Hotel Reservations

Project Part 6: Final Report

Monte Anderson, Nathan Carmine, Nolan Cretney, Jackson Mediavilla

Implemented Features

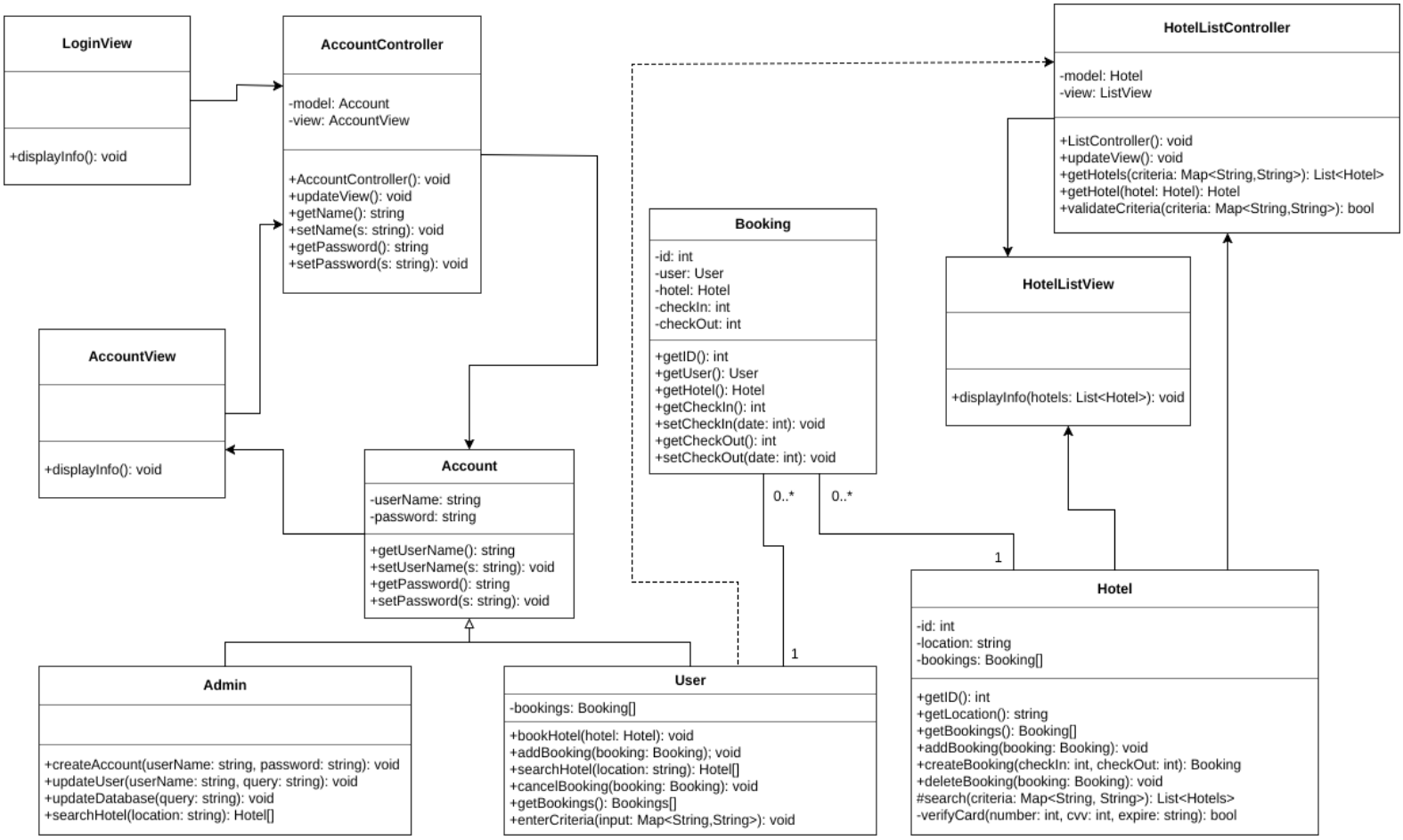
ID	Requirement	Actor
UR-001	User should be able to book a reservation	SearchUser
UR-002	User should be able to cancel a reservation	BookedUser
UR-003	User should be able to pay with credit card only	SearchUser, BookedUser
UR-005	User should be able to compare different hotels to each other	SearchUser, BookUser
UR-008	User should be able to search for hotels and sort by name, location, price range, or ratings*	SearchUser, Admin, BookedUser
BR-001	If a user is paying by card, we can only accept MasterCard or Visa	SearchUser
NFR-001	If an admin is editing DB info, the downtime of the DB should be less than 1 hour	Admin
NRF-002	When a user goes to book a hotel room, it should not take less than 10 seconds before timing out	SearchUser
NRF-004	When a user tries to search and compare hotels, they should be able to do so without logging in	SearchUser, BookedUser, Admin

Unimplemented Features

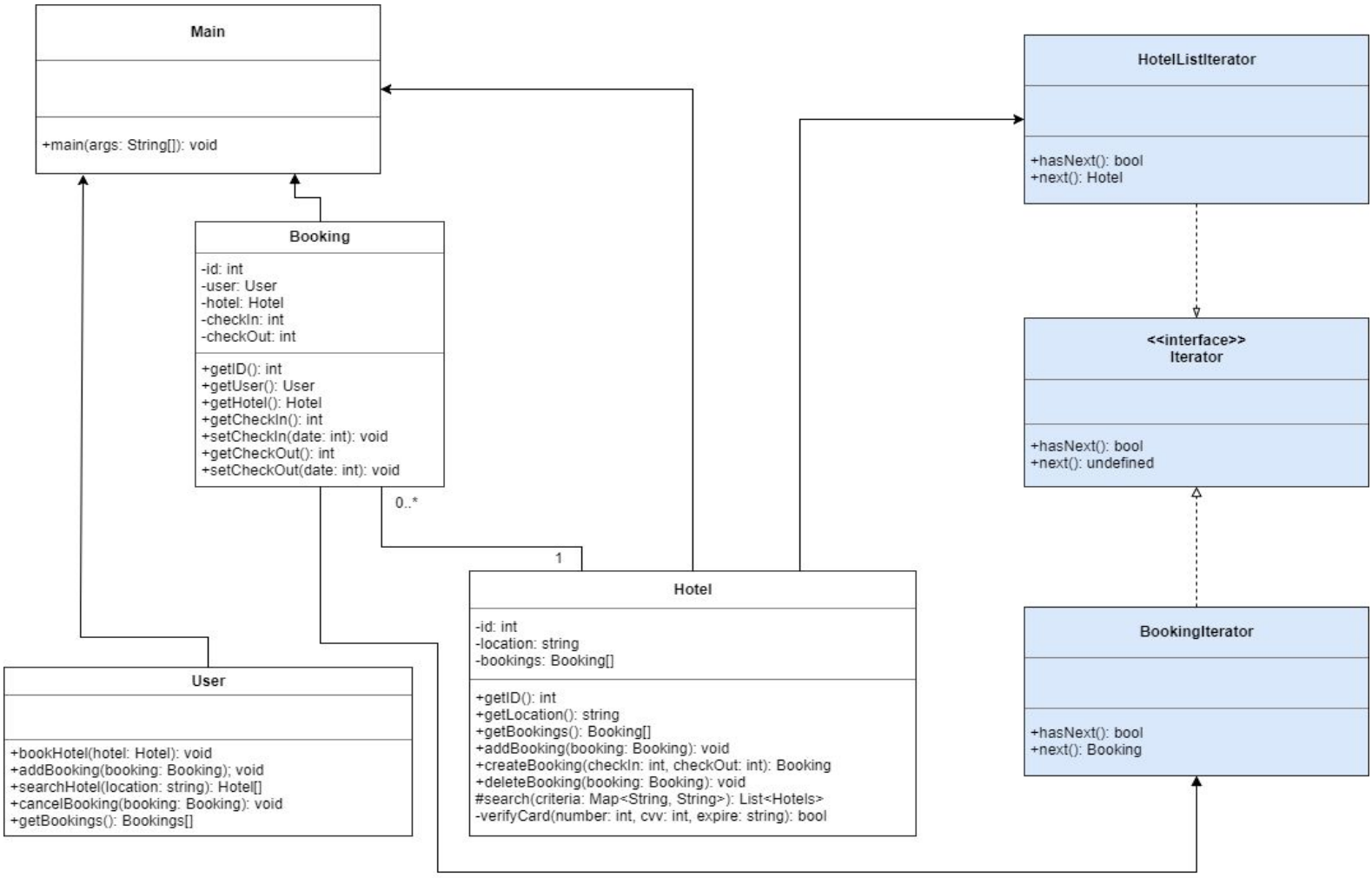
ID	Requirement	Actor
UR-004	User should be able to see the amenities of the hotel they are looking at	SearchUser, BookedUser
UR-006	User should be able to change userinfo or DB info	Admin
UR-007	User should be able to create an account that has their booking information as well as credit information, etc.	SearchUser, BookedUser
BR-002	If a user is paying by card, need to validate card info with bank	SearchUser, BookedUser
BR-003	If a hotel requires a minimum amount of days to make a reservation, that information needs to displayed to the user	SearchUser, BookedUser,
BR-004	The information that displayed for the rates comes directly from the hotels themselves	SearchUser, BookedUser
NRF-003	When user tries to perform an action on the app and they have not logged in yet, they should be prompted to	SearchUser, BookedUser, Admin
NRF-005	When an admin goes to edit a DB, a backup is instantly created	Admin

Class Diagrams

Part 2 Class Diagram



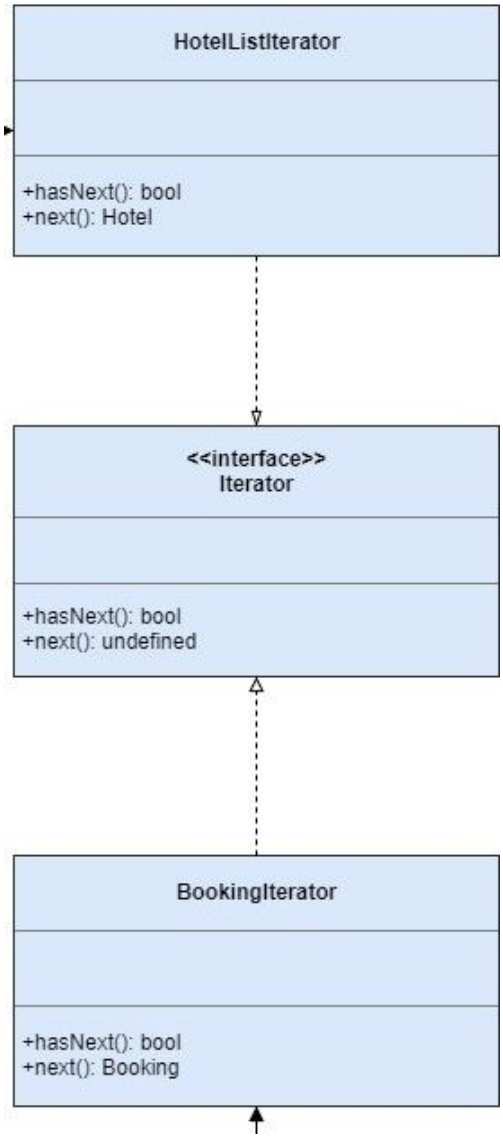
Final Class Diagram



Our final class diagram differs from our Part 2 diagram for two reasons. First, we implemented the Iterator design pattern to aid in Hotel and Booking list iterations. This added the Iterator, BookingIterator, and HotelIterator classes to our design. Second, we did not fully implement all of the features listed in the initial design. The Account and Admin classes were not included because we only implemented User functionality. The AccountController, LoginView, AccountView, and HotelListView classes are not in the final design because they deal with the user interface, which was not implemented.

Design Pattern

Iterator



According to the Gang of Four, iterators “provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.” We implemented this design pattern to reduce the number of list iterations. Using the initial design, we would have had to create a discrete list of all hotel locations separate from hotel objects and go through that list (again) to find matches. This results in two list iterations. Using the Iterator design pattern, the system iterates through hotel objects, gets their locations, and adds the results to another list that is returned. This only requires a single iteration through the hotel objects.

Reflection

Through creating, designing, and implementing an object-oriented system, we gained invaluable knowledge about the process of program analysis and design. First, we learned how important it is to design the system well prior to writing code. A good design facilitates adding features, changing requirements, makes the system easy to understand and implement, and promotes system efficiency.

We all learned how to program in Java in a much more object oriented way, including using different classes, constructions, objects, private and public methods, and overriding methods, to create an object oriented application. Learning how objects interact with each other in an object oriented program was also kind of a learning curve, as some of our initial ideas would not work in an object oriented pattern as they would in a non-object oriented way. As mentioned, our primary design pattern we implemented was iterators, and this increased our programed efficiency by two times when iterating through hotel objects to query matches to user input. Adhering to proper object-oriented design also afforded us the ability add and remove features and shape classes to our changing requirements at ease. Had we practiced bad programming design principles, this would have posed much more of a challenge for us.

We also learned how to work efficiently as a team and delegate tasks to accomplish checkpoints, and through the calming power of jazz when we were feeling *Kind of Blue* (we actually did listen to jazz as a team). Pair/group programming on a large TV with every member was a large part of this project, which also allowed us to learn to work as a team and solve problems in the project together. This also enabled us to all learn how to work through the same problems.