Arturo Chaidez III
SE 450 Final Report

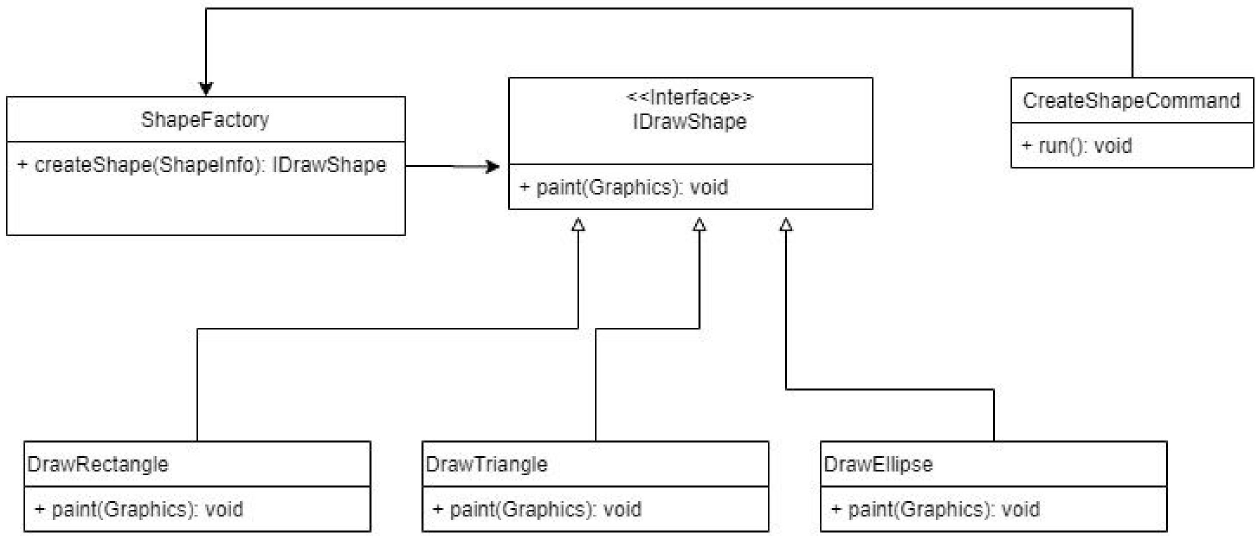
<u>List of missing features, bugs, extra credit, and miscellaneous note</u>

- Pick a shape - all three work
  - Ellipse
  - Triangle
  - Rectangle
- Pick a primary color - works
- Pick a secondary color -works
- Select shading type (outline only, filled-in, outline and filled-in) -works
- Click and drag to draw a shape -works
- Click and drag to select shapes - works, but has bugs
  - The collision aspect does not work. To select a shape, the entire shape needs to be dragged over to be selected.
  - Overall, selecting and deselecting a shape can be inconsistent.
  - When there are multiple shapes, there seems to be a bug that JPaint will only select the first drawn shape. The first shape is selected regardless of what is selected. This does not happen every time.
- Click and drag to move selected shapes -works
  - Be aware that highlighting shapes was not implemented and selecting shapes is inconsistent. Because of this, you may not be aware you did not select a shape when trying to move it.
- Copy selected shapes -works
- Paste copied shapes -works
- Delete selected shapes -works
- Undo last action -works
- Redo last action -works, but has bugs
  - After moving a shape and undoing it, you can use redo to move the shape twice. For example, if the shape was moved to the right, you can click redo to move it right twice.
  - You can redo an action multiple times in other commands as well, but very rarely.
- Group selected shapes - does not work
- Ungroup selected shapes - does not work
- Selected shapes have dashed outline -does not work

<u>Notes on Design</u>

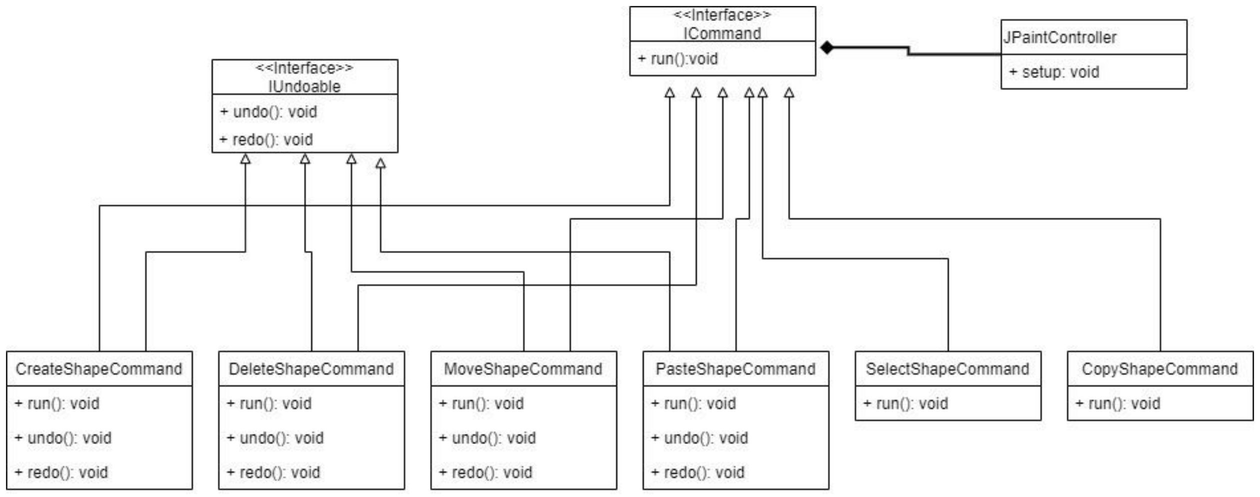
<u>Abstract Factory Pattern:</u>

An abstract factory is a creational pattern and allows the creation of any of the three shapes. Each shape has its own class. The CreateShapeCommand calls ShapeFactory to draw whatever shape was made. This pattern also allows new shapes to be added, as new shapes can be added without having to change any of the working classes in this pattern. The factory uses a switch statement to confirm which shape was drawn, so code can be easily added to ShapeFactory for new shapes to work.



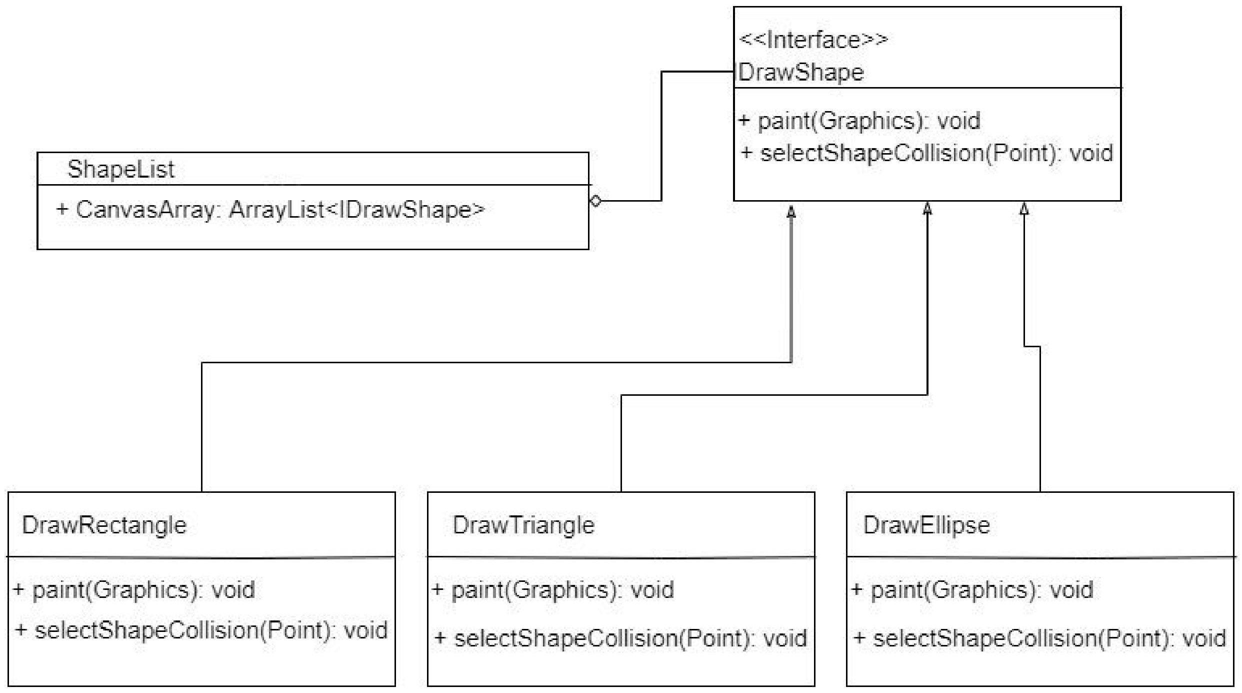
Command Pattern

The command pattern is a behavioral pattern and is used to trigger actions. There are six commands: CreateShape, DeleteShape, MoveShape, PasteShape, SelectShape, and CopyShape. They are all named after the functionality they do. Each command has a run method, but some have undo and redo methods as well. Implementing this pattern early helped in the long term, as new commands were easily added later on.



Strategy Pattern

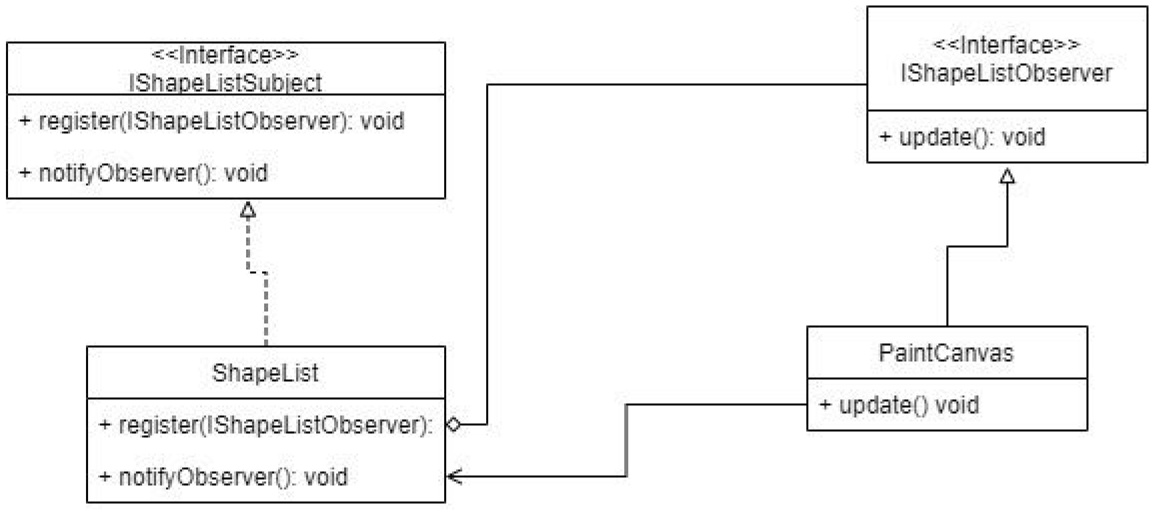
The strategy pattern behavior pattern that encapsulates behavior and does some of the algorithm. This pattern's interface is implemented by three classes: DrawTriangle, DrawRectangle, and DrawEllipse. This is where the shapes are drawn and ensures everything that is needed to make the shape. For example, the shading type is checked here. This reduces having to have a class for each shape and shading type combination.



<<Interface>>
DrawShape
+ paint(Graphics): void
+ selectShapeCollision(Point): void
ShapeList
+ CanvasArray: ArrayList<IDrawShape>
DrawRectangle
+ paint(Graphics): void
+ selectShapeCollision(Point): void
DrawTriangle
+ paint(Graphics): void
+ selectShapeCollision(Point): void
DrawEllipse
+ paint(Graphics): void
+ selectShapeCollision(Point): void

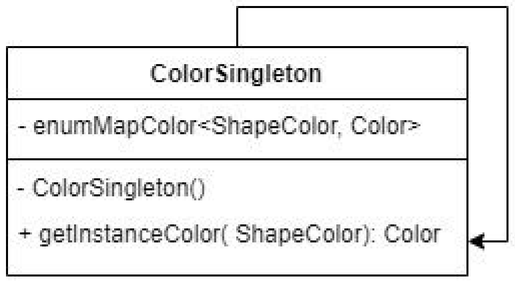
Observer Pattern

The observer pattern is a behavior pattern. This pattern keeps track of any event being done and updates others when something happens. When something is drawn, Shapelist is updated and tells the PaintCanvas to draw the shape. ShapeList contains multiple lists, so whatever is changed or added can be iterated over easily.



Singleton Pattern

The singleton pattern is a creational pattern where a single instance of an object is made and other classes can only access this object. In my earlier code, an object was being made for colors everytime a shape was made. By changing the code into singleton, only one object is made and the shapes can access what colors they need. This is way more efficient by not wasting memory space.


ColorSingleton
- enumMapColor<ShapeColor, Color>
- ColorSingleton()
+ getInstanceColor(ShapeColor): Color

Success and Failures

Overall, I was able to implement a variety of different patterns and features. However, there were also many roadblocks. Figuring out how to get a simple feature to work, such as colors and shading types, was a major hurdle. Refactoring this code into a pattern was also not easy. You not only have to refactor the code, but choosing the correct pattern was important. I was trying to figure out long term if this pattern will allow more features to be added easily, or if a future feature could even be added to this pattern. Other than the Singleton Pattern, all patterns were difficult to figure out and implement.

I am disappointed I was not able to figure out how to highlight shapes and group/ungroup shapes. It seemed classmates were able to use the composite pattern to group shapes and used wrapper patterns for highlighting, but I could not figure these out. Of course, I did start by trying to code it within other classes in an ugly way but could not get the feature to work at all. I could not get the outline_and_filled_in shade type to work at first. It was only filling in one color. I also had issues with my MoveShapeCommand in its run and redo methods. For both methods, I could not get the shapes to move. The MouseAdapters gave me issues as well, as I had to do a lot of research to understand how it works. I am also disappointed my code has a handful of bugs. However, I was able to improve selecting shapes later on. There were some other bugs I was able to resolve as well, so I can at least be positive that I fixed those.

This was a difficult project and plan to fix these last few things over winter break. As someone who has been playing video games since he was young, working on this project humbled me how hard video game development is. This project was much more simpler than any modern game or even games I played growing up. While I have zero interest in this field of computer science, I will be way less harsh on a game developer about bugs or missing features in a game!