```java
import java.time.LocalDateTime;
public class AssistantOnShift {

 // Used for displaying the availability of an assistant.
 enum StatusValue{
 FREE,
 BUSY
 }
 // Formatted yyyy-mm-dd-HH-MM-ss-ns
     private LocalDateTime timeSlot;
     private Assistant assistant;
     private StatusValue status = StatusValue.FREE;
     private static AssistantOnShift[] assistantOnShifts = new
    AssistantOnShift[100];
     private static int numAssistantsOnShift = 0;
     /**
     * The cconstrutor method for Assistants on Shift.
     *
     * @param timeSlot a LocalDateTime used to store the time the assistant on the
    shift.
     * @param assistant this is the class representing the actual assistant.
     */
     public AssistantOnShift(LocalDateTime timeSlot, Assistant assistant){
     this.timeSlot = checkTimeSlot(timeSlot);
     this.assistant = assistant;
     addAssistantsOnShift(this);
     iterateNumAssistantsOnShift();
     }
     /**
     * The getter method for time slots
     *
     * @return the timeslot of the assistant on shift.
     */
     public LocalDateTime getTimeSlot(){
     return timeSlot;
     }
     /**
     * The getter method for the assistant
     *
     * @return the assistant
     */
     public Assistant getAssistant(){
     return assistant;
     }
     /**
     * The getter method for the status.
     *
     * @return the status from the enum, StatusValue.
     */
     public StatusValue getStatus(){
```

```
return status;
}
/**
* The getter method for the list of assistants on
shift.
*
* @return the list of assistants on shift.
*/
public static AssistantOnShift[]
getAssistantOnShifts(){
return assistantOnShifts;
}
/**
* The getter method for the number of assistants on
shift
*
* @return the number of assistants on shift.
*/
public static int getnumAssistantsOnShifts(){
return numAssistantsOnShift;
}
/**
* The setter method for the time slot.
*
* @param timeSlot a new time slot for the assistant
on shift.
*/
public void setTimeSlot(LocalDateTime timeSlot){
this.timeSlot = checkTimeSlot(timeSlot);
}
/**
* The setter method for the status of the assistant
on shift.
*
* @param busy true - sets the status to busy, false -
sets the status to free.
*/
public void setStatus(boolean busy){
if (busy){
status = StatusValue.BUSY;
}else{
status = StatusValue.FREE;
}
}
/**
* The method to add an assistant on shift to the list
of assistants on shift.
*
* @param assistantOnShift this is an assistant on
shift to be added to the
list.
*/
private static void
addAssistantsOnShift(AssistantOnShift assistantOnShift){
```

```
assistantOnShifts[numAssistantsOnShift] = assistantOnShift;
}
/**
* The method to remove an assistant on shift from the list of assistants on
shift.
*
* @param assistantOnShift this is an assistant on shift to be removed from the
list.
*/
public static void removeAssistantsOnShift(AssistantOnShift assistantOnShift){
if (assistantOnShift.status != StatusValue.FREE){
throw new IllegalArgumentException("To remove an assistant on shift it
cannot be busy.");
}
// linear search to find booking in the array.
boolean found = false;
int index = -1;
while (!found){
index += 1;
if (assistantOnShifts[index] == assistantOnShift){
found = true;
}else if (index >= numAssistantsOnShift){
throw new IllegalArgumentException("The assistant on shift was not
found.");
}
}
// Shifting the last elements.
for (int i=index;i<=numAssistantsOnShift-1;i++){
assistantOnShifts[i] = assistantOnShifts[i+1];
}
assistantOnShifts[numAssistantsOnShift] = null;
numAssistantsOnShift -= 1;
}
/**
* A method to increment the number of assistants
* Usually called in the constructor of AssistantOnShift.
*/
private static void iterateNumAssistantsOnShift(){
numAssistantsOnShift += 1;
}
/**
* The method to check if a time slot is valid.
*
* @param timeSlot this is the timeslot being checked
* @return this will return the timeslot if it is valid.
*/
```

```java
    private LocalDateTime checkTimeSlot(LocalDateTime timeSlot){
    LocalDateTime currentTime = LocalDateTime.now();
    if (timeSlot.isBefore(currentTime)){
    throw new IllegalArgumentException("The time slot has already
   passed.");
    }else if (timeSlot.getHour() < 7 || timeSlot.getHour() > 10){
    throw new IllegalArgumentException("The time slot must be between 7 and
   10.");
    }else if (timeSlot.getMinute() != 0 || timeSlot.getSecond() != 0){
    throw new IllegalArgumentException("The time slot must be at the start
   of the hour.");
    }
    for (int i=0;i<numAssistantsOnShift;i++){
    if (timeSlot == assistantOnShifts[i].getTimeSlot() &&
   assistantOnShifts[i].getAssistant().equals(assistant)){
    throw new IllegalArgumentException("Duplicate time slot.");
    }
    }
    return timeSlot;
    }
    /**
    * This is the static method to format the date correctly
    *
    * @param time this is the time that needs to be formatted.
    * @return A string of the formatted date.
    */
    public static String formatDate(LocalDateTime time){
    return time.getDayOfMonth()+"/"+time.getMonthValue()+"/"+time.getYear()+"
   "+time.getHour()+":"+time.getMinute()+"0";
    }
    /**
    * This is the method to convert the index and the sequential id
    * when removing an assistant on shift.
    *
    * @param status this is the status to be filtering the assistants on shift by.
    * @return this is the list of indexes used to convert the id to the idex.
    */
    public static int[] convertIndex(StatusValue status){
    int[] indexList = new int[numAssistantsOnShift+1];
    int index = 11;
```

```
int i;
for (i=0;i<numAssistantsOnShift;++i){
if (assistantOnShifts[i].status == status){
indexList[i] = index++;
}
}
indexList[numAssistantsOnShift] = index-11; // To store the actual length
of the list.
return indexList;
}
/**
* This is the overloaded method to return the the string of all assistants on
shift
*
* @return a string of all assistants on shift.
*/
public static String toStringAll(){
String allAssistantsOnShift = "Assistants on Shift-\n";
for (int i=0;i<numAssistantsOnShift;i++){
allAssistantsOnShift = allAssistantsOnShift.concat((i+11)+".
"+assistantOnShifts[i].toString()+"\n");
}
return allAssistantsOnShift;
}
/**
* This is the overloaded method to return the the string of all assistants on
shift
*
* @param status this is the status to filter the assistants on shift by.
* @return a string of all assistants on shift that are valid for the status.
*/
public static String toStringAll(StatusValue status){
String allAssistantsOnShift = "Assistants on Shift-\n";
int index = 0;
for (int i=0;i<numAssistantsOnShift;i++){
if (assistantOnShifts[i].status == status){
allAssistantsOnShift = allAssistantsOnShift.concat((index+11)+".
"+assistantOnShifts[i].toString()+"\n");
index++;
}
}
return allAssistantsOnShift;
}
/**
* This is the method to return a string of an
```

```
assistant on shift.
     *
     * @return a string of an assistant on shift.
     */
     public String toString(){
     return "| "+formatDate(timeSlot)+" | "+status+" |
"+assistant.getEmail()+"
    |";
     }
    }
```