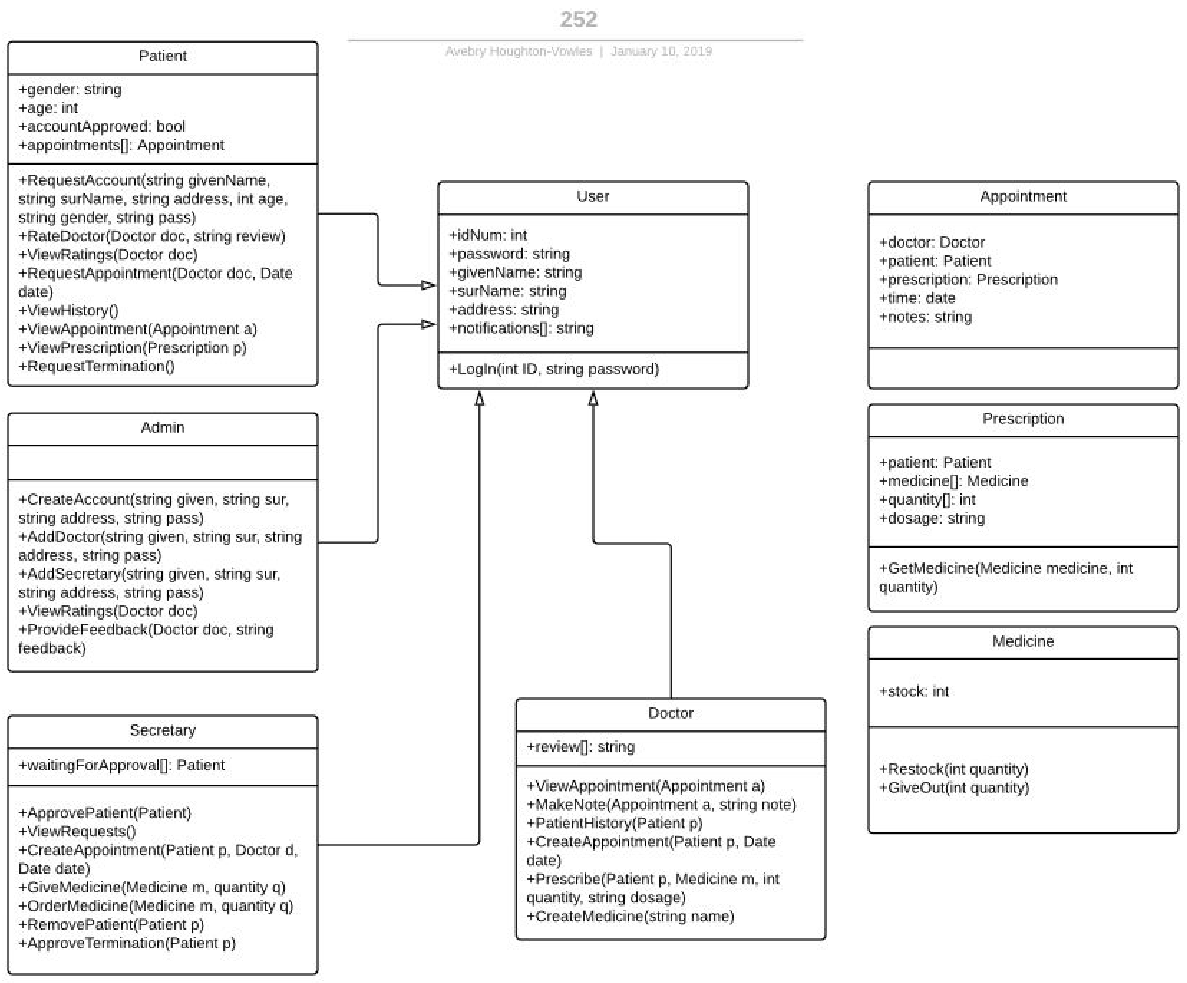
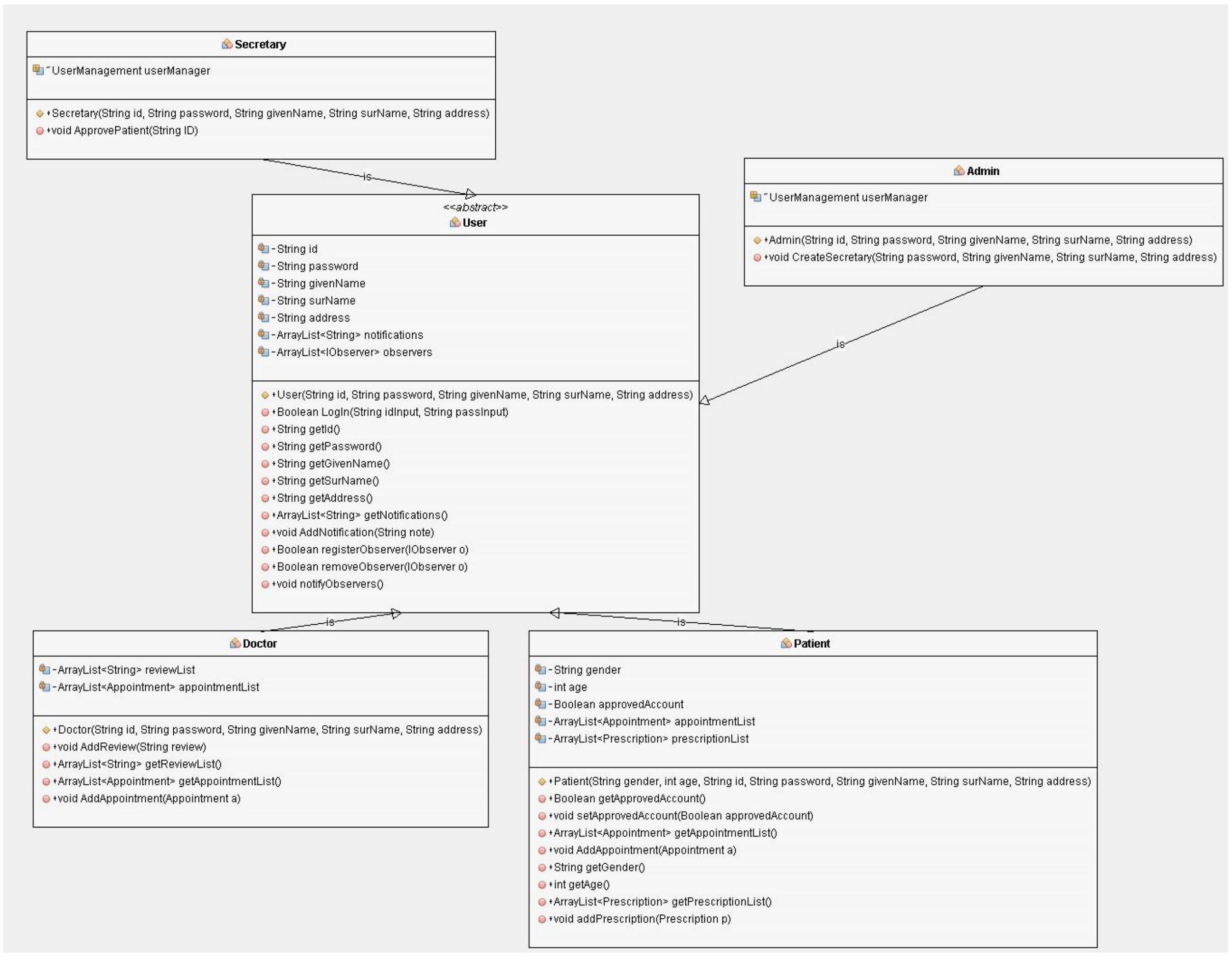
SOFT252 Coursework Reflective Report

Design Process

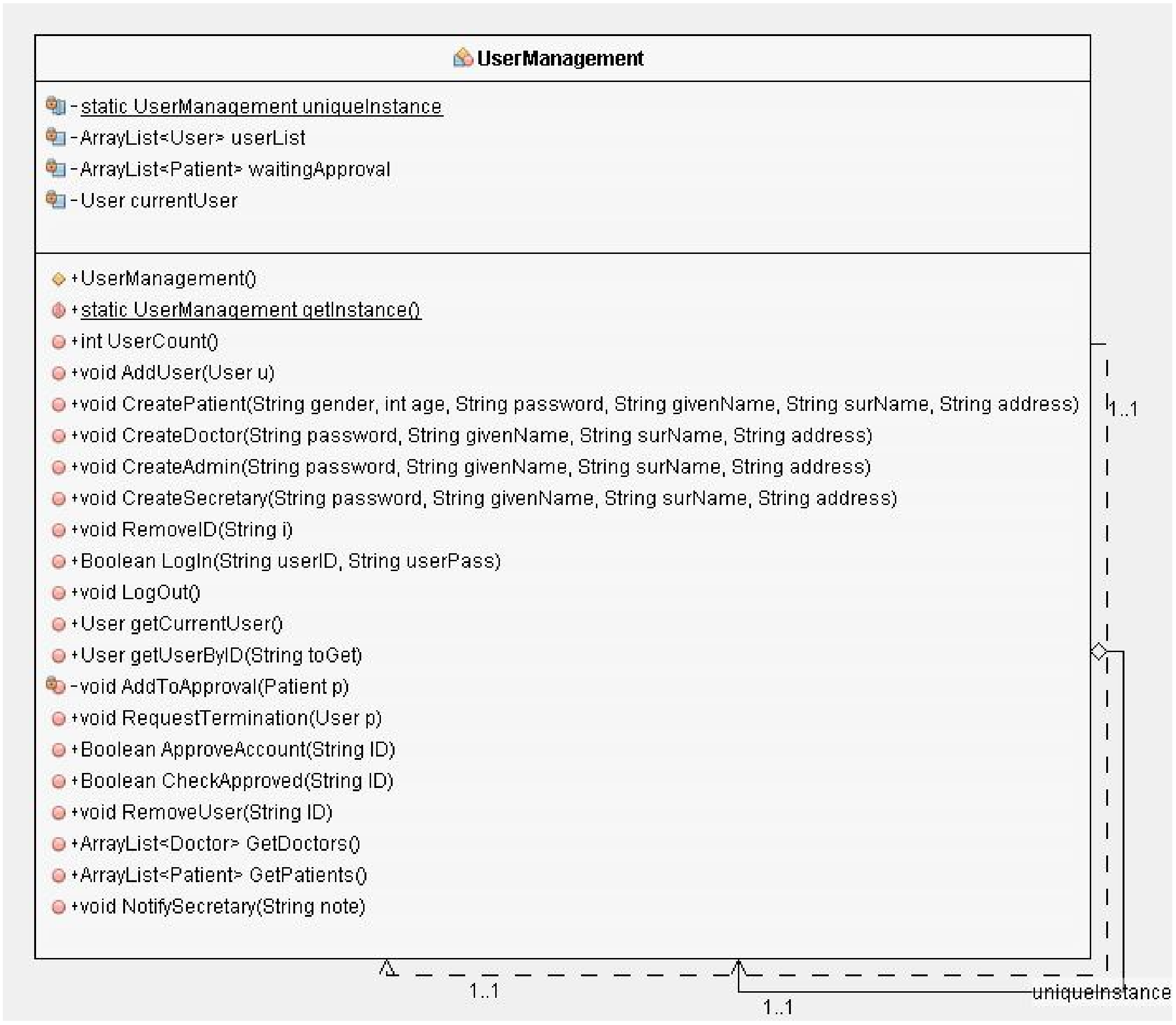
Early UML diagram of the user classes:


252
Avebry Houghton-Vowles | January 10, 2019
Patient
+gender: string
+age: int
+accountApproved: bool
+appointments[]: Appointment
+RequestAccount(string givenName, string surName, string address, int age, string gender, string pass)
+RateDoctor(Doctor doc, string review)
+ViewRatings(Doctor doc)
+RequestAppointment(Doctor doc, Date date)
+ViewHistory()
+ViewAppointment(Appointment a)
+ViewPrescription(Prescription p)
+RequestTermination()
User
+idNum: int
+password: string
+givenName: string
+surName: string
+address: string
+notifications[]: string
+LogIn(int ID, string password)
Appointment
+doctor: Doctor
+patient: Patient
+prescription: Prescription
+time: date
+notes: string
Admin
+CreateAccount(string given, string sur, string address, string pass)
+AddDoctor(string given, string sur, string address, string pass)
+AddSecretary(string given, string sur, string address, string pass)
+ViewRatings(Doctor doc)
+ProvideFeedback(Doctor doc, string feedback)
Prescription
+patient: Patient
+medicine[]: Medicine
+quantity[]: int
+dosage: string
+GetMedicine(Medicine medicine, int quantity)
Medicine
+stock: int
+Restock(int quantity)
+GiveOut(int quantity)
Secretary
+waitingForApproval[]: Patient
+ApprovePatient(Patient)
+ViewRequests()
+CreateAppointment(Patient p, Doctor d, Date date)
+GiveMedicine(Medicine m, quantity q)
+OrderMedicine(Medicine m, quantity q)
+RemovePatient(Patient p)
+ApproveTermination(Patient p)
Doctor
+review[]: string
+ViewAppointment(Appointment a)
+MakeNote(Appointment a, string note)
+PatientHistory(Patient p)
+CreateAppointment(Patient p, Date date)
+Prescribe(Patient p, Medicine m, int quantity, string dosage)
+CreateMedicine(string name)

Final UML diagram:

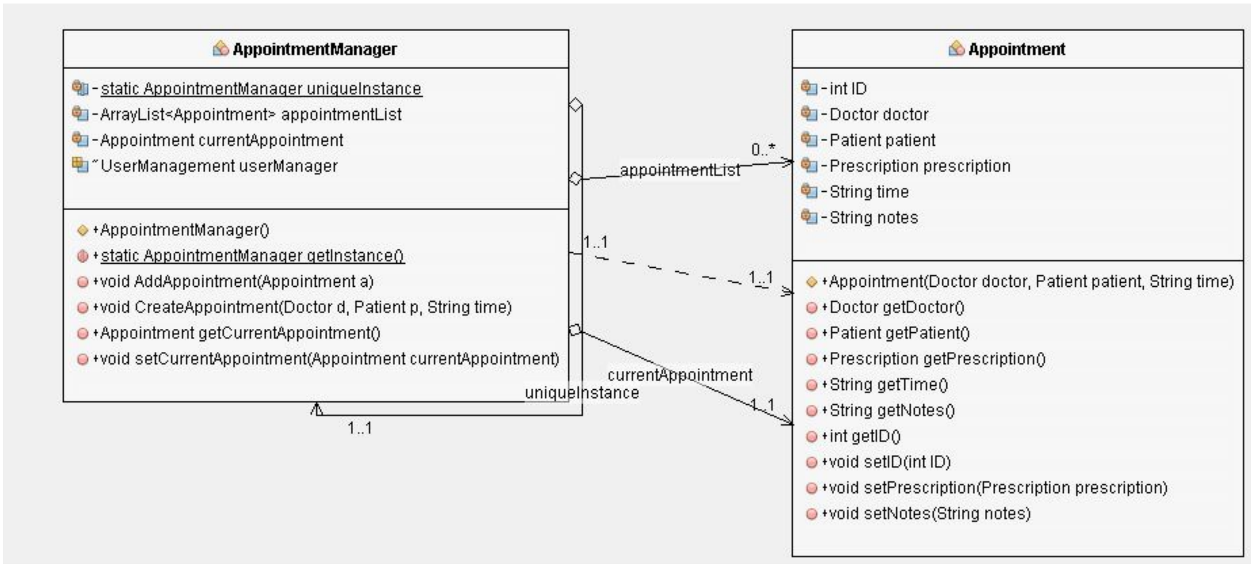

Secretary
~UserManagement userManager
+Secretary(String id, String password, String givenName, String surName, String address)
+void ApprovePatient(String ID)
is
Admin
~UserManagement userManager
+Admin(String id, String password, String givenName, String surName, String address)
+void CreateSecretary(String password, String givenName, String surName, String address)
is
<<abstract>>
User
-String id
-String password
-String givenName
-String surName
-String address
-ArrayList<String> notifications
-ArrayList<IObserver> observers
+User(String id, String password, String givenName, String surName, String address)
+Boolean LogIn(String idInput, String passInput)
+String getId()
+String getPassword()
+String getGivenName()
+String getSurName()
+String getAddress()
+ArrayList<String> getNotifications()
+void AddNotification(String note)
+Boolean registerObserver(IObserver o)
+Boolean removeObserver(IObserver o)
+void notifyObservers()
is
is
Doctor
-ArrayList<String> reviewList
-ArrayList<Appointment> appointmentList
+Doctor(String id, String password, String givenName, String surName, String address)
+void AddReview(String review)
+ArrayList<String> getReviewList()
+ArrayList<Appointment> getAppointmentList()
+void AddAppointment(Appointment a)
Patient
-String gender
-int age
-Boolean approvedAccount
-ArrayList<Appointment> appointmentList
-ArrayList<Prescription> prescriptionList
+Patient(String gender, int age, String id, String password, String givenName, String surName, String address)
+Boolean getApprovedAccount()
+void setApprovedAccount(Boolean approvedAccount)
+ArrayList<Appointment> getAppointmentList()
+void AddAppointment(Appointment a)
+String getGender()
+int getAge()
+ArrayList<Prescription> getPrescriptionList()
+void addPrescription(Prescription p)

User Management UML:

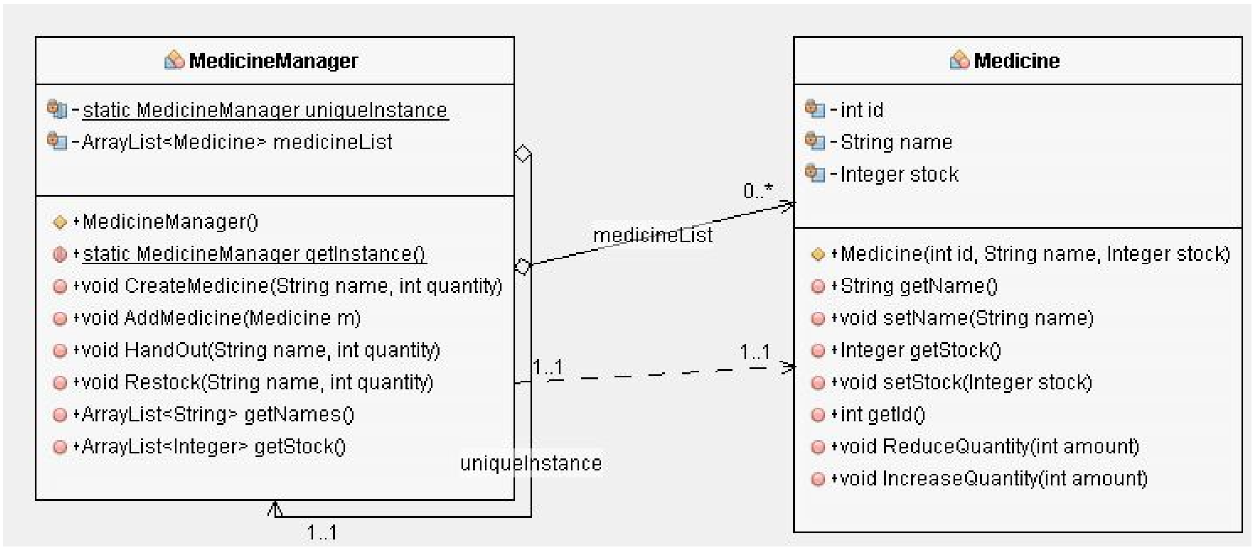


```
UserManagement

-static UserManagement uniqueInstance
-ArrayList<User> userList
-ArrayList<Patient> waitingApproval
-User currentUser

+UserManagement()
+static UserManagement getInstance()
+int UserCount()
+void AddUser(User u)
+void CreatePatient(String gender, int age, String password, String givenName, String surName, String address)
+void CreateDoctor(String password, String givenName, String surName, String address)
+void CreateAdmin(String password, String givenName, String surName, String address)
+void CreateSecretary(String password, String givenName, String surName, String address)
+void RemoveID(String i)
+Boolean LogIn(String userID, String userPass)
+void LogOut()
+User getCurrentUser()
+User getUserByID(String toGet)
-void AddToApproval(Patient p)
+void RequestTermination(User p)
+Boolean ApproveAccount(String ID)
+Boolean CheckApproved(String ID)
+void RemoveUser(String ID)
+ArrayList<Doctor> GetDoctors()
+ArrayList<Patient> GetPatients()
+void NotifySecretary(String note)
```

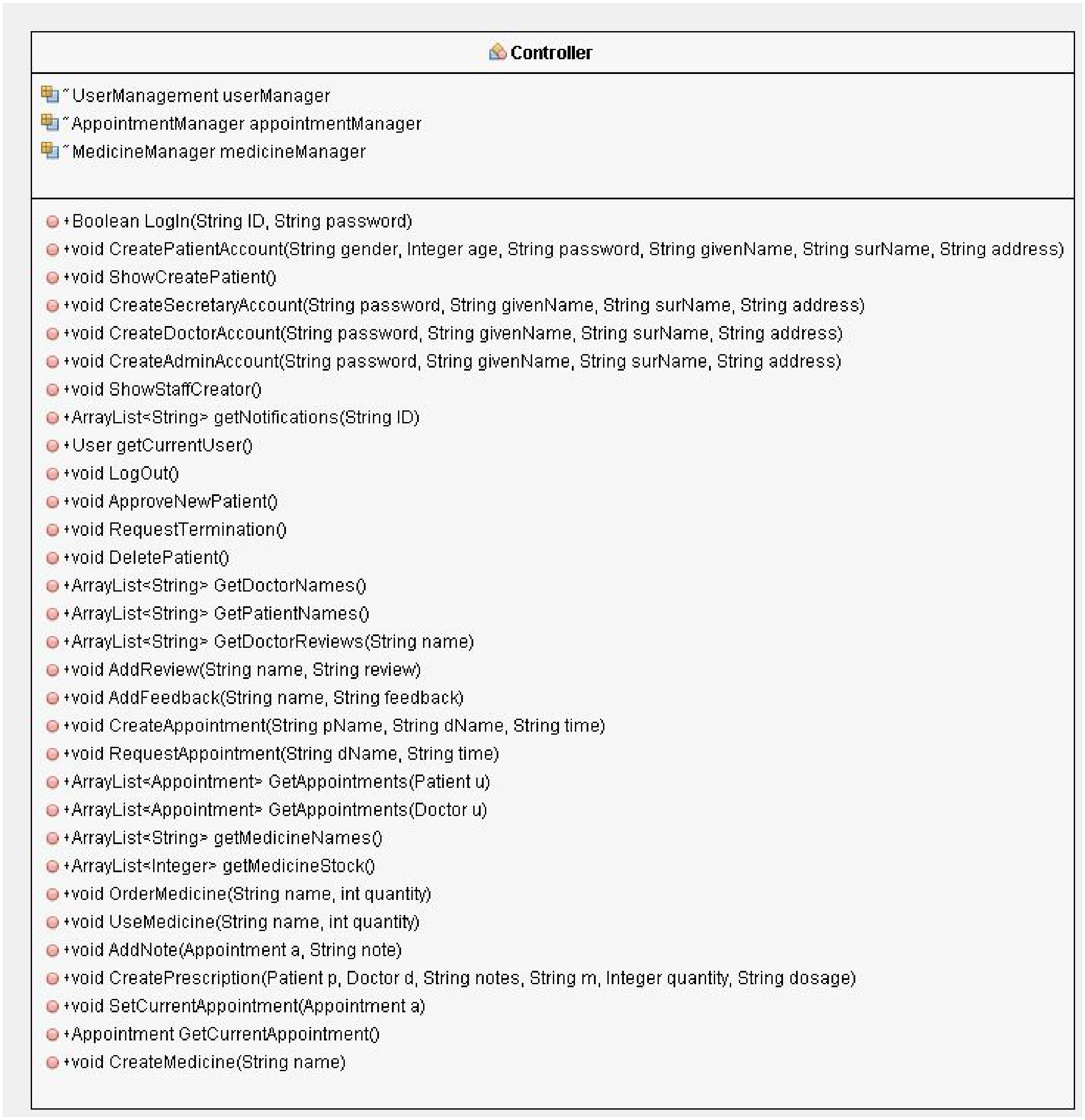
Appointment Manager UML:


AppointmentManager
-static AppointmentManager uniqueInstance
-ArrayList<Appointment> appointmentList
-Appointment currentAppointment
~UserManagement userManager
+AppointmentManager()
+static AppointmentManager getInstance()
+void AddAppointment(Appointment a)
+void CreateAppointment(Doctor d, Patient p, String time)
+Appointment getCurrentAppointment()
+void setCurrentAppointment(Appointment currentAppointment)
Appointment
-int ID
-Doctor doctor
-Patient patient
-Prescription prescription
-String time
-String notes
+Appointment(Doctor doctor, Patient patient, String time)
+Doctor getDoctor()
+Patient getPatient()
+Prescription getPrescription()
+String getTime()
+String getNotes()
+int getID()
+void setID(int ID)
+void setPrescription(Prescription prescription)
+void setNotes(String notes)
0..*
appointmentList
1..1
1..1
currentAppointment
uniqueInstance
1..1
1..1

Medicine Manager UML:


MedicineManager
-static MedicineManager uniqueInstance
-ArrayList<Medicine> medicineList
+MedicineManager()
+static MedicineManager getInstance()
+void CreateMedicine(String name, int quantity)
+void AddMedicine(Medicine m)
+void HandOut(String name, int quantity)
+void Restock(String name, int quantity)
+ArrayList<String> getNames()
+ArrayList<Integer> getStock()
Medicine
-int id
-String name
-Integer stock
+Medicine(int id, String name, Integer stock)
+String getName()
+void setName(String name)
+Integer getStock()
+void setStock(Integer stock)
+int getId()
+void ReduceQuantity(int amount)
+void IncreaseQuantity(int amount)
0..*
medicineList
1..1
1..1
uniqueInstance
1..1

Controller UML:



```
Controller

~UserManagement userManager
~AppointmentManager appointmentManager
~MedicineManager medicineManager

+Boolean LogIn(String ID, String password)
+void CreatePatientAccount(String gender, Integer age, String password, String givenName, String surName, String address)
+void ShowCreatePatient()
+void CreateSecretaryAccount(String password, String givenName, String surName, String address)
+void CreateDoctorAccount(String password, String givenName, String surName, String address)
+void CreateAdminAccount(String password, String givenName, String surName, String address)
+void ShowStaffCreator()
+ArrayList<String> getNotifications(String ID)
+User getCurrentUser()
+void LogOut()
+void ApproveNewPatient()
+void RequestTermination()
+void DeletePatient()
+ArrayList<String> GetDoctorNames()
+ArrayList<String> GetPatientNames()
+ArrayList<String> GetDoctorReviews(String name)
+void AddReview(String name, String review)
+void AddFeedback(String name, String feedback)
+void CreateAppointment(String pName, String dName, String time)
+void RequestAppointment(String dName, String time)
+ArrayList<Appointment> GetAppointments(Patient u)
+ArrayList<Appointment> GetAppointments(Doctor u)
+ArrayList<String> getMedicineNames()
+ArrayList<Integer> getMedicineStock()
+void OrderMedicine(String name, int quantity)
+void UseMedicine(String name, int quantity)
+void AddNote(Appointment a, String note)
+void CreatePrescription(Patient p, Doctor d, String notes, String m, Integer quantity, String dosage)
+void SetCurrentAppointment(Appointment a)
+Appointment GetCurrentAppointment()
+void CreateMedicine(String name)
```

Design Choices

The class UserManagement stores a list of all users in the program, and handles most operations related to them. As this would need to be accessed by most classes in the program, I decided to use the Singleton design pattern for it. This allows the same instance of UserManagement storing the list of users to be accessed everywhere. As one instance of the classes AppointmentManager and MedicineManager are also required throughout the program, they also utilise the Singleton pattern.

As the notification output should subscribe to a user and update whenever a new notification is added, it should be built using the Observer pattern.

Successes

In the view, the GUI for each type of user is constructed in a separate JFrame. This helps prevent overlap between the permissions and methods available to each type, and means that each user will only see the options relevant to them. Having separate frames for each major task also means that frames can be shared between functions if necessary, such as viewing and writing doctor reports for both admins and patients.

Another success is the implementation of the userManager class. By gathering most of the user-related methods and lists into one class, then applying the Singleton pattern to it, these functions and data can easily be accessed throughout the program in other packages without needing multiple imports. The Singleton pattern also guarantees that the main storage lists will not be overwritten.

When an input with limited options would have to be entered into a JFrame, such as selecting patients and doctors to be assigned an appointment, the input is done via a Combo Box instead of an open text field. When the combo box model is properly assigned, this eliminates the risk that the user will enter an invalid input.

Shortcomings

A shortcoming in the submitted work is that I feel the implementation of MVC is lacking. This is due to the fact that the View still contains its own methods, instead of passing everything to the Controller. An example of this is in the JFrame DoctorForm, which has its own methods to retrieve the list of notifications instead of handing it off to the controller. This could be remedied by better planning out the functions in the Controller.

Another shortcoming in the code is that the notifications for a user are only currently checked when the user logs in, or directly adds a new notification. If a user were to receive a new notification while logged in, it would not display until next login. This could be solved by properly implementing the Observer pattern with some kind of notification manager, which I failed to do due to poor planning and time constraints.
The submitted program is also incapable of saving data, as I was unable to figure out how to get the Serialiser working in time. Again, this is due to poor planning and time management, and could be solved if I had fewer pieces of coursework to do at the time.

Project Management Link:

https://github.com/avebryhv/SOFT252Coursework