```java
import java.time.LocalDateTime;
import java.util.ArrayList;
public class Booking {

 enum StatusValue{
 SCHEDULED,
 COMPLETED
 }
 private BookableRoom bookableRoom;
	private AssistantOnShift assistantOnShift;
	private int identifiationCode;

	private String studentEmail;
	private StatusValue status = StatusValue.SCHEDULED;
	private static Booking[] bookings = new Booking[100];
	private static int numBookings = 0;
	/**
	* The constructor for the bookings of the tests.
	*
	* @param identifiationCode the code to identify the booking.
	* @param studentEmail the email of the student who will be tested.
	* @param bookableRoom the room that has been booked for the test.
	* @param assistantOnShift the assistant that will be doing the test.
	*/
	public Booking(int identifiationCode, String studentEmail,
	BookableRoom bookableRoom, AssistantOnShift assistantOnShift){
	this.identifiationCode = checkIdentificationCode(identifiationCode);
	this.studentEmail = checkEmail(studentEmail);
	this.bookableRoom = checkBookableRoom(bookableRoom);
	this.assistantOnShift = checkAssistantOnShift(assistantOnShift);
	checkTimeSlot();
	assistantOnShift.setStatus(true);
	bookableRoom.setOccupancy(bookableRoom.getOccupancy()+1);
	addBookings(this);
	iterateNumBookings();
	}
	/**
	* The getter method for the room this test is booked in.
	*
	* @return the bookable room.
	*/
	public BookableRoom getBookableRoom(){
	return bookableRoom;
	}
```

```
/**
* The getter method for the assistant that will be doing the test.
*
* @return the assistant on shift.
*/
public AssistantOnShift getAssistantOnShift(){
return assistantOnShift;
}
/**
* The getter method for the identification code
*
* @return the code to identify the booking.
*/
public int getIdentificationCode(){
return identifiationCode;
}
/**
* The getter method for the student's email.
*
* @return the email of the student.
*/
public String getStudentEmail(){
return studentEmail;
}
/**
* The getter method for the status of the booking.
*
* @return the current status of the booking.
*/
public StatusValue getStatus(){
return status;
}
/**
* The static getter method for the number of bookings.
*
* @return the number of bookings.
*/
public static int getNumBookings(){
return numBookings;
}
/**
* The static getter method for the list of all bookings.
*
* @return the list of all bookings.
*/
public static Booking[] getBookings(){
return bookings;
}
/**
* The setter method for the identification code
*
```

```
    * @param identifiationCode a new identification code for the booking to be set
   to.
    */
    public void setIdentificationCode(int identifiationCode){
    this.identifiationCode = checkIdentificationCode(identifiationCode);
    }
    /**
    * The setter method for the student's email.
    *
    * @param studentEmail a new student's email for this booking.
    */
    public void setStudentEmail(String studentEmail){
    this.studentEmail = checkEmail(studentEmail);
    }
    /**
    * The setter method for the current status of the booking
    *
    * @param complete true - the booking is complete, false - the booking is
   scheduled.
    */
    public void setStatus(boolean complete){
    if (complete){//The status can only go from scheduled to complete not the
   other way round.
    this.status = StatusValue.COMPLETED;
    }
    }
    /**
    * The static method to incement the number of bookings.
    */
    private static void iterateNumBookings(){
    numBookings += 1;
    }
    /**
    * The static method to add a new booking to the list of bookings
    *
    * @param booking a new booking to be added to the list.
    */
    private static void addBookings(Booking booking){
    bookings[numBookings] = booking;
    }
    /**
    * The static method to remove a booking from the list of bookings
    *
```

```
* @param booking a booking to be removed from the list.
*/
public static void removeBookings(Booking booking){
if (booking.status != StatusValue.SCHEDULED){
throw new IllegalArgumentException("To remove the booking it cannot
already been completed.");
}
boolean found = false;
int index = -1;
// linear search to find booking in the array.
while (!found){
index += 1;
if (bookings[index] == booking){
found = true;
}else if (index >= numBookings){
throw new IllegalArgumentException("The booking was not found.");
}
}
// Shifting the last elements.
for (int i=index;i<=numBookings-1;i++){
bookings[i] = bookings[i+1];
}
bookings[numBookings] = null;
numBookings -= 1;
}
/**
* The email checker private method.
* Checks if the email of an student ends with "@uok.ac.uk" and is unique.
*
* @param email the email of a specific student.
* @return the email of a specific student if it is valid.
*/
private String checkEmail(String email){
if (!email.endsWith("@uok.ac.uk")){
throw new IllegalArgumentException("The email string should always end
with @uok.ac.uk.");
}
for (int i=0;i<numBookings;i++){
if (bookings[i].getStudentEmail().equals(email)){
throw new IllegalArgumentException("The email should be
unique.");
}
}
return email;
}
/**
* The method to check the identification code is
```

```
valid
    *
    * @param identifiationCode an identification code being checked.
    * @return the identfication code if it valid.
    */
    private int checkIdentificationCode(int identifiationCode){
    for (int i=0;i<numBookings;i++){
    if (bookings[i].getIdentificationCode() == identifiationCode){
    throw new IllegalArgumentException("The identification code should
   be unique.");
    }
    }
    return identifiationCode;
    }
    /**
    * The method to check the bookable room is valid.
    *
    * @param bookableRoom a bookable room being checked.
    * @return the bookable room if it is valid.
    */
    private BookableRoom checkBookableRoom(BookableRoom bookableRoom){
    if (bookableRoom.getStatus().equals(BookableRoom.StatusValue.FULL)){
    throw new IllegalArgumentException("The bookable room is full so not
   booking can be made.");
    }
    return bookableRoom;
    }
    /**
    * The method is check if an assistant on shift is valid.
    *
    * @param assistantOnShift an assistant on shift being check
    * @return the assistant on shift if it is valid.
    */
    private AssistantOnShift checkAssistantOnShift(AssistantOnShift
   assistantOnShift){
    if
   (assistantOnShift.getStatus().equals(AssistantOnShift.StatusValue.BUSY)){
    throw new IllegalArgumentException("The assistant on shift cannot be
   busy.");
    }
    return assistantOnShift;
```

```
}
/**
* The method to check if the time slot is valid.
*/
private void checkTimeSlot(){
if (!assistantOnShift.getTimeSlot().isEqual(bookableRoom.getTimeSlot())){
throw new IllegalArgumentException("The timeslots do not match with the
assistant on shift and the bookable room.");
}
}
/**
* The static method to remove duplicates from a list.
*
* @param <e> the elements of that list.
* @param list the list that contains duplicates.
* @return the list that has no ducplicates.
*/
private static <e>ArrayList<e> removeDuplicate(ArrayList<e> list) {
ArrayList<e> correctList = new ArrayList<e>(); // Uses array list since it
is easier.
for (e element : list) {
if (!correctList.contains(element)) {
correctList.add(element);
}
}
return correctList;
}
/**
* The method to return the list of current valid time slots for new bookings.
*
* @return the list of valid time slots.
*/
public static LocalDateTime[] validTimeSlots(){
LocalDateTime[] validDateTime = new
LocalDateTime[AssistantOnShift.getnumAssistantsOnShifts()];
for (int i=0;i<AssistantOnShift.getnumAssistantsOnShifts();i++){
for (int j=0;j<BookableRoom.getNumBookableRooms();j++){
if
(AssistantOnShift.getAssistantOnShifts()[i].getTimeSlot().equals(BookableRoom.getBo
okableRooms()[j].getTimeSlot())
&&
AssistantOnShift.getAssistantOnShifts()[i].getStatus().equals(AssistantOnShift.Stat
usValue.FREE)
```

```
 &&
!BookableRoom.getBookableRooms()[j].getStatus().equals(BookableRoom.StatusValue.FUL
L)){
 validDateTime[i] =
AssistantOnShift.getAssistantOnShifts()[i].getTimeSlot();
 break;
 }
 }
 }
 return validDateTime;
 }
 /**
 * The method to return the string of all valid time slots.
 *
 * @return the string of all valid time slots in the correct format.
 */
 public static String toStringTimeSlots(){
 String allTimeSlots = "List of available time-slots-\n";
 ArrayList<LocalDateTime> validDateTime = new ArrayList<LocalDateTime>();
 for (int i=0;i<AssistantOnShift.getnumAssistantsOnShifts();i++){
 for (int j=0;j<BookableRoom.getNumBookableRooms();j++){
 if
(AssistantOnShift.getAssistantOnShifts()[i].getTimeSlot().equals(BookableRoom.getBo
okableRooms()[j].getTimeSlot())
 &&
AssistantOnShift.getAssistantOnShifts()[i].getStatus().equals(AssistantOnShift.Stat
usValue.FREE)
 &&
!BookableRoom.getBookableRooms()[j].getStatus().equals(BookableRoom.StatusValue.FUL
L)){

validDateTime.add(AssistantOnShift.getAssistantOnShifts()[i].getTimeSlot());
 break;
 }
 }
 }
 validDateTime = removeDuplicate(validDateTime);
 for (int i=0;i<validDateTime.size();i++){
 allTimeSlots = allTimeSlots.concat((i+11)+".
"+BookableRoom.formatDate(validDateTime.get(i))+"\n");
 }
 return allTimeSlots;
```

```
}
/**
* This is the overloaded method to return the the string of all bookings.
*
* @return a string of all bookings.
*/
public static String toStringAll(){
String allBookings = "Bookings-\n";
for (int i=0;i<numBookings;i++){
allBookings = allBookings.concat((i+11)+".
"+bookings[i].toString()+"\n");
}
return allBookings;
}
/**
* This is the overloaded method to return the the string of all bookings
*
* @param status this is the status to filter the bookings by.
* @return a string of bookings that are valid for the status.
*/
public static String toStringAll(StatusValue status){
String allbookings = "Bookings-\n";
int index = 0;
for (int i=0;i<numBookings;i++){
if (bookings[i].status == status){
allbookings = allbookings.concat((index+11)+".
"+bookings[i].toString()+"\n");
index++;
}
}
return allbookings;
}
/**
* This is the method to return a string of a booking.
*
* @return a string of a booking.
*/
public String toString(){
return "| "+assistantOnShift.formatDate(assistantOnShift.getTimeSlot())+
" | "+status+" | "+assistantOnShift.getAssistant().getEmail()+" |
"+bookableRoom.getRoom().getCode()+" | "+studentEmail+" |";
}
}
```