University of Plymouth

**SOFT252:**Object-oriented Software Engineering

with Design Patterns

Reflective report

*Student number: 10576156*

Word count

Report: 792

January 16th, 2019

**The address to the GitHub repository:**

<https://github.com/Dawidgrad/Patient-Management-System>

**The design process:**

The design process started from reading the specification and looking for nouns to identify potential classes in the system. The classes that were produced from this process were then created in UML diagram software to get a general design of the system with the connections between classes.

Next, the design was split into the smaller subsystems that each had their own responsibility. At this point, the classes only had most crucial methods included on the diagram.

After that design step was done, the focus was switched to each of the smaller systems, starting from account creation subsystem. The list of methods for each class were added during that step. After finishing the design of the subsystem, the implementation was produced.

After the implementation of the subsystem, the focus was switched again to the detailed design of the next subsystem and repeating the previous steps until the product was finished.

**Design choices:**

Throughout the implementation of the software 4 patterns were used, 3 for the model and 1 architecture pattern:

1. MVC pattern – used as a general structure of the program, the View and Model were separated, and Controllers were implemented to convey the information between them.
2. Factory pattern – used to create medicine and account classes. As the class types had to be decided at a run time, such as Tablet, Capsule and Liquid medicine, the best solution for that was a Factory pattern. This also leaves a possibility to extend the implementation with other classes in the future.
3. Singleton pattern – this pattern was used to store the information and act as a source of it for other classes. The pattern allowed for a simple Serialisation just by adding every singleton to the list and serialising it.
4. Observer pattern – the pattern was used for appointment notification. The patient would be subscribed to the Appointment request that they have created, and when its status changes, the patient would be notified.

**Degree to which design meets good design criteria:**

The solution meets good design criteria for the most part, especially the model part of it. There are some shortcomings though, which will be discussed in the next section.

The implementation uses various patterns, which allow extension and easier management of the code. Additionally, the classes were properly separated into smaller parts and the functionalities were delegated between them e.g. Prescription being split into Notes and PrescriptionMedicine. That also allows for future extension without having to modify the existing methods. For some of the classes e.g. Account and Medicine, abstraction was used to store shared fields and methods.

As the design was split into smaller subsystems, it is possible to use each of them separately, which promotes low coupling. Furthermore, the enumerator classes were used where possible to avoid semantic errors.

**Shortcomings:**

While all of the functionality was included, and the design patterns were used appropriately, the solution could use some improvements. The biggest challenge for that solution was the time limit which means that some of the solutions in the implementation are not the most optimal.

The biggest part of the project that could be designed better is the GUI. The main issue is the code redundancy such as methods refreshing and populating content in JList, which are separately implemented for every view. Additionally, navigation (Back and Log out) listeners could have been abstracted.

There was an attempt to incorporate state pattern into switching views, which would involve abstracted views and controllers, but unfortunately too many problems were encountered and after some time, the simpler solution was created.

Another shortcoming that could be addressed in the future is better error handling. Custom exceptions could have been used to provide more meaningful error messages to the users.

**Additional comments:**

Some of the elements on the UML diagrams were omitted to keep them readable and not too cluttered. The elements that are not included are:

* Getter and setter methods, instead the fields are included on the diagram.
* Parameters that methods use, as some of the methods had more than 5 of them.
* Constructors, as they don’t provide any information on the diagram without the parameters.

Furthermore, as advised, the UML diagram has been split into several smaller subsystems, to further increase readability. The UML diagrams are included in Appendix 1, as well as the folder in the main submission folder.

One additional functionality was implemented for the patient to change their password.

Finally, the JUnit tests are not fully implemented as they are missing for the singleton classes. This is due to limited time and rushed implementation of the tests at the end of development process instead of the beginning. In the future this approach could be switched around to properly test the functionalities.

**Appendix 1: UML Diagrams**

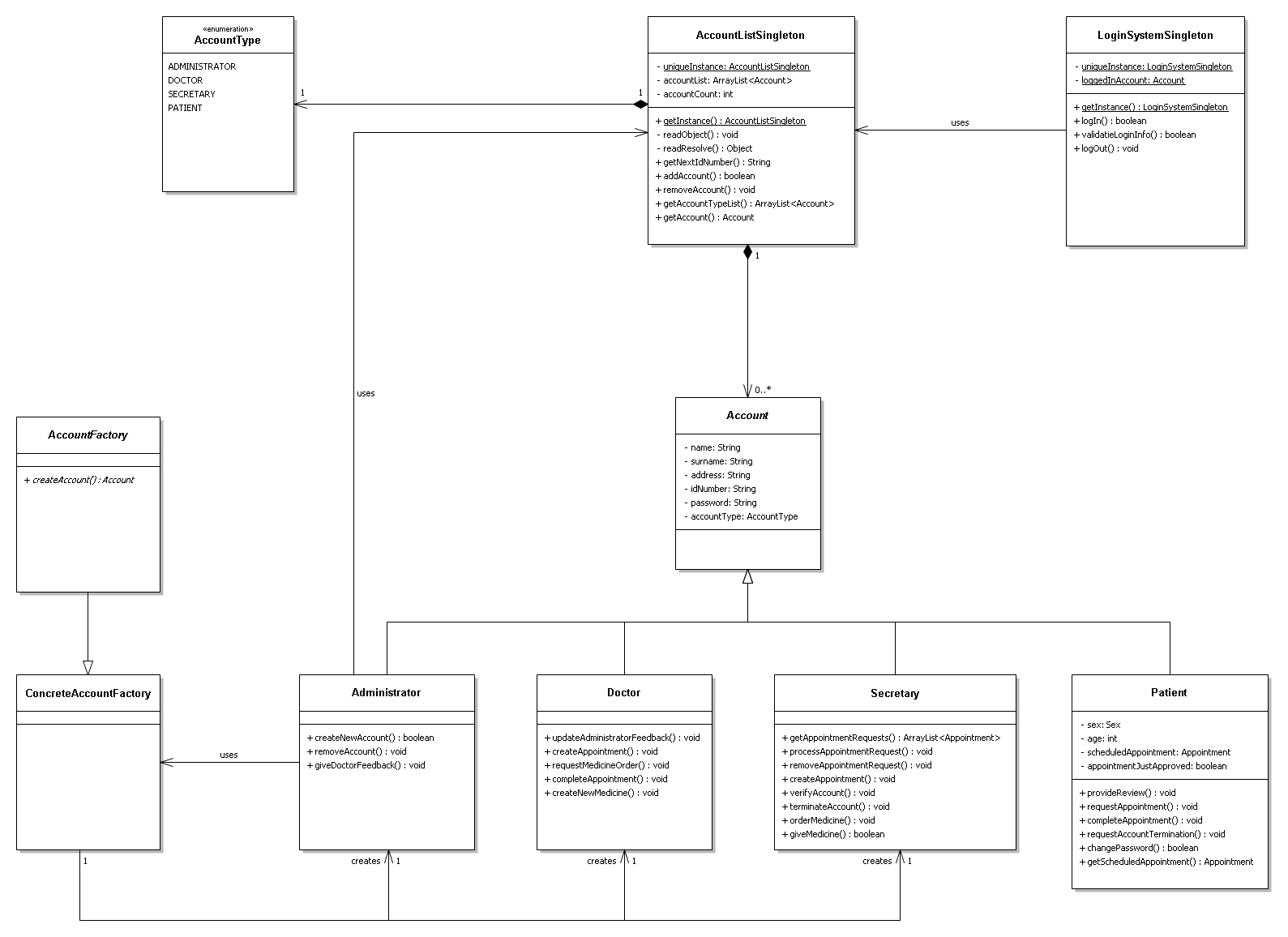


Figure 1 Account subsystem

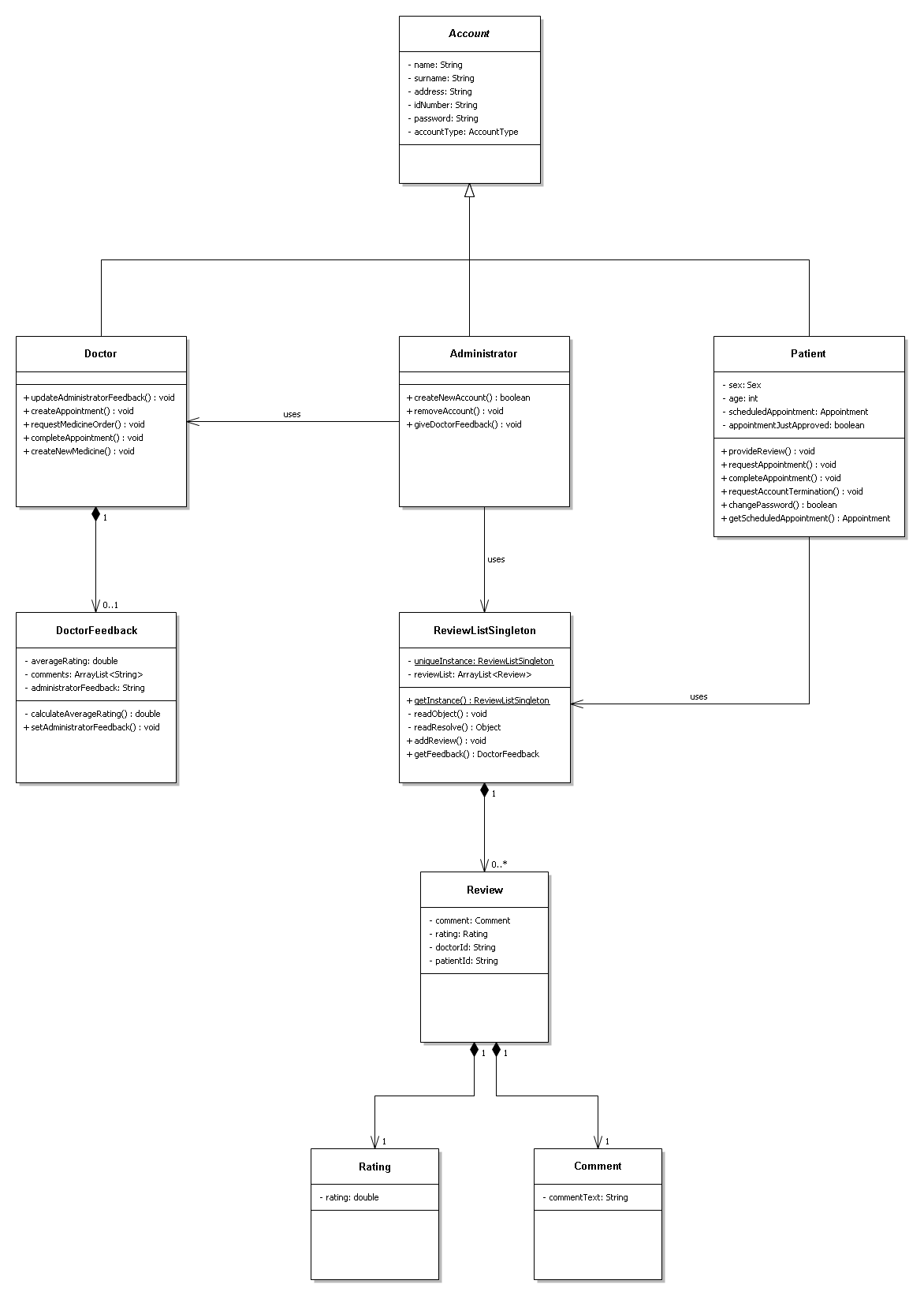


Figure 2 Review and feedback subsystem

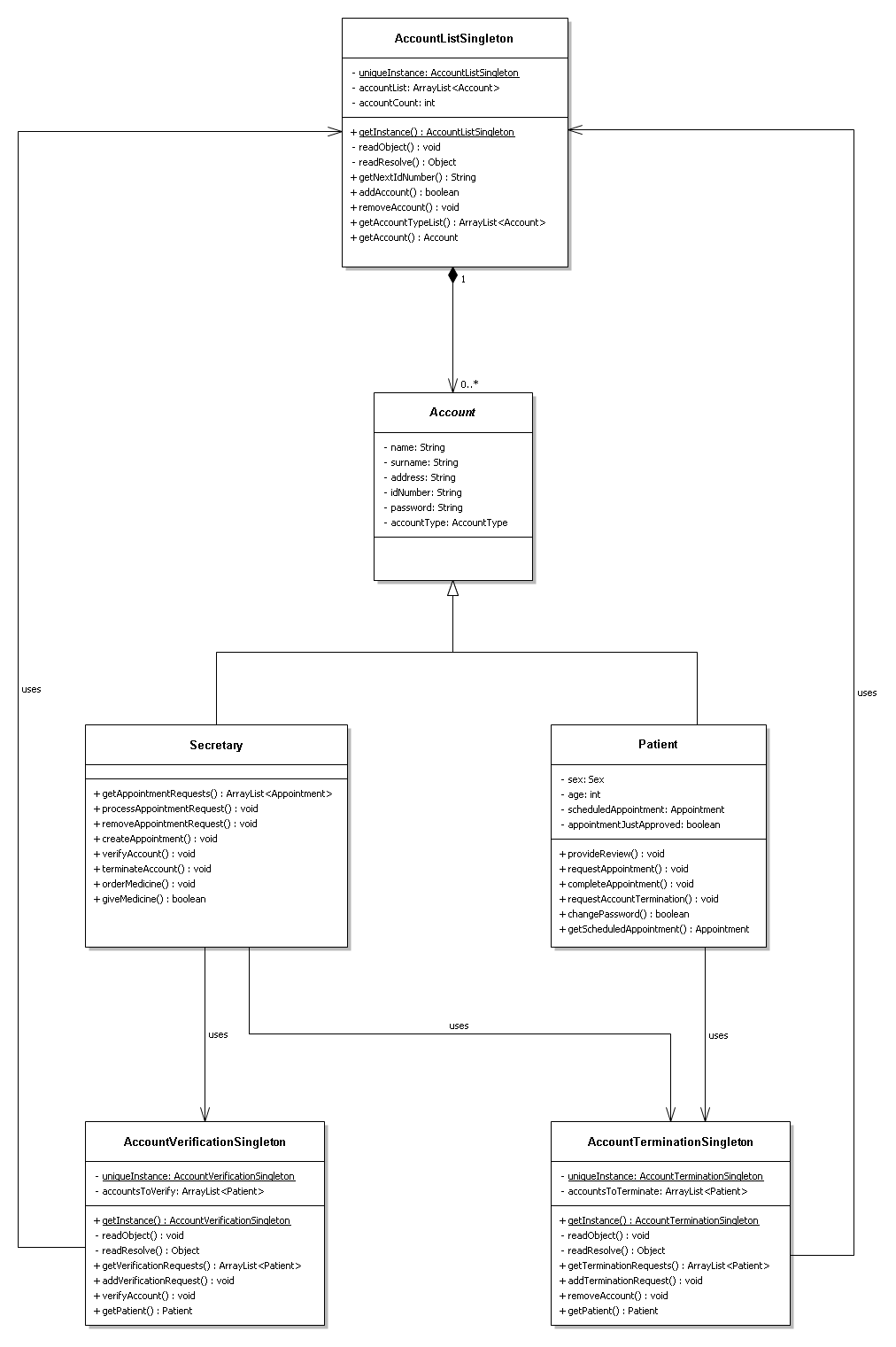


Figure 3 Patient account management system

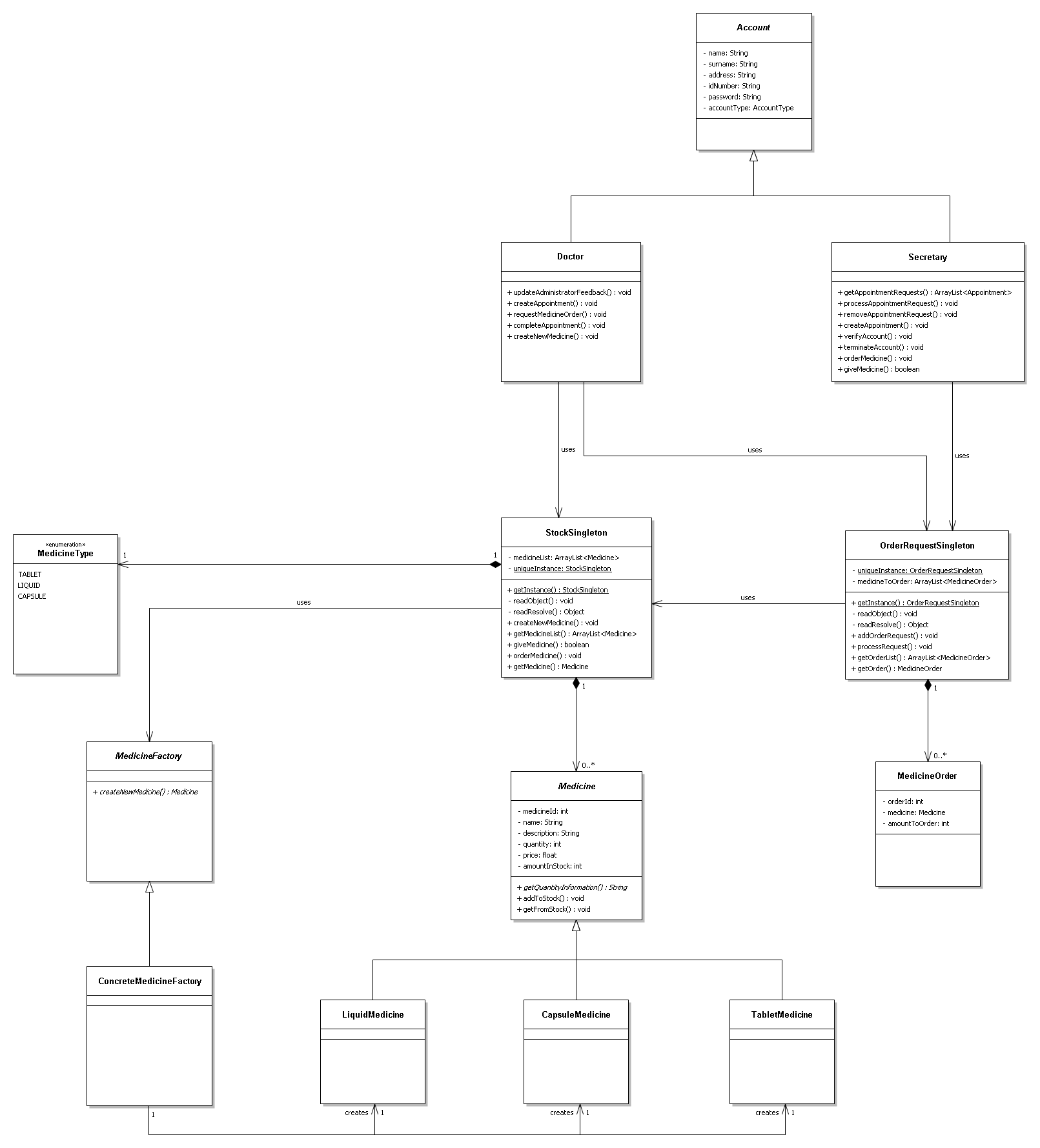


Figure 4 Medicine subsystem

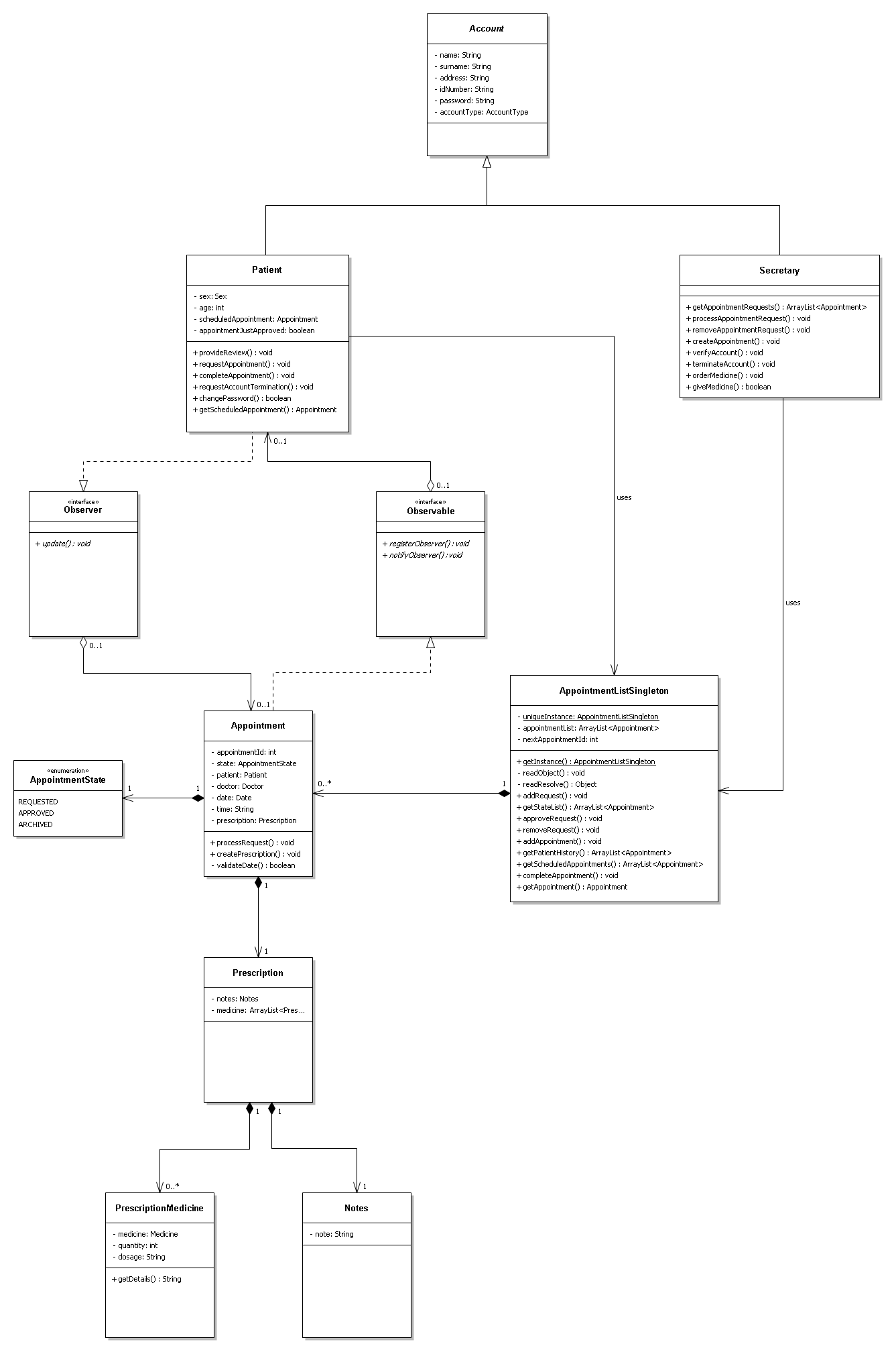


Figure 5 Appointment subsystem