```
import java.util.Scanner;
/**This class defines an assistant (object) who works at the university, and can be used as an assitant on shift */
public class Assistant {

    public String email;
    public String name;
    public String assistantsOnShifts;

    public String getEmail(){
        /** @return a unique emai for each assistant */
        Scanner sc= new Scanner(System.in); //System.in is a standard input stream.

        if (assistantsOnShifts.contains(this.email)) {
            System.out.print("This email already exists for an assistant, please try again");
            String newEmail= sc.nextLine(); //reads string.
            return newEmail;
        }
        else {
            return this.email;
        }
    }
    public String setName() {
        /** Sets the name of the assistant if it is not a blank
         * @return a string that is either the name, or a message saying that the name cannot be blank
        */
        if (this.name != null){
            return this.name;
        }
        else{return "Name cannot be blank";}
    }
```

```

  //System.out.print(getName + getEmail);

  //connstructor
  public Assistant(String email, String name) {
    /**Constructs the asssistant */
    this.email = email;
    this.name = name;
  }
}

//Print template: | <name> | <email> |
```

```java
//import java.lang.reflect.Method;
import java.time.LocalDateTime;
import java.util.ArrayList;
//import java.util.function.Function;
/**This class defines an asssistant on shift (object) who is has shifts and is present during the booked covid tests*/
public class AssistantOnShift {
    public LocalDateTime dateAndTime;
    public String AssistantOnShiftstatus;
    boolean available;
    public Boolean removable;

    public String FREE;
    public String BUSY;
    //constructor:
    public AssistantOnShift(LocalDateTime dateAndTime, String AssistantOnShiftstatus) {
        /**Constructs the asssistant on shift class*/
        this.dateAndTime = dateAndTime;
        this.AssistantOnShiftstatus = AssistantOnShiftstatus;
    }

    //public string assistantEmail
    public String getAssistantStatus(){
        /** Calculates the status of the assistant of shift (based on its availability) and returns said status
         * @return the status of the assistant on shift
         */
        if (available == true) {
            this.AssistantOnShiftstatus = FREE;
        }
        else if (available == false){
            this.AssistantOnShiftstatus = BUSY;
        }
        return this.AssistantOnShiftstatus;
    }
    public boolean isRemovable(){
        /** Returns whether or not the  assistant on shift is removable based on its status
         * @return the boolean value of removeable
         */
        if (this.AssistantOnShiftstatus == FREE){
            removable = true;
        }
        else {
            removable = false;
        }
        return removable;
    }

    //  list assistant on shifts status:FREE
    public ArrayList FREEassistantsOnShiftlist;
    public ArrayList getFREEassistantsOnShiftlist(){
     /** This function loops through all of the asssitants on shift in the list, and selects the ones with a FREE status and puts them in a list specific for that type of assitants on shift
     *@return the list of assistants on shift with a FREE status*/

     for(int i = 0; i < FREEassistantsOnShiftlist.size(); i++) {
        if (this.AssistantOnShiftstatus == FREE){
         this.dateAndTime = dateAndTime;
         this.AssistantOnShiftstatus = AssistantOnShiftstatus;
         FREEassistantsOnShiftlist.add(new AssistantOnShift(dateAndTime, AssistantOnShiftstatus));
        }
     }
     return FREEassistantsOnShiftlist;
```

```
    }

//assistant is registered to shifts for the entire day (7 AM to 10 AM). Given the current 60-minute duration of a
//time-slot, when selecting a date, the system will be creating three assistant on shifts.
//6. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> |
}

```

```
import java.lang.reflect.Method;
import java.time.LocalDateTime;
import java.util.ArrayList;
/** This class defines a bookable room (object) in which the tests will take place once they are booked. */
public class BookableRoom {
    public LocalDateTime dateAndTime;
    public int occupancy;
    public String BookableRoomStatus;
    public String code;

    public int Capacity;
    //Room.getCapacity();
    public String EMPTY;
    public String AVAIABLE;
    public String FULL;

    public ArrayList EMPTYBookableRoomslist;
    public ArrayList getEMPTYBookableRoomslist(){
      /** This function loops through all of the bookable rooms in the list, and selects the ones with an EMPTY status and puts them in a list specific for that type of bookable rooms
      *@return the list of bookablerooms with an EMPTY status*/
      //FREEBookableRoomslist.size() = size
      //for (bookableRooms in bookableRooms) {
      for(int i = 0; i < BookableRoomslist.size(); i++) {
        if (this.BookableRoomStatus == EMPTY){

          this.dateAndTime = dateAndTime;
          this.occupancy = occupancy;
          this.BookableRoomStatus = BookableRoomStatus;
          this.code = code;
          EMPTYBookableRoomslist.add(new BookableRoom(dateAndTime, occupancy, BookableRoomStatus, code));
        }
      }
      return EMPTYBookableRoomslist;
    }

    public String removeEBR(String usersRoom){
      /**This function removes the bookable room from the list of bookable rooms status:EMPTY)*/
      EMPTYBookableRoomslist.remove(usersRoom);
      //getEMPTYBookableRoomslist.remove(room);
      String message;
      message = "The room is no longer empty";
      return message;
    }
    public String getremovedRoom(String usersRoom){
      /**@returns the bookable room that was removed*/
      return usersRoom;
    }

    public String getBookableRoomStatus(){
        /** Calaculates and returns the status of the bookable room based on its occupancy
         * @return the status of the bookable room as a string
         */
        if (occupancy == 0){
            this.BookableRoomStatus = EMPTY;
        }
        else if (occupancy < Capacity) {
            this.BookableRoomStatus = AVAIABLE;
        }
        else if (occupancy == Capacity) {
            this.BookableRoomStatus = FULL;
        }
        else if (occupancy > Capacity){
            //print error message
            System.out.print("The occupancy of a room cannot be higher than itds capacity, please enter a lower occupancy");

           // System.out.print("")
        }
        return this.BookableRoomStatus;
    }
    public boolean removable = false;
    public boolean removable(){
        /** Returns whether the bookable room is removable or not based on its status
         * @return the boolean value of removable
         */
        if (this.BookableRoomStatus == "EMPTY"){
            removable = true;
        }
        else {
            removable = false;
        }
        return removable;
    }
    public int updateOccupancy() {
        /** Adds one to the occupancy of the bookable room, this is called whenever a booking is made i n a particular bookable room
         * @return the updated occupancy as an integer
         */
        return occupancy += 1;
    }
    public int getOccupancy(){
        /** @return the occupancy of the room */
        return this.occupancy;
    }

    //constructor:
public BookableRoom(LocalDateTime dateAndTime, int occupancy, String BookableRoomStatus, String code){
  /**This function constructs the BookableRoom class*/
  this.dateAndTime = dateAndTime;
  this.occupancy = occupancy;
  this.BookableRoomStatus = BookableRoomStatus;
  this.code = code;
}
}
  //if the occupancy changes, then the getBookableRoomStatus method is called to update the BookableRoomStatus of the bookable room
//to do check

//2. A bookable room is a room allocated in a specific time-slot (dd/mm/yyyy HH:MM). Since rooms are available
//from 7 AM - 10 AM, the system will offer at most three bookable rooms (time-slots) per room per day.

//6. The BookableRoomStatus of a bookable room must be updated whenever its occupancy changes.

//7. Print template: | <dd/mm/yyyy HH:MM> | <BookableRoomStatus> | <roomCode> | occupancy: <occupancy> |

//@param
//@return

```

```
import java.util.ArrayList;
import java.util.Scanner;
import java.lang.reflect.Method;
import java.time.LocalDateTime;
/**This booking class defines what variables belong to the ooking object */
public class Booking {

    public LocalDateTime dateAndTime;
    public String status;
    public String studentEmail;
    public int bookingIDcode;
    public String roomCode;
    public String assistantEmail;

    public String COMPLETED;
    public String FREE;
    public String SHEDULED;
    public String FULL;

    public String AssistantOnShiftstatus; //HOW CAN I GET THIS FROM THE ASSISTANT ON SHIFT CLASS? CHECK!
    public String BookableRoomStatus; // HOW CAN I CGET THIS FROM THE BOOKABLE ROOM CLASS? CHECK!

    public String emailInput;

    public String getEmail(){
      /**This function is used to get the students' email
      @return the students' email*/
        System.out.print("What is the start of the students email?");
        Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
        String emailInput = sc.nextLine(); //reads string.
```

```
  return studentEmail = (emailInput + "@uok.ac.uk");
}
public boolean happened = false;
public String getBookingStatus(){
 /**This function calculates tje booking status based on whether or not the test has occured
 @return the status of the booking*/
  if (happened = true){
    this.status = "SCHEDULED";
  }
  else if (happened = false) {
    this.status = "COMPLETED";
  }
  return this.status;
}
public boolean available = false;
public boolean AvailableResources() {
 /**This function calculates whether or not a resource is available based on its statuses
 @return all of the available resources*/
  if (this.BookableRoomStatus == "FULL"){
    available = false;
    System.out.print("This Bookable Room is full, choose another room");
  }
  else if (this.AssistantOnShiftstatus != "FREE"){
    available = false;
    System.out.print("This Assistant on shift isn't availabe right now, please choose another assistant.");
  }
  //return AvailableResources = true;
  else {
    return available = true;}
  //do i need to make this method a boolean? how do i do this
```

```
      return available;
    }

    public boolean rb = false; //rb = removable Booking
    public boolean rb() {
     /**This function calculates whether or not a boking is removable based on whether it has been completed or not which we can find out from its status
     @return the boolean value of the removable booking variable */
      if (status.equals("COMPLETED")) {
          rb = true;
      }
      else {
          rb = false;
      }
      return rb;
    }
    public Booking getBooking(){
     return this.booking;
    }
    //constructor:
public Booking(LocalDateTime dateAndTime, String status, String assistantEmail, String roomCode, String studentEmail, int bookingIDcode) {
  /**This function constructs the booking class*/
    this.dateAndTime = dateAndTime;
    this.status = status;
    this.assistantEmail = assistantEmail;
    this.roomCode = roomCode;
    this.studentEmail = studentEmail;
    this.bookingIDcode = bookingIDcode;
}
}

```

```

//1. 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.

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

//3. To create a booking in a time-slot, the system must certify the availab
//5. A booking not COMPLETED can be cancelled, i.e., deleted from the system.
//After cancellation, the resources (room and assistant) should be released for booking again, i.e., their statuses must be updated.



//1. 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.
//6. A booking SCHEDULED can become COMPLETED. Once completed, the booking cannot be deleted due to audit processes.
//7. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> | <roomCode> | <studentEmail> |



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

      Scanner input = new Scanner(System.in);
      String newAssistant = Scanner.nextline();

     // String newAssistant = input.next();
      assistants.add(newAssistant);
      //return assistants;

   }
   /*public function addRoom(){
      rooms.add(userInput.next());
      return rooms;
   } */

```

```
import java.util.*;
//import java.util.Scanner;
/**This application class is where the covid test bookinjg application is run, it's where things actually happen*/
public class BookingApp {
    /**The main class for the program, where the menus are loaded and all of the functions are called. */
    public static void someMethod() {
        /*This function is used for outputting the original menu screen*/
        System.out.println("University of Knowledge - COVID test");
        System.out.println();
        System.out.println("Manage Bookings");
        System.out.println();

        System.out.println("Please, enter the number to select your option:");
        System.out.println();
        System.out.println("To manage Bookable Rooms:");
        System.out.println("1. List");
        System.out.println("2. Add");
        System.out.println("3. Remove");
        System.out.println("To manage Assistants on Shift:");
        System.out.println("4. List");
        System.out.println("5. Add");
        System.out.println("6. Remove");
        System.out.println("To manage Bookings:");
        System.out.println("7. List");
        System.out.println("8. Add");
        System.out.println("9. Remove");
        System.out.println("10. Conclude");

    }

```

```
public static final String QUIT = "-1. Quit application.";
public static final String BACK = "0. Back to main menu.";
public static final String ERROR = "Error!";

public static void main(String[] args) {
  /** This is the main method where the application is run*/
  someMethod();
  System.out.println("Enter a string: ");

  Scanner input = new Scanner(System.in);
  int userOption = input.nextInt();
  if (userOption == 1) {
    //collection.clear(); //clear interface
    System.out.println("University of Knowledge - COVID test");
    System.out.println();
    UniversityResources.getRooms();
    System.out.println(BACK);
    System.out.println(QUIT);
    System.out.println();

  }
  else if (userOption == 2) {
    //collection.clear(); //clear interface
    Boolean Valid = true;
    System.out.println("University of Knowledge - COVID test");
    System.out.println();
    System.out.println("Adding bookable room");
    System.out.println();
    System.out.println(UniversityResources.getRooms());
    System.out.println("Please, enter one of the following:");
    System.out.println();
    System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time (HH:MM), separated by a white space.");
    System.out.println(BACK);
    System.out.println(QUIT);
    System.out.println();

    //Adding bookable room
    if (Valid = true) {
       System.out.println("Bookable Room added successfully:");
       System.out.println(BookingSystem.getBookableRoom());
       System.out.println("Please, enter one of the following:");
       System.out.println();
       System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time (HH:MM), separated by a white space.");
       System.out.println(BACK);
       System.out.println(QUIT);
       System.out.println();
    }
    else if (Valid = false) {
       System.out.println(ERROR);

       System.out.println("There is no availability for this room at this particular time, please try again rooom isn't available");
       System.out.println("This room does not exist, please try again");
       System.out.println("This time isn't available, you can only book for 07:00, 08:00 and 09:00");
       /*If the entry is NOT valid, the system remains unchanged. In this way, you should append to the screen the following message explaining the problem:
       Error!
       <message explaining the error>*/

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

```
        System.out.println("The sequential ID listed to a room, a date (dd/mm/yyyy), and a time (HH:MM), separated by a white space.");
        System.out.println(BACK);
        System.out.println(QUIT);

        System.out.println();
      }
    }
    else if (userOption == 3) {
      //collection.clear(); //clear interface
      Boolean Valid = true;
        System.out.println("University of Knowledge - COVID test");
        System.out.println();
        System.out.println(BookableRoom.getEMPTYBookableRoomslist()); //static error
        System.out.println("Removing bookable room");
        System.out.println();
        System.out.println("Please, enter one of the following:");
        System.out.println();
        System.out.println("The sequential ID to select the bookable room to be removed.");
        Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
        String usersRoom= sc.nextLine(); //reads string.
        System.out.println(BACK);
        System.out.println(QUIT);
        System.out.println();


        if (Valid = true) {

          BookableRoom.removeEBR(usersRoom);  //static error  //remove the bookable room from the list of bookable rooms status:EMPTY)
          System.out.println("Bookable Room removed successfully:");
```

```
            System.out.println(BookableRoom.getremovedRoom(usersRoom)); //that was deleted);
            System.out.println("Please, enter one of the following:");
            System.out.println();
            System.out.println("The sequential ID to select the bookable room to be removed.");
            System.out.println(BACK);
            System.out.println(QUIT);
            System.out.println();
        }
        else if (Valid = false) {
            System.out.println(ERROR);
            System.out.println("This room is not an empty room that can be removed from the system, please try again");
            System.out.println("Please, enter one of the following:");
            System.out.println();
            System.out.println("The sequential ID to select the bookable room to be removed.");
            System.out.println(BACK);
            System.out.println(QUIT);


        }
    }
    else if (userOption == 4) {
        //collection.clear(); //clear interface
        System.out.println("University of Knowledge - COVID test");
        System.out.println();
        //System.out.println(assistant on shifts List);
        System.out.println(BACK);
        System.out.println(QUIT);
        System.out.println();
    }
    else if (userOption == 5) {
```

```
//collection.clear(); //clear interface
Boolean valid = true;
System.out.println("University of Knowledge - COVID test");
System.out.println();
System.out.println("Adding assistant on shift");
System.out.println();
//System.out.println(assistants);
System.out.println("Please, enter one of the following:");
System.out.println();
System.out.println("The sequential ID of an assistant, and date (dd/mm/yyyy), separated by a white space.");
System.out.println(BACK);
System.out.println(QUIT);
System.out.println();

//Adding assistant on shift
if (valid = true) {
  //add the asssistant to the list of assistants
  System.out.println("Assistant on Shift added successfully:");
  //System.out.println(Assistant on Shift);
  System.out.println("Please, enter one of the following:");
  System.out.println();
  System.out.println("The sequential ID listed of an assistant, a date (dd/mm/yyyy), separated by a white space.");
  System.out.println(BACK);
  System.out.println(QUIT);
  System.out.println();
}
else if (valid = false) {
  System.out.println(ERROR);

  //System.out.println(message explaining the error)
```

```

                System.out.println("There is no availability for this room at this particular time, please try again rooom isn't available");
                System.out.println("This room does not exist, please try again");
                System.out.println("This time isn't available, you can only book for 07:00, 08:00 and 09:00");
                /*If the entry is NOT valid, the system remains unchanged.In this way, you should append to the screen the following
message explaining the problem:
Error!
<message explaining the error>*/

                System.out.println("Please, enter one of the following:");
                System.out.println();
                System.out.println("The sequential ID of an assistant and date (dd/mm/yyyy), separated by a white space.");
                System.out.println(BACK);
                System.out.println(QUIT);
                System.out.println();
            }
        }
        else if (userOption == 6){
            //collection.clear(); //clear interface
            Boolean Valid = true;
            System.out.println("University of Knowledge - COVID test");
            System.out.println();
            System.out.println(AssistantOnShift.getFREEassistantsOnShiftlist());
            System.out.println("Removing assistant on shift");
            System.out.println();
            System.out.println("Please, enter one of the following:");
            System.out.println();
            System.out.println("The sequential ID to select the assistant on shift to be removed.");
            System.out.println(BACK);
            System.out.println(QUIT);
            System.out.println();

            //removing an assistants
            if (Valid = true) {
                //remove the assistant from the list of assistant on shifts status:FREE)
                System.out.println("Assistant on Shift removed successfully:");
                //System.out.println(Assistant on Shift); //that was deleted);
                System.out.println("Please, enter one of the following:");
                System.out.println();
                System.out.println("The sequential ID to select the assistant on shift to be removed");

                System.out.println(BACK);
                System.out.println(QUIT);
                System.out.println();
            }
            else if (Valid = false) {
                /*If the entry is NOT valid, the system remains unchanged. In this way, you should append to the screen the following
                message explaining the problem:
                Error!
                <message explaining the error>*/
                System.out.println(ERROR);
                System.out.println("This is not an assistant on shift that is free to be removed from the system, please try again");

                System.out.println("Please, enter one of the following:");
                System.out.println();
                System.out.println("The sequential ID to select the bookable room to be removed.");

                System.out.println(BACK);
                System.out.println(QUIT);
                System.out.println();
            }

        }
        else if (userOption == 7){
            //collection.clear(); //clear interface
            System.out.println("University of Knowledge - 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(BACK);
            System.out.println(QUIT);
            System.out.println();
            if (userOption == 1 || userOption == 2 || userOption == 3) {
                append:
                System.out.println(BACK);
                System.out.println(QUIT);
                System.out.println();
            }
            else {
                /*If the entry is NOT valid, show by default ALL bookings. Append the screen with the following to explain the problem:
                <list bookings>
                0. Back to main menu.
                -1. Quit application.
                <new line>*/
                //System.out.println(list bookings);
                System.out.println(BACK);
                System.out.println(QUIT);
                System.out.println();
            }
        }
        else if (userOption == 8) {
            //collection.clear(); //clear interface
            Boolean valid = true;
            System.out.println("University of Knowledge - COVID test");
            System.out.println();
            System.out.println("Adding booking (appointment for a COVID test) to the system");
            System.out.println();
            System.out.println("List of available time-slots:");
            //system.out.println(List of available slots + 11, 12,...)
           // 11. dd/mm/yyyy HH:MM
            //12. dd/mm/yyyy HH:MM
            //13. dd/mm/yyyy HH:MM
            //...
            System.out.println();
            System.out.println("Please, enter one of the following:");
            System.out.println();
            System.out.println("The sequential ID of an available time-slot and the student email, separated by a white space.");
            System.out.println(BACK);
            System.out.println(QUIT);
            System.out.println();

            if (valid = true){
                //create a booking in the System
                //If in the selected time-slot there are more than one combination of bookable rooms and assistants on shift, you can implement any strategy to choose the resources for the booking. T
                //just sequentially pick the first ones in the list
                System.out.println("Booking added successfully:");
```

```
          //System.out.println(print booking);
          System.out.println();
          System.out.println("List of available time-slots:");
          //system.out.println(List of available slots + 11, 12,...)
          //11. dd/mm/yyyy HH:MM;
          //12. dd/mm/yyyy HH:MM;
          //13. dd/mm/yyyy HH:MM;
         // ...;
          System.out.println();
          System.out.println("Please, enter one of the following:");
          System.out.println();
          System.out.println("The sequential ID of an available time-slot and the student email, separated by a white space.");
          System.out.println(BACK);
          System.out.println(QUIT);
          System.out.println();
        }
        else if (valid = false) {
          System.out.println(ERROR);

          //add in some potential errors
          //System.out.println(message explaining the error);

          System.out.println("Please, enter one of the following:");
          System.out.println();
          System.out.println("The sequential ID of an available time-slot and the student email, separated by a white space.");
          System.out.println(BACK);
          System.out.println(QUIT);
          System.out.println();
        }
    }
```

```
else if (userOption == 9) {
    //collection.clear(); //clear interface
    Boolean valid = true;
    System.out.println("University of Knowledge - COVID test");
    System.out.println();
    //System.out.println(list booking status:SCHEDULED); //list of bookings with the status scheduled
    System.out.println("Removing booking from the system");
    System.out.println();
    System.out.println("Please, enter one of the following:");
    System.out.println();
    System.out.println("The sequential ID to select the booking to be removed from the listed bookings above.");
    System.out.println(BACK);
    System.out.println(QUIT);
    System.out.println();
    if (valid = true) {
        //remove the booking from the system
        System.out.println("Booking removed successfully:");
        //System.out.println(print booking); //print the booking that was just removed
        System.out.println("Please, enter one of the following:");
        System.out.println();
        System.out.println("The sequential ID to select the booking to be removed from the listed bookings above.");
        System.out.println(BACK);
        System.out.println(QUIT);
        System.out.println();
    }
    else if (valid = false) {
        System.out.println(ERROR);

        //add in some potential errors
        //System.out.println(message explaining the error);



        System.out.println("Please, enter one of the following:");
        System.out.println();
        System.out.println("The sequential ID to select the booking to be removed from the listed bookings above.");
        System.out.println(BACK);
        System.out.println(QUIT);
        System.out.println();
      }
    }
    else if (userOption == 10) {
      //collection.clear(); //clear interface
      Boolean valid = true;
      //Conclude Booking: If a user selects 10 from the Manage Booking menu, they can conclude (finish) a booking.
      //That is, the testing was performed as planned and the record can no longer be deleted from the system.
      //The screen shows:
      System.out.println("University of Knowledge - COVID test");
      System.out.println();

      //System.out.println(list booking status:SCHEDULED) //print the list of bookings with the status: SHEDULED

      System.out.println("Conclude booking");
      System.out.println();
      System.out.println("Please, enter one of the following:");
      System.out.println();
      System.out.println("The sequential ID to select the booking to be completed.");
```

```
System.out.println(BACK);
System.out.println(QUIT);
System.out.println();


//If the sequential ID is valid, complete the respective booking in the system, and append the screen with the following:
//complete booking, add to scheduled bookings

if (valid = true) {
  System.out.println("Booking completed successfully:");
  System.out.println(print Booking.getBooking()); //print the successful booking
  System.out.println("Please, enter one of the following:");
  System.out.println();
  System.out.println("The sequential ID to select the booking to be completed.");
  System.out.println(BACK);
  System.out.println(QUIT);
  System.out.println();
}
else if (valid = false) {

  //If the entry is NOT valid, the system remains unchanged. Append the screen with the following to explain the problem:
  System.out.println(ERROR);
  System.out.println("Your booking id is invalid, please try again");
  System.out.println("Please, enter one of the following:");
  System.out.println();
  System.out.println("The sequential ID to select the booking to be completed.");
  System.out.println(BACK);
  System.out.println(QUIT);
  System.out.println();
}
```

```

          //public static final String BACK = "0. Back to main menu.";
          else if (userOption == -1){ //"-1. Quit application."
            System.exit(1);
          }
          else if (userOption == 0){ //"0. Back to main menu."
            someMethod();
          }
        }

    }

}

```

```

import java.util.ArrayList;
import java.util.Scanner;
import java.time.LocalDateTime;
     /** This class is where bookings are made, added to the list of bookings, and also removed, thus it is also where assistants in shift and bookable rooms are created, added and removed */
public class BookingSystem {




    private ArrayList<BookableRoom> bookableRooms=new ArrayList<BookableRoom>();
    private ArrayList<AssistantOnShift> assistantsOnShifts=new ArrayList<AssistantOnShift>();
    private ArrayList<Booking>bookings=new ArrayList<Booking>();
    public static final String ENTERDAT = "Please enter a date and time";
    public ArrayList createBookableRooms(){
     /** This function creates the bookable rooms by adding them to the list of bookable rooms, with approriate varibles
      * @return the list of bookable rooms after the new bookable rooms have been added
      */
      bookableRooms.add(new BookableRoom(2021-05-25 08:00:00, "EMPTY", BR12, 0)); //1
      bookableRooms.add(new BookableRoom(2021-06-24 09:00:00, "AVAILABLE", BR23, 2)); //2
      bookableRooms.add(new BookableRoom(2021-09-12 07:00:00, "EMPTY", BR34, 0)); //3
      bookableRooms.add(new BookableRoom(2021-10-09 07:00:00, "FULL", BR43, 2)); //4
      bookableRooms.add(new BookableRoom(2021-12-06 09:00:00, "AVAILABLE", BR49, 3)); //5
      bookableRooms.add(new BookableRoom(2021-11-21 08:00:00, "FULL", BR51, 1)); //6
      bookableRooms.add(new BookableRoom(2021-06-15 09:00:00, "AVAILABLE", BR60, 4)); //7
      bookableRooms.add(new BookableRoom(2021-07-04 08:00:00, "AVAILABLE", BR73, 2)); //8
      bookableRooms.add(new BookableRoom(2021-05-16 07:00:00, "FULL", BR98, 4)); //9

      return bookableRooms;
    }
```

```
//• 6 assistants on shift
public ArrayList createAssistantsOnShift(){
 /** This function creates the assistants on shift by adding them to the list of assistants on shift, with approriate varibles
  * @return the list of assistantsOnShifts after the new assistantsOnShifts have been added
  */
   assistantsOnShifts.add(new AssistantOnShift(2021-05-10 07:00:00, "JA32@uok.ac.uk")); //1
   assistantsOnShifts.add(new AssistantOnShift(2021-06-21 07:00:00, "Molly20@uok.ac.uk")); //2
   assistantsOnShifts.add(new AssistantOnShift(2021-09-09 07:00:00, "Sam43@uok.ac.uk")); //3
   assistantsOnShifts.add(new AssistantOnShift(2021-07-07 07:00:00, "Bryan90@uok.ac.uk")); //4
   assistantsOnShifts.add(new AssistantOnShift(2021-10-16 07:00:00, "Jose56@uok.ac.uk")); //5
   assistantsOnShifts.add(new AssistantOnShift(2021-11-05 07:00:00, "Amy72@uok.ac.uk")); //6

   return assistantsOnShifts;
}



public BookableRoom getBookableRoom(){
 /**@return the bookable room*/
 return this.BookableRoom;
}



//list of bookable rooms status:EMPTY
ArrayList EmptyBookableRooms;
String EMPTY;
```

```

public void addBookableRooms() {
      /** add a new bookable room to the list*/

   //bookableRooms.add(dateAndTime, status,roomCode, occupancy);
   //bookableRooms.add(12/03/2022, AVAILABLE,A567, 1);
   //Print template: | <dd/mm/yyyy HH:MM> | <status> | <roomCode> | occupancy: <occupancy> |
   Scanner scanner = new Scanner(System.in);
   System.out.print(ENTERDAT);
   String dateAndTimeString = scanner.next();
   LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
   System.out.print("Please enter a room code");
   String code = scanner.next();
   int oocupancy = BookableRoom.updateOccupancy(); //increase the occupancy by 1 //STATIC ERRORR CHECK
   int status = BookableRoom.getRoomBookableRoomStatus(); //STATIC ERRORR CHECK //get the statuis of the room
   bookableRooms.add(new BookableRoom(dateAndTime, Status, code, occupancy));

}

public void addAssistantsOnShift() {
      //** add a new assistanmt on shift to the list*/

   // Print template: | <0dd/mm/yyyy HH:MM> | <status> | <assistantEmail> |
   Scanner scanner = new Scanner(System.in);

   System.out.print(ENTERDAT);
   String dateAndTimeString = scanner.next();
   LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
   System.out.print("Please enter an assistant's email");
```

```
    String email = scanner.next();
    //AssistantOnShift.getAssistantStatus(); //get the status of the assistant
    assistantsOnShifts.add(new AssistantOnShift(dateAndTime, email));
    }

public void addBookings() {
        /** add a new booking to the list*/

        //7. Print template: | <dd/mm/yyyy HH:MM> | <status> | <assistantEmail> | <roomCode> | <studentEmail> |
        Scanner scanner = new Scanner(System.in);

        System.out.print(ENTERDAT);
        String dateAndTimeString = scanner.next();
        LocalDateTime dateAndTime = LocalDateTime.parse(dateAndTimeString);
        System.out.print("Please enter an assistant's email");
        String email = scanner.next();
        System.out.print("Please enter a room code");
        String code = scanner.next();
        System.out.print("Please enter a student's email");
        String studentEmail = scanner.next();
        //BookableRoom.updateOccupancy(); //increase the occupancy of the room by 1 //CHECK

        BookableRoom.getBookableRoomStatus(); //get the status of the room //CHECK
        AssistantOnShift.getAssistantStatus(); //get the status of the assistant  //CHECK
        Status = Booking.getBookingStatus(); //get the status of the booking
        //Booking b = new Booking();
        bookings.add(new Booking(dateAndTime, Status, email, code, studentEmail));

}
public String removeBookableRooms() {
        /** remove a bookable room from the list
         * @return a string confirming to the user that the bookable room has been removed
        */
        Scanner scanner = new Scanner(System.in);
        System.out.print("Please enter the code of the room you wish to delete");
        String UserCode = scanner.next();
        int deleteIndex1 = bookableRooms.indexOf(UserCode);
        bookableRooms.remove(deleteIndex1);
        return "The bookable room has been removed";
    }

    public String removeAssistantOnShift() {
        /** remove an asssitant on shift from the list
         * @return a string confirming to the user that the assistant on shift has been removed
        */
        Scanner scanner = new Scanner(System.in);
        System.out.print("Please enter the email address of the assistant on shift you would like to remove");
        String UserEmail1 = scanner.next();
        int deleteIndex2 = assistantsOnShifts.indexOf(UserEmail1);
        assistantsOnShifts.remove(deleteIndex2);
        return "The assistant on shift has been removed";
    }

    public String removeBookings() {
        /** remove a booking from the list
         * @return a string confriming to the user that the booking jas beem removed
        */
        Scanner scanner = new Scanner(System.in);
        System.out.print("Please enter the students email of the booking you would like to remove");
        String UserEmail2 = scanner.next();
        int deleteIndex3 = bookings.indexOf(UserEmail2);
        bookings.remove(deleteIndex3);
        return "The booking has been removed";
    }

    public ArrayList showAssistantsOnShift(){
        /** @return the list of assistants on shift */
        return assistantsOnShifts;
    }
    public ArrayList showBookableRooms(){
        /** @return the list of bookable rooms */
        return bookableRooms;
    }
    public ArrayList showBookings(){
        /** @return the list bookings */
        return bookings;
    }
}//DO: check
//3. There is a time-slot concept that will guide the booking system. For instance, rooms will be available, and
//assistants will work at a specific time-slot, i.e., date, time and duration. Hence, tests should be booked at
//available slots.
//4. Every time-slot has a fixed duration – a positive number representing the duration of a test, in minutes. This
//quantity includes the time spent doing the test and the time to sanitize the room. The current policy establishes
//this duration to be 60 minutes.

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

```

3. There is a time-slot concept that will guide the booking System. For instance, rooms will be available, and assistants will work at a specific time-slot, i.e., date, time and duration. Hence, tests should be booked at available slots.

4. Every time-slot has a fixed duration – a positive number representing the duration of a test, in minutes. This quantity includes the time spent doing the test and the time to sanitize the room. The current policy establishes this duration to be 60 minutes.

*/

```
import java.util.ArrayList;
/** This class defines a room (object) incthe unicversity trhat can be used as a bookable room for covid tests */
public class Room {
    public String code;
    public int capacity;


    public String getCode() {
     /**This function is a getter method to retrieve the unique room getCode
       @return the unique room code*/
        this.code = code;
        UniversityResources.getRooms();
        ArrayList rooms = UniversityResources.getRooms();
        if(rooms.contains(this.code)){
            System.out.print("This room code is already being used for another room, please use another room code.");
        }
        else {
            return this.code;
        }
    }

    public int getCapacity() {
     /**This function is a getter method used to retrieve the capacity of a room if it is not < or = to 0
       @return the capacity if it is legitimate, otherwise @return 1 as a defult room capacity */
        this.capacity = capacity;
        if (this.capacity<=0){
            System.out.print("The capacity needs to be an integer value greater than 0, please try again");
            return 1;
        }
        else {
            return this.capacity;
        }
    }

//constructor:
public Room(String code, int capacity) {
  /**This function sonstructs the room class*/
    this.code = code;
    this.capacity = capacity;
}
}
 //System.out.print(getCode() + getCapacity());

/*1.1.2 Room
public Module(int year, byte term, ModuleDescriptor module, StudentRecord[] records, double finalAverageGrade) {
    this.year = year;
1. The university has several rooms, and some of the rooms can be allocated to apply COVID tests.
2. A room must have a string code (e.g., IC215) and a capacity.
3. The code is used to identify the room and, therefore, must be unique.
4. The capacity must be an integer value greater than zero. It represents the number of concurrent assistants that
can be safely allocated in the room to perform tests.
5. Print template: | <code> | capacity: <capacity> | */

```

```
import java.lang.reflect.Method;
import java.util.ArrayList;
import java.util.Scanner;
/**This class is where all of the university resources are made, such s rooms and assitants. */
public class UniversityResources {
    private ArrayList<Assistant> assistants = new ArrayList<Assistant>();
    public static ArrayList<Room> rooms = new ArrayList<Room>();

    public static ArrayList getRooms(){
        /** @return the list of rooms */
        return rooms;
    }
    public ArrayList getAssistants() {
        /** @return the list of assistants*/
        return assistants;
    }

    public ArrayList createRooms(){
        /** This function creates the rooms by adding them to the list of rooms, with approriate varibles for the room code and capcity
         * @return the list of rooms after the new rooms have been added
         */
        //getcode();
        rooms.add(new Room("A33",5));
        rooms.add(new Room("B22", 1));
        rooms.add(new Room("C39", 2));
        rooms.add(new Room("D54", 3));
        rooms.add(new Room("E28", 1));
        return rooms;
    }
    public ArrayList createAssistants() {
```

```
    /** This function creayes the assistants by adding them to the list of assistants, with appropriate variables for the name and email
     * @return the list of assistants after the new assistants have been added
     */
    assistants.add(new Assistant("Julie", "JA32@uok.ac.uk"));
    assistants.add(new Assistant("Sandy", "MrsSandMan@uok.ac.uk"));
    assistants.add(new Assistant("Louise", "LoulOU00@uok.ac.uk"));
    assistants.add(new Assistant("Kathrin", "Kay7890@uok.ac.uk"));
    assistants.add(new Assistant("Robbert", "Bobbles@uok.ac.uk"));
    return assistants;
  }
    //private ArrayList<Assistant> assistants = new ArrayList<Assistant>();


public String addRoom() {
  /** This function allows the user to add a room to the list of rooms (if a new room is available for testing in the university for example)
   * @return a message confirming to the user that the room has been added
   */

  Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
  System.out.print("Please enter a room code");
  String code = sc.nextLine();  //reads string
  System.out.print("Please enter a capacity");
  String cap = sc.nextLine();
  int capInt=Integer.parseInt(cap);
  rooms.add(new Room(code, capInt));
  return "The room has been added";
}

public String addAssistant() {
  /**This function allows the user to add an assistant to the list of assists (if for example a new memeber of staff at the university now has the time to jelp with covid tests)
   *@return a message confirming to the user that the assistant has been added
   */
  Scanner sc= new Scanner(System.in); //System.in is a standard input stream.
  System.out.print("Please enter an assistant name");
  String name = sc.nextLine();  //reads string
  System.out.print("Please enter an email");
  String email = sc.nextLine();  //reads string
  assistants.add(new Assistant(name, email));
  return "The assistant has been added";
}
//constructor:
public UniversityResources(ArrayList assistants, ArrayList rooms) {
  /**This function sonstructs the University Recourses class*/
  this.assistants = assistants;
  this.rooms = rooms;
}
}
/*University Resources
1. The University has a list of assistants and a list of rooms.
2. You should implement functions to add, both assistants and rooms.
3. Due to time constraints, you don’t need to develop screen to manage the university resources, but you need to
pre-load the system with instances of rooms and assistants. Please, check section 1.3 for more details.
*/

```