SE 450: Summer 2019 Amy Aumpansub
Professor Jeffrey Sharpe

Final Project: JPaint Application

Introduction

The goal of this project is to utilize the knowledge of object-oriented software development and apply design patterns to build the JPaint application that contains various features. The application can be compiled and run successfully in Java 1.8. This project relies on the uses of public interface and utilizes several design patterns including Factory Method, Decoration pattern, Strategy pattern, Command Pattern, and Composition pattern.

Feature Descriptions

- Draw: The users can click mouse drag to draw shape on canvas. They can pick various shape types (Ellipse, Rectangle, Triangle), primary and secondary colors, and shading type The default values are filled-in Ellipse with blue (primary color) and orange (secondary color).
- Select: The AABB collision detection is applied to determine selected shapes when user clicks mouse on canvas. The selected shapes will be showed with a dashed outline.
- Move: The selected shapes can be move around with mouse drag. The original shape's starting points will be offset by a mouse drag distance. Its width and height do not change.
- Copy/Paste: When user select the shapes to be copied, they will be cloned by using the overridden Clone function in class Object and will be added to the list. When the paste mode is selected, the copied shapes will be pasted on the right side of the original shapes.
- Delete: The user can select any shapes to be deleted from canvas.
- Undo/Redo: The user can choose to undo and redo the previous operations.
- Group/Ungroup: The user can select multiple shapes to group and ungroup them.

Design Patterns

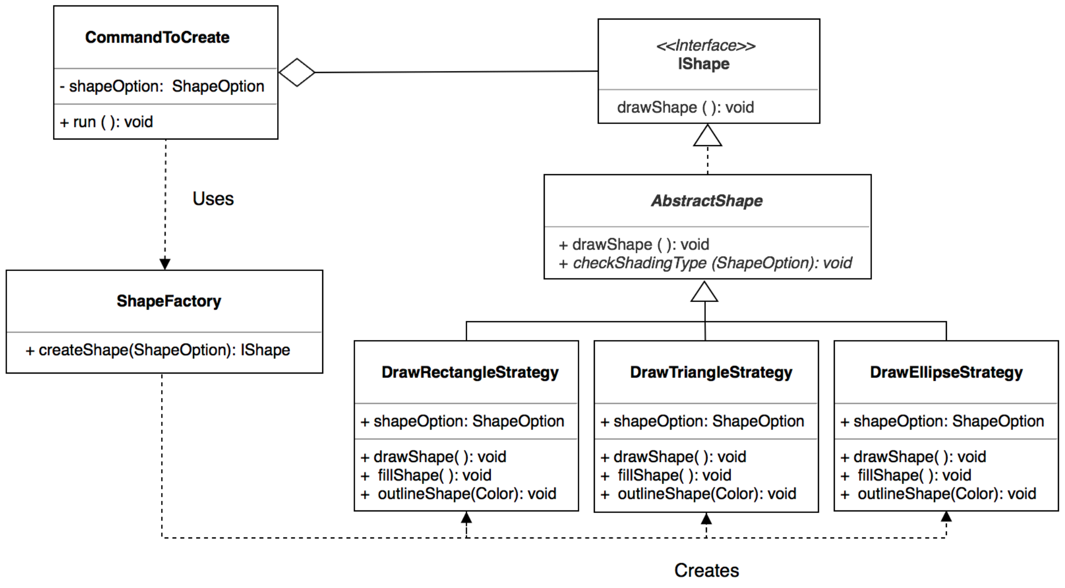
Several design patterns in creational, behavioral, and structural categories were applied to encapsulate the inside information and improve program's efficiency.

- **Creational Category:** The "factory method" is used to encapsulate the creation of related objects which are Draw Strategy of each shape type.
- **Behavioral Category:** The "strategy" pattern is applied to hide the draw method from the client. The "command" pattern is also used to encapsulate data for executing command. It contains state and command history which will be applied for the Undo/Redo commands.
- **Structural Category:** The "decorator" pattern is applied to decorate the selected shapes with a dash outline around them. The "composition" pattern is utilized for performing group and ungroup actions.

SE 450: Summer 2019 Amy Aumpansub
Professor Jeffrey Sharpe

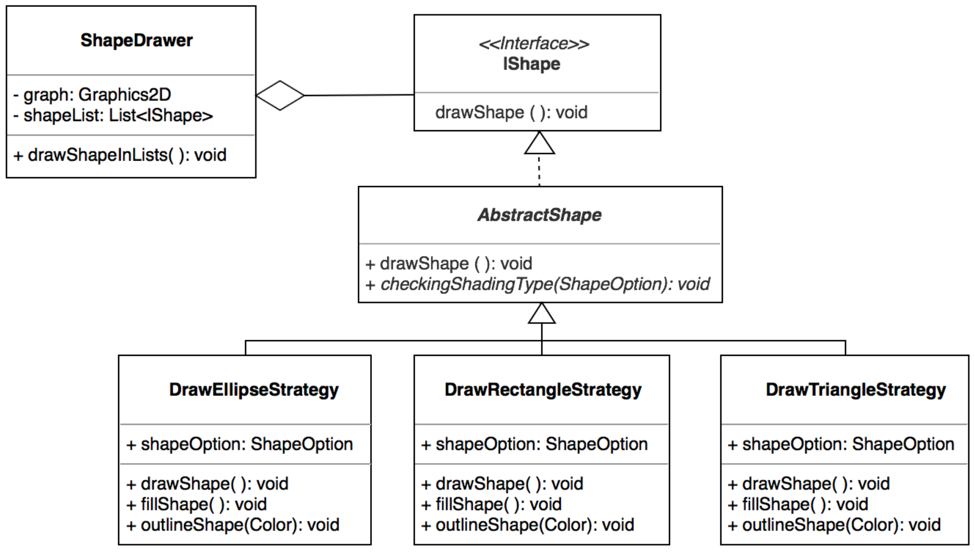
The Factory Method

The factory method is implemented in the ShapeFactory class. Using this pattern, the client "CommandToCreate" declares a new ShapeFactory object and call the CreateShape method to create and return the related draw strategy (implementing IShape) in regard to the active shape type acquired from a shapeOption class. This pattern solves the shortcomings of static factory method. The client can use Factory to create the related object at run-time without knowing the concrete type of object.



Strategy Pattern

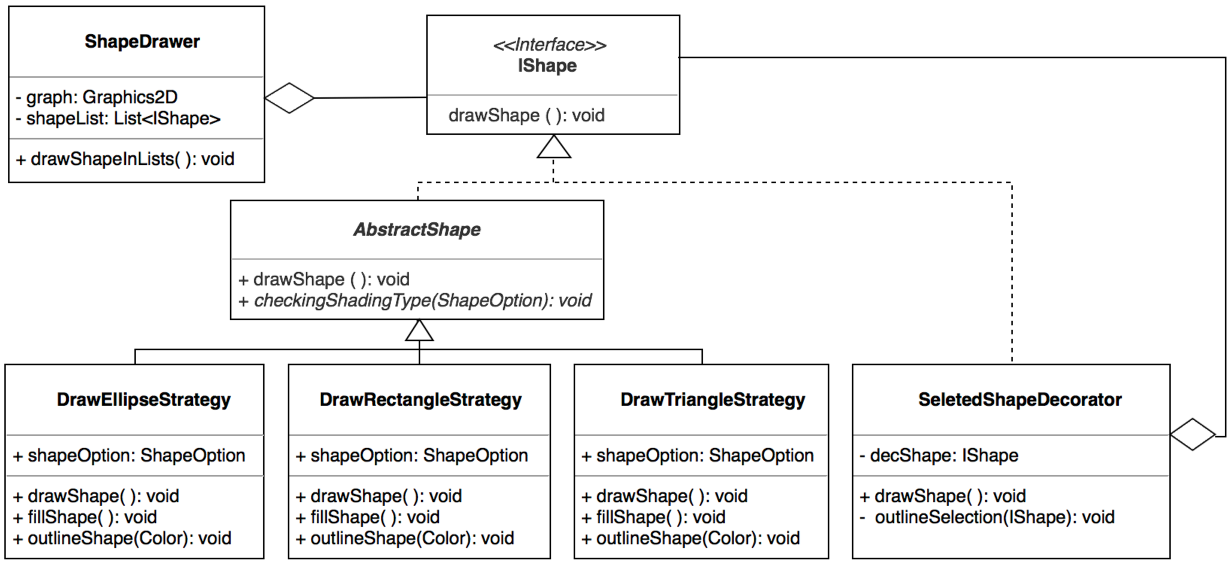
Using this pattern, we can encapsulate the draw shape implementation and hide the details of drawShape method from client. Another benefit is that it allows us to add a new strategy for a new shape type without modifying other strategies. There are 3 strategies implementing IShape interface. Each strategy contains drawShape method. The abstract shape class checks the shading types and calls 2 methods: fillShape and outlineShape regarding to shading type.



SE 450: Summer 2019 Amy Aumpansub
Professor Jeffrey Sharpe

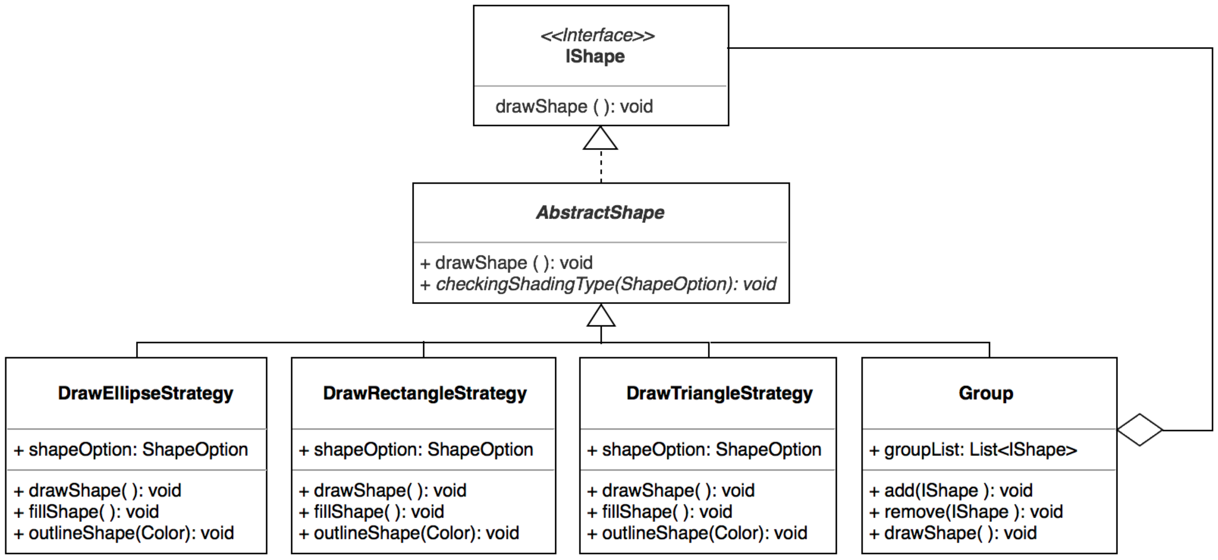
Decorator Pattern

Using this pattern, it allows us to add more functionality and format basic shapes drawn by the 3 DrawStrategy classes.The SelectedShapeDecorator class implements the same IShape interface as the subject, so it also contains the public drawShape method, which calls the private outlineSelection method to draw a dashed outline around the selected shapes.



Composite Pattern

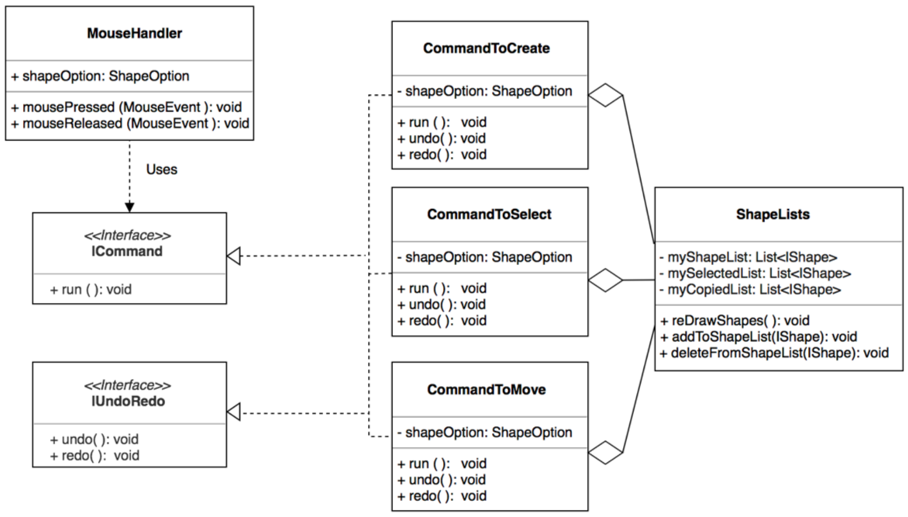
Using this pattern, the client can interact with an individual object and composition in the same manner. So, we can utilize this pattern to group shapes which composite is a group of shapes.



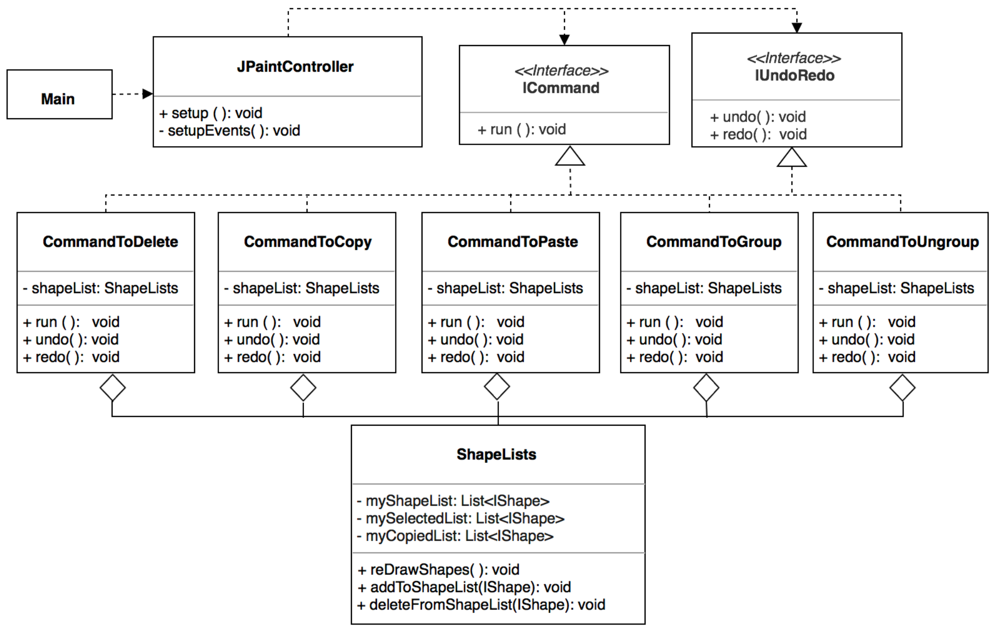
Command Pattern

Using the command pattern, we can encapsulate information as an object needed for executing the action. The program relies on command patterns (ICommand and IUndoRedo).

- <u>ICommand (Create, Select, Move):</u> When the user releases mouse, a new Command object is created in real-time in regard to the Draw, Select, Move mode that user selects on the screen. The program will execute the run method inside the Command object. For the CommandToCreate class, it will use the ShapeFactory to create the Draw Strategy.


MouseHandler
+ shapeOption: ShapeOption
+ mousePressed (MouseEvent): void
+ mouseReleased (MouseEvent): void
Uses
<<Interface>>
ICommand
+ run ( ): void
<<Interface>>
IUndoRedo
+ undo( ): void
+ redo( ): void
CommandToCreate
- shapeOption: ShapeOption
+ run ( ): void
+ undo( ): void
+ redo( ): void
CommandToSelect
- shapeOption: ShapeOption
+ run ( ): void
+ undo( ): void
+ redo( ): void
CommandToMove
- shapeOption: ShapeOption
+ run ( ): void
+ undo( ): void
+ redo( ): void
ShapeLists
- myShapeList: List<IShape>
- mySelectedList: List<IShape>
- myCopiedList: List<IShape>
+ reDrawShapes( ): void
+ addToShapeList(IShape): void
+ deleteFromShapeList(IShape): void

- <u>ICommand(Copy/Delete/Paste/Group/Ungroup):</u> Using this pattern, we can add the command object which contains all necessary information needed to execute the action into the command history class, so we are able to undo or redo the previous operations.



SE 450: Summer 2019 Amy Aumpansub
Professor Jeffrey Sharpe

Successes and Failures

This project is very useful and challenging. It took me time to understand the linkages among each java class and to plan on how I would apply design patterns to my project. This course is the first software engineering course that I have ever took, so it is my first time to create an application related to the object-oriented software development. I found the implementations in Sprint2 to be the most challenging part of this project. I updated my codes and refactored my programs in the Sprint 2 and found that when I refactored the static factory to the factory method, I need to declare a new method to set a new point in my program. From visiting your office hours, I learned how to solve this issue and learned how to use breakpoint and debug my program in eclipse. I found that the eclipse debugger is really useful especially when we need to identify and fix the errors.

Overall, my program can implement all features such as draw, select, move, delete, undo/redo, group/ungroup and copy/paste, but I will need to further refractor my codes to implement composite pattern. I was able to apply the knowledge I acquired in this course to the final project. I enjoyed working on this project.

For the future studies, I will update my codes for composite pattern and further refractor my program to improve its efficiency.