```java
import java.util.ArrayList;
import java.util.Scanner;
import java.io.IOException;

public class BookingSystem {
    ArrayList<BookableRoom> bookableRooms = new ArrayList<BookableRoom>();
    ArrayList<ShiftAssistant> assistantsOnShift = new ArrayList<ShiftAssistant>();
    ArrayList<Booking> bookings = new ArrayList<Booking>();
    University uniResources = new University();
    Scanner userInput = new Scanner(System.in);

    public BookingSystem(String[][] namesAndEmails, String[] roomCodes, int[] roomCaps,
    String defaultStartTime, String defaultEndTime, int duration){
        this.initialiseUniResources(namesAndEmails, roomCodes, roomCaps);

        if(getDateFromString(defaultStartTime).equals(getDateFromString(defaultEndTime)))
        { //Check the items are on the same day
            this.addToArrayLists(defaultStartTime, defaultEndTime, duration);
        } else {
            System.out.println("The start and end dates are not on the same day");
        }
    }

    public static void emptyConsole() {  //Empties the console
        try {
            if (System.getProperty("os.name").contains("Windows"))
                new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
            else
                Runtime.getRuntime().exec("clear");
        } catch (IOException | InterruptedException ex) {}
    }

    public void mainMenu(String[] args){ //Prints out the main menu and handles the
    choices
        boolean exitInput = false;
        boolean inMenu = false;
        while (exitInput == false && inMenu == false){
            emptyConsole();
            System.out.println("University of Knowledge - COVID test \n\n" +
            "Manage Bookings \n\n" +
            "Please, enter the number to select your option: \n\n" +
            "To manage Bookable Rooms: \n" +
            "   1. List\n" +
            "   2. Add\n" +
            "   3. Remove\n" +
            "To manage Assistants on Shift:\n" +
            "   4. List\n" +
            "   5. Add\n" +
            "   6. Remove\n" +
            "To manage Bookings:\n" +
            "   7. List\n" +
            "   8. Add\n" +
            "   9. Remove\n" +
            "   10. Conclude\n" +
            "After selecting one the options above, you will be presented other
            screens.\n" +
            "If you press 0, you will be able to return to this main menu.\n" +
            "Press -1 (or ctrl+c) to quit this application.\n"
            );
            String currentInput = userInput.nextLine();
            switch (currentInput){
                case "1": //List rooms
                    exitInput = listMenu("ROOMS",  inMenu);
                    break;
                case "2": //Add rooms
                    exitInput = addRoomMenu(inMenu);
                    break;
                case "3": //Remove rooms
                    exitInput = removeRoomMenu(inMenu);
```

```
                    break;
                case "4": //List assistants
                    exitInput = listMenu("ASSISTANTS", inMenu);
                    break;
                case "5": //Add assistants
                    exitInput = addAssistantMenu(inMenu);
                    break;
                case "6": //Remove assistants
                    exitInput = removeAssistantMenu(inMenu);
                    break;
                case "7": //List bookings
                    exitInput = listMenu("BOOKINGS", inMenu);
                    break;
                case "8": //Add bookings
                    exitInput = addBookingsMenu(inMenu);
                    break;
                case "9": //Remove bookings
                    exitInput = removeBookingsMenu(inMenu);
                    break;
                case "10": //Conclude booking
                    exitInput = concludeBookingsMenu(inMenu);
                    break;
                case "-1": //Quit the app
                    exitInput = true;
                    break;
                default:
                    System.out.println("Invalid input, please try again");
            }
        }
    }

    private boolean concludeBookingsMenu(boolean inMenu){
        inMenu = true;
        boolean exitInput = false;
        while(inMenu == true){
            System.out.println("University of Knowledge - COVID test \n\n");
            this.showBookings("SCHEDULED");
            System.out.println("Conclude booking\n\n" +
                "Please, enter one of the following:\n\n"+
                "The sequential ID to select the booking to be removed from the listed
                bookings above\n"+
                "0. Back to main menu.\n"+
                "-1. Quit application.\n");
            String currentInput = userInput.nextLine();
            switch(currentInput){
                case "0":
                    inMenu = false;
                    emptyConsole();
                    break;
                case "-1":
                    inMenu = false;
                    exitInput = true;
                    break;
                default:
                    String bookingIDString = "";
                    for(int i = 0; i < currentInput.length(); i++){
                        if(Character.isDigit(currentInput.charAt(i))){
                            bookingIDString = bookingIDString + currentInput.charAt(i);
                        }
                    }
                    if(bookingIDString != ""){
                        bookings.get(Integer.parseInt(bookingIDString) -
                        11).completeBooking();
                    } else {
                        System.out.println("Invalid input, please try again");
                    }
            }
        }
        return exitInput;
```

```
132         }
133
134         private boolean removeBookingsMenu(boolean inMenu){
135             inMenu = true;
136             boolean exitInput = false;
137             while(inMenu == true){
138                 System.out.println("University of Knowledge - COVID test \n\n");
139                 this.showBookings("SCHEDULED");
140                 System.out.println("Conclude booking\n\n" +
141                     "Please, enter one of the following:\n\n"+
142                     "The sequential ID to select the booking to be removed from the listed
                        bookings above\n"+
143                     "0. Back to main menu.\n"+
144                     "-1. Quit application.\n");
145                 String currentInput = userInput.nextLine();
146                 switch(currentInput){
147                     case "0":
148                         inMenu = false;
149                         emptyConsole();
150                         break;
151                     case "-1":
152                         inMenu = false;
153                         exitInput = true;
154                         break;
155                     default:
156                         String bookingIDString = "";
157                         for(int i = 0; i < currentInput.length(); i++){
158                             if(Character.isDigit(currentInput.charAt(i))){
159                                 bookingIDString = bookingIDString + currentInput.charAt(i);
160                             }
161                         }
162                         if(bookingIDString != ""){
163                             removeBooking(Integer.parseInt(bookingIDString));
164                         } else {
165                             System.out.println("Invalid input, please try again");
166                         }
167                 }
168             }
169             return exitInput;
170         }
171
172         private boolean addBookingsMenu(boolean inMenu){
173             emptyConsole();
174             System.out.println("University of Knowledge - COVID test \n\n" +
175             "Adding booking (appointment for a COVID test) to the system \n");
176             this.showBookableRooms("");
177             inMenu = true;
178             boolean exitInput = false;
179             System.out.println("Please, enter one of the following:\n\n"+
180             "A time-slot and the start of a student email, separated by a white space.\n"+
181             "0. Back to main menu.\n"+
182             "-1. Quit application.\n");
183             while(inMenu == true){
184                 String currentInput = userInput.nextLine();
185                 switch(currentInput){
186                     case "0":
187                         inMenu = false;
188                         emptyConsole();
189                         break;
190                     case "-1":
191                         inMenu = false;
192                         exitInput = true;
193                         break;
194                     default:
195                         int spaceCount = 0;
196                         String timeSlotID = "";
197                         String email = "";
198                         for(int i = 0; i < currentInput.length(); i++){
199                             if(currentInput.charAt(i) != ' '){
```

```
                            switch(spaceCount){
                                case 0:

                                    if(Character.isDigit(currentInput.charAt(i))){timeSlotID = timeSlotID + currentInput.charAt(i);}
                                    break;
                                case 1:
                                    email = email + currentInput.charAt(i);
                                    break;
                            }
                        } else {
                            spaceCount++;
                        }
                    }
                    if(spaceCount == 1 && email != ""){
                        createBooking(bookableRooms.get(Integer.parseInt(timeSlotID) - 11).getTimeSlot(), email);
                    } else {
                        System.out.println("Invalid entry, please try again");
                    }
                }
            }
            return exitInput;
        }

        private boolean removeAssistantMenu(boolean inMenu){
            inMenu = true;
            boolean exitInput = false;
            while(inMenu == true){
                System.out.println("University of Knowledge - COVID test \n\n");
                showShiftAssistants("FREE");
                System.out.println("Please, enter one of the following:\n\n"+
                    "The sequential ID to select the assistant on shift to be removed.\n"+
                    "0. Back to main menu.\n"+
                    "-1. Quit application.\n");
                String currentInput = userInput.nextLine();
                switch(currentInput){
                    case "0":
                        inMenu = false;
                        emptyConsole();
                        break;
                    case "-1":
                        inMenu = false;
                        exitInput = true;
                        break;
                    default:
                        String assistantIDString = "";
                        for(int i = 0; i < currentInput.length(); i++){
                            if(Character.isDigit(currentInput.charAt(i))){
                                assistantIDString = assistantIDString + currentInput.charAt(i);
                            }
                        }
                        if(assistantIDString != ""){
                            removeAssistantOnShift(Integer.parseInt(assistantIDString));
                        } else {
                            System.out.println("Invalid input, please try again");
                        }
                }
            }
            return exitInput;
        }

        private boolean addAssistantMenu(boolean inMenu){
            emptyConsole();
            System.out.println("University of Knowledge - COVID test \n\n" +
            "Adding assistant on shift \n");
            this.showAssistants();
            inMenu = true;
```

```
265         boolean exitInput = false;
266         System.out.println("Please, enter one of the following:\n\n"+
267         "The sequential ID of an assistant, a date (dd/mm/yyyy) separated by a white
            space.\n"+
268         "0. Back to main menu.\n"+
269         "-1. Quit application.\n");
270         while(inMenu == true){
271             String currentInput = userInput.nextLine();
272             switch(currentInput){
273                 case "0":
274                     inMenu = false;
275                     emptyConsole();
276                     break;
277                 case "-1":
278                     inMenu = false;
279                     exitInput = true;
280                     break;
281                 default:
282                     int spaceCount = 0;
283                     String assistantIDString = "";
284                     String date = "";
285                     for(int i = 0; i < currentInput.length(); i++){
286                         if(currentInput.charAt(i) != ' '){
287                             switch(spaceCount){
288                                 case 0:
289
                                        if(Character.isDigit(currentInput.charAt(i))){assista
                                        ntIDString = assistantIDString +
                                        currentInput.charAt(i);}
290                                     break;
291                                 case 1:
292                                     date = date + currentInput.charAt(i);
293                                     break;
294                             }
295                         } else {
296                             spaceCount++;
297                         }
298                     }
299                     if(spaceCount == 1 && date != "" && date.length() == 10){
300                         addAssistantOnShift(Integer.parseInt(assistantIDString), date);
301                     } else {
302                         System.out.println("Invalid entry, please try again");
303                     }
304             }
305         }
306         return exitInput;
307     }
308
309     private boolean removeRoomMenu(boolean inMenu){
310         inMenu = true;
311         boolean exitInput = false;
312         while(inMenu == true){
313             System.out.println("University of Knowledge - COVID test \n\n");
314             showBookableRooms("EMPTY");
315             System.out.println("Please, enter one of the following:\n\n"+
316                 "The sequential ID to select the bookable room to be removed\n"+
317                 "0. Back to main menu.\n"+
318                 "-1. Quit application.\n");
319             String currentInput = userInput.nextLine();
320             switch(currentInput){
321                 case "0":
322                     inMenu = false;
323                     emptyConsole();
324                     break;
325                 case "-1":
326                     inMenu = false;
327                     exitInput = true;
328                     break;
329                 default:
```

```
                    String roomIDString = "";
                    for(int i = 0; i < currentInput.length(); i++){
                        if(Character.isDigit(currentInput.charAt(i))){
                            roomIDString = roomIDString + currentInput.charAt(i);
                        }
                    }
                    if(roomIDString != ""){
                        removeBookableRoom(Integer.parseInt(roomIDString));
                    } else {
                        System.out.println("Invalid input, please try again");
                    }
                }
            }
            return exitInput;
        }

        private boolean addRoomMenu(boolean inMenu){
            emptyConsole();
            System.out.println("University of Knowledge - COVID test \n\n" +
            "Adding bookable room \n");
            this.showRooms();
            inMenu = true;
            boolean exitInput = false;
            System.out.println("Please, enter one of the following:\n\n"+
            "The sequential ID listed to a room, a date (dd/mm/yyyy), and a time (HH:MM),
            separated by a white space.\n"+
            "0. Back to main menu.\n"+
            "-1. Quit application.\n");
            while(inMenu == true){
                String currentInput = userInput.nextLine();
                switch(currentInput){
                    case "0":
                        inMenu = false;
                        emptyConsole();
                        break;
                    case "-1":
                        inMenu = false;
                        exitInput = true;
                        break;
                    default:
                        int spaceCount = 0;
                        String roomIDString = "";
                        String date = "";
                        String time = "";
                        for(int i = 0; i < currentInput.length(); i++){
                            if(currentInput.charAt(i) != ' '){
                                switch(spaceCount){
                                    case 0:

                                        if(Character.isDigit(currentInput.charAt(i))){roomIDS
                                        tring = roomIDString + currentInput.charAt(i);}
                                        break;
                                    case 1:
                                        date = date + currentInput.charAt(i);
                                        break;
                                    case 2:
                                        time = time + currentInput.charAt(i);
                                        break;
                                }
                            } else {
                                spaceCount++;
                            }
                        }
                        if(spaceCount == 2 && time != "" && date.length() == 10 &&
                        time.length() == 5){
                            addBookableRoom(Integer.parseInt(roomIDString), date, time);
                        } else {
                            System.out.println("Invalid entry, please try again");
                        }
```

```
            }
        }
        return exitInput;
    }

    private boolean listMenu(String typeSelected, boolean inMenu){ //Open the menu to show a list of a given type
        emptyConsole();
        System.out.println("University of Knowledge - COVID test \n\n");
        switch(typeSelected){
            case "ROOMS":
                this.showBookableRooms("");
                break;
            case "ASSISTANTS":
                this.showShiftAssistants("");
                break;
            case "BOOKINGS":
                this.showBookings("");
                break;
        }
        inMenu = true;
        System.out.println("0. Back to main menu. \n -1. Quit application.\n\n");
        boolean exitInput = false;
        while(inMenu == true){
            String currentInput = userInput.nextLine();
            switch(currentInput){
                case "0":
                    inMenu = false;
                    emptyConsole();
                    break;
                case "-1":
                    inMenu = false;
                    exitInput = true;
                    break;
                default:
                    System.out.println("Invalid input, please try again");
            }
        }
        return exitInput;
    }

    public void addAssistantOnShift(int assistantID, String date){
        String startTime = "07:00"; //Create a start time
        for(int i = 0; i < 3; i++){  //For 3 shifts
            String currentTime = getTimeFromMinutes(getMinutes(startTime) + (60 * i));
            //Get the time for each time slot
            String timeSlot = ("<" + date + " " + currentTime + ">"); //Add the date and the time slot to make a timeSlot with the format <dd/mm/yyyy HH:MM>
            Assistant newAssistant = uniResources.getAssistants().get(assistantID - 11);
            ShiftAssistant assistantOnShift = new ShiftAssistant(newAssistant, timeSlot);
            assistantsOnShift.add(assistantOnShift);
            System.out.println("Assistant successfully added");
            System.out.println(assistantOnShift.getTranscript());
        }
    }

    public void addBookableRoom(int roomID, String date, String time){
        String timeSlot = ("<" + date + " " + time + ">");
        Room newRoom = uniResources.getRooms().get(roomID - 11);
        BookableRoom newBookableRoom = new BookableRoom(newRoom, timeSlot);
        bookableRooms.add(newBookableRoom);
        System.out.println("Bookable Room added successfully:\n");
        System.out.println(newBookableRoom.getTranscript());
    }

    public void removeAssistantOnShift(int id){
        int indexToGet = id - 11;
        for(int index = 0; index < assistantsOnShift.size(); index++){ //For every assistant on shift
```

```
460             if(index == indexToGet){ //Check if the index is the same as the ID - 10
461                 if(assistantsOnShift.get(index).getStatus().equals("FREE")){ //If it is
                    FREE
462                     System.out.println("Assistant removed successfully");
463                     System.out.println(assistantsOnShift.get(index).getTranscript());
464                     assistantsOnShift.remove(index); //Remove the assistant from the list
465                 } else {
466                     System.out.println("You cannot remove BUSY assistants on shift");
467                 }
468             }
469         }
470     }
471
472     public void removeBookableRoom(int id){
473         int indexToGet = id - 11;
474         for(int index = 0; index < bookableRooms.size(); index++){ //For every bookable
            room
475             if(index == indexToGet){ //Check if the index is the same as the ID - 10
476                 if(bookableRooms.get(index).getStatus().equals("EMPTY")){ //If it is
                    EMPTY
477                     System.out.println("Room removed successfully");
478                     System.out.println(bookableRooms.get(index).getTranscript());
479                     bookableRooms.remove(index); //Remove the bookable room from the list
480                 } else {
481                     System.out.println("You cannot remove non EMPTY rooms");
482                 }
483             }
484         }
485     }
486
487     //--- BOOKING METHODS ---
488     public void removeBooking(int id){ //Removes a booking at a given ID
489         int indexToGet = id - 11;
490         for(int index = 0; index < bookings.size(); index++){ //For every booking
491             if(index == indexToGet){ //Check if the index is the same as the ID - 10
492                 if(bookings.get(index).getStatus().equals("SCHEDULED")){ //If it is
                    scheduled
493                     bookings.get(index).getAssistant().setFree(); //Free up the assistant
494                     bookings.get(index).getRoom().decreaseOccupancy(); //Decrease the
                        occupancy of the room
495                     bookings.remove(index); //Remove the booking from the list
496                 }
497             }
498         }
499     }
500
501     public void createBooking(String timeSlot, String emailStarter){ //Creates a
        booking from a given timeSlot and emailStarter
502         if(checkRoomAvailability(timeSlot) && checkAssistantAvailability(timeSlot)){
            //Check that you can make a booking at this time
503             int roomIndex = getSpareRoomIndex(timeSlot); //Get a usable room
504             int assistantIndex = getSpareAssistantIndex(timeSlot); //Get a usable
                assistant
505             String email = Assistant.createEmail(emailStarter); //Create an email from
                the emailStarter
506             Booking newBooking = new Booking(timeSlot, bookableRooms.get(roomIndex),
                assistantsOnShift.get(assistantIndex), email);
507             bookings.add(newBooking);
508             System.out.println("Booking successfully created");
509             System.out.println(newBooking.getTranscript());
510             bookableRooms.get(roomIndex).increaseOccupancy(); //Increase the occupancy
                of the room
511             assistantsOnShift.get(assistantIndex).setBusy(); //Set the assistantOnShift
                to busy
512         }
513     }
514
515     public int getSpareAssistantIndex(String timeSlot){ //Returns the index of a free
        assistant for a given timeslot
```

```
516             boolean assistantFound = false;
517             int assistantIndex = 0;
518             while(assistantFound == false && assistantIndex < assistantsOnShift.size()){
                //Whilst you haven't found an assistant, and you haven't exceeded the length of
                the list
519                 if(assistantsOnShift.get(assistantIndex).getTimeSlot().equals(timeSlot) &&
                    (assistantsOnShift.get(assistantIndex).getStatus().equals("FREE"))){ //If
                    you have found a room with the same timeslot
520                     assistantFound = true; //Note that you have found a room
521                 } else {
522                     assistantIndex++; //Iterate by 1 along the arraylist
523                 }
524             }
525             return assistantIndex;
526         }
527 
528         public int getSpareRoomIndex(String timeSlot){ //Returns the index of an available
                room for a given timeslot
529             boolean roomFound = false; //Note that you have not yet found an available room
530             int roomIndex = 0;
531             while(roomFound == false && roomIndex < bookableRooms.size()){ //Whilst you
                haven't found a room, and you haven't exceeded the length of the list
532                 if(bookableRooms.get(roomIndex).getTimeSlot().equals(timeSlot) &&
                    bookableRooms.get(roomIndex).getStatus().equals("FULL") != true){ //If you
                    have found a room with the same timeslot
533                     roomFound = true; //Note that you have found a room
534                 } else {
535                     roomIndex++; //Iterate by 1 along the arraylist
536                 }
537             }
538             return roomIndex;
539         }
540 
541         public boolean checkAssistantAvailability(String timeSlot){ //Checks if an
                assistant is available for a specific timeSlot
542             boolean availableAssistant = false;
543             for(ShiftAssistant assistantOnShift : assistantsOnShift){ //Check if there is
                an available assistant on shift for that time
544                 if(assistantOnShift.getTimeSlot().equals(timeSlot) &&
                    (assistantOnShift.getStatus().equals("FREE"))){
545                     availableAssistant = true;
546                 }
547             }
548             if(availableAssistant != true){System.out.println("There are no available
                assistants for that date and time");} //Print an error message if there is not
549             return availableAssistant;
550         }
551 
552         public boolean checkRoomAvailability(String timeSlot){ //Checks if a room is
                available for a specific timeSlot
553             boolean availableRoom = false;
554             for (int roomIndex = 0; roomIndex < bookableRooms.size(); roomIndex++){ //For
                every room
555                 if(bookableRooms.get(roomIndex).getTimeSlot().equals(timeSlot) &
                    (bookableRooms.get(roomIndex).getStatus().equals("FULL") != true)){ //Check
                    if there is an available room for that time
556                     availableRoom = true;
557                 }
558             }
559             if(availableRoom != true){System.out.println("There are no available rooms for
                that date and time");} //Print an error message if there is not
560             return availableRoom;
561         }
562 
563         //--- SHOWING METHODS ---
564         public void showBookings(String statusToSearch){ //Show the transcripts of all
                Bookings
565             int count = 11;
566             switch (statusToSearch){
```

```
            case "":
                System.out.println("List of Bookings:");
                break;
            default:
                System.out.println("List of Bookings: " + statusToSearch);
                break;
        }
        if(statusToSearch != ""){ //If you are searching with a condition
            for(Booking booking : bookings){
                if(booking.getStatus().equals(statusToSearch)){
                    System.out.println(count + ". " + booking.getTranscript());
                }
                count++;
            }
        } else {
            for(Booking booking : bookings){
                System.out.println(count + ". " + booking.getTranscript());
                count++;
            }
        }
    }

    public void showRooms(){ //Show the transcripts of all Rooms
        int count = 11;
        System.out.println("List of Rooms");
        for(Room room : uniResources.getRooms()){
            System.out.println(count + ". " + room.getTranscript());
            count++;
        }
    }

    public void showAssistants(){ //Show the transcripts of all Assistants
        int count = 11;
        System.out.println("List of Assistants");
        for(Assistant assistant : uniResources.getAssistants()){
            System.out.println(count + ". " + assistant.getTranscript());
            count++;
        }
    }

    public void showShiftAssistants(String statusToSearch){ //Show the transcripts of
    all ShiftAssistants
        int count = 11;
        switch (statusToSearch){
            case "":
                System.out.println("List of Assistants on Shift:");
                break;
            default:
                System.out.println("List of Assistants on Shift: " + statusToSearch);
                break;
        }
        if(statusToSearch != ""){ //If you are searching with a condition
            for(ShiftAssistant assistantOnShift : assistantsOnShift){ //For every
            assistant
                if(assistantOnShift.getStatus().equals(statusToSearch)){
                    System.out.println(count + ". " +
                    assistantOnShift.getTranscript()); //Print if its status matches
                    the status to search
                }
                count++;
            }
        } else {
            for(ShiftAssistant assistantOnShift : assistantsOnShift){
                System.out.println(count + ". " + assistantOnShift.getTranscript());
                count++;
            }
        }
    }

```

```
632     public void showBookableRooms(String statusToSearch){ //Show the transcripts of all
        bookableRooms
633         int count = 11;
634         switch (statusToSearch){
635             case "":
636                 System.out.println("List of Bookable Rooms: ");
637                 break;
638             default:
639                 System.out.println("List of Bookable Rooms: " + statusToSearch);
640                 break;
641         }
642         if(statusToSearch != ""){ //If you are searching with a condition
643             for(BookableRoom room : bookableRooms){
644                 if(room.getStatus().equals(statusToSearch)){
645                     System.out.println(count + ". " + room.getTranscript());
646                 }
647                 count++;
648             }
649         } else {
650             for(BookableRoom room : bookableRooms){
651                 System.out.println(count + ". " + room.getTranscript());
652                 count++;
653             }
654         }
655     }
656 
657     //--- TIME AND SETUP FUNCTIONS ---
658     public void addToArrayLists(String startTimeString, String endTimeString, int
        duration){
659     //Add BookableRooms and ShiftAssistants to the list arrays based on start and end
        time strings, with a duration
660         String startTime = getTimeFromString(startTimeString); //Get the times from
            ecah of the strings, as dates are unimportant for calculating time
661         String endTime = getTimeFromString(endTimeString);
662         int timeDifference = getTimeDifference(startTime, endTime); //Get the
            difference in time between the start and end times
663         int numberOfSlots = timeDifference/duration; //Divide the time difference to
            get the number of potential slots
664         String date = getDateFromString(startTimeString);  //Get the date from the
            timeString
665         for(int i = 0; i < numberOfSlots; i++){  //For every possible slot
666             String currentTime = getTimeFromMinutes(getMinutes(startTime) + (duration *
                i)); //Get the time for each time slot
667             String timeSlot = ("<" + date + " " + currentTime + ">"); //Add the date
                and the time slot to make a timeSlot with the format <dd/mm/yyyy HH:MM>
668             for (Assistant currentAssistant: uniResources.getAssistants()){ //Create an
                assistantOnShift with each assistant and time slot
669                 ShiftAssistant assistantOnShift = new ShiftAssistant(currentAssistant,
                    timeSlot);
670                 assistantsOnShift.add(assistantOnShift);
671             }
672             for (Room currentRoom: uniResources.getRooms()){ //Create a bookableRoom to
                make with each room and time slot
673                 BookableRoom newBookableRoom = new BookableRoom(currentRoom, timeSlot);
674                 bookableRooms.add(newBookableRoom);
675             }
676         }
677     }
678 
679     public static String getDateFromString(String dateTime){ //Get the date from a
        given stringin the format <dd/mm/yyyy HH:MM>
680         String date = "";
681         if(dateTime.length() == 18) { //Check the string has the same length of the
            format <dd/mm/yyyy HH:MM>
682             for(int index = 1; index < 11; index++) { //For the characters between 1
                and 11
683                 date = date + dateTime.charAt(index); //Add them to the date string
684             }
685         } else {
```

```
686                 System.out.println("The inputted time and date were not the correct length");
687             }
688             return date;
689         }
690 
691         public static String getTimeFromMinutes(int minutes){ //Return a string in the
        format "HH:MM" from a number of minutes
692             String timeString = "";
693             int tenHours = minutes / (60*10); //Work out how many sets of 10 hours are in
                the minutes
694             minutes -= (tenHours * 60 * 10); //Subtract the sets of 10 hours from the total
695             int oneHours = minutes / 60; //Work out the remaining number of hours
696             minutes -= (oneHours * 60); //Subtract the hours from the total
697             int tenMinutes = minutes / 10; //Work out how many sets of 10 minutes are in
                the minutes
698             minutes -= (tenMinutes * 10); //Subtract the sets of 10 minutes from the total
699             int oneMinutes = minutes; //Get the remaining number of minutes
700             timeString = timeString + tenHours + oneHours + ":" + tenMinutes + oneMinutes;
                //Concatenate the numbers to form a time
701             return timeString;
702         }
703 
704         public static int getTimeDifference(String startTime, String endTime){ //Return the
        difference in time in minutes between two times of format "HH:MM"
705             int timeDifference = getMinutes(endTime) - getMinutes(startTime);
706             if (timeDifference < 0) {
707                 System.out.println("The end time is before the start time");
708                 timeDifference = 0;
709             }
710             return timeDifference;
711         }
712 
713         public static int getMinutes(String timeString){ //Return the number of minutes
        from a time of format "HH:MM"
714             int totalMinutes = 0;
715             if(timeString.length() == 5){ //Check the string is the correct length
716                 for(int index = 0; index < 5; index++){ //For every character in the string
717                     if(Character.isDigit(timeString.charAt(index))){ //Check it is a digit
718                         int multiplier = 1;
719                         switch (index) { //Decide how much the digit should be multiplied
                            based on position within the string
720                             case 0:
721                                 multiplier = 60*10; //The first number represents 10 hours,
                                    which are 60 minutes each
722                                 break;
723                             case 1:
724                                 multiplier = 60; //The second number represents 1 hour,
                                    which is 60 minutes
725                                 break;
726                             case 3: //The third number represents 10 minutes
727                                 multiplier = 10;
728                                 break;
729                             case 4: //The fourth number is the remainder of minutes
730                                 multiplier = 1;
731                                 break;
732                         }
733                         totalMinutes += multiplier *
                            Character.getNumericValue(timeString.charAt(index));
734                     }
735                 }
736             } else {
737                 System.out.println("An inputted time was not the correct length");
738             }
739             return totalMinutes;
740         }
741 
742         public static String getTimeFromString(String dateTime){ //Return the time from a
        string format  <dd/mm/yyyy HH:MM>
743             String time = ""; //Create an empty time string
```

```
744             if(dateTime.length() == 18) { //Check the string has the same length of the
                format <dd/mm/yyyy HH:MM>
745                 for(int index = 12; index < 17; index++) { //For the characters between 12
                    and 18
746                     time = time + dateTime.charAt(index); //Add them to the time string
747                 }
748             } else {
749                 System.out.println("The inputted time and date were not the correct length");
750             }
751             return time;
752         }
753 
754         public void initialiseUniResources(String[][] namesAndEmails, String[] roomCodes,
                int[] roomCaps){
755             uniResources.setAssistants(namesAndEmails); //Add the assistants to the
                university resources
756             uniResources.setRooms(roomCodes, roomCaps); //Add the rooms to university
                resources
757         }
758     }
```