1. **Polymorphism** (*DisplayPanel*)
   * Display Panel is a jPanel with an additional User System attribute and methods to access and mutate that attribute. TradeCenter, and PortfolioPanel are both subclasses of DisplayPanel. When the Main renders, uses, and changes the currently displayed View, it does so by referencing a variable defined as a DisplayPanel. In reality that variable may be one of several subclasses. This is to enable uniform functionality and simple transitions across all components of the View. It also greatly encourages extensibility through allowing designers to add any new subclasses, all of which may be used without any redesign of the existing class structure.
2. **Interface** (*Iterator*)
   * The interface design pattern was used as a means of modularizing the StockList class. Its use shortened the overall length of the StockList source code and increased readability for StepThrough, the class nested in StockList.
3. **Decorator** (*DisplayPanel*)
   * MainView uses DisplayPanel as a decorator polymorphically by being decorated with either the Portfolio or the Trade Center
4. **Encapsulation/Data-Hiding** (*StockQuote*)
   * This design pattern was used to protect any StockQuote object from being used incorrectly by other programmers.
5. **Iterator** (*StockList.StepThrough*)
   * This design pattern was used due to its efficiency in moving through an ArrayList object type.
6. **Events/Listeners** (*GUI*)
   * Many listeners are used for various components in the GUI. Each button has a corresponding callback that listens for buttons presses. Additionally, there is a ComboBox in PortfolioPanel that listens for changes in its index in order to for commands from the Controller to pass to the View to change its display of the data in the Model.
7. **Strategy Pattern** (*User, UserSystem*)
   * This design pattern was used to store and manipulate data easily through the user interface, allowing variables to be assigned and used as they were needed.
8. **Composite Pattern** *(OwnedStock)*
   * OwnedStock composites StockList and a HashMap in order to easily associate quantities with each StockQuote within the StockList.
9. **MVC Passive** *(entire package)*
   * The Stock Trader uses the MVC Passive design pattern. The Model processes stock quotes and performs operations upon their data according to input from the user through the Controller. The View displays the data in the Model, though it does not actively interact with it in order to update itself. This distinguishes it from MVC Active. The buttons and objects on the Display Panels allow the user to control both the data in the model as well as how it is used, with several filtering capabilities. In addition, there is an Update button that queries for new input in the model from Yahoo, performs operations on that input with the business logic, and then displays the data to the user in the view. File input and output is also available through the Save button.