } else {
          return false;
       }
    }

    // allows you to set the students email in the booking
    public void setEmail(String newEmail) {
       // Checks email ends with '@uok.ac.uk'.
       if (newEmail.endsWith("@uok.ac.uk") == true) {
          this.studentEmail = newEmail;
       } else {
          System.out.println("Could not set this email address as it does not end with '@uok.ac.uk'.");
       }
    }

    // Getters -
    // These are needed to return the values of the attributes as they are private...

    public String getStatus() {
```

```
47        return status;
48     }
49
50     public AssistantOnShift getAssistantOnShift() {
51        return assistant;
52     }
53
54     public BookableRoom getBookableRoom() {
55        return room;
56     }
57
58     public String getStudentEmail() {
59        return studentEmail;
60     }
61
62     // Constructor:
63     public Booking(BookableRoom room, AssistantOnShift assistant, String email) {
64        this.room = room;
65        this.assistant = assistant;
66        setEmail(email);
67        this.status = "SCHEDULED";
68
69     }
70
71  }
```

5 BookingApp.java

```
1  import java.util.Scanner;
2  import java.io.IOException;
3
4  public class BookingApp {
5
6     // 2 instance attributes:
7     private BookingSystem uokBS;
8     public boolean quitter;
9
10
11    // METHODS:
12
13    public void menuUserInput(){
14       boolean validInput;
15       Scanner in = new Scanner(System.in);
16       do {
17          // main menu section (very cool) and very self explaitory. They enter number it do the thing it
                says next to it on the menu.
18          printMainMenu();
19          String inStr = in.next();
20          validInput = true;
21          switch(inStr){
22          case "1":
23             listBookableRooms();
24             break;
25          case "2":
```

```
26              addBookableRoom();
27              break;
28          case "3":
29              removeBookableRoom();
30              break;
31          case "4":
32              listAssistantOnShifts();
33              break;
34          case "5":
35              addAssistantOnShift();
36              break;
37          case "6":
38              removeAssistantOnShift();
39              break;
40          case "7":
41              listBookings();
42              break;
43          case "8":
44              addBooking();
45              break;
46          case "9":
47              removeBooking();
48              break;
49          case "10":
50              concludeBooking();
51              break;
52          case "-1":
53              this.quitter = true;
54              break;
55          default:
56              System.out.println("Invalid input.");
57              validInput = false;
58          }
59      } while (!validInput);
60  }
61
62  public static void clearScreen() {
63      // i must confess i found this on stack overflow, it clears the console on Windows. If you are not
            using windows, I do not know if it will work, but I can assure you it works well on windows. :)
64      try {
65          if (System.getProperty("os.name").contains("Windows"))
66                  new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
67              else
68                  Runtime.getRuntime().exec("clear");
69        } catch (IOException | InterruptedException ex) {}
70      System.out.flush();
71  }
72
73  private int convertStringToInt(String number) {
74      //just a little string to int function cause I gor fed up of typing it out each time.
75        return Integer.parseInt(number);
76   }
77
78  public void printMainMenu() {
79      // main menu printer
```

```
80        System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
81        System.out.println("");
82        System.out.println("Manage Bookings");
83        System.out.println("");
84        System.out.println("Please, enter the number to select your option:");
85        System.out.println("");
86        System.out.println("To manage Bookable Rooms:");
87        System.out.println(" 1. List");
88        System.out.println(" 2. Add");
89        System.out.println(" 3. Remove");
90        System.out.println("To manage Assistants on Shift:");
91        System.out.println(" 4. List");
92        System.out.println(" 5. Add");
93        System.out.println(" 6. Remove");
94        System.out.println("To manage Bookings:");
95        System.out.println(" 7. List");
96        System.out.println(" 8. Add");
97        System.out.println(" 9. Remove");
98        System.out.println(" 10. Conclude");
99        System.out.println("After selecting one the options above, you will be presented other screens.");
100       System.out.println("If you press 0, you will be able to return to this main menu.");
101       System.out.println("Press -1 (or ctrl+c) to quit this application.");
102       System.out.println("");
103       // wow very cool
104    }
105
106    public void listBookableRooms() {
107       // 1.
108       boolean validInput;
109       Scanner in = new Scanner(System.in);
110       do {
111          System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
112          System.out.println("");
113          for (int i = 0; i < uokBS.getRoomsLength(); i++) {
114             System.out.println(i+11 +". "+ uokBS.getBookableRoom(i));
115          }
116          //prints all bookable rooms
117          System.out.println("");
118          System.out.println("0. Back to main menu.");
119          System.out.println("-1. Quit application.");
120          System.out.println("");
121          String inStr = in.next(); // take input
122          validInput = true;
123          switch(inStr){
124          case "0":
125             break;
126          case "-1":
127             this.quitter = true;
128             break;
129          default:
130             System.out.println("Invalid input.");
131             validInput = false;
132          }
133       } while (!validInput);
134    }
```

```
135
136     public void addBookableRoom() {
137        // 2.
138        boolean validInput;
139        Scanner in = new Scanner(System.in);
140        System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
141        System.out.println("");
142        System.out.println("Adding bookable room");
143        System.out.println("");
144        (uokBS.getUniversity()).printRooms(); //prints all rooms in class university
145        do {
146           System.out.println("Please, enter one of the following:");
147           System.out.println("");
148           System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time (HH:MM),");
149           System.out.println("separated by a white space.");
150           System.out.println("0. Back to main menu.");
151           System.out.println("-1. Quit application.");
152           System.out.println("");
153           String inStr = in.nextLine();
154           validInput = true;
155           try {
156              switch(inStr){
157              case "0":
158                 break;
159              case "-1":
160                 this.quitter = true;
161                 break;
162              default:
163                 String[] splitInput = inStr.split(" ");
164                 //split string into array at spaces
165                 int index = convertStringToInt(splitInput[0]);
166                 if (index > 10) {
167                    if (index < ((uokBS.getUniversity()).getRoomsLength() + 11)) {
168                       String timeSlot = ("" + splitInput[1] + " " + splitInput[2]);
169                       if (uokBS.checkTimeSlotValid(timeSlot) == true) {
170                          uokBS.createBookableRoom(timeSlot, (uokBS.getUniversity()).getRoom(index - 11));
171                          System.out.println("Bookable Room added successfully:");
172                          System.out.println(uokBS.getBookableRoom(uokBS.getRoomsLength()-1));
173                          validInput = false;
174                       } else {
175                          System.out.println("Invalid date / time.");
176                          validInput = false;
177                       }
178                    } else {
179                       System.out.println("Invalid input. Room index not found!");
180                       validInput = false;
181                    }
182                 } else {
183                    System.out.println("Invalid input. Room index not found!");
184                    validInput = false;
185                 }
186              }
187           }
188           catch (Exception e) {
189                   System.out.println("Invalid input.");
```

```
190                validInput = false;
191             }
192          } while (!validInput);
193       }
194
195       public void removeBookableRoom() {
196          // 3.
197          boolean validInput;
198          Scanner in = new Scanner(System.in);
199          System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
200             System.out.println("");
201             for (int i = 0; i < uokBS.getRoomsLength(); i++) {
202                if ((uokBS.getBookableRoom(i)).isEmpty() == true) {
203                   System.out.println(i+11 +". "+ uokBS.getBookableRoom(i));
204                }
205             }
206             System.out.println("Removing bookable room");
207             System.out.println("");
208          do {
209             System.out.println("Please, enter one of the following:");
210             System.out.println("");
211             System.out.println("The sequential ID to select the bookable room to be removed.");
212             System.out.println("0. Back to main menu.");
213             System.out.println("-1. Quit application.");
214             System.out.println("");
215             String inStr = in.next(); //input
216             validInput = true;
217             try {
218                switch(inStr){
219                case "0":
220                   break;
221                case "-1":
222                   this.quitter = true;
223                   break;
224                default:
225                   int index = convertStringToInt(inStr);
226                   if (index > 10) {
227                      if (index < (uokBS.getRoomsLength() + 11)) {
228                         BookableRoom roomToDelete = uokBS.getBookableRoom(index-11);
229                         if (roomToDelete.isEmpty() == true) {
230                            //checks room is empty
231                            uokBS.deleteBookableRoom(roomToDelete);
232                            System.out.println("Bookable Room removed successfully:");
233                            System.out.println(roomToDelete);
234                            validInput = false;
235                         } else {
236                            System.out.println("Error!");
237                            System.out.println("Invalid input. Room not EMPTY!");
238                            validInput = false;
239                         }
240                      } else {
241                         System.out.println("Error!");
242                         System.out.println("Invalid input. Room index not found! The index should be < " +
                               (uokBS.getRoomsLength() + 11) + ".");
243                         validInput = false;
```

```
                }
            } else {
                System.out.println("Error!");
                System.out.println("Invalid input. Room index not found! The index should be > 10.");
                validInput = false;
            }
         }
      }
      catch (Exception e) {
         System.out.println("Error!");
         System.out.println("Invalid input.");
         validInput = false;
      }
   } while (!validInput);
}

public void listAssistantOnShifts() {
   //4.
   boolean validInput;
   Scanner in = new Scanner(System.in);
   do {
      System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
      System.out.println("");
      for (int i = 0; i < uokBS.getAssistantsLength(); i++) {
         System.out.println(i+11 +". "+ uokBS.getAssistantOnShift(i));
         //print all AssistantOnShifts
      }
      System.out.println("");
      System.out.println("0. Back to main menu.");
      System.out.println("-1. Quit application.");
      System.out.println("");
      String inStr = in.next();
      validInput = true;
      switch(inStr){
      case "0":
         break;
      case "-1":
         this.quitter = true;
         break;
      default:
         System.out.println("Invalid input.");
         validInput = false;
      }
   } while (!validInput);
}

public void addAssistantOnShift() {
   //5.
   boolean validInput;
   Scanner in = new Scanner(System.in);
   System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
   System.out.println("");
   System.out.println("Adding assistant on shift");
   System.out.println("");
   (uokBS.getUniversity()).printAssistants();
```

```
299         do {
300            System.out.println("Please, enter one of the following:");
301            System.out.println("");
302            System.out.println("The sequential ID of an assistant and date (dd/mm/yyyy), separated by a white
                   space.");
303            System.out.println("0. Back to main menu.");
304            System.out.println("-1. Quit application.");
305            System.out.println("");
306            String inStr = in.nextLine();
307            validInput = true;
308            try {
309               switch(inStr){
310               case "0":
311                  break;
312               case "-1":
313                  this.quitter = true;
314                  break;
315               default:
316                  String[] splitInput = inStr.split(" ");
317                  //split string into array at spaces
318                  int index = convertStringToInt(splitInput[0]);
319                  if (index > 10) {
320                     if (index < ((uokBS.getUniversity()).getAssistantsLength() + 11)) {
321                        String timeSlot = ("" + splitInput[1] + " 07:00");
322                        if (uokBS.checkTimeSlotValid(timeSlot) == true) {
323                           uokBS.createAssistantOnShift(timeSlot, (uokBS.getUniversity()).getAssistant(index -
                                  11));
324                           System.out.println("Assistant on Shift added successfully:");
325                           System.out.println(uokBS.getAssistantOnShift(uokBS.getAssistantsLength()-1));
326                           timeSlot = ("" + splitInput[1] + " 08:00");
327                           uokBS.createAssistantOnShift(timeSlot, (uokBS.getUniversity()).getAssistant(index -
                                  11));
328                           System.out.println("Assistant on Shift added successfully:");
329                           System.out.println(uokBS.getAssistantOnShift(uokBS.getAssistantsLength()-1));
330                           timeSlot = ("" + splitInput[1] + " 09:00");
331                           uokBS.createAssistantOnShift(timeSlot, (uokBS.getUniversity()).getAssistant(index -
                                  11));
332                           System.out.println("Assistant on Shift added successfully:");
333                           System.out.println(uokBS.getAssistantOnShift(uokBS.getAssistantsLength()-1));
334                           validInput = false;
335                           // creates 3 Assistants (they work 3 shifts per date)
336                        } else {
337                           System.out.println("Invalid date / time.");
338                           validInput = false;
339                        }
340                     } else {
341                        System.out.println("Invalid input. Assistant index not found!");
342                        validInput = false;
343                     }
344                  } else {
345                     System.out.println("Invalid input. Assistant index not found!");
346                     validInput = false;
347                  }
348               }
349            }
```

```
350         catch (Exception e) {
351                 System.out.println("Invalid input.");
352                 validInput = false;
353             }
354     } while (!validInput);
355 }
356 
357 public void removeAssistantOnShift() {
358     // 6.
359     boolean validInput;
360     Scanner in = new Scanner(System.in);
361     System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
362         System.out.println("");
363         for (int i = 0; i < uokBS.getAssistantsLength(); i++) {
364             if ((uokBS.getAssistantOnShift(i)).isBusy() == false) {
365                 //prints all free assistants
366                 System.out.println(i+11 +". "+ uokBS.getAssistantOnShift(i));
367             }
368         }
369         System.out.println("Removing assistant on shift");
370         System.out.println("");
371     do {
372         System.out.println("Please, enter one of the following:");
373         System.out.println("");
374         System.out.println("The sequential ID to select the assistant on shift to be removed.");
375         System.out.println("0. Back to main menu.");
376         System.out.println("-1. Quit application.");
377         System.out.println("");
378         String inStr = in.next();
379         validInput = true;
380         try {
381             switch(inStr){
382             case "0":
383                 break;
384             case "-1":
385                 this.quitter = true;
386                 break;
387             default:
388                 int index = convertStringToInt(inStr);
389                 if (index > 10) {
390                     if (index < (uokBS.getAssistantsLength() + 11)) {
391                         AssistantOnShift assistantToDelete = uokBS.getAssistantOnShift(index-11);
392                         if (assistantToDelete.isBusy() == false) {
393                             // check the AssistantOnShift is FREE
394                             uokBS.removeAssistantOnShift(assistantToDelete);
395                             System.out.println("Assistant on Shift removed successfully:");
396                             System.out.println(assistantToDelete);
397                             validInput = false;
398                         } else {
399                             System.out.println("Error!");
400                             System.out.println("Invalid input. AssistantOnShift not FREE!");
401                             validInput = false;
402                         }
403                     } else {
404                         System.out.println("Error!");
```

```
                    System.out.println("Invalid input. AssistantOnShift index not found! The index should be
                        < " + (uokBS.getAssistantsLength() + 11) + ".");
                    validInput = false;
                }
            } else {
                System.out.println("Error!");
                System.out.println("Invalid input. AssistantOnShift index not found! The index should be >
                    10.");
                validInput = false;
            }
        }
        catch (Exception e) {
            System.out.println("Error!");
            System.out.println("Invalid input.");
            validInput = false;
        }
    } while (!validInput);
}

public void listBookings() {
    // 7.
    boolean validInput = false;
    Scanner in = new Scanner(System.in);
    System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
        System.out.println("");
        System.out.println("Select which booking to list:");
        System.out.println("1. All");
        System.out.println("2. Only bookings status:SCHEDULED");
        System.out.println("3. Only bookings status:COMPLETED");
        System.out.println("0. Back to main menu.");
        System.out.println("-1. Quit application.");
        String inStr = in.next(); // input
        System.out.println("");
        switch(inStr){
        case "2":
            for (int i = 0; i < uokBS.getBookingsLength(); i++) {
                if ((uokBS.getBooking(i)).isComplete() == false) {
                    // ONLY SCHEDULED
                    System.out.print((i+11) +". "+ uokBS.getBooking(i));
                }
            }
            validInput = false;
            break;
        case "3":
            for (int i = 0; i < uokBS.getBookingsLength(); i++) {
                if ((uokBS.getBooking(i)).isComplete() == true) {
                    // ONLY COMPLETED
                    System.out.print((i+11) +". "+ uokBS.getBooking(i));
                }
            }
            validInput = false;
            break;
        case "0":
            validInput = true;
```

```
458             break;
459          case "-1":
460             this.quitter = true;
461             validInput = true;
462             break;
463          default:
464             for (int i = 0; i < uokBS.getBookingsLength(); i++) {
465                System.out.print(i+11 +". "+ uokBS.getBooking(i));
466                validInput = false;
467                // ALL BOOKINGS
468             }
469             break;
470          }
471       if (validInput == false) { // if they haven't asked to leave
472          do {
473             System.out.println("");
474             System.out.println("0. Back to main menu.");
475             System.out.println("-1. Quit application.");
476             System.out.println("");
477             inStr = in.next();
478             validInput = true;
479             switch(inStr){
480             case "0":
481                break;
482             case "-1":
483                this.quitter = true;
484                break;
485             default:
486                System.out.println("Invalid input.");
487                validInput = false;
488             }
489          } while (!validInput);
490       }
491
492    }
493
494    public void addBooking() {
495       // 8.
496       boolean validInput;
497       Scanner in = new Scanner(System.in);
498       System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
499       System.out.println("");
500       System.out.println("Adding booking (appointment for a COVID test) to the system");
501       do {
502          System.out.println("");
503          System.out.println("List of available time-slots:");
504          uokBS.createBookingTimeSlots(); //create timeslots for bookings
505          for (int i = 0; i < uokBS.getTSLength(); i++) {
506             System.out.println(i+11 +". "+ uokBS.getTimeSlot(i)); //print time slots
507          }
508          System.out.println("Please, enter one of the following:");
509          System.out.println("");
510          System.out.println("The sequential ID of an available time-slot and the student email, separated by
                 a white space.");
511          System.out.println("0. Back to main menu.");
```

```
512         System.out.println("-1. Quit application.");
513         System.out.println("");
514         String inStr = in.nextLine();
515         validInput = true;
516         try {
517            switch(inStr){
518            case "0":
519               break;
520            case "-1":
521               this.quitter = true;
522               break;
523            default:
524               String[] splitInput = inStr.split(" ");
525               //split string into array at spaces
526               int index = convertStringToInt(splitInput[0]);
527               if (index > 10) {
528                  if (index < ((uokBS.getTSLength() + 11))) {
529                     String email = ("" + splitInput[1]);
530                     uokBS.createBooking(uokBS.getTimeSlot(index-11).getRoom(),
                          uokBS.getTimeSlot(index-11).getAssistant(), email); //create booking
531                     System.out.println("Booking added successfully:");
532                     System.out.println(uokBS.getBooking(uokBS.getBookingsLength()-1)); // get from list
533                     validInput = false;
534                  } else {
535                     System.out.println("Invalid input. Room index not found!");
536                     validInput = false;
537                  }
538               } else {
539                  System.out.println("Invalid input. Room index not found!");
540                  validInput = false;
541               }
542            }
543         }
544         catch (Exception e) {
545                  System.out.println("Invalid input.");
546                  validInput = false;
547               }
548      } while (!validInput);
549   }
550
551   public void removeBooking() {
552      // 9.
553      boolean validInput;
554      Scanner in = new Scanner(System.in);
555      System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
556         System.out.println("");
557         for (int i = 0; i < uokBS.getBookingsLength(); i++) {
558            if ((uokBS.getBooking(i)).getStatus() == "SCHEDULED") {
559               System.out.println(i+11 +". "+ uokBS.getBooking(i)); //show all removeable bookings
560            }
561         }
562         System.out.println("Removing booking from the system");
563         System.out.println("");
564      do {
565         System.out.println("Please, enter one of the following:");
```

```
566         System.out.println("");
567         System.out.println("The sequential ID to select the booking to be removed from the listed bookings
                above.");
568         System.out.println("0. Back to main menu.");
569         System.out.println("-1. Quit application.");
570         System.out.println("");
571         String inStr = in.next();
572         validInput = true;
573         try {
574            switch(inStr){
575            case "0":
576               break;
577            case "-1":
578               this.quitter = true;
579               break;
580            default:
581               int index = convertStringToInt(inStr);
582               if (index > 10) {
583                  if (index < (uokBS.getBookingsLength() + 11)) {
584                     Booking bookingToDelete = uokBS.getBooking(index-11);
585                     if (bookingToDelete.getStatus() == "SCHEDULED") {
586                        uokBS.removeBooking(bookingToDelete); //delete
587                        System.out.println("Booking removed successfully:");
588                        System.out.println(bookingToDelete);
589                        validInput = false;
590                     } else {
591                        System.out.println("Error!");
592                        System.out.println("Invalid input. Booking is already COMPLETED!");
593                        validInput = false;
594                     }
595                  } else {
596                     System.out.println("Error!");
597                     System.out.println("Invalid input. Booking index not found! The index should be < " +
                           (uokBS.getBookingsLength() + 11) + ".");
598                     validInput = false;
599                  }
600               } else {
601                  System.out.println("Error!");
602                  System.out.println("Invalid input. Booking index not found! The index should be > 10.");
603                  validInput = false;
604               }
605            }
606         }
607         catch (Exception e) {
608            System.out.println("Error!");
609            System.out.println("Invalid input.");
610            validInput = false;
611         }
612      } while (!validInput);
613   }
614
615   public void concludeBooking() {
616      // 10.
617      boolean validInput;
618      Scanner in = new Scanner(System.in);
```

```
619         System.out.println((uokBS.getUniversity()).getName() + " - COVID test");
620            System.out.println("");
621            for (int i = 0; i < uokBS.getBookingsLength(); i++) {
622               if ((uokBS.getBooking(i)).getStatus() == "SCHEDULED") {
623                  //only not completed bookings
624                  System.out.println(i+11 +". "+ uokBS.getBooking(i));
625               }
626            }
627            System.out.println("Conclude booking");
628            System.out.println("");
629         do {
630            System.out.println("Please, enter one of the following:");
631            System.out.println("");
632            System.out.println("The sequential ID to select the booking to be completed.");
633            System.out.println("0. Back to main menu.");
634            System.out.println("-1. Quit application.");
635            System.out.println("");
636            String inStr = in.next();
637            validInput = true;
638            try {
639               switch(inStr){
640               case "0":
641                  break;
642               case "-1":
643                  this.quitter = true;
644                  break;
645               default:
646                  int index = convertStringToInt(inStr);
647                  if (index > 10) {
648                     if (index < (uokBS.getBookingsLength() + 11)) {
649                        Booking bookingToComplete = uokBS.getBooking(index-11);
650                        if (bookingToComplete.getStatus() == "SCHEDULED") {
651                           uokBS.completeBooking(bookingToComplete); //similar to removeBooking()
652                           System.out.println("Booking completed successfully:");
653                           System.out.println(bookingToComplete);
654                           validInput = false;
655                        } else {
656                           System.out.println("Error!");
657                           System.out.println("Invalid input. Booking is already COMPLETED!");
658                           validInput = false;
659                        }
660                     } else {
661                        System.out.println("Error!");
662                        System.out.println("Invalid input. Booking index not found! The index should be < " +
                              (uokBS.getBookingsLength() + 11) + ".");
663                        validInput = false;
664                     }
665                  } else {
666                     System.out.println("Error!");
667                     System.out.println("Invalid input. Booking index not found! The index should be > 10.");
668                     validInput = false;
669                  }
670               }
671            }
672            catch (Exception e) {
```

```
673             System.out.println("Error!");
674             System.out.println("Invalid input.");
675             validInput = false;
676          }
677       } while (!validInput);
678    }
679
680    // Constructor
681    public BookingApp(BookingSystem uokBS) {
682       this.uokBS = uokBS;
683       this.quitter = false;
684    }
685
686    public static void main(String[] args) {
687       // hardcoding initialization
688       boolean quitter = false;
689
690       Assistant[] setAssistants;
691       setAssistants = new Assistant[12];
692       setAssistants[0] = new Assistant("Dr John Doe", "JD912@uok.ac.uk");
693       setAssistants[1] = new Assistant("Mr Bob Bobson", "BB103@uok.ac.uk");
694       setAssistants[2] = new Assistant("Miss Persona Reala", "PR753@uok.ac.uk");
695       setAssistants[3] = new Assistant("Dr Percy McPersonface", "PP816@uok.ac.uk");
696       setAssistants[4] = new Assistant("Dr Reginald Body", "RB617@uok.ac.uk");
697       setAssistants[5] = new Assistant("Mrs Agatha Harkness", "AH666@uok.ac.uk");
698       setAssistants[6] = new Assistant("Mr Victor Blisk", "VB348@uok.ac.uk");
699       setAssistants[7] = new Assistant("Miss Emily Wattson", "EW900@uok.ac.uk");
700       setAssistants[8] = new Assistant("Miss Hana Bacon", "HB517@uok.ac.uk");
701       setAssistants[9] = new Assistant("Professor Bennyg Ames", "BA282@uok.ac.uk");
702       setAssistants[10] = new Assistant("Dr Walter Black", "WB878@uok.ac.uk");
703       setAssistants[11] = new Assistant("Mrs Lori Driver", "LD421@uok.ac.uk");
704
705       Room[] setRooms;
706       setRooms = new Room[3];
707       setRooms[0] = new Room("IC215", 3);
708       setRooms[1] = new Room("SH102", 1);
709       setRooms[2] = new Room("HB108", 2);
710
711       University uok = new University(setAssistants, setRooms);
712       BookingSystem uokBS = new BookingSystem(uok);
713
714       uokBS.createBookableRoom("25/02/2021 09:00", uok.getRoom(2));
715       uokBS.createBookableRoom("26/02/2021 07:00", uok.getRoom(0));
716       uokBS.createBookableRoom("26/02/2021 08:00", uok.getRoom(0));
717       uokBS.createBookableRoom("26/02/2021 09:00", uok.getRoom(0));
718       uokBS.createBookableRoom("26/02/2021 07:00", uok.getRoom(1));
719       uokBS.createBookableRoom("26/02/2021 08:00", uok.getRoom(1));
720       uokBS.createBookableRoom("26/02/2021 09:00", uok.getRoom(1));
721       uokBS.createBookableRoom("26/02/2021 07:00", uok.getRoom(2));
722       uokBS.createBookableRoom("26/02/2021 08:00", uok.getRoom(2));
723       uokBS.createBookableRoom("26/02/2021 09:00", uok.getRoom(2));
724
725       uokBS.createAssistantOnShift("25/02/2021 09:00", uok.getAssistant(0));
726       uokBS.createAssistantOnShift("26/02/2021 07:00", uok.getAssistant(3));
727       uokBS.createAssistantOnShift("26/02/2021 08:00", uok.getAssistant(3));
```

```
728         uokBS.createAssistantOnShift("26/02/2021 09:00", uok.getAssistant(3));
729         uokBS.createAssistantOnShift("26/02/2021 07:00", uok.getAssistant(5));
730         uokBS.createAssistantOnShift("26/02/2021 08:00", uok.getAssistant(5));
731         uokBS.createAssistantOnShift("26/02/2021 09:00", uok.getAssistant(5));
732         uokBS.createAssistantOnShift("26/02/2021 07:00", uok.getAssistant(9));
733         uokBS.createAssistantOnShift("26/02/2021 08:00", uok.getAssistant(9));
734         uokBS.createAssistantOnShift("26/02/2021 09:00", uok.getAssistant(9));
735 
736         uokBS.createBooking(uokBS.getBookableRoom(0), uokBS.getAssistantOnShift(0), "SD420@uok.ac.uk");
737         uokBS.createBooking(uokBS.getBookableRoom(1), uokBS.getAssistantOnShift(1), "HK433@uok.ac.uk");
738         uokBS.createBooking(uokBS.getBookableRoom(5), uokBS.getAssistantOnShift(5), "SP782@uok.ac.uk");
739         uokBS.createBooking(uokBS.getBookableRoom(9), uokBS.getAssistantOnShift(9), "LA205@uok.ac.uk");
740 
741         uokBS.completeBooking(uokBS.getBooking(0));
742 
743         //end of hardcoding
744 
745         BookingApp thisApp;
746         thisApp = new BookingApp(uokBS);
747         do {
748            thisApp.clearScreen();
749            thisApp.menuUserInput();
750         } while (!thisApp.quitter);
751      }
752   }
```

6 BookingSystem.java

```
1  import java.util.ArrayList; // import the ArrayList class
2  public class BookingSystem {
3  
4     // The University has a list of assistants and a list of rooms.
5  
6     // 6 instance attributes
7     private ArrayList<BookableRoom> rooms;
8     private ArrayList<AssistantOnShift> assistants;
9     private ArrayList<Booking> bookings;
10    private ArrayList<TimeSlot> listTimeSlots;
11    private int numberOfBookings;
12    private University universityResources;
13 
14    public boolean checkTimeSlotValid(String timeSlot) {
15       // check in format "dd/mm/yyyy HH:MM"
16       boolean isValid = false;
17       if (timeSlot.length() == 16) {
18          // Check timeSlot isn't null and is correct length (16 chars).
19          char timeSlotArray[] = timeSlot.toCharArray();
20          String symbols = "" + timeSlotArray[2] + timeSlotArray[5] + timeSlotArray[10] + timeSlotArray[13];
21          if (symbols.equals("// :") == true) {
22             String year = "" + timeSlotArray[6] + timeSlotArray[7] + timeSlotArray[8] + timeSlotArray[9];
23             String month = "" + timeSlotArray[3] + timeSlotArray[4];
24             String day = "" + timeSlotArray[0] + timeSlotArray[1];
25             String hour = "" + timeSlotArray[11] + timeSlotArray[12];
26             String minute = "" + timeSlotArray[14] + timeSlotArray[15];
```

```
27              boolean hourCorrect = false;
28              //checks it is between 7 and 10 am
29              if (hour.equals("07")) {
30                 hourCorrect = true;
31              } else if (hour.equals("08")) {
32                 hourCorrect = true;
33              } else if (hour.equals("09")) {
34                 hourCorrect = true;
35              }
36              if (hourCorrect == true) {
37                 int intMinute;
38                 try {
39                       intMinute = Integer.parseInt(minute);
40                    }
41                    catch (NumberFormatException e) {
42                       intMinute = -1;
43                    }
44                 if ((intMinute < 60) && (intMinute >= 0)) {
45                    // checks minutes are valid
46                    int intYear;
47                    try {
48                       intYear = Integer.parseInt(year);
49                    }
50                    catch (NumberFormatException e) {
51                       intYear = -1;
52                    }
53                    if (intYear > 2020) {
54                       // checks its atleast 2021
55                       int intDay;
56                       try {
57                          intDay = Integer.parseInt(day);
58                       }
59                       catch (NumberFormatException e) {
60                          intDay = -1;
61                       }
62                       if (month.equals("01")) {
63                          //JANUARY
64                          if ((intDay <= 31) && (intDay > 0)) {
65                             isValid = true;
66                          }
67                       } else if (month.equals("02")) {
68                          //FEBUARY
69                          if ((intDay <= 28) && (intDay > 0)) {
70                             isValid = true;
71                          } else if (intDay == 29) {
72                             if (intYear % 4 == 0) {
73                                isValid = true;
74                                // leap year cause i am very pedantic
75                             }
76                          }
77                       } else if (month.equals("03")) {
78                          //MARCH
79                          if ((intDay <= 31) && (intDay > 0)) {
80                             isValid = true;
81                          }
```

```
82                  } else if (month.equals("04")) {
83                     //APRIL
84                     if ((intDay <= 30) && (intDay > 0)) {
85                        isValid = true;
86                     }
87                  } else if (month.equals("05")) {
88                     //MAY (THE BEST MONTH CLEARLY SUPERIOR)
89                     if ((intDay <= 31) && (intDay > 0)) {
90                        isValid = true;
91                     }
92                  } else if (month.equals("06")) {
93                     //JUNE
94                     if ((intDay <= 30) && (intDay > 0)) {
95                        isValid = true;
96                     }
97                  } else if (month.equals("07")) {
98                     //JULY
99                     if ((intDay <= 31) && (intDay > 0)) {
100                       isValid = true;
101                    }
102                 } else if (month.equals("08")) {
103                    //AUSUST
104                    if ((intDay <= 31) && (intDay > 0)) {
105                       isValid = true;
106                    }
107                 } else if (month.equals("09")) {
108                    //SEPTEMBER
109                    if ((intDay <= 30) && (intDay > 0)) {
110                       isValid = true;
111                    }
112                 } else if (month.equals("10")) {
113                    //OCTOBER spoooookyy
114                    if ((intDay <= 31) && (intDay > 0)) {
115                       isValid = true;
116                    }
117                 } else if (month.equals("11")) {
118                    //NOVEMBER
119                    if ((intDay <= 30) && (intDay > 0)) {
120                       isValid = true;
121                    }
122                 } else if (month.equals("12")) {
123                    //DECEMBER (also cool)
124                    if ((intDay <= 31) && (intDay > 0)) {
125                       isValid = true;
126                    }
127                 }
128              }
129           }
130        }
131     }
132  }
133
134  return isValid;
135  // this is all very unnecessary but I am very thorough
136 }
```

```
137
138     public void createBookingTimeSlots() {
139        // a cool algorithm i came up with on the spot at 2am to create time slots for the booking,
140        for (int i = 0; i < getRoomsLength(); i++) {
141           if (getBookableRoom(i).isFull() == false) {
142              String currentTimeSlot = getBookableRoom(i).getTimeSlot();
143              boolean match = false;
144              for (int j = 0; j < getTSLength(); j++) {
145                 if (currentTimeSlot == getTimeSlot(j).getTimeSlot()) {
146                    match = true;
147                 }
148              }
149              // checks if its already an option
150              if (match == false) {
151                 int l = 0;
152                 for (int k = 0; k < getAssistantsLength(); k++) {
153                    if (getAssistantOnShift(k).isBusy() == false) {
154                       if (getAssistantOnShift(k).getTimeSlot() == currentTimeSlot) {
155                          // finds free assistant
156                          l = k;
157                          k = getAssistantsLength();
158                          break;
159                       }
160                    }
161                 }
162
163                 AssistantOnShift bookingAssistant = getAssistantOnShift(l);
164                 TimeSlot timeSlotToAdd;
165                 // make new timeslot object
166                 timeSlotToAdd = new TimeSlot(currentTimeSlot, getBookableRoom(i), bookingAssistant);
167                 listTimeSlots.add(timeSlotToAdd);
168                 // added to TimeSlot array list!!!!
169              }
170           }
171        }
172     }
173
174     // to create bookable room and assistant on shift
175     public void createBookableRoom(String timeSlot, Room room) {
176        BookableRoom newBookableRoom;
177        newBookableRoom = new BookableRoom(timeSlot, room);
178        rooms.add(newBookableRoom);
179     }
180
181     public void createAssistantOnShift(String timeSlot, Assistant assistant) {
182        AssistantOnShift newAssistantOnShift;
183        newAssistantOnShift = new AssistantOnShift(timeSlot, assistant);
184        assistants.add(newAssistantOnShift);
185     }
186
187     public void removeBookableRoom(BookableRoom room) {
188        //completes booking in the room and removes if then needed to
189        room.completeBooking();
190        if (room.isEmpty() == true) {
191           int index = rooms.indexOf(room);
```

```
192             rooms.remove(index);
193         }
194     }
195 
196     public void deleteBookableRoom(BookableRoom room) {
197         //like remove but far more extreme :O
198         if (room.isEmpty() == true) {
199             int index = rooms.indexOf(room);
200             rooms.remove(index);
201         }
202     }
203 
204     public void removeAssistantOnShift(AssistantOnShift assistant) {
205         int index = assistants.indexOf(assistant);
206         assistants.remove(index);
207         //removes AssistantOnShift from list (they did their shift etc)
208     }
209 
210     public void removeBooking(Booking booking) {
211         int index = bookings.indexOf(booking);
212         bookings.remove(index);
213         // removes non completed booking
214     }
215 
216     public boolean createBooking(BookableRoom room, AssistantOnShift assistant, String email) {
217         //creates a booking
218         boolean created = false;
219         if (assistant.isBusy() == false) {
220             if (room.isFull() == false) {
221                 //checks it should make the booking
222                 assistant.isBooked(true);
223                 //updates assistant status
224                 room.addBooking();
225                 // updates room status
226                 Booking newBooking;
227                 // creates a new booking object
228                 newBooking = new Booking(room, assistant, email);
229                 bookings.add(newBooking);
230                 // adds to list of bookings
231                 created = true;
232             } else {
233                 System.out.println("Cannot create booking, BookableRoom is full...");
234             }
235         } else {
236             System.out.println("Cannot create booking, AssistantOnShift is busy...");
237         }
238         return created;
239     }
240 
241     public void completeBooking(Booking booking) {
242         //completes a booking
243         booking.testCompleted();
244         //sets booking as completed
245         removeBookableRoom(booking.getBookableRoom());
246         // removes a person from the room
```

```
247         removeAssistantOnShift(booking.getAssistantOnShift());
248         // removes assistantonshift
249     }
250 
251     // Getters -
252     // These are needed to return the values of the attributes as they are private...
253 
254     public BookableRoom getBookableRoom(int index) {
255         return rooms.get(index);
256     }
257 
258     public int getRoomsLength() {
259         return rooms.size();
260     }
261 
262     public AssistantOnShift getAssistantOnShift(int index) {
263         return assistants.get(index);
264     }
265 
266     public int getAssistantsLength() {
267         return assistants.size();
268     }
269 
270     public Booking getBooking(int index) {
271         return bookings.get(index);
272     }
273 
274     public int getBookingsLength() {
275         return bookings.size();
276     }
277 
278     public int getTSLength() {
279         return listTimeSlots.size();
280     }
281 
282     public TimeSlot getTimeSlot(int index) {
283         return listTimeSlots.get(index);
284     }
285 
286     public University getUniversity() {
287         return universityResources;
288     }
289 
290     // Constructor
291     public BookingSystem(University university) {
292         this.numberOfBookings = 0;
293         this.universityResources = university;
294         this.rooms = new ArrayList<BookableRoom>();
295         this.assistants = new ArrayList<AssistantOnShift>();
296         this.bookings = new ArrayList<Booking>();
297         this.listTimeSlots = new ArrayList<TimeSlot>();
298     }
299 
300 }
```

7 Room.java

```java
// The Room class:
public class Room{

   // The university has several rooms, and some of the rooms can be allocated to apply COVID tests.

   // 2 instance attributes:
   private String code;
   private int capacity;

   // Methods:

   // toString Method -
   public String toString(){
     return "| "+code+" | capacity: "+capacity+" |";
   }

   // Setters -

   public void setCode(String code) {
      if (code != "") {
         // Check code isn't null.
         this.code = code;
      }
   }

   public void setCapacity(int maxCapacity) {
      if (maxCapacity > 0) {
         // Check capacity is greater than 0.
         this.capacity = maxCapacity;
      }
   }

   // Getters -
   // These are needed to return the values of the attributes as they are private...

   public String getCode() {
      return code;
   }

   public int getCapacity() {
      return capacity;
   }

   // Constructor:
   public Room(String code, int maxCapacity) {
      setCode(code);
      setCapacity(maxCapacity);
   }

}
```

8 TimeSlot.java

```
// The TimeSlot class:
public class TimeSlot {

   // A TimeSlot is a funky class I just made cause I was like why the heck not I have no idea how to do
       this otherwise for the add booking section.

   // 3 instance attributes:
   private String timeSlot;
   private BookableRoom room;
   private AssistantOnShift assistant;

   // toString Method -
   public String toString(){
     return timeSlot;
   }

   // Getters -
   // These are needed to return the values of the attributes as they are private...

   public String getTimeSlot() {
      return timeSlot;
   }

   public BookableRoom getRoom() {
      return room;
   }

   public AssistantOnShift getAssistant() {
      return assistant;
   }

   // Constructor

   public TimeSlot(String timeSlot, BookableRoom room, AssistantOnShift assistant) {
      this.timeSlot = timeSlot;
      this.room = room;
      this.assistant = assistant;
   }

}
```

9 University.java

```
public class University {

   // The University has a list of assistants and a list of rooms.

   // 3 instance attributes
   private Assistant[] assistants;
   private Room[] rooms;
   private String name = "University of Knowledge";

```

```
10      // Getters -
11      // These are needed to return the values of the attributes as they are private...
12
13      public String getName() {
14         return name;
15      }
16
17      public Room getRoom(int index) {
18         return rooms[index];
19      }
20
21      public Assistant getAssistant(int index) {
22         return assistants[index];
23      }
24
25      // Gets length of arrays:
26
27      public int getRoomsLength() {
28         return rooms.length;
29      }
30
31      public int getAssistantsLength() {
32         return assistants.length;
33      }
34
35      // Prints the values in the arrays:
36
37      public void printRooms() {
38         for (int i = 0; i < rooms.length; i++) {
39            System.out.println((i+11)+". "+rooms[i]);
40         }
41         System.out.println("");
42      }
43
44      public void printAssistants() {
45         for (int i = 0; i < assistants.length; i++) {
46            System.out.println((i+11)+". "+assistants[i]);
47         }
48         System.out.println("");
49      }
50
51      // Constructor
52      public University(Assistant[] assistants, Room[] rooms) {
53         this.assistants = assistants;
54         this.rooms = rooms;
55      }
56
57      // second constructor incase you want to change the name.
58      public University(Assistant[] assistants, Room[] rooms, String name) {
59         this.assistants = assistants;
60         this.rooms = rooms;
61         this.name = name;
62      }
63
64  }
```