**Application Overview**

This application is a tool which will help users build their ideal optical solution, when the user first starts the application they will be prompted to enter which type of optics they would like to avail of. This will range from generic framed optics to contact lenses and contactless, ensuring a wide range of choices is vital as again to reiterate; the objective is to create the ideal optical solution which is unique to each user experience. As the user enters their preferred optic type, brand and lenses etc. there will be a textArea being updated as new information is added, this allows the user to see exactly what they’re selecting as they build their optics. Once all the necessary fields have been filled the final build will be displayed to the user along with the price.

**Functional Requirements**

The application must implement at least four different design patterns and they all must work together in order to accomplish the given task, what this means is instead of each pattern “doing their own thing” they all must logically work together.

**User Requirements/User Interface**

**Design Pattern Explanation**

There are four design patterns which have been implemented to ensure efficiency and decrease overall workload in terms of programming in this application. Each design pattern will be looked at in detail and then the use of said design pattern will be justified.

**Factory Pattern**

The factory pattern is a simple design pattern which allows for a base template to be created (in our case optics) and can then be extended out into other classes which will inherit the base templates methods and attributes. All attributes common to each optic variant have been added to the base “Ocular” class, this is along with any methods which may be needed across all classes.

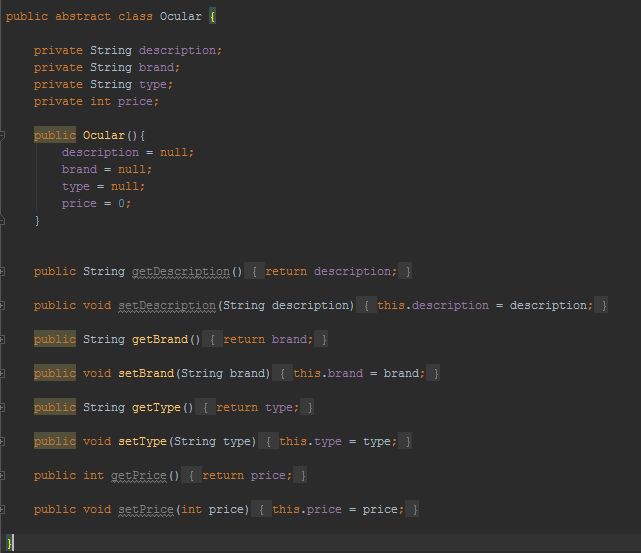


Figure : Ocular class which all other classes will extend.

The only attribute which is unique to each class is their type i.e. framed, contact & contact lenses. This not only helps to identify what type of optics have been chosen but also cuts down on the overall workload, if the factory pattern was not implemented each individual class for optics would need all attributes and methods present instead of inheriting them from the Ocular superclass.

**Singleton Pattern**

**Command Pattern**

The command pattern was chosen in order to ease the implementation of certain functions within the application. A menu bar was added to the GUI with a series of functions available to the user such as, the ability to view your current build which would be displayed in a JOptionPane, an option to close out of the application & an option displaying author information.

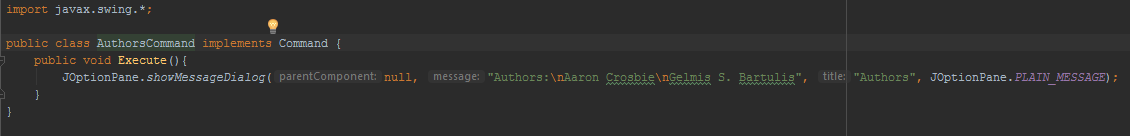


Figure : AuthorCommand, in the Execute method a JOptionPane is created with the relevant dialog displayed.

A Command class was created, this class is implemented in all classes which need an Execute function. A CommandHandler class was then created in order to:

1. Set commands with a custom function.
2. Get commands, which would return the command which had previously been set.

The cmdMenu class implements this CommandHolder so that each new menu item can be initialised with a custom command defined in a separate class, this custom command would be unique for each menu item as each item preforms a different function.

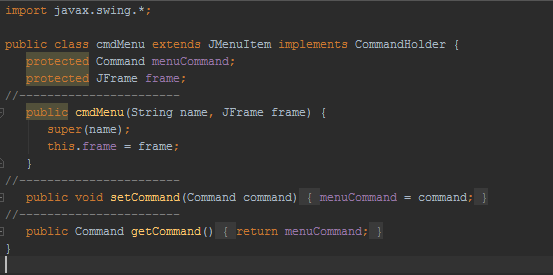


Figure : cmdMenu showing its default constructor, get & set methods.

**UML Diagrams**