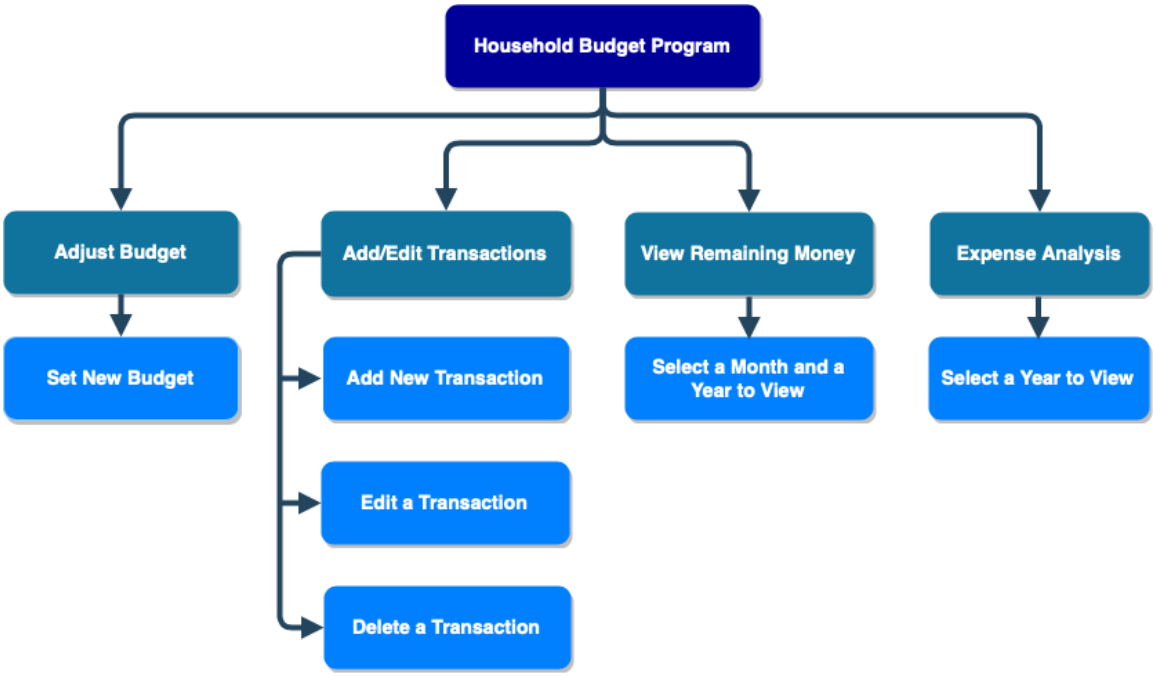
Criterion B: Design

Structure Chart

First, I decided to create a structure chart to organize the different elements of the program in relation to one another. This allowed me to visualize the tasks and develop the preliminary structure of the program.



After creating this diagram, I decided to make an outline of the classes that need to be included in the program to perform those tasks.

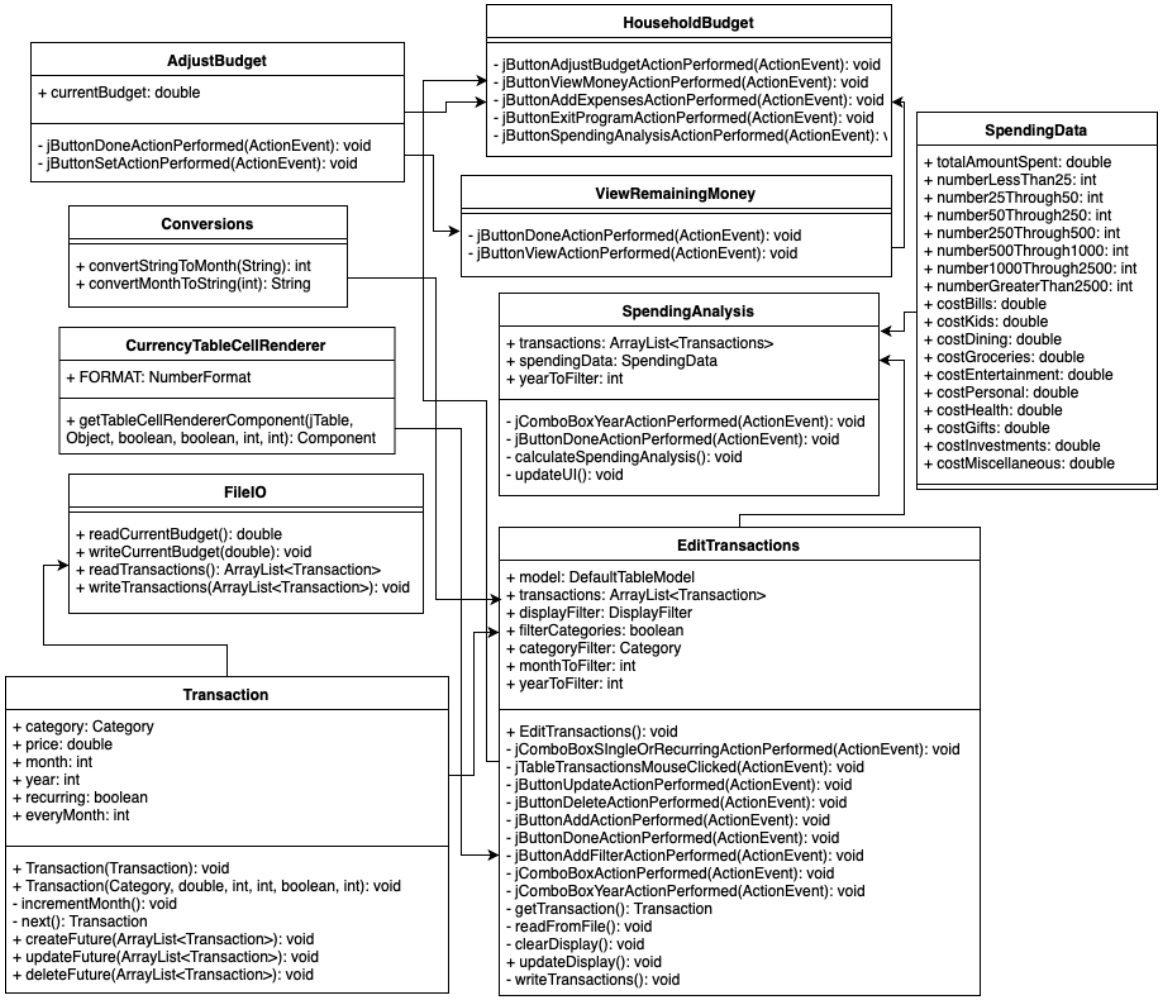
The responsibilities of each class are as follows:

- **HouseholdBudget:** contains the program's main method and allows the client to interact with the rest of the program through a menu system
- **AdjustBudget:** stores current budget and allows client to change that value
- **Transaction:** serializes transactions
- **EditTransactions:** sets the table model, stores the transactions, and allows the client to view, add, edit, and delete transactions
- **ViewRemainingMoney:** allows client to view the remaining money from budget after selecting a month and a year

- **ExpenseAnalysis:** calculates client's spending statistics based on the existing transactions and allows client to select a year to view
- **Conversions:** converts between integer and string representation of the months
- **CurrencyTableCellRenderer:** sets the formatting in the transactions table
- **FileIO:** reads and writes the budget and transactions
- **SpendingAnalysis:** computes the spending analysis data for display to the user
- **SpendingData:** contains variables to hold client's spending data

UML Class Diagrams

Next, I created UML class diagrams to design the static view of the program, which would help me determine how the different classes should interact with each other before I create the program.



Prototype

I created a prototype for the program using the NetBeans IDE to determine whether the program would be feasible. Also, the prototype would allow my client to evaluate the design and provide feedback on the user interface before I begin coding the program.

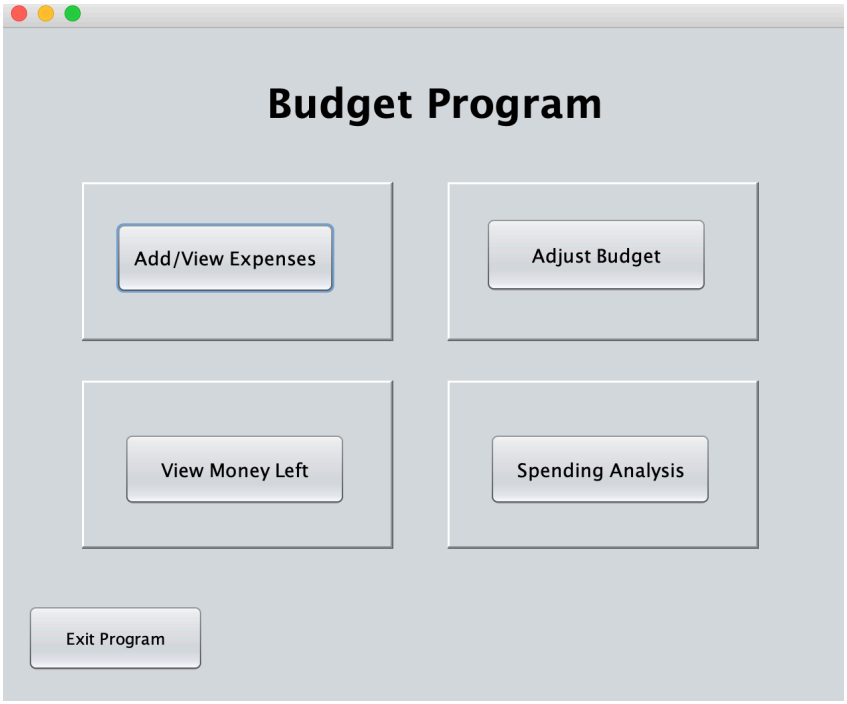


Figure 1 - Main Menu

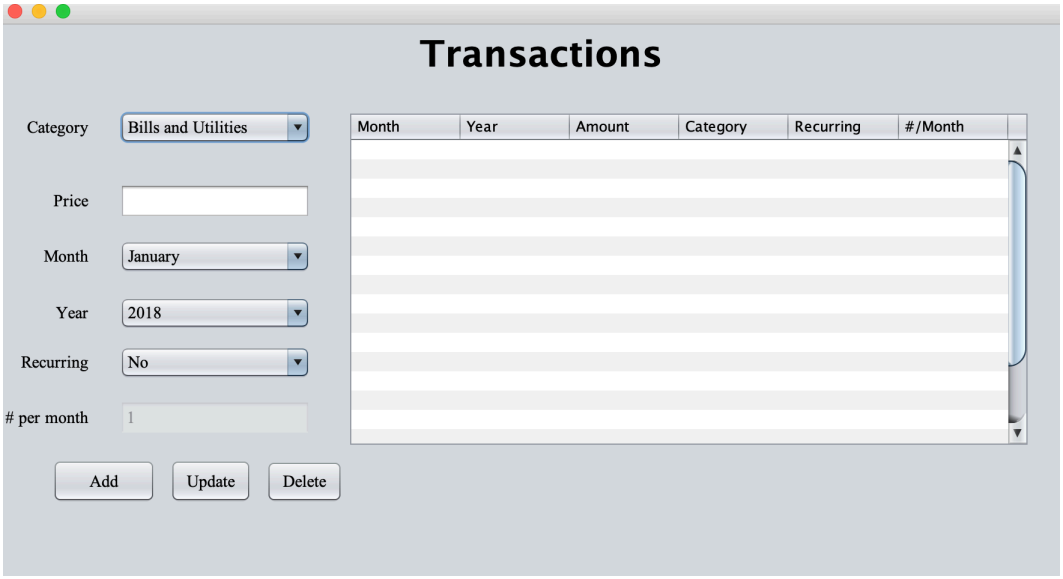


Figure 2 - View, enter, update, and delete transactions

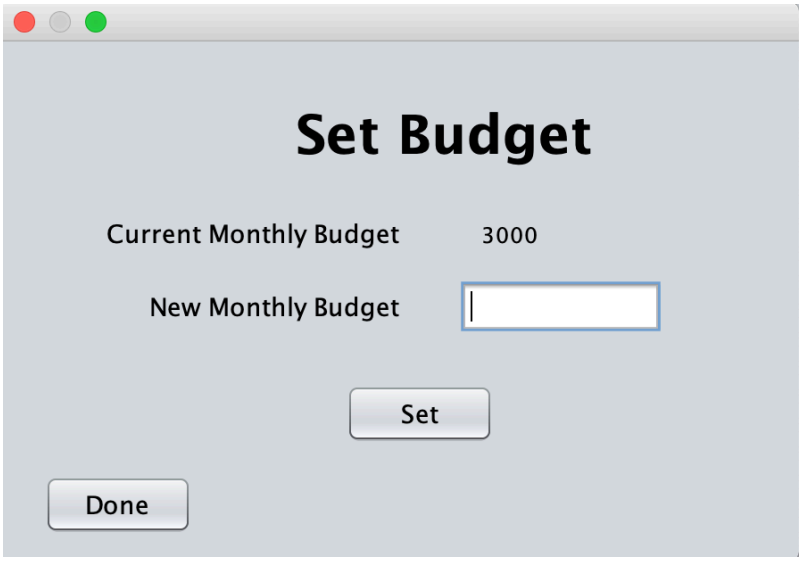


Figure 3 - Set or adjust budget

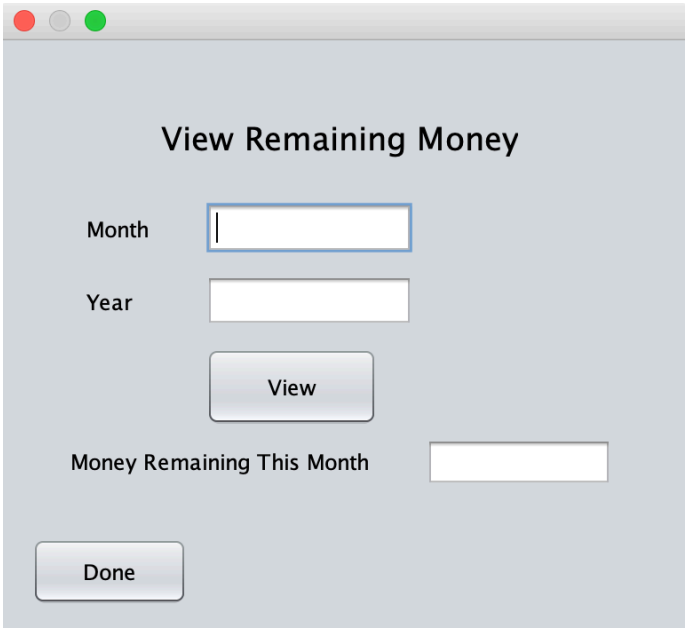


Figure 4 - View remaining money in budget for a chosen month and year

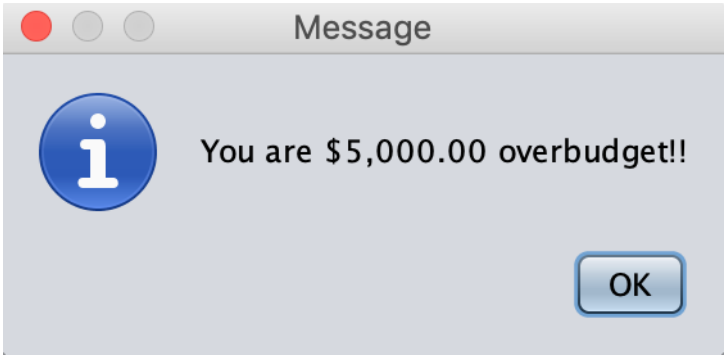


Figure 5 – Warn client if expenses exeed the budget

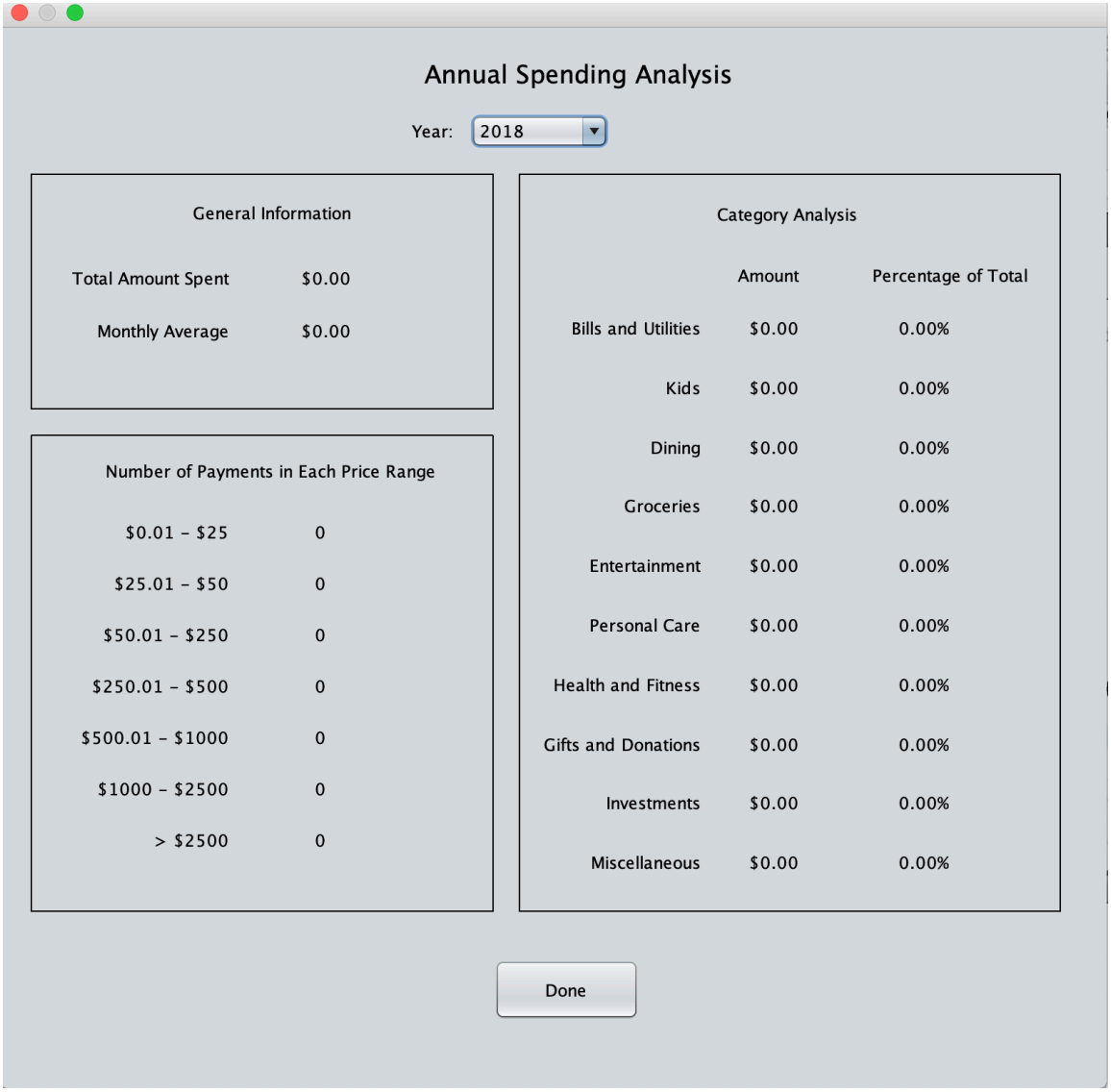


Figure 6: View expense analysis

After showing the prototype to my client and explaining the different components of the program design, my client was generally satisfied. However, she made a few suggestions for improvement. She said she wanted to be able to sort the transactions by their attributes or search for a specific transaction. Furthermore, she wanted recurring transactions to be displayed once a month so that she can make changes easily.

Modifications to Initial Design

In order to honor the requests of my client, I decided to make some minor changes. I added an AddFilter class to allow the my client to choose which categories, dates, and types of transactions she would like to see in the table and a Filters class to implement the filtering. I also modified the display function of recurring transactions so that they would be displayed in the table once a month.

Figure 7: Add Filters

(479 words)

Test Plan

Action to be tested	Test method
Graphical User Interface system	• Run the program and check to make sure the graphical elements are functioning as desired. • Test all buttons, combo boxes, and other features on the user interface
Adding/updating/deleting transactions	• Add and update a variety of single and recurring transactions to make sure all the categories are functioning properly. Verify that the program prompts user to either update one transaction or all future transactions occurrences when updating or deleting • Delete one transaction and delete multiple transactions at once.
File input/output	• Add, update, and delete transactions and make sure they are still shown in the table after the program is exited and executed again. • Adjust the budget and verify that the new budget is maintained after the program is exited and executed again.
Sorting transactions by date	• Enter five transactions of varying months and years to make sure that the transactions are sorted chronologically.
Filtering transactions	• Enter single and recurring transactions that have varying months, years, and categories and verify that the filtering process filters the correct transactions.

Calculating remaining money	• Enter five transactions and manually calculate the remaining money to make sure this feature is working as desired.
Setting budget	• Adjust the budget to varying values and make sure the budget is being updated.
Budget analysis calculations	• Enter ten transactions of varying categories, years, and amounts. Manually calculate the results of the expense analysis. Verify that the manually calculated values match the values calculated by the program.
Data validation	• Verify that if the user enters invalid data, such as a letter instead of a number, the program will allow user to re-enter data instead of crashing.