Retrospective Activity

The two main focal points that our group has pinpointed have to do with the DataBase(DB) Design Structure as well as the GraphicalUserInterface(GUI) Layout Structure.

In regards to the DB Design Structure, we noticed that with the increase in the number of features we had more and more data to work with and store. The way our persistent data access worked was by interfacing with the DB and going through the DBProxy as well as the StubDB(for testing)/RealDB(real application and purpose) resulting in these classes becoming overloaded. We have determined 2 potential solutions as well as their respective evaluations for success.

One such way to combat the DB overload issue is to break the DB into the following 2 components: User storage for data that is specific to the user and their fridge, and general storage for data that is fixed and the same for all users (this could potentially be hosted on the web).The Evaluation of Success relies upon if all the classes in the persistenceLayer and the DBProxy can be trimmed. The code should also be dispersed enough that it is possible to add a new major method without risking overloading the class.

Another way is to disperse the data processing more evenly; this could be accomplished by having classes for specific databases that are used for raw data fetching, having RealDB process this data into more structured Java data structures, and then having DBProxy create or initialise data processing for the setup of new objects. The Evaluation of Success relies upon method call passing (1-2 line methods only meant to pass the call to another method) being eliminated and being able to distinctly distinguish the responsibility of each data access method.

Focusing on the GUI Layout Structure, our most used layout format for the UI is currently the GridBagLayout. While this layout format does allow detailed and specific formats, it is at the expense of multiple lines of extra code to specify said details. In an attempt to combat this issue the class GridConstraintsSpec was used to provide static utility methods to reduce duplicate code and better specify layout details. Although this made the use of GridBadLayout more convenient, it also resulted in method calls with a large number of parameters. The large number of parameters resulted in lengthy horizontal code that reduced readability and increased clutter. We have determined 2 potential solutions as well as their respective evaluations for success.

The first solution entails the creation and use of a TableUI class that can provide an interface to the GridBadLayout. This would be able to handle most of the layout setup in the background as long as the client initialised the TableUI correctly at the beginning.The Evaluation of Success relies on if we can utilise the features of GridBagLayout without it taking up more lines of code than the process that initially setup the UI elements that are being laid out.

The second solution would be to make use of the Decorator design pattern; allowing us to build up layout specifications without the need to call methods with many parameters (5+). This would essentially be an expansion on the current fix. The Evaluation of Success relies on limiting the specifying of layouts to at most 2 lines while keeping method parameters to at most 3 lines for each method call.