```
1. import java.util.InputMismatchException;
2. import java.util.Scanner;
3. import java.time.DateTimeException;
4. import java.time.LocalDateTime;
5. public class Menu {
6.  /**
7.  * Private constructor that should not be called.
8.  */
9.  private Menu(){
10.      throw new IllegalStateException("Utility class");
11.      }
12.      /**
13.      * The function to initialize all the other classes.
14.      * It creates 4 rooms, 3 assistants, 9 assistants on
   shift, 9 bookable rooms
15.      * and 5 bookings.
16.      */
17.      static void initialization(){
18.      new Room("SR01", 20);
19.      new Room("PQ25", 10);
20.      new Room("SR04", 15);
21.      new Room("SR06", 2);
22.      new Assistant("Bill21@uok.ac.uk", "Bill Jones");
23.      new Assistant("Watt@uok.ac.uk", "Bob Watt");
24.      new Assistant("SHobs102@uok.ac.uk", "Steve Hobs");
25.      LocalDateTime timeSlot = LocalDateTime.of(2022, 4, 5,
   7, 0, 0);
26.      new AssistantOnShift(timeSlot,
   Assistant.getAssistants()[0]);
27.      new AssistantOnShift(timeSlot.plusHours(1),
   Assistant.getAssistants()[0]);
28.      new AssistantOnShift(timeSlot.plusHours(2),
   Assistant.getAssistants()[0]);
29.      new AssistantOnShift(timeSlot,
   Assistant.getAssistants()[1]);
30.      new AssistantOnShift(timeSlot.plusHours(1),
   Assistant.getAssistants()[1]);
31.      new AssistantOnShift(timeSlot.plusHours(2),
   Assistant.getAssistants()[1]);
32.      new AssistantOnShift(timeSlot,
   Assistant.getAssistants()[2]);
33.      new AssistantOnShift(timeSlot.plusHours(1),
   Assistant.getAssistants()[2]);
34.      new AssistantOnShift(timeSlot.plusHours(2),
   Assistant.getAssistants()[2]);
35.      new BookableRoom(Room.getRooms()[0], timeSlot);
36.      new BookableRoom(Room.getRooms()[0],
   timeSlot.plusHours(1));
37.      new BookableRoom(Room.getRooms()[0],
   timeSlot.plusHours(2));
38.      new BookableRoom(Room.getRooms()[1], timeSlot);
39.      new BookableRoom(Room.getRooms()[1],
   timeSlot.plusHours(1));
40.      new BookableRoom(Room.getRooms()[1],
```

```
   timeSlot.plusHours(2));
41.      new BookableRoom(Room.getRooms()[2], timeSlot);
42.      new BookableRoom(Room.getRooms()[2],
   timeSlot.plusHours(1));
43.      new BookableRoom(Room.getRooms()[2],
   timeSlot.plusHours(2));
44.      new BookableRoom(Room.getRooms()[3],
   timeSlot.plusHours(2));
45.      new Booking(1, "Maddy420@uok.ac.uk",
   BookableRoom.getBookableRooms()[9],
46.     AssistantOnShift.getAssistantOnShifts()[2]);
47.      new Booking(2, "Paddy440@uok.ac.uk",
   BookableRoom.getBookableRooms()[9],
48.     AssistantOnShift.getAssistantOnShifts()[5]);
49.      new Booking(3, "Steven84@uok.ac.uk",
   BookableRoom.getBookableRooms()[3],
50.     AssistantOnShift.getAssistantOnShifts()[0]);
51.      new Booking(4, "MathsGenius@uok.ac.uk",
   BookableRoom.getBookableRooms()[4],
52.     AssistantOnShift.getAssistantOnShifts()[7]);
53.      new Booking(5, "JimBob@uok.ac.uk",
   BookableRoom.getBookableRooms()[5],
54.     AssistantOnShift.getAssistantOnShifts()[8]);
55.      Booking.getBookings()[3].setStatus(true);
56.      }
57.      /**
58.      * The function for the main menu screen.
59.      */
60.      static void mainMenu(){
61.      String menuText = "University of Knowledge - COVID
   test\n\n"
62.      +"Manage Bookings\n\nPlease, enter the number to
   select your option: \n\n"
63.      +"To manage Bookable Rooms:\n\t1. List\n\t2.
   Add\n\t3. Remove\n"
64.      +"To manage Assistants on Shift:\n\t4. List\n\t5.
   Add\n\t6. Remove\n"
65.      +"To manage Bookings:\n\t7. List\n\t8. Add\n\t9.
   Remove\n\t10. Conclude\n"
66.      +"After selecting one the options above, you will be
   presented other
67.     screens.\n"
68.      +"If you press 0, you will be able to return to this
   main menu.\n"
69.      +"Press -1 (or ctrl+c) to quit this application.\n";
70.      Scanner scanner = new Scanner(System.in);
71.      System.out.println(menuText);
72.      int option = -2;
73.      try {
74.      option = scanner.nextInt();
75.      } catch (InputMismatchException e) {
76.      System.out.println("InputMismatchException - Invalid
   input type
77.     entered.");
```

```
78.      clearScreen();
79.      Menu.mainMenu();
80.      }
81.      switch(option){
82.      case -1:
83.      scanner.close();
84.      System.out.println("Quit the application selected.");
85.      System.exit(0);
86.      break;
87.      case 0:
88.      clearScreen();
89.      System.out.println("This is already the main menu.");
90.      Menu.mainMenu();
91.      break;
92.      case 1:
93.      listMenu(BookableRoom.toStringAll());
94.      break;
95.      case 2:
96.      clearScreen();
97.      System.out.println("University of Knowledge - COVID
   test\n\n"
98.      +"Adding Bookable room\n\n"+Room.toStringAll());
99.      addBookableRooms();
100.     break;
101.     case 3:
102.     if (BookableRoom.getNumBookableRooms() < 1){
103.     System.out.println("Error!\nNo Bookable Rooms :
   Please create a
104.    bookable room before deleting one.");
105.     mainMenu();
106.     }
107.     clearScreen();
108.     System.out.println("University of Knowledge - COVID
   test\n\n"
109.     +"Removing Bookable
110.    room\n\n"+BookableRoom.toStringAll(BookableRoom.Status
   Value.EMPTY));
111.     removeBookableRoom();
112.     break;
113.     case 4:
114.     listMenu(AssistantOnShift.toStringAll());
115.     break;
116.     case 5:
117.     clearScreen();
118.     System.out.println("University of Knowledge - COVID
   test\n\n"
119.     +"Adding assistant on
   shift\n\n"+Assistant.toStringAll());
120.     addAssistantsOnShift();
121.     break;
122.     case 6:
123.     if (AssistantOnShift.getnumAssistantsOnShifts() < 1){
124.     System.out.println("Error!\nNo Assistants on
   Shifts :"
```

```
125.     +" Please create an Assistant on Shift before
   deleting one.");
126.     mainMenu();
127.     }
128.     clearScreen();
129.     System.out.println("University of Knowledge - COVID
   test\n\n"
130.     +"Removing Assistants on
131.    Shift\n\n"+AssistantOnShift.toStringAll(AssistantOnShi
   ft.StatusValue.FREE));
132.     removeBookableRoom();
133.     break;
134.     case 7:
135.     listMenu(Booking.toStringAll());
136.     break;
137.     case 8:
138.     clearScreen();
139.     System.out.println("University of Knowledge - COVID
   test\n\n"
140.     +"Adding booking (appointment for a COVID test) to
   the system\n"
141.     +Booking.toStringTimeSlots());
142.     addBookings();
143.     break;
144.     case 9:
145.     clearScreen();
146.     System.out.println("University of Knowledge - COVID
   test\n\n"
147.     +Booking.toStringAll(Booking.StatusValue.SCHEDULED)
148.     +"Removing booking from the system\n");
149.     removeBookings();
150.     break;
151.     case 10:
152.     clearScreen();
153.     System.out.println("University of Knowledge - COVID
   test\n\n"
154.     +Booking.toStringAll(Booking.StatusValue.SCHEDULED)
155.     +"Conclude booking\n");
156.     concludeBookings();
157.     break;
158.     default:
159.     System.out.println("Error - Invalid choice
   selected.");
160.     clearScreen();
161.     Menu.mainMenu();
162.     }
163.     }
164.     /**
165.     * The function for going back to the main menu or
   quitting the program
166.     *
167.     * @param number this is the number to choose whether
   to quit or return.
168.     * @param scanner this is the input scanner that needs
```

```
   to be closed to prevent
169.    memory leaks.
170.     */
171.     static void backQuitMenu(int number,Scanner scanner){
172.     if (number == -1){
173.     scanner.close();
174.     System.out.println("Quit the application selected.");
175.     System.exit(0);
176.     } else if (number == 0){
177.     System.out.println("Returning to the main menu.");
178.     clearScreen();
179.     Menu.mainMenu();
180.     }
181.     }
182.     /**
183.     * This is the function to display a list of all the
   different objects.
184.     *
185.     * @param list this is the list that will be
   displayed. This is a string of the
186.    list of objects.
187.     */
188.     static void listMenu(String list){
189.     clearScreen();
190.     System.out.println("University of Knowledge - COVID
   test\n\n"+list+"\n");
191.     Scanner scanner = new Scanner(System.in);
192.     System.out.println("0. Back to main menu\n-1. Quit
   application.\n");
193.     int option = -2;
194.     try {
195.     option = scanner.nextInt();
196.     } catch (InputMismatchException e) {
197.     System.out.println("InputMismatchException - Invalid
   input type
198.    entered.");
199.     listMenu(list);
200.     }
201.     backQuitMenu(option, scanner);
202.     System.out.println("InputMismatchException - Invalid
   input type entered");
203.     listMenu(list);
204.     }
205.     /**
206.     * This is the function to add a new bookable room.
207.     */
208.     static void addBookableRooms(){
209.     String addString = "Please, enter one of the
   following:\n\n"
210.     +"The sequential ID listed to a room, a data
   (dd/mm/yyyy), "
211.     +"and a time (HH:MM), separated by a white space.\n"
212.     +"0. Back to main menu\n-1. Quit application.\n";
213.     Scanner scanner = new Scanner(System.in);
```

```
214.     System.out.println(addString);
215.     int id = -1,day = -1,month = -1,year = -1,hour =
   -1,minute = -1;
216.     LocalDateTime timeSlot = LocalDateTime.now();
217.     try {
218.     id = scanner.nextInt();
219.     backQuitMenu(id, scanner);
220.     day = scanner.nextInt();
221.     month = scanner.nextInt();
222.     year = scanner.nextInt();
223.     hour = scanner.nextInt();
224.     minute = scanner.nextInt();
225.     timeSlot = LocalDateTime.of(year, month, day, hour,
   minute, 0);
226.     } catch (InputMismatchException | DateTimeException
   e) {
227.     System.out.println("Error!\n"+e+" :
   "+e.getMessage());
228.     addBookableRooms();
229.     }
230.     if (id >= 11 || id <=
   BookableRoom.getNumBookableRooms()){
231.     try {
232.     new BookableRoom(Room.getRooms()[id-11], timeSlot);
233.     } catch (IllegalArgumentException e) {
234.
   System.out.println("Error!\nIllegalArgumentException :
235.    "+e.getMessage());
236.     addBookableRooms();
237.     } catch (Exception e){
238.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
239.     addBookableRooms();
240.     }
241.     System.out.println("Bookable Room added
   successfully:\n"
242.
243.    +BookableRoom.getBookableRooms()
   [BookableRoom.getNumBookableRooms()-1]);
244.     addBookableRooms();
245.     } else{
246.     System.out.println("Error!\nInvalid id : The id is
   not a valid id for a
247.    room.");
248.     addBookableRooms();
249.     }
250.     }
251.     /**
252.     * This is the function to remove a bookable room.
253.     */
254.     static void removeBookableRoom(){
255.     String removeString = "\nPlease, enter one of the
   following:\n\n"
256.     +"The sequential ID to select the bookable room to be
```

```
   removed."
257.     +"\n0. Back to main menu\n-1. Quit application.\n";
258.     int[] indexList =
259.    BookableRoom.convertIndex(BookableRoom.StatusValue.EMP
   TY);
260.     Scanner scanner = new Scanner(System.in);
261.     System.out.println(removeString);
262.     int id = -1;
263.     try {
264.     id = scanner.nextInt();
265.     backQuitMenu(id, scanner);
266.     } catch (IllegalArgumentException e) {
267.
   System.out.println("Error!\nIllegalArgumentException :
268.    "+e.getMessage());
269.     removeBookableRoom();
270.     } catch (Exception e){
271.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
272.     removeBookableRoom();
273.     }
274.     if (id >= 11 || id <=
   indexList[BookableRoom.getNumBookableRooms()]){
275.     int i = 0;
276.     boolean found = false;
277.     // Linear Search to convert the id into the actual
   index of the
278.    bookable room.
279.     while (i < BookableRoom.getNumBookableRooms() && !
   found){
280.     if (indexList[i++] == (id)){
281.     found = true;
282.     }
283.     }
284.     if (!found){
285.     System.out.println("Error!\nInvalid id : The id is
   not a valid id
286.    for a room.");
287.     removeBookableRoom();
288.     }
289.     System.out.println("Bookable Room removed
290.    successfully:\n"+BookableRoom.getBookableRooms()[i-
   1]);
291.     BookableRoom.removeBookableRoom(i-1);
292.     removeBookableRoom();
293.     } else{
294.     System.out.println("Error!\nInvalid id : The id is
   not a valid id for a
295.    room.");
296.     removeBookableRoom();
297.     }
298.     }
299.
300.     /**
```

```
301.     * This is the function so add a new assistant on
   shift
302.     */
303.     static void addAssistantsOnShift(){
304.     String addString = "Please, enter one of the
   following:\n\n"
305.     +"The sequential ID listed to a room, a data
   (dd/mm/yyyy)"
306.     +", separated by a white space.\n"
307.     +"0. Back to main menu\n-1. Quit application.\n";
308.     Scanner scanner = new Scanner(System.in);
309.     System.out.println(addString);
310.     int id = -1,day = -1,month = -1,year = -1;
311.     LocalDateTime timeSlot = LocalDateTime.now();
312.     try {
313.     id = scanner.nextInt();
314.     backQuitMenu(id, scanner);
315.     day = scanner.nextInt();
316.     month = scanner.nextInt();
317.     year = scanner.nextInt();
318.     timeSlot = LocalDateTime.of(year, month, day, 7, 0,
   0);
319.     } catch (InputMismatchException | DateTimeException
   e) {
320.     System.out.println("Error!\n"+e+" :
   "+e.getMessage());
321.     addAssistantsOnShift();
322.     }
323.     if (id >= 11 || id <= Assistant.getnumAssistants()){
324.     try {
325.     new AssistantOnShift(timeSlot,
   Assistant.getAssistants()[id-11]);
326.     new AssistantOnShift(timeSlot.plusHours(1),
327.    Assistant.getAssistants()[id-11]);
328.     new AssistantOnShift(timeSlot.plusHours(2),
329.    Assistant.getAssistants()[id-11]);
330.     } catch (IllegalArgumentException e) {
331.
   System.out.println("Error!\nIllegalArgumentException :
332.    "+e.getMessage());
333.     addAssistantsOnShift();
334.     } catch (Exception e){
335.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
336.     addAssistantsOnShift();
337.     }
338.     System.out.println("Assistant on Shift added
   successfully:\n"
339.
340.    +AssistantOnShift.getAssistantOnShifts()
   [AssistantOnShift.getnumAssistantsOnShifts(
341.    )-3]+"\n"
342.
343.    +AssistantOnShift.getAssistantOnShifts()
```

```
   [AssistantOnShift.getnumAssistantsOnShifts(
344.     )-2]+"\n"
345.
346.     +AssistantOnShift.getAssistantOnShifts()
   [AssistantOnShift.getnumAssistantsOnShifts(
347.     )-1]);
348.      addAssistantsOnShift();
349.      } else{
350.      System.out.println("Error!\nInvalid id : The id is
   not a valid id for a
351.     assistant.");
352.      addAssistantsOnShift();
353.      }
354.      }
355.      /**
356.      * This is the function to remove an assistant on
   shift.
357.      */
358.      static void removeAssistantsOnShift(){
359.      String removeString = "\nPlease, enter one of the
   following:\n\n"
360.      +"The sequential ID to select the assistant on shift
   to be removed."
361.      +"\n0. Back to main menu\n-1. Quit application.\n";
362.      int[] indexList =
363.     AssistantOnShift.convertIndex(AssistantOnShift.StatusV
   alue.FREE);
364.      Scanner scanner = new Scanner(System.in);
365.      System.out.println(removeString);
366.      int id = -1;
367.      try {
368.      id = scanner.nextInt();
369.      backQuitMenu(id, scanner);
370.      } catch (IllegalArgumentException e) {
371.
   System.out.println("Error!\nIllegalArgumentException :
372.     "+e.getMessage());
373.      removeBookableRoom();
374.      } catch (Exception e){
375.      System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
376.      removeBookableRoom();
377.      }
378.      if (id >= 11 || id <=
379.     indexList[AssistantOnShift.getnumAssistantsOnShifts()]
   ){
380.      int i = 0;
381.      boolean found = false;
382.      // Linear Search to convert the id into the actual
   index of the
383.     assistant on shift.
384.      while (i <
   AssistantOnShift.getnumAssistantsOnShifts() && !found){
385.      if (indexList[i++] == (id)){
```

```
386.     found = true;
387.     }
388.     }
389.     if (!found){
390.     System.out.println("Error!\nInvalid id : The id is
   not a valid id
391.    for an Assistant on Shift.");
392.     removeBookableRoom();
393.     }
394.     System.out.println("Assistant on Shift removed
395.    successfully:\n"+BookableRoom.getBookableRooms()[i-
   1]);
396.     BookableRoom.removeBookableRoom(i-1);
397.     removeBookableRoom();
398.     } else{
399.     System.out.println("Error!\nInvalid id : The id is
   not a valid id for
400.    an Assistant on Shift.");
401.     removeBookableRoom();
402.     }
403.     }
404.     /**
405.     * This is the function to add a new booking.
406.     */
407.     static void addBookings(){
408.     String addString = "\nPlease, enter one of the
   following:\n\n"
409.     +"The sequential ID of an available time-slot and the
   student email,
410.    separated by a white space."
411.     +"\n0. Back to main menu\n-1. Quit application.\n";
412.     Scanner scanner = new Scanner(System.in);
413.     System.out.println(addString);
414.     int id = -1;
415.     String email = "";
416.     try {
417.     id = scanner.nextInt();
418.     backQuitMenu(id, scanner);
419.     email = scanner.next();
420.     } catch (InputMismatchException e) {
421.     System.out.println("Error!\n"+e+" :
   "+e.getMessage());
422.     addBookings();
423.     }
424.     LocalDateTime[] validDateTime =
   Booking.validTimeSlots();
425.     int index = 0;
426.     for (int i=0;i<validDateTime.length;i++){
427.     if (validDateTime[i] != null){
428.     index = i;
429.     }
430.     }
431.     try {
432.     new Booking(id, email,
```

```
   BookableRoom.getBookableRooms()[index],
433.    AssistantOnShift.getAssistantOnShifts()[index]);
434.     } catch (IllegalArgumentException e) {
435.
   System.out.println("Error!\nIllegalArgumentException :
436.    "+e.getMessage());
437.     addBookings();
438.     } catch (Exception e){
439.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
440.     addBookings();
441.     }
442.     System.out.println("Booking added successfully:\n"
443.     +Booking.getBookings()[Booking.getNumBookings()-
   1]+"\n"
444.     +Booking.toStringTimeSlots());
445.     addBookings();
446.     }
447.     /**
448.     * This is the function to remove a booking
449.     */
450.     static void removeBookings(){
451.     String removeString = "\nPlease, enter one of the
   following:\n\n"
452.     +"The sequential ID to select the booking to be
   removed from the listed
453.    bookings above."
454.     +"\n0. Back to main menu\n-1. Quit application.\n";
455.     Scanner scanner = new Scanner(System.in);
456.     System.out.println(removeString);
457.     int id = -1;
458.     try {
459.     id = scanner.nextInt();
460.     backQuitMenu(id, scanner);
461.     } catch (InputMismatchException e) {
462.     System.out.println("Error!\n"+e+" :
   "+e.getMessage());
463.     removeBookings();
464.     }
465.     if (id >= 11 || id <= Booking.getNumBookings()){
466.     System.out.println("Booking removed
467.    successfully:\n"+Booking.getBookings()[id-11]);
468.     try{
469.     BookableRoom.removeBookableRoom(id-11);
470.     } catch (IllegalArgumentException e) {
471.
   System.out.println("Error!\nIllegalArgumentException :
472.    "+e.getMessage());
473.     addBookings();
474.     } catch (Exception e){
475.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
476.     addBookings();
477.     }
```

```
478.     removeBookableRoom();
479.     } else{
480.     System.out.println("Error!\nInvalid id : The id is
   not a valid id for
481.    the booking.");
482.     removeBookableRoom();
483.     }
484.     }
485.     /**
486.     * This is the function to conclude a booking.
487.     * This changes its state from scheduled to complete.
488.     */
489.     static void concludeBookings(){
490.     String concludeString = "\nPlease, enter one of the
   following:\n\n"
491.     +"The sequential ID to select the booking to be
   completed."
492.     +"\n0. Back to main menu\n-1. Quit application.\n";
493.     Scanner scanner = new Scanner(System.in);
494.     System.out.println(concludeString);
495.     int id = -1;
496.     try {
497.     id = scanner.nextInt();
498.     backQuitMenu(id, scanner);
499.     } catch (InputMismatchException e) {
500.     System.out.println("Error!\n"+e+" :
   "+e.getMessage());
501.     concludeBookings();
502.     }
503.     if ((id >= 11 || id <= Booking.getNumBookings()) &&
504.
505.    Booking.getBookings()[id-
   11].getStatus().equals(Booking.StatusValue.SCHEDULED)){
506.     try{
507.     Booking.getBookings()[id-11].setStatus(true);
508.     } catch (IllegalArgumentException e) {
509.
   System.out.println("Error!\nIllegalArgumentException :
510.    "+e.getMessage());
511.     concludeBookings();
512.     } catch (Exception e){
513.     System.out.println("Error!\n"+e.getCause()+" :
   "+e.getMessage());
514.     concludeBookings();
515.     }
516.     System.out.println("Booking completed
517.    successfully:\n"+Booking.getBookings()[id-11]);
518.     concludeBookings();
519.     } else{
520.     System.out.println("Error!\nInvalid id : The id is
   not a valid id for
521.    the booking.");
522.     concludeBookings();
523.     }
```

```
524.     }
525.     /**
526.     * This is the function to clear screen.
527.     */
528.     static void clearScreen() {
529.     //System.out.print("\033\143"); could be used but
   would result in incorrect
530.    formatting.
531.     for(int i = 0; i < 100; i++)
532.     {
533.     System.out.println("\b");
534.     }
535.     }
536.     }
```