Tushar Kumar
SE-450

Final Project Report

Missing Features and Bugs

Missing features:

There are no missing features in the project. But there is one feature that have been implemented ineffectively.

The ungroup feature just uncouples all the shapes at once. It does not ungroup the last group shape or shapes.

Bugs:

Outlining of shapes does not work as expected. When a new group of shapes is selected instantly after selecting another group of shapes, the dashed outline does not get removed from the old shape.

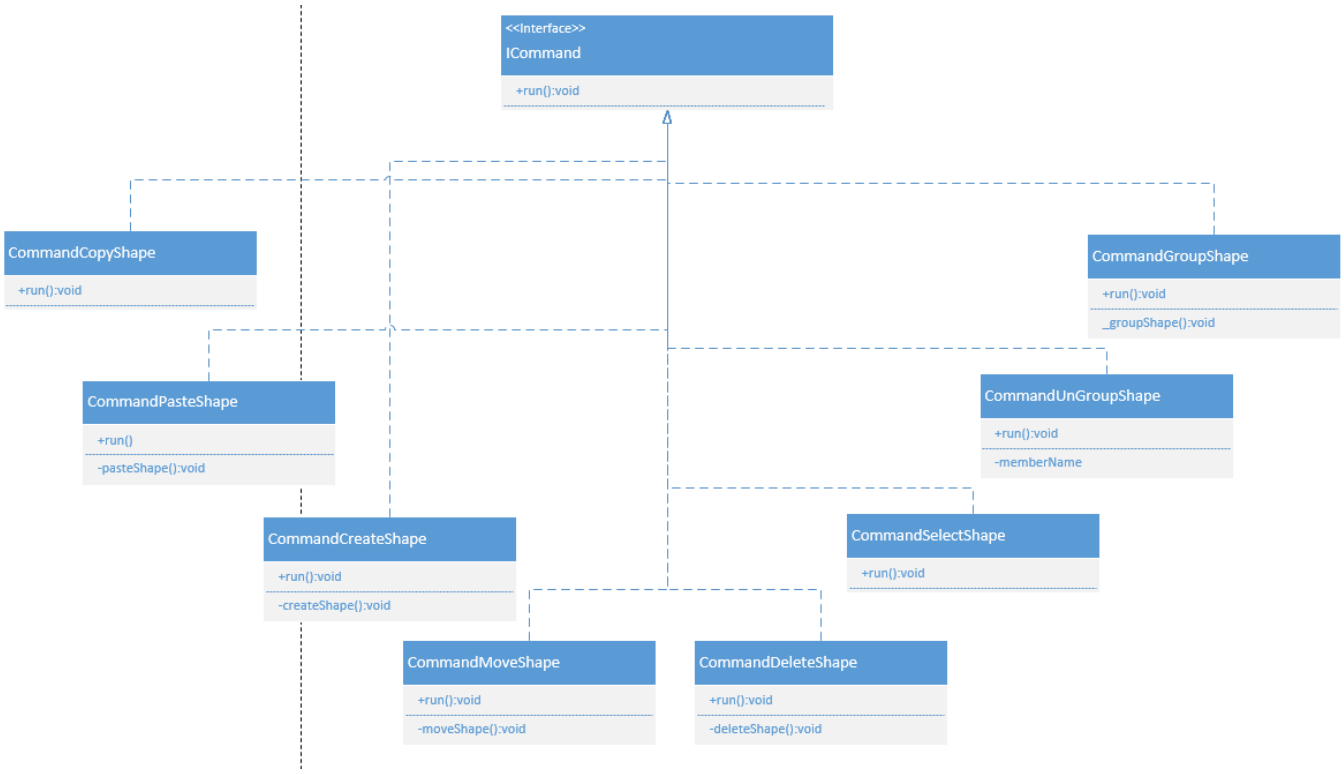
Sometimes ApplicationState does not initialize dialog provider so the code does not compile. Closing the code in the editor and opening it up again fixes the issue.

Sometimes the first shape drawn does not show up. After going back to the editor and then going back to the application window makes it show up. Everything works after that as expected.

Tushar Kumar
SE-450

Notes on Design

Command Pattern



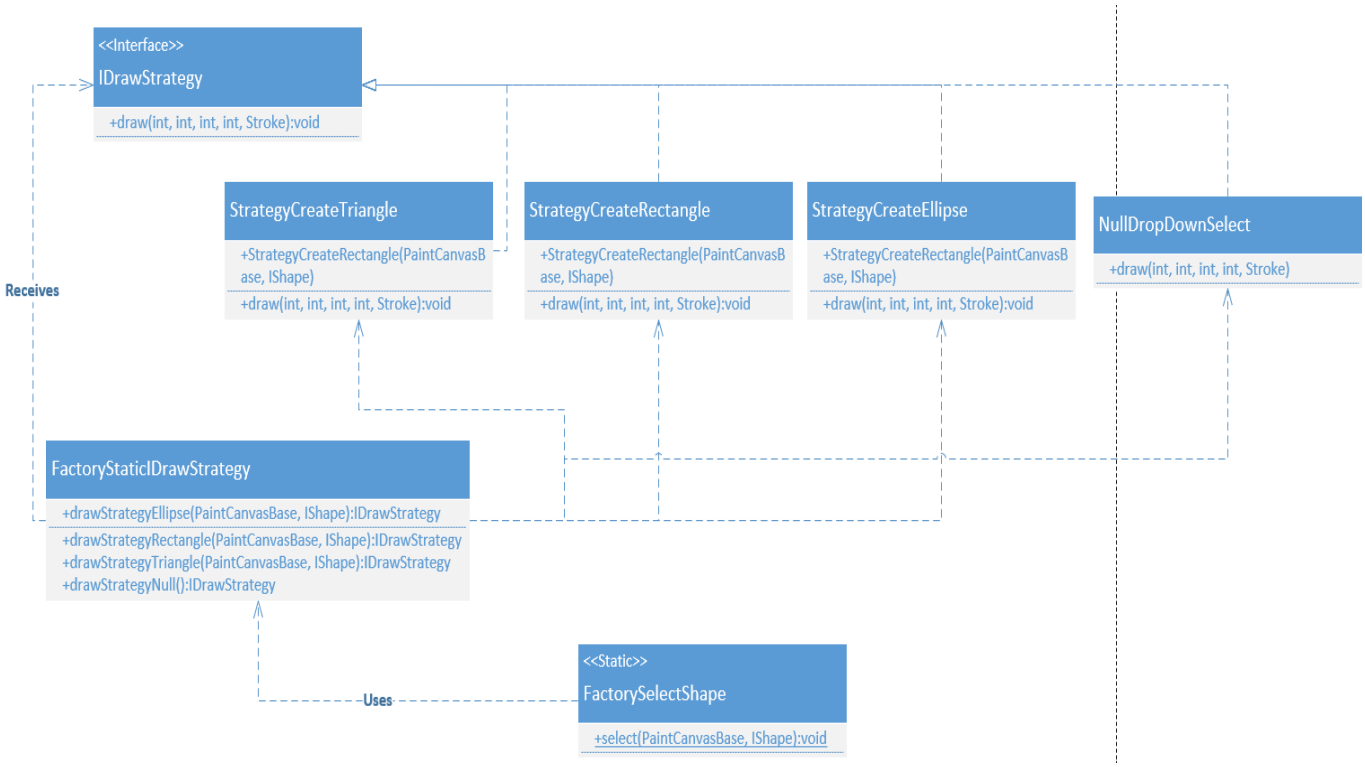
Command Pattern is a behavioral pattern that encapsulates a method call. The method call acts as a intermediary for another object. It wraps up functionality and all necessary information into an object to be run later.

Command pattern encapsulates a method call to either draw, select, move, copy, paste, undo, redo, group and ungroup. These are performed by classes which have their functionality in their name. These classes are CommandCreateShape, CommandSelecShape, CommandMoveShape, CommandCopyShape, CommandPasteShape, CommandUndo, CommandRedo, CommandGroupShape, CommandUnGroupShape and they implement the ICommand interface. Among these classes, the classes which draw, move, paste and delete also implement the IUndoRedo interface to implement the undo and redo functionality.

Command Pattern was chosen as it would make it easier to keep adding functionality to the application without having to make changed to any previous functionalities and because it was strongly suggested by the professor. The implementation of all the above functionality is encapsulated by the run() method declared in the ICommand interface. It allows for implementation of multiple functionalities with a single interface.

Tushar Kumar
SE-450

Combined UML (Strategy, Static Factory and Null Object Pattern)



Strategy Pattern

Strategy Pattern is a behavioral pattern. It allows for the implementation of part of the algorithm and the other part is interchangeable at runtime. It provides a consumer with information about what it needs to do.

StrategyCreateTriangle, StrategyCreateRectangle, StrategyCreateEllipse, NullDropDownSelect are the classes that implement the IDrawStrategy interface. Each class draws a shape, these shapes can either be outline only, filled in or outline and filled in both combined.

While drawing three different kind of shapes the common behavior is to draw a shape. The strategy pattern allows for selection of the shape to be drawn at runtime. This pattern also allows more shapes to be added at a later stage if required. It helped with the implementation of single responsibility principle as each class is only responsible for one shape. The implementation of each shape is in their own classes.

Static Factory Pattern

Factory is an object that makes other objects. Static factory is a creational pattern utilizes principle of least knowledge. This ensures that the clients do not need to know the name of the specific classes. It allows for the encapsulation of the shape creation.

FactoryStaticIDrawStrategy creates the factory of shapes. The shapes are selected using the FactorySelectShape.

The static factory is used to create the three required shapes, ellipse, triangle and rectangle dynamically. It allows for the implementation of the open/close principle. The factory is open to change where multiple new shapes and be added easily. I chose this pattern after reading a discussion on the discussion forums about using this pattern together with the strategy pattern. This helped me eliminate three shape classes which had Identical code for selecting which strategy to use to draw the shapes.

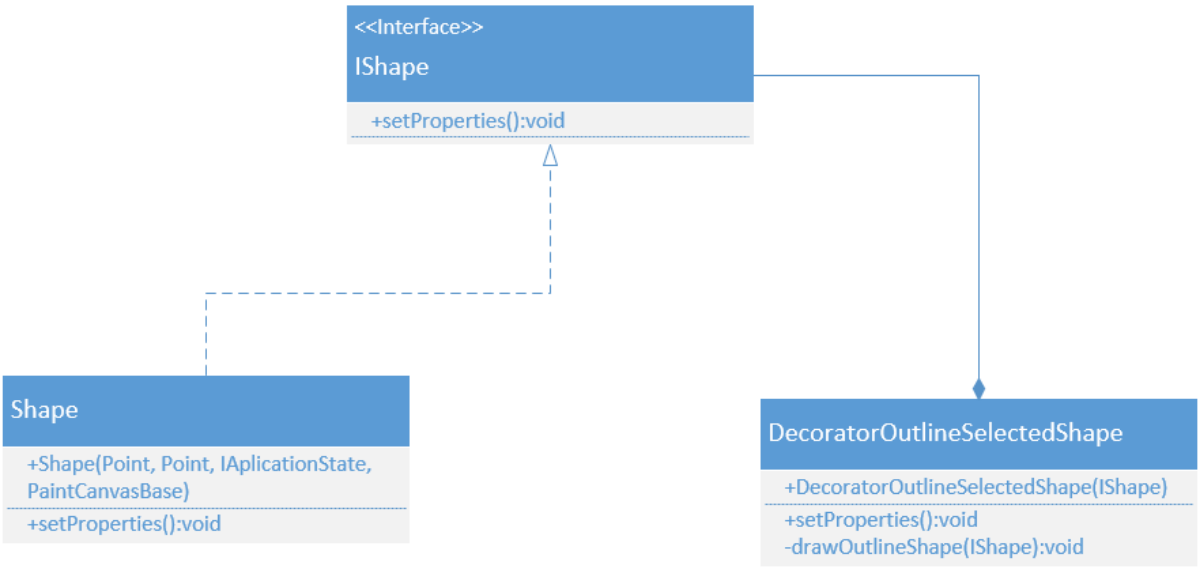
Null Object Pattern

Null Object is a behavioral pattern, which is an object that provides default behavior that can be used in place of null in code. Since the null object is not null, it does not throw NullReferenceExceptions when methods are called.

NullDropDownSelect class is used to define the default behavior. It implements the IDrawStrategy interface. The default behavior is defined inside the draw() method.

When selecting a shape to be drawn, the IDrawStrategy instance was needed to be initialized as null. To give a default behavior if there was a null, NullDropDownSelect was created. This was done to help debug the code if there was a NullReferenceException while choosing a shape.

Decorator Pattern



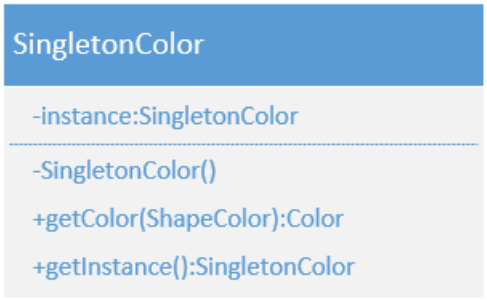
Decorator is a structural pattern, which is used to wrap functionality around objects that implement a certain interface. The decorator objects can implement same interface as the

subjects. Optionally, they can also implement their own interface, however, the decorator interface will extend the subject's interface.

Shape and DecoratorOutlineSelectedShape classes were used and they implemented the IShape interface. The interface has multiple setters and getters for various shape properties. It has a setProperties() method which is used to initially set the properties of a shape. In DecoratorOutlineSelectedShape this method is used to outline the selected shapes.

Outlining a selected shape can be achieved by wrapping the outline function around the selected shape. This is the reason why decorator pattern was used. This implementation allowed for the use of the classes used to select and draw the original shapes to also draw the outline to those shapes.

Singleton Pattern



Singleton pattern is creational pattern, which is used when there is single instance of an object and other instances cannot be initialized. The single instances can be access statically.

The pattern is implemented in the SingletonColor class.

The singleton pattern was used to convert the string values in ShapColor into color. Since color only needed to be initialized once so that it could be used in other places inside the project, using singleton pattern made sense.

Successes and Failures

Successes:

The biggest success was learning java and object-oriented programming principles like interfaces, different kinds of classes (abstract, static), different types of concrete relationships between classes, drawing UML diagrams etc.

Implementing strategy pattern to draw different shapes was a good decision in the beginning of the project. As more features were added to the project I did not have to change or update anything in the draw strategy as it worked perfectly. I was even able to combine it with the static

factory pattern which eliminated the need for separate shape classes which had similar code in them.

Creating a shape class which created a shape object which contained all the set and get methods for all the shape properties was very helpful. It prevented a lot of redundant code and made is easy to pass shapes around different classes.

Lastly, learning about the various design patterns and SOLID principles was very helpful for the project and helped me understand more about developing software. The patterns and the principles helped in refactoring whenever required and avoid duplicate code. One of the major takeaways was to keep in mind to design and program everything for long term so that things only need to added not changed.

Failures:

One problem I overcame was when I created a new shape for pasting the shape and for outlining the selected shapes, I had to learn to set all their required properties by creating and using a set and get methods for individual properties.

Sprint 1 and 4 were the toughest sprints to complete.

During sprint 1 I had a lot of duplicate code and getting to draw the shapes using a design pattern was very difficult. But reading the discussion forums and some research really helped me overcome these problems.

The biggest failure was during sprint 4. I tried to implement the group and ungroup functionality using composite pattern but was unsuccessful. I had to abandon using the pattern for this functionality. I was able to group and ungroup the shapes, but it does not work as expected.

Overall implementing each pattern took time and patience, since along with implementing I was also learning how java works. But since the professor basically guided us through about how various patterns could be used to implement different functionalities, it was easier to choose which patterns to use for which functionality.