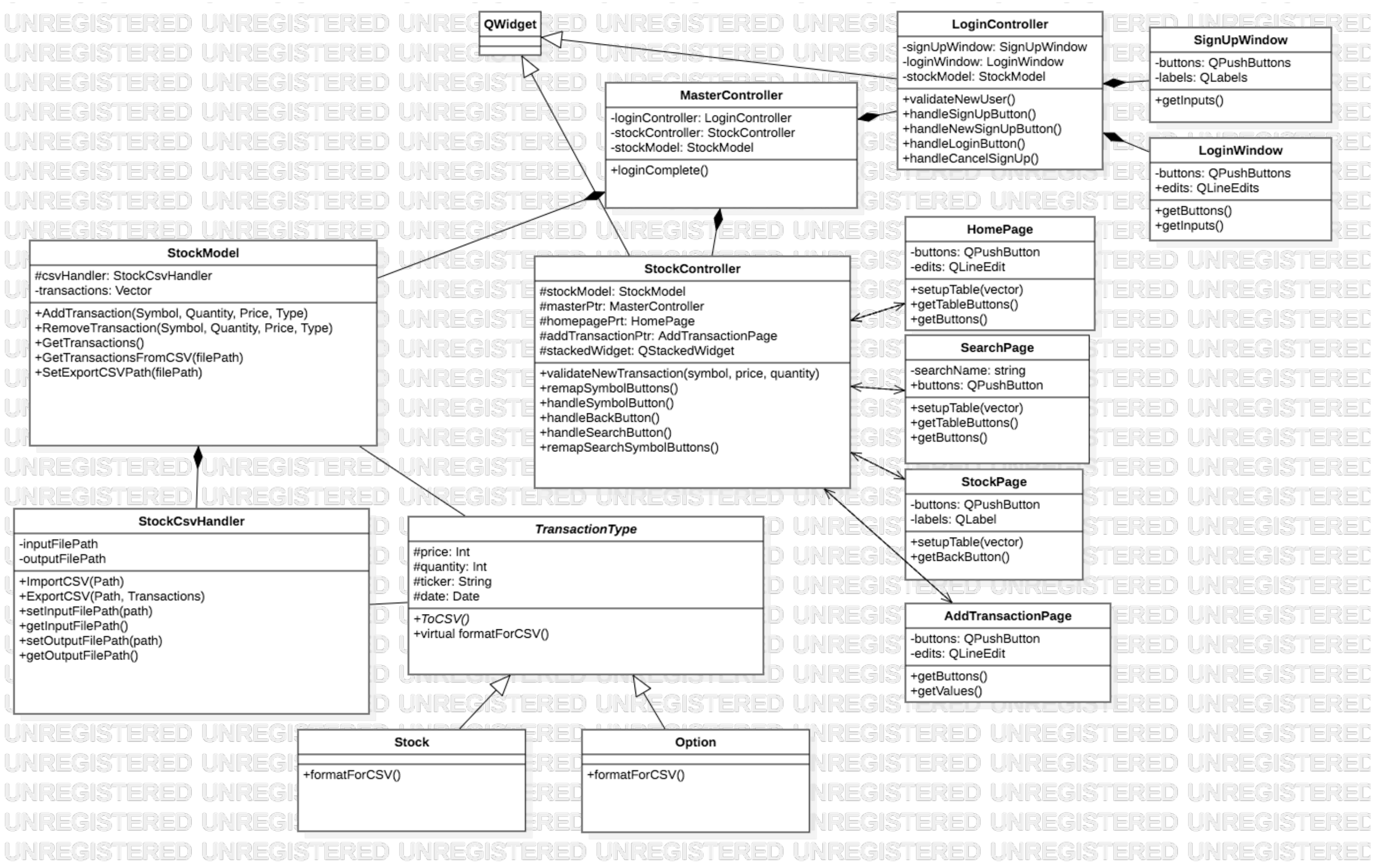
1. Kirk Partridge
2. Project Summary: A GUI application that will aid with tracking stock purchases and sales. Will allow users input recent stock transactions and will keep track of performance of said stocks. Users will also be able to request a dump of all previous transactions as well as import transactions from a csv file.
3. Features Implemented

ID	Description
US-01	As a user, I want to be able to sign up to use the application.
US-02	As a user, I want to be able to login using credentials established during sign up.
US-03	As a user, I want to be able to input a new stock transaction.
US-05	As a user, I want to be able to view details of a specific stock transaction.
US-07	As a user, I want to be able to search for transactions by ticker symbol.
US-08	As a user, I want to be able to view search results in a single window.
US-09	As a user, I want to be able to select from search results to display a specific transactions details.
US-10	As a user, I want to be able to request a csv download of all transaction data.
US-11	As a user, I want to be able to import a csv of transaction data.
US-12	As a user, I want to be able to follow a URL hyperlink for a specific stock.

4. Features Not Implemented

ID	Description
US-04	As a user, I want to be able to remove old stock transactions.
US-06	As a user, I want to be able to view current stock performance.

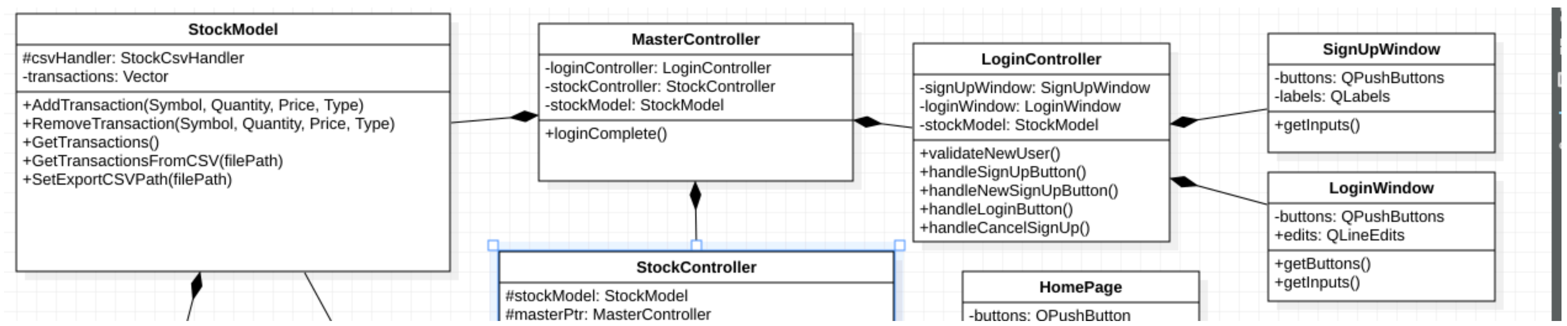
5. Class Diagram



My class diagram changed a bit throughout the development process but not as much as I expected. I ended up breaking apart the controller into two separate classes as it quickly became apparent the single controller was turning into a god class. I also opted to go with a file storage system as opposed to a database. This simplified the import and export process and avoided the overhead involved with a true database.

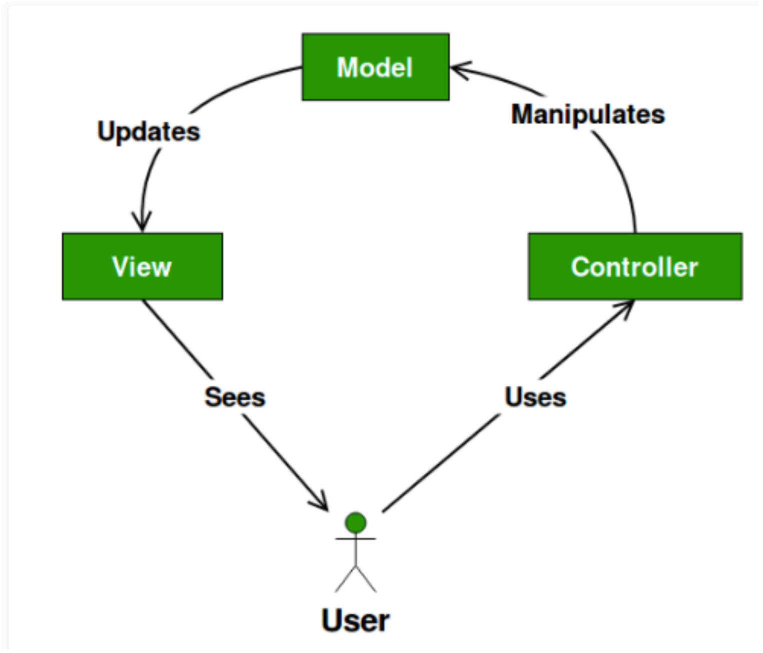
Thanks to the design process up front, I was able to start coding with a pretty solid vision of all the pieces and how they come together. This made the coding process much more effective and clean as the relationships between classes were already established.

6. Design Patterns
   a. MVC –
      i. My Implementation



      ii. Design Pattern Diagram

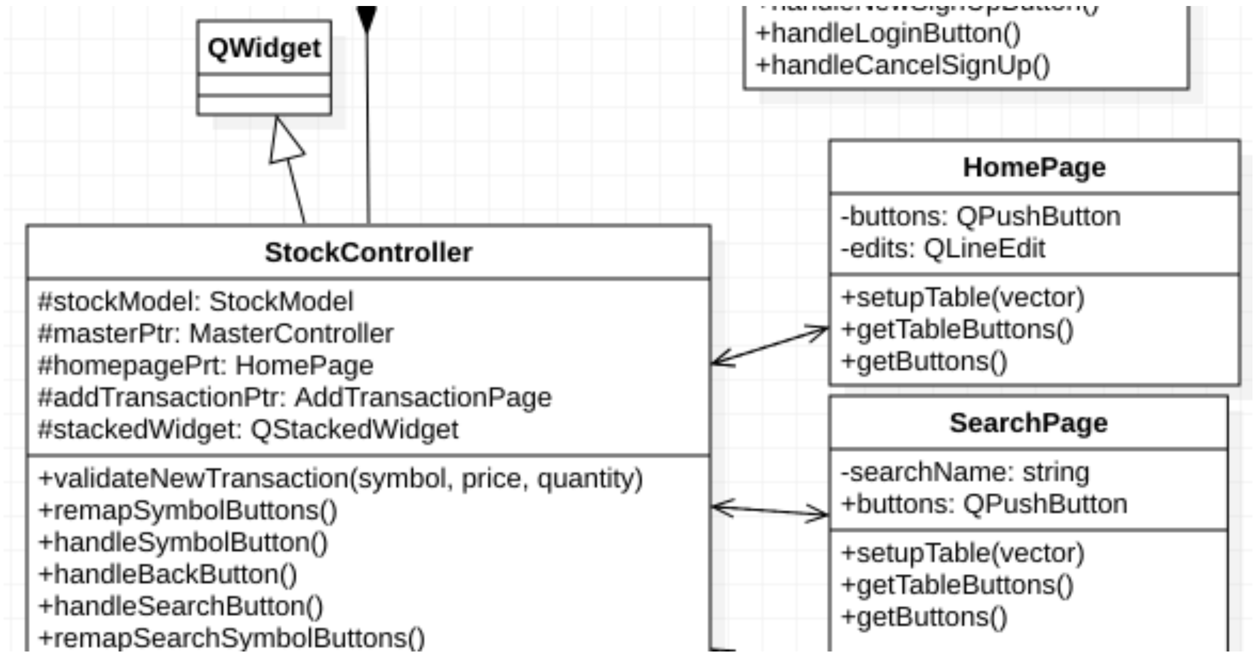
         1. Source - https://www.geeksforgeeks.org/mvc-design-pattern/



      iii. Why / How I Implemented:
MVC is a classic pattern for GUI applications. It allows a backend vs frontend to be separated and developed in parallel. For my application, StockModel acts as the Model, MasterController / LoginController as the controllers and SignUpWindow/LoginWindow as the views. I chose this pattern as it separates concerns and handles all model interaction through a single controller as opposed to randomly within views. It creates a simple data flow and helps manage view to model updates and vice versa.

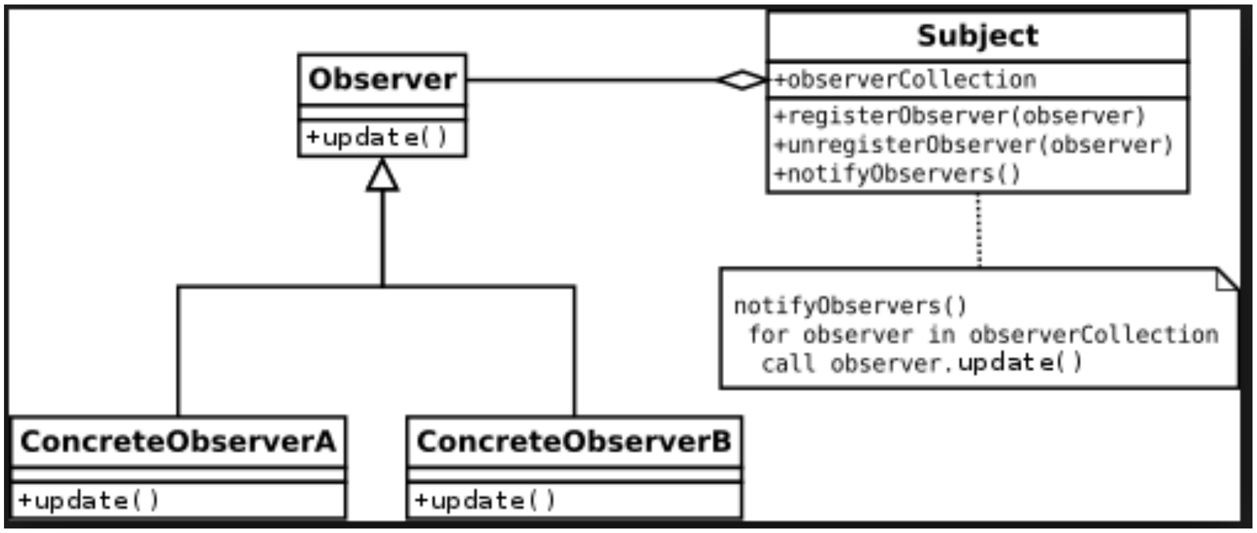
b. Observer

i. My Implementation



ii. Design Pattern Diagram

1. Source - https://en.wikipedia.org/wiki/Observer_pattern

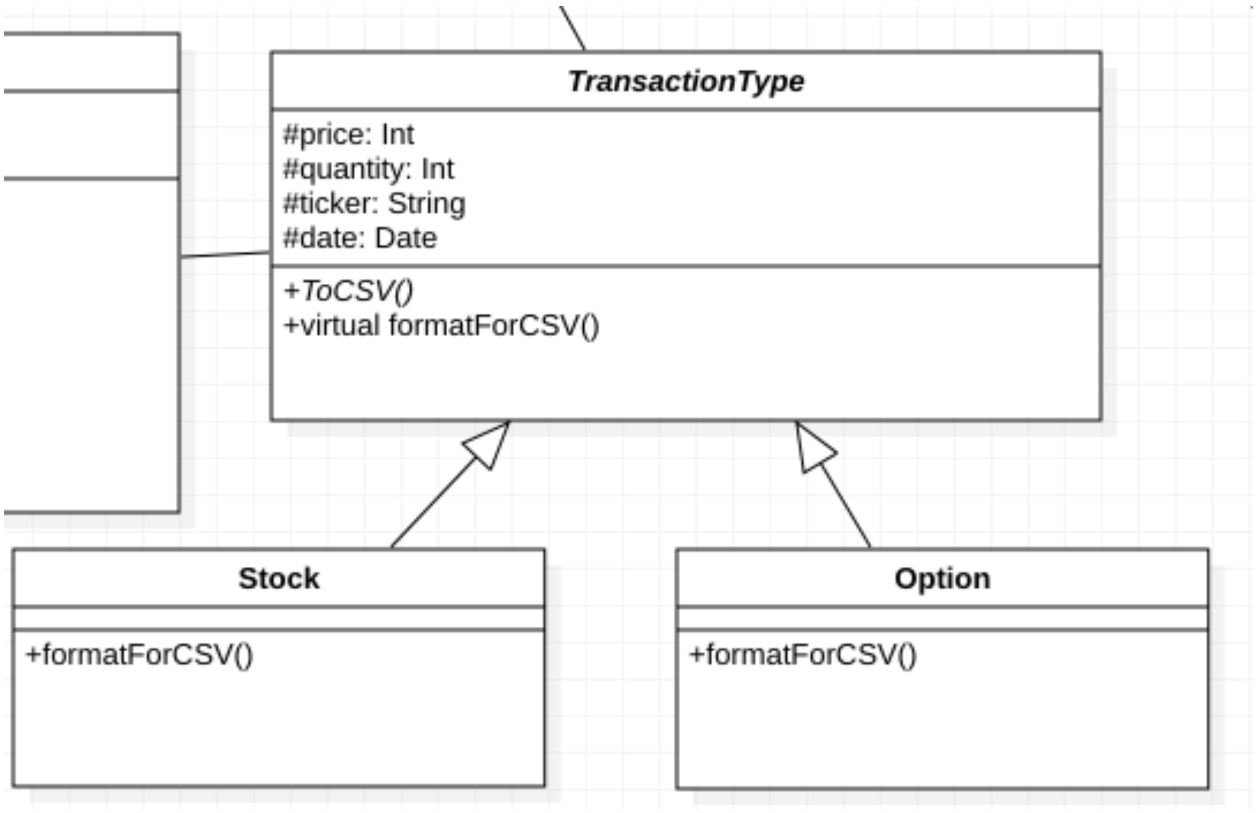


iii. Why / How I Implemented:

Thanks to QT, this design pattern was an obvious choice to implement. This allows easy communication between the HomePage/SearchPage buttons (Subject) and the StockController (ConcreteObserver). Anytime a button is clicked in either page, the StocController is notified and can handle the press appropriately. This allows for easy navigation throughout the GUI as well as necessary data input and output.

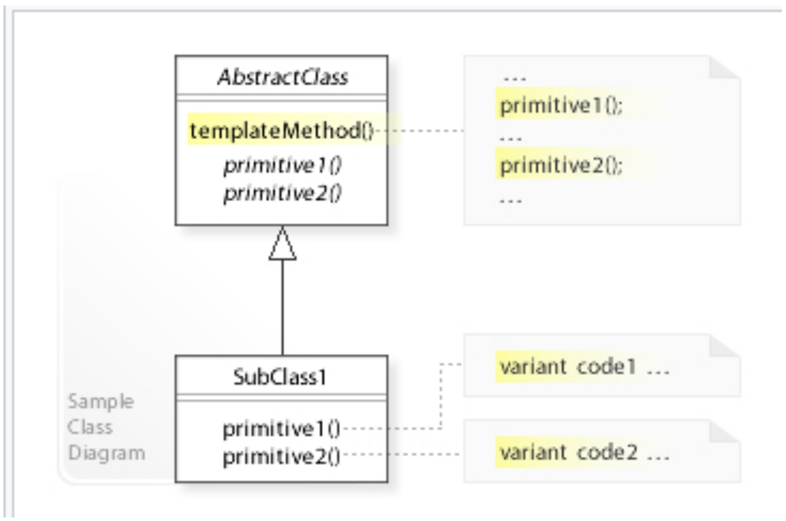
c. Template

  i. My Implementaition



  ii. Design Pattern Diagram

    1. Source - https://en.wikipedia.org/wiki/Template_method_pattern



iii. Why / How I Implemented:
I used the template design pattern to allow for future extensions of the program. Because the “formatForCSV()” function in the TransactionType class is virtual, its implementation is left up to those who inherit from it. And since ToCSV() calls the formatForCSV(), export functionality among all TransactionTypes are consistent. Today this application only supports stocks but if someone wanted to add options or another type of transaction, this would allow for easy integration with the current system and not break the ToCSV function.

7. What I Learned:

Going through the entire software development process was a great opportunity to better understand good development. Previously I have attacked most projects with a code first mentality, get something working and figure out specifics later. This can lead to messy and unorganized code as well as multiple hacks to get things working. Actually going through the process of what you want to accomplish (requirements), a high level view of how the classes will look and interact (class diagram), and then actually coding has lead to much cleaner code. It also made it much easier to recognize where design patterns and other efficiencies are most effective. Thanks to the requirements, it also was much easier to know when something was done and working, as opposed to constantly fiddling with a piece due to vague understanding of what it needs to accomplish.

I also learned that spending time upfront on the design really does pay off with the amount of time saved down the road. A concrete vision paired with a strong class diagram can make the coding much quicker and effective. It also leads to less bugs down the road and easier debugging thanks to the high level view the class diagram provides.