Object Oriented Programming | Hospital Application

KARL-JOHAN BAILEY | 2018212008

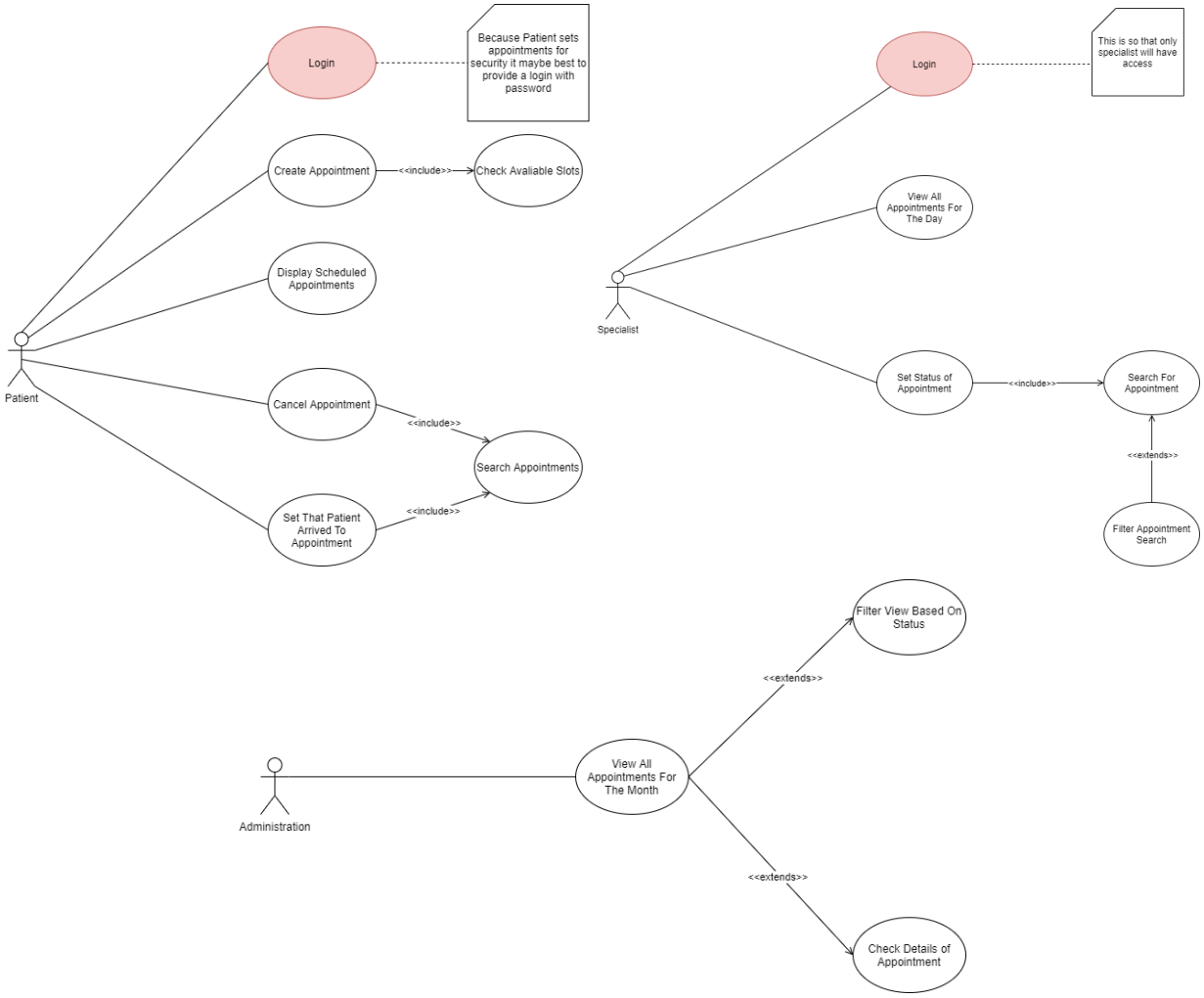
INTRODUCTION:

For this project I used many different design patterns and OOP principles. This document will provide diagrams, explanations and reasons for the change from the original UML assignment.

KARL-JOHAN BAILEY | 2018212008

USE CASE DIAGRAMS

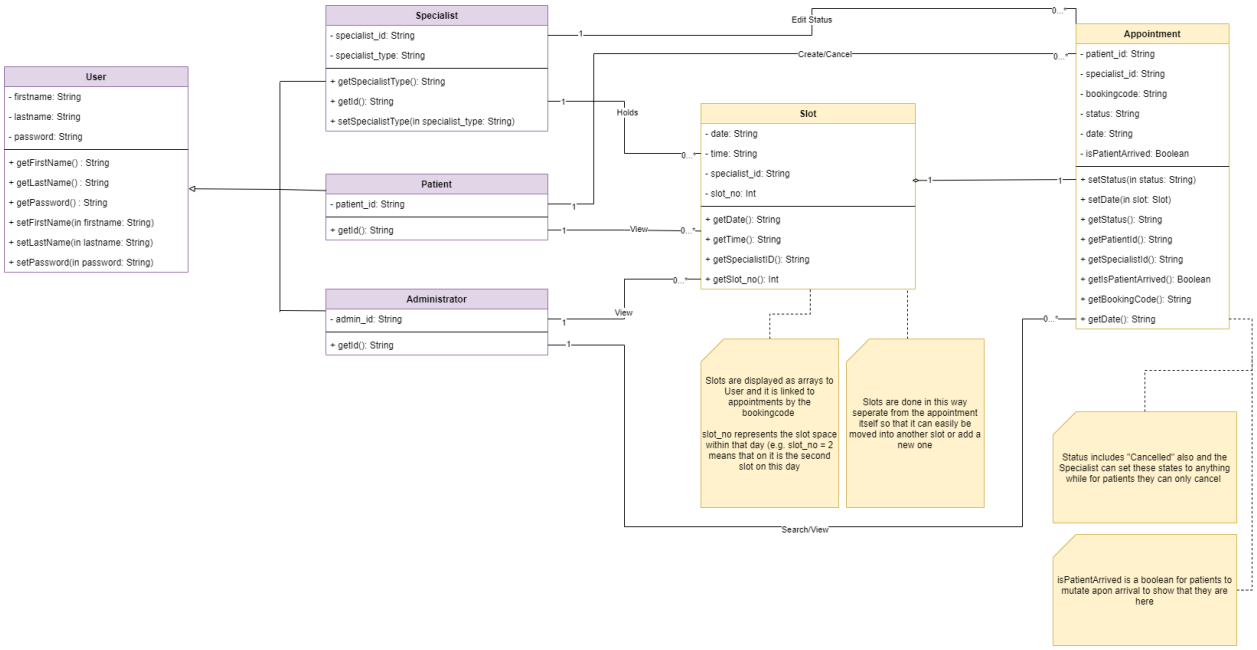
These Use Case diagrams are from the original assignment and has mostly stayed the same with the features granted to each class



Within these diagrams a change was made to the Administrator class. I gave this class the ability to set Patient's appointment as "waiting", because the Admin should have that sort of accessibility

Object Oriented Programming | Hospital Application

KARL-JOHAN BAILEY | 2018212008



CLASS DIAGRAM

Above shows the diagram from the original assignment. Many changes were made and new design patterns were implemented.

1. User and Its Inheritance: For the most part this class has stayed the same however a lot of the attributes were reconsidered. For one the ID was made universal amongst all children of the class.
2. Appointment: This has changed a little to suit the statuses the appointments can have. Instead of “isPatientArrived” I switched it to a String called status. This is so that it can be changed around from “waiting”, “finished” etc. Also appointment holds the entire User object instead of just their id
3. Slot: We got rid of the slot class and just kept it as an Integer which will be in Appointments.
4. API: This is a new class we have implemented which uses a Singleton Pattern. This class was made to keep track of all users and appointments. It is like a database.

KARL-JOHAN BAILEY | 2018212008

DESIGN PATTERNS

For patterns we used Singleton Pattern and Observer Pattern. In an abstract sense we used Strategy Pattern at a higher abstraction level with the GUI and what it allowed different users to do.

Here we will show the different patterns within the code:

Singleton Pattern

```
//Singleton

static API api = new API();
private API(){




}
public static API getInstance() { return api; }

//Each method should take object and search for it in this
```

This is so that anywhere in the program and search and modify ONE set of data

Observer Pattern

```
public Appointment(int slotNumber, LocalDate date, Patient patient, Specialist specialist){

    this.slotNumber = slotNumber;
    this.date = date;
    this.patient = patient;
    this.specialist = specialist;

}

public void subscribeAdministrator(Administrator administrator) { administrators.add(administrator); }

public void notifyUsers(){ //Do this everytime change is made to appointment

    //Notify the users who are linked to this appointment

    for(Administrator administrator : administrators)
    {
        administrator.appointmentUpdated(this.bookingCode, this.status, this.slotNumber);
    }
    specialist.appointmentUpdated(this.bookingCode, this.status, this.slotNumber);
    patient.appointmentUpdated(this.bookingCode, this.status, this.slotNumber);

    API api = API.getInstance();

    api.searchAppointments(this.bookingCode).setStatus(this.status);
}
```

```
public void notificationCheck(){
    if(notifications.size() != 0)
    showMessageDialog( parentComponent: null, notifications.get(notifications.size()-1)); //Call the last notification
}

public void appointmentSubscribed(Appointment appointment) { appointments.add(appointment); }

public void appointmentUpdated(String bookingCode, String status, int slotNumber){ //Test this method

    //Return data to notify user

    int i = 0;

    while(appointments.size() > i){
        if(appointments.get(i).getBookingCode().equalsIgnoreCase(bookingCode)){
            appointments.get(i).setStatus(status);
            appointments.get(i).setSlotNumber(slotNumber);
            //return appointments.get(i);

            break;
        }
        i++;
    }
    // return null;
    if(isActive){
        showMessageDialog( parentComponent: null,  message: "Appointment With Dr." + appointments.get(i).getSpecialist().getFirstname() + " " + appointment
    }
    else{
        notifications.add( "Appointment With Dr." + appointments.get(i).getSpecialist().getFirstname() + " " + appointments.get(i).getSpecialist().ge
    }
}
```

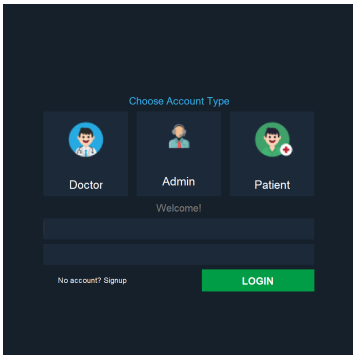
This was modified a little to suit my program. The goal was to notify users who are connected to this appointment.

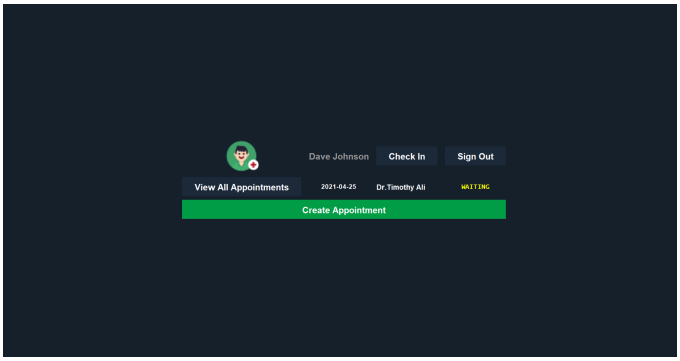
KARL-JOHAN BAILEY | 2018212008

GUI

Within the JAVA Swing classes most of the code is written. While the other classes had main functionality the Interface had to form and bring it all together.

The main thing for each GUI Class was to get the instantiation of the api class to work with. Overall that carried the entire user experience. Due to rushing a very simple GUI was implemented especially for the Creating Appointments class.





KARL-JOHAN BAILEY | 2018212008

USER CREDENTIALS

These were made to help with testing, you can always add your own users with registration

Pre-Made Users

- Patient: Dave MedID: 1000, Password: admin123
- Specialist
  - Timothy MedID: 1000, Password admin123
  - Ryan MedID: 1001, Password admin123
  - Luis MedID: 1002, Password admin123
  - Hillary MedID: 1003, Password admin123
  - Ethan MedID: 1004, Password admin123
- Administrator: Gabby MedID: 100, Password admin123