Nick Pierce

4/19/2021

CSCI 507

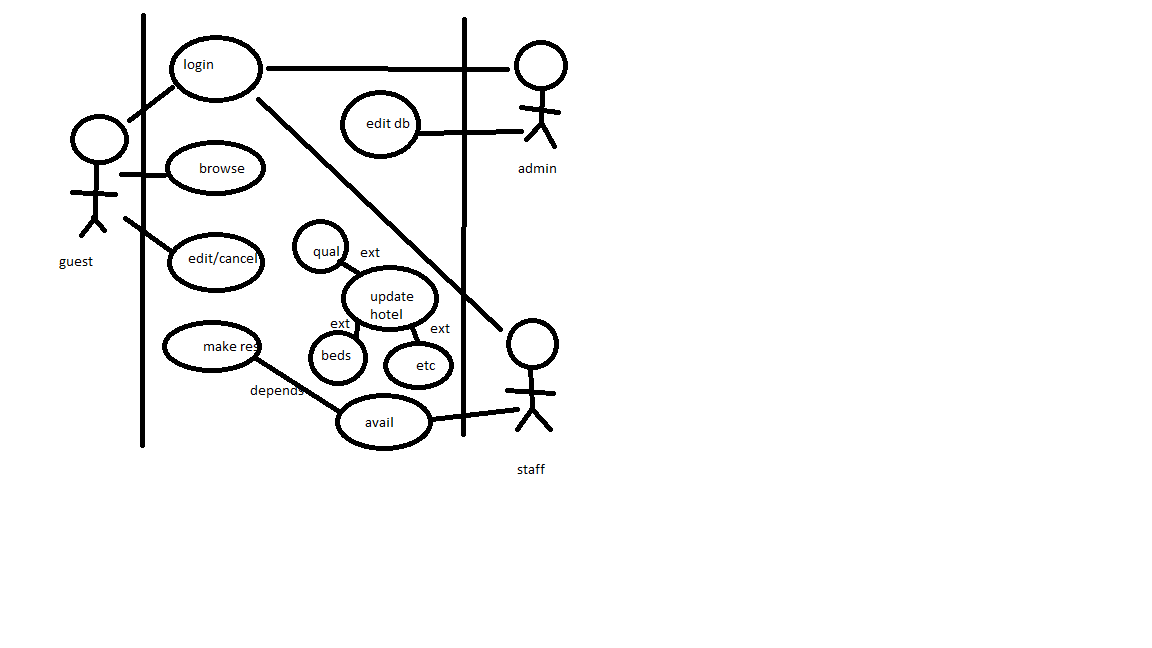
Assignment 4 Report

Introduction

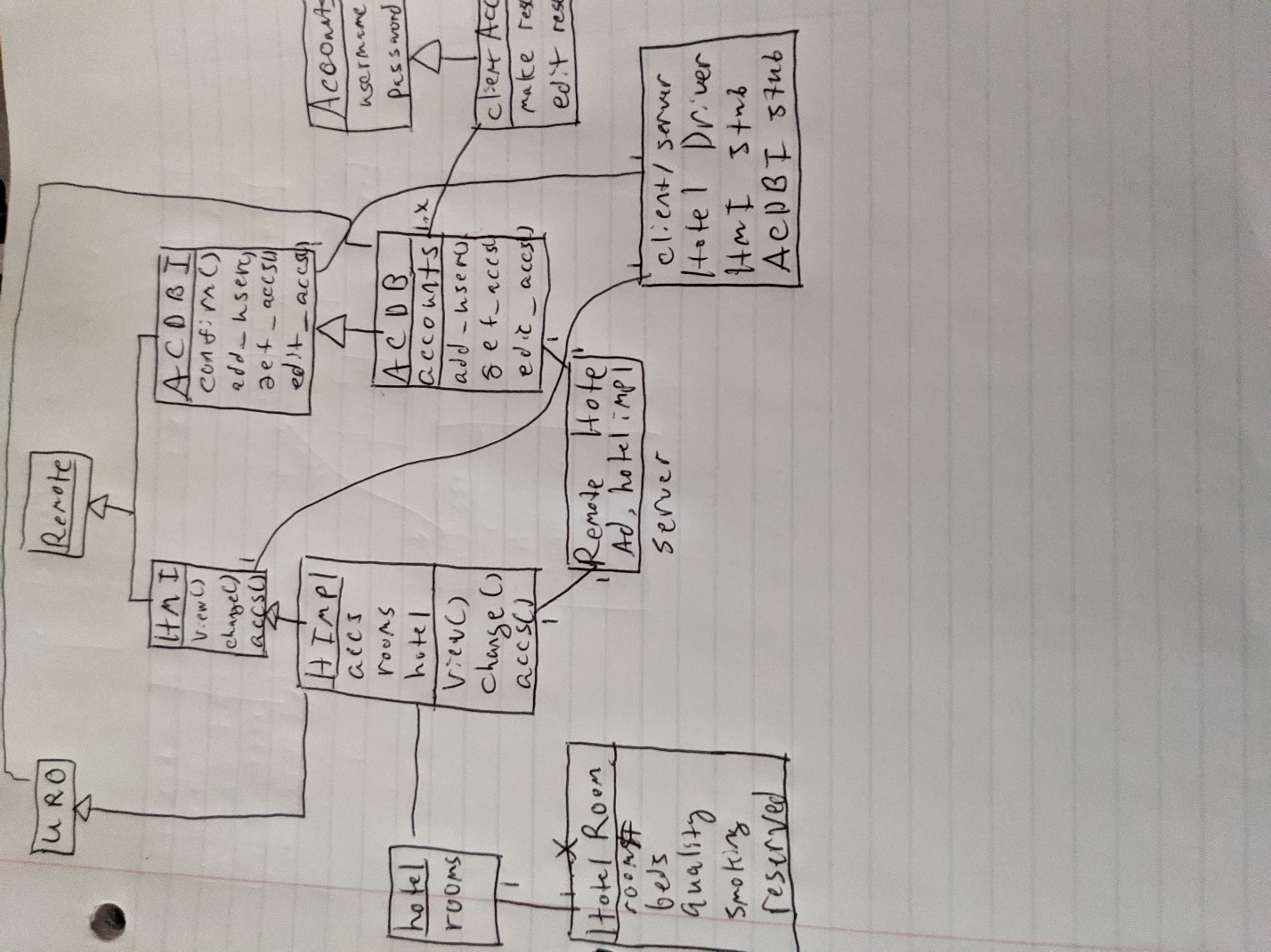
For assignment 4, we were tasked with building upon our original design by implementing the command, template, and factory patterns. I use two hierarchies of views and dispatchers to adhere to the requirements of the command and template patterns. I then use an abstract factory hierarchy for the factories, one concrete one builds dispatchers, and the other showing how to reuse the hierarchy, builds different hotels.

Use-Cases

In terms of the use case diagram, it is essentially the same, but I refactored it to show the distinct types of users the system will be using. For now, I have the administrators editing the database, adding users/deleting users, and looking at accounts. And I have the hotel staff accounts editing the parts of the hotel and rooms when they desire, the normal hotel guests have not really changed.

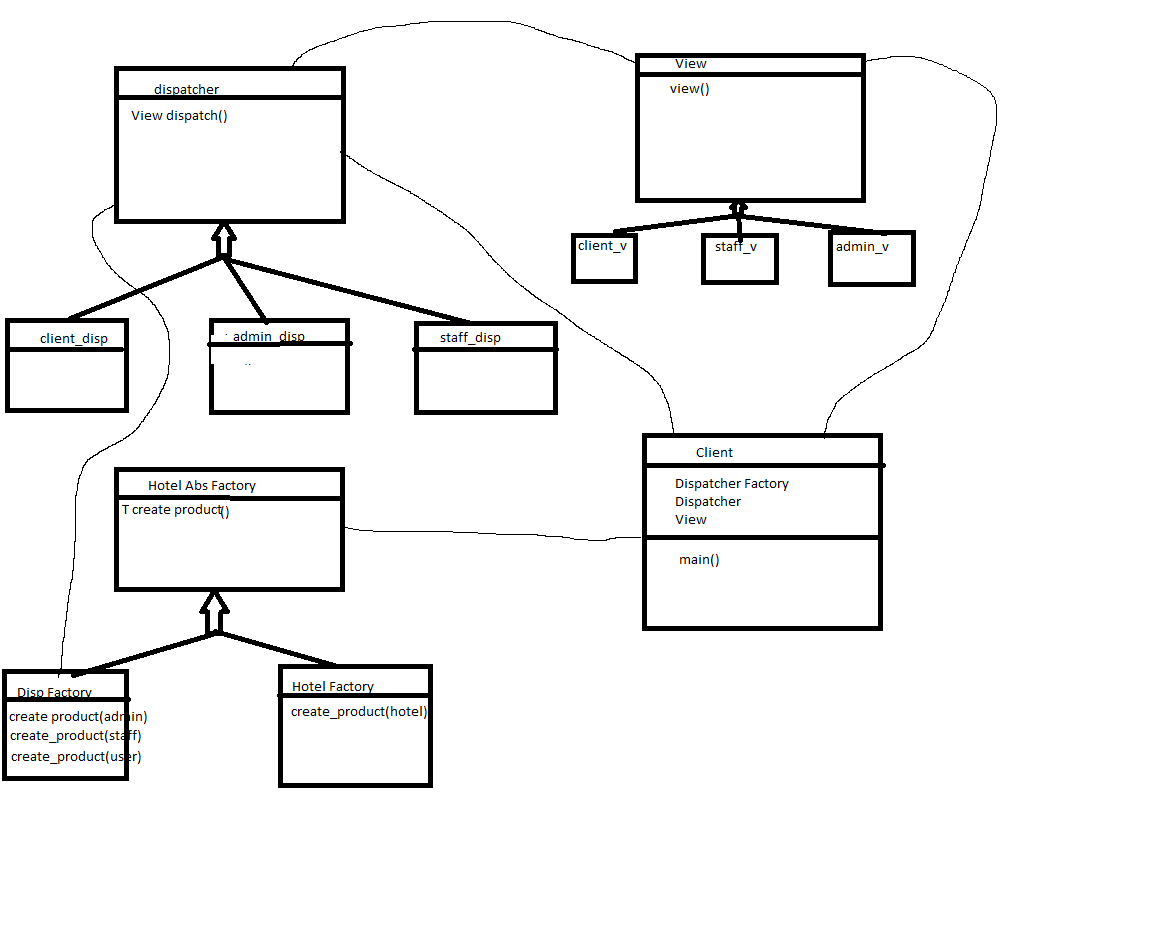


Class Diagram



New additional classes

So my class model added a few classes/hierarchies to implement the patterns requested in the assignment.



All other classes are valid, so nothing really changes there, but the main thing is adding two hierarchies and a factory to the new class diagram. So I use the command and template patterns to create the dispatchers and views that work together to show the front work based on whoever logs in. So my classes are based on the templated polymorphic hierarchies, then each has a specialized “dispatch” or “view” method that is core to the command pattern. Many times the command pattern will blanketly be called “execute” but I chose to use a more specific name as it helped me understand my code base better.

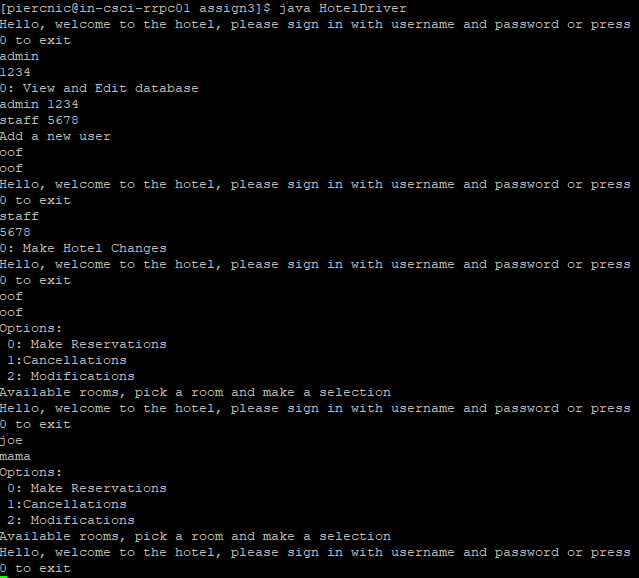
As for the abstract/factory I chose to have the abstract factory only have a templated create method so it could be reused multiple times. For my first concrete factory, I have the dispatcher factory. So in the client, based on who logs in the factory creates the dispatcher object and then the client runs the dispatch method, then the view method. I use the client as the receiver and executor to make this simpler, but a more fleshed out command pattern setup would have those objects as well. I also have a hotel factory showing how the abstract factory can be used to create other products, it simply creates and returns a hotel.

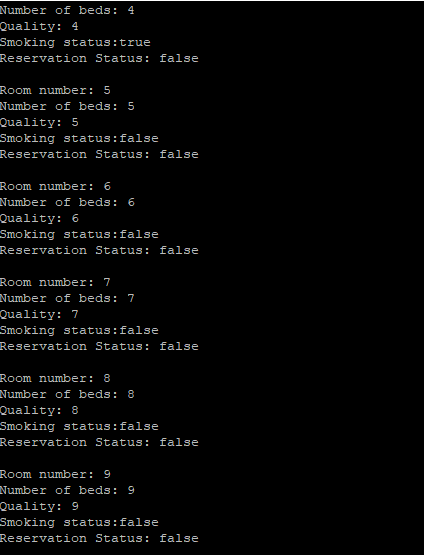
In terms of the executing section, nothing really changed so I just took pretty much the same screenshots again.

Domain Model

I didn’t think it was entirely needed to remake an entire domain model diagram as most of my classes are just wrapper classes for data. Most of the actual functionality is driven by the two RMI servers. The main management server that management interacts with, updates the hotel and all of the users so this is where most of the behavioral/logical changes take place. For example, the management might want to view/update accounts so they’ll initiate a change on the ACDB from the management server. The other main components is the client server. This main server runs the main UI components that the clients and managers interact with. This server establishes a connection to the bound objects created by the remote hotel server and uses this as the persistent connection.

Proof of Running





Not shown here, but I do check for available accounts and if it does not exist it spits you back out to the main menu, you can see this functionality in the account database class.