I wanted to design an app that I could apply my financial background to while highlighting the new skills I’ve acquired with my education at MATC. I thought a budget app would be a good start. I knew I wouldn’t be able to do everything I wanted with this app, like include business valuation and investment portfolio analysis/tracking, but this is a good start. Knowing that I want to add more to this app down the road I tried to keep things as adjustable as possible employing object oriented programming techniques. I created a base class of “budgeteer” and a sub class that inherits the base class where I do all my SQL querying. This will also explain why you might notice a “Settings” table in my JDBC, because I want to expand the usability to allow for users to choose which settings they want to see next time they run the app. I plan on using this application in my portfolio.

For first time users they’re going to have to create an account using the “CreateNewUserFrame” by clicking on the “Create User” button on the “LoginFrame”. While the user enters their information I’m verifying each item doesn’t exceed the SQL table limits so that the user doesn’t get a nasty error. After the user has entered all their information I validate that the email is actually an email address, that their full name consists of more than one name, and that the login name they chose isn’t already being used. If all the information checks out, then the status bar at the bottom will display a message that the user has been added to the database. On exiting this window the user’s chosen login name and password are automatically filled in for them in the login screen for easier access into the budget.

Upon entering the budget frame, the table is populated with the simple or summary version of the user’s budget. If this is the user’s first time entering the frame, they’re only going to see their balance with no items increasing or decreasing it’s amount. The user is going to want to click on the “Edit Budget Items” menu item to go to the “BudgetItemsFrame” where they can add and edit the items associated with their user id. Upon entering the budget items frame it loads all the budget items the user has listed, which of course would be blank for new users. Once items are added, the user can select any of the rows in the table and click the “Edit Selected Item” button that will pull all the information of the selected table item into the form above for the user to edit. Once the user is done editing they can click the save button and the information will be updated in the database. Once the user has items set up for their account they can enjoy either the simple or detailed views of their budget.

I enjoyed working on this project and trying to put good object oriented programming techniques to work for me. This is still a work in progress so I look forward to any feedback you could give me to improve the functionality or programming design of my app. Below is a summary of how my app meets the requirements of this semester project:

1. I have 4 easy to use jFrame windows I designed using the GUI builder in NetBeans (LoginFrame.java, CreateNewUserFrame.java, BudgetFrame.java and BudgetItemsFrame.java).
2. I used JDBC:derby for building my relational database. All my tables are based on the USERS table, including the UserID field as a foreign key in all of them so that I don’t duplicate the storage of data in a table.
3. I have 4 classes outside the 4 classes I have for the frames (Start.java, SimpleDataSource.java, Budgeteer.java and BudgeteerSQL.java). Like I said before I created the BudgeteerSQL class to inherit the properties of the Budgeteer class and provide all the necessary SQL methods in an object separate of the base class while still pulling from the base class. In the base class I encapsulated all my property data.
4. I’ve create the JavaDoc and descriptively named all my variables, methods and classes using camel case
5. I used exception handling every time I queried the SQL database and when I was loading and editing tables. I found it very useful for debugging and notifying the user when something they entered was incorrect or already in use.
6. I used loops and arrays specifically in the BudgeteerSQL class for querying the database to return an array to my frame classes so I could loop through the results and populate my tables. As I said before the BudgeteerSQL class inherits the Budgeteer class, adding the SQL methods. All the tables in my SQL database are linked using the user id as a foreign key as well as each table having their own auto incrementing id. I have 4 screens total that either pop up in addition to the window already present on the screen or closing one window before opening another and passing variables and objects from one frame class to another.