Amr Al Dayeh, Elena Rawlinson, Javier Gutierrez Bach
COSC 112
Professor Alfeld
May 9, 2023

Grade

Functionality

- Our program has robust functionality on several fronts. Firstly, our program takes in direct user input on the registration page via a text box. The user can either navigate to the new user registration page and input their name and email or they can navigate to the existing user page and enter their email to log in as an existing user. We built in some safe-guards for this direct interface with the user; if they select existing user registration and then enter an email that is not in the ACRinse.txt file, they receive an error message in the graphics and are prompted so they can register as a new user. We could make this more robust by checking that every email entered (by both new users and a returning users) has the correct format of "user@domain.tld" and if we made this a real app we would have to have some type of email confirmation process to make sure that users weren't just creating false emails to make new accounts. Secondly, when the user clicks somewhere on the JFrame a series of nested if loops determine what happens (if buttons change colors/the page the user is one switches, etc). We attempted to prevent bugs by making the "buttons" (areas on the JFrame that are reactive to mouse click events) change with the value of the static page variable (a field in world). This way, if someone's on their profile and they click in the middle of the page, they don't get switched to the Put Load In page because they clicked the area where the put load button is on the home screen because that button only exists when page == 1, and they're on page ==8. Thirdly, if the user tries to run the app without connecting to the server, they get an error message that simply says "please run the server." We could have made this error message more informative, for example telling them where to navigate in the code to connect to the server and what IP address they should put in.

Design: Our design at the end of this project didn't change dramatically from the design we proposed in the original Project Specification document.

- Using the JSon file to organize and update our data
    - One design choice we made was to organize and store our data in a JSON file. Choosing to use this data format has made the coding of our app much easier than it would have been otherwise. JSON files are composed of nested dictionaries and arrays that make storing and editing large sets of related data straightforward. Whenever a user changes any of the machine fields, user fields, or load fields the JSON file is updated (via the writeJSON method) and the new

JSON file is uploaded to all other users and used to repaint the app (via the readJSON method).

- Machine class and washer/dryer subclasses
    - One design aspect of our program is that we made Machine an abstract parent class and then washer/dryer subclasses of this parent class. This allowed us to bundle a good amount of functionality while still being able to have more specific implementations of the machine and washer classes as needed.
- Making our own queue structure
    - We decided to make our own FIFO Linked List structure instead of using Java's premade LinkedList or ArrayList data structures because we wanted to add some customized functionalities to the queue. Namely, we added a PeopleInQueue method that turns the linked list of users into an array of users, and we have a contains method which checks if the linked list contains a specific user. By making our own Queue and Node classes, we were able to add functionalities to our linked list as needed.
- Semantics so that the User, Load, and Machine all have references to one another
    - Making a machine have a load and a person have a load
        - One semantics thing we did that ended up making the code more organized is give all the machines a load pointer and all the users a load pointer. This way, when a user is created, they have one load that their user object always has a reference to. When they put this load in a machine, the machine then also contains a reference to the user's load. This lets us trace which user is using which machine from either end (the user end or the machine end).
    - Load has machine index or machine index = -1
        - Another semantics thing we did was make each load have a machine index. As described above, each machine can keep track of what load it is pointing to. However, we also wanted each load to contain a reference to the machine that is pointing to it (the machine that it is in). We addressed this by making a field in Load called machine index. Machine index is set to -1 when the load isn't in any machine. This is the load equivalent of the machine field boolean occupied. When the load is placed in a machine, the machine load = this.load and this.load.machineindex = machine.index. Again, this creates a two way reference (like the one described above), where the load contains a record of the machine it's in and the machine contains a reference to the load.
- Machine can be occupied, reserved, or unoccupied
    - The machine can be in three states; occupied (in use) or unoccupied (not in use). Having these three states represented by a boolean made it easier to write a series of nested conditional statements because we always had to start with evaluating the state of the washer. From there, we broke it down into user = you vs !user = you.
- World contains array of dryers and array of washers (instead of project)

    - Main makes an instance of a project, project makes an instance of the world (which contains an array of washers and dryers). This way, if we ultimately implement this app for AC campus, different dorms will be different “worlds” because they have different size dryer and washer arrays.

Creativity

- While using an app to track availability of shared resources is an idea that has been implemented many times in different ways, Amherst doesn’t have one and desperately needs one. Thus, the idea is not entirely unique but our application in the context of Amherst is both genuinely useful and novel. We also have a couple creative twists to our program. The bubbles as a social score to encourage people to take their loads out in time. We also added washing tips and a feedback button that redirects the user to a google form page where we can take suggestions. The leaderboard showing the top 10 people to encourage them to keep good laundry etiquette and possibly we can offer some gifts to people that maintain their position in the leaderboard for a certain amount of time.

Sophistication

- Our program had two main sophisticated parts. The first one is loading and organizing the data from a JSON file. This was particularly helpful and useful because it allowed us to maintain a file with users and information about washers and dryers. The second part is that our program can run on a server and creates a thread for each client and keeps the file updated across users. Every time any user makes a change, the program rewrites the local JSON file, sends it to the server which then broadcasts it to the rest of the users.

Broadness

- We ticked the boxes 1,2,4,5,6. For box number 1 and 6, we used the JSON java library for reading and editing the data as the program runs to maintain a shared data between all users. We also used the java socket library to use in making our server and communicating with it. We used subclassing (box 2) for the washers and dryers from the machine class because they have a lot in common with only a few details that were different. For box 5, there are numerous times where we used bult in data structures such as arrays and linked lists. For example, we used a linkedlist to keep track of the users connected to the server and to broadcast the file in case there is a change to it. However, we still created a data structure (box 4) very close to a linked list with some extra functionality to handle the queue for the washer and the dryer.

Code quality

    - Our final product looks semi-professional because of the graphics and the functionality linked to each page. We tried to make it look as much as an app as possible. We also added comments explaining the code throughout and tried to stay consistent with the way we handled it.