George Costas
Hostel21 Memo

Project Status: Functioning, but unfinished
Number of Hours Required: 35-50 hours
Hours spent preparing program: 5

Solved Requirement Issues:

Parsing command line arguments. Specifically ignoring the space in "San Francisco" and delimiting using regex. Implemented search logic to iterate through the date as presented in command arguments (i.e. 20140701). Handling actions after booking and canceling, to allow revenue and list management.

Unsolved Requirement Issues: The project does not have a robust usage of date or comparisons based on time. This project lacks persistence. We could not achieve the use variable # of command prompts based on "- -" delimiters. A document outlining proper command line arguments is included.

Write-up:

As the Hostel21 application stands now there are still a variety of problems to be worked through. The main application functions well for all use cases except search by bed. The application is designed that the main application is called HostelTwentyOne; this class has a list of a Hostel class, a list of a Client class, and an instance of the record class. The Hostel class has a list of beds in each hotel. The Record class contains lists of a Booking class. The Hostel class has a list of Beds. Finally the each Booking has a Bed. This design format allowed me to have a simple representation of the situation as it actually is. The HostelTwentyOne Company has hostels, clients and a record. I used this intuitive design so I would not be confused and it has worked well. The design could be improved, though. I was rigid in sticking to the object oriented design I came up with when I saw the problem. The has-a and is-a relationships did not change once I started the application. Because of this the main program, HostelTwentyOne, is cluttered and has too many functions. If I could design this program again I would go back and put appropriate functions like, or searchBed into their respective classes.

The first issue I found was how to make the application persistent. I could not get persistence to function. It was not stated as an explicit requirement, but it is obvious that the user wants and needs this feature. I looked into implementing Serializable, a java class that supports persistence. I found that implementing persistence is non trivial and decided that it should come later in my programming. The second and most obvious issue is that of timing and date. I looked into the java object date and attempted to implement it but there are many support issues with this class. Many of the functions applied to date have been moved to a more complicated object called Calendar. The functions used to manipulate the calendar were too weak to allow me the abilities needed in the application. Because of this I have implemented comparisons in the Search by date function to workaround this issue. Date and persistence are huge issues in java programming and getting these

two subsets of code functioning would likely take a great deal of time. Finally it would have been nice to get the command prompts to have a variable number of commands based on the "- -file_name" format. As it stands only specific command prompts will functions. They will be listed below for each working use case.

Working functions/ Use Cases:

adminLoad- This function takes in a list of hostels. This list should only be loaded once, as to avoid conflicting lists. There is no reason to load an xml file of data twice as that data is already stored for that hostel. Any actions taken on that data place it in the appropriate location. For example when a booking is made the available bed is removed from the list of beds being searched, and if that booking is canceled the bed is then re-added to the bookable beds in a hostel.

adminRevenue- this function goes through the record (two lists of bookings), and finds all relevant hostels booked in the date range. From here it sums the prices associated with all relevant bookings.

adminOccupancy-this function divides the list of all available hostels (over all stored days) by the list of hostels booked. This gives us a percentage of occupancy for the hostel company.

bookAdd- this function looks through the list of hostels (in a desired city), the list of beds in that hostel and then the list of search ids that each bed has been involved in. If the search id of the bed that the user wants to book is contained in any of the beds this bed will be booked for the user

bookCancel- This function works but could be improved. Currently the book cancel function goes through the list of current bookings and cancels the booking if the booking id matches the booking id the admin is looking for. It could be improved in that I could not figure out how to make the cancellation policy function. This requires the use of dates, and after trying to use Java's date object I still could not get the deadline and penalty to function.

bookView- This function searches the list of current and past booking and finds the booking with the matching id value to the administrators search. Following this it displays the booking.

userAdd – this function takes the command from the admin and uses each field to make a user. If the user does not want to upload financial information they must only enter the first three pieces of data, first name, last name and email.

userChange- this function goes through a list of users. If a user id matches the id in the search, the function will take each field of data pertaining to a user and then replace that data with data the admin entered. If a piece of data should stay the same then enter the same value that is already contained within the user.

userView- this function goes through the list of users if the user id matches the id being searched for then it displays the user.

Search- this function actually was two functions in my mind. The first is a search by date, which is implemented in the application. The second is a search by bed, which is also included. Both searches have problems with long stretches of date. This is because I am not using an explicit date object rather I am just finding the number of days that the booking should occur across, and iterating through the "date" (which is really just a string of numbers). To this end the search works for

multiple days so long as those days don't cross the line of a month, or a year. Future implementations will improve upon this shortcoming.

Non Functional

search(by bed)- so far this function and the results of the function confuse me. A search doesn't output results, it simply alters the search ids associated with a bed-date. This makes booking multiple beds at once difficult at best. For this reason I have decided to make this search similar to bookByDate. This function gives each bed that passes the searches' parameters an individual searchID and allows the booking function to operate. The commands and responses required to add a booking make sense. These requirements are one searchID and one userID. Finally, to implement this function I would have had to alter the bookAdd function to see if a searchID (just an integer in my implementation) referred to multiple beds. This function could be implemented eventually via a small amount of tweaking to the Booking, Bed, and HostelTwentyOne classes.

Future Recommendations:

Please make the technical details of the project available to students earlier. The object oriented design requirements for the project are not that confusing and the bulk of my time was spent working on technical issues like parsing the XML file (which didn't fully work until a week and a half before due date). It seems like you wanted steady work on the program but with the lack of technical details that seems difficult. Of course, this could all simply be because we as students did not ask the correct questions quickly enough. Aside from this, I believe that this is a good exercise in object oriented design, technical implementation and business programming.