**SE 450 Object-Oriented Software Development**

**Final Report of J-Paint Application**

**Harry Chen**

**Appendix**

1. Abstract.
2. List of Features.
3. Note on Design.
4. Success and Failure.

**Part I: Abstract**

J-Paint is a Java Swing application which has functionality of painting shape on canvas. Four design patterns are being applied to implement the application. This final report basically summarized how specific software development problems are solved using those patterns and supportive ULM diagram are provided to explain how classes arranged and the relationship between them. This Final report compose three parts which includes “List of Features”, “Note on Design” and “Success and Failure”, each of them will be discussed in their own section.

**Part II: List of Features.**

Features that don’t work are mark with \* in red color.

Features that count as extra credit are mark with \* in green color.

1. **Draw** (When mouse-mode is in draw mode)
2. Draw “Rectangle”, “Ellipse”, “Triangle”
3. Draw shapes that are with color “Filled-in” or “Outline” or “Filled-in & Outline”.
4. Draw shapes with primary color of outline and secondary color of filled-in.
5. **Select** (When mouse-mode is in select mode)
6. Mouse clicked inside shape/shapes, shape/shapes got selected.
7. Mouse clicked outside shapes, all selected shapes got deselected.
8. **Move** (When mouse-mode is in move mode)
9. Click inside any shape to select, drag the shape to new position, and when mouse released the shape got move to the new position.
10. \* Select multiple shapes first and move them simultaneously.
11. **Copy & Paste**
12. Selected shapes got copy and new shaped got pasted in the right-bottom position of the old shape.
13. **Delete**
14. Selected shapes got deleted.
15. **Undo & Redo**
16. Undo & Redo new created shapes.
17. Undo & Redo deleted shapes.
18. Undo & Redo new pasted shapes.
19. \* Undo & Redo moved shapes to original position.
20. **\* Save (export) & Read (import) files to canvas.**
21. Export drawn shapes with reserved shape-type, position, color to file.
22. Retrieve the shapes that was inside the shape list and show them on canvas again.
23. **\* Keyboard shortcut for copy, paste, delete, undo and redo.**

* Copy (⌘ + C)
* Paste (⌘ + P)
* Delete (⌘ + Delete)
* Undo (⌘ + U)
* Redo (⌘ + Redo)

1. **\* Selected Shapes can change primary color, secondary color and shape shading type.**

* Select your target shapes
* Change the application state in dialog, selected shape got updated

1. **\* Unit test for create, copy, delete shapes command.**

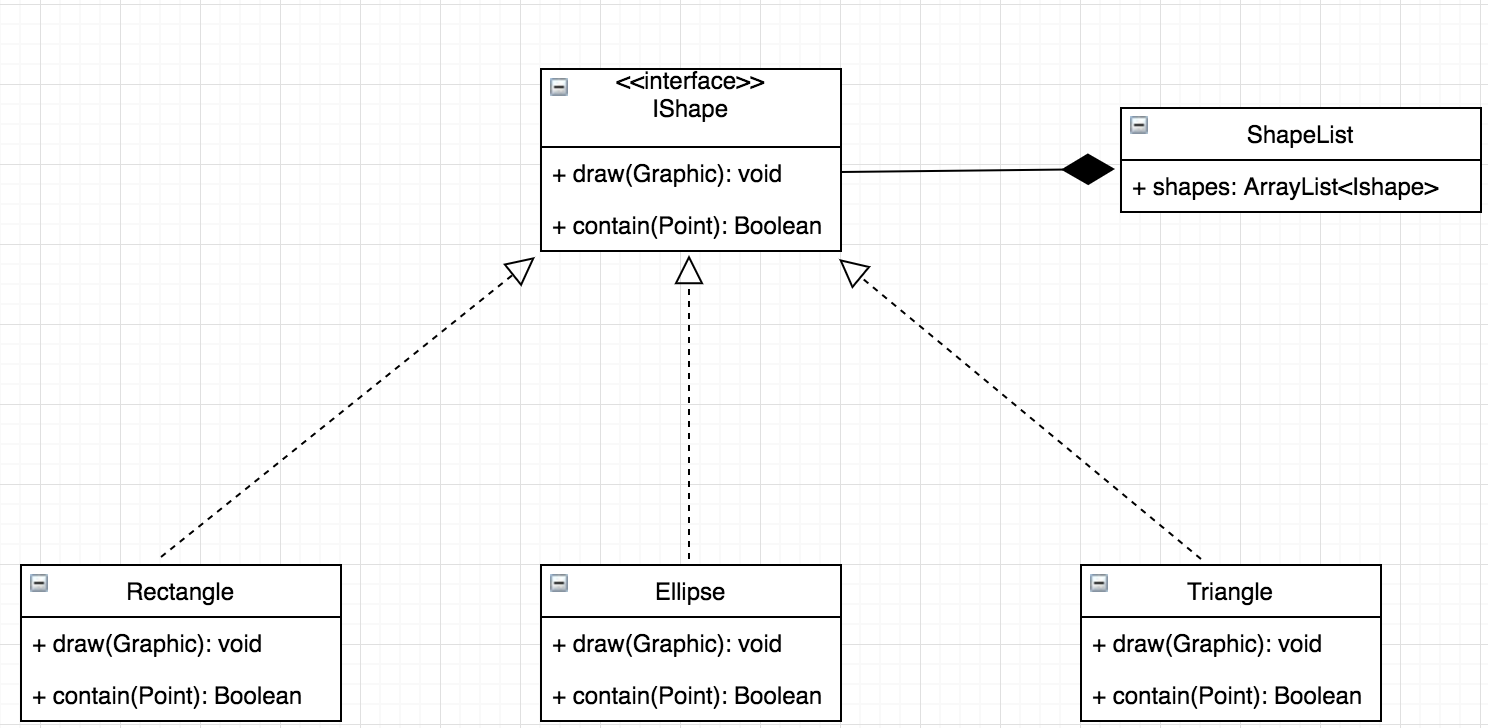
**Part III: Note on Design**

In order to develop a robust application and write maintainable code, several software design patterns were use during the development including Strategy Pattern, Observer Pattern, simple factory pattern, Command Pattern, and Adaptor Pattern. Besides the patterns mentioned above, there are several java build-in functions use other patterns such as iterator for collection object, proxy pattern for serializable and stub object, in J-Paint application, I uses iterator for my Shape-List and I use serializable for any class related to save and read functionality.

The following paragraphs will explain how each of the 5 patterns are used in the application, what issue does the pattern solved, and what relationship between the classes that involved in the pattern design. UML diagram will be provided to support explanation.

1. **Strategy Pattern**

Canvas should be able to draw different type of shapes with different attributes applied, strategy pattern provides an excellent mechanism to separate the aspects that vary from what stay the same. Another advantage of strategy pattern is object are created by interfaces rather than concrete object, that way, shape classes won’t need to know any of the implementation details for their own behaviors.

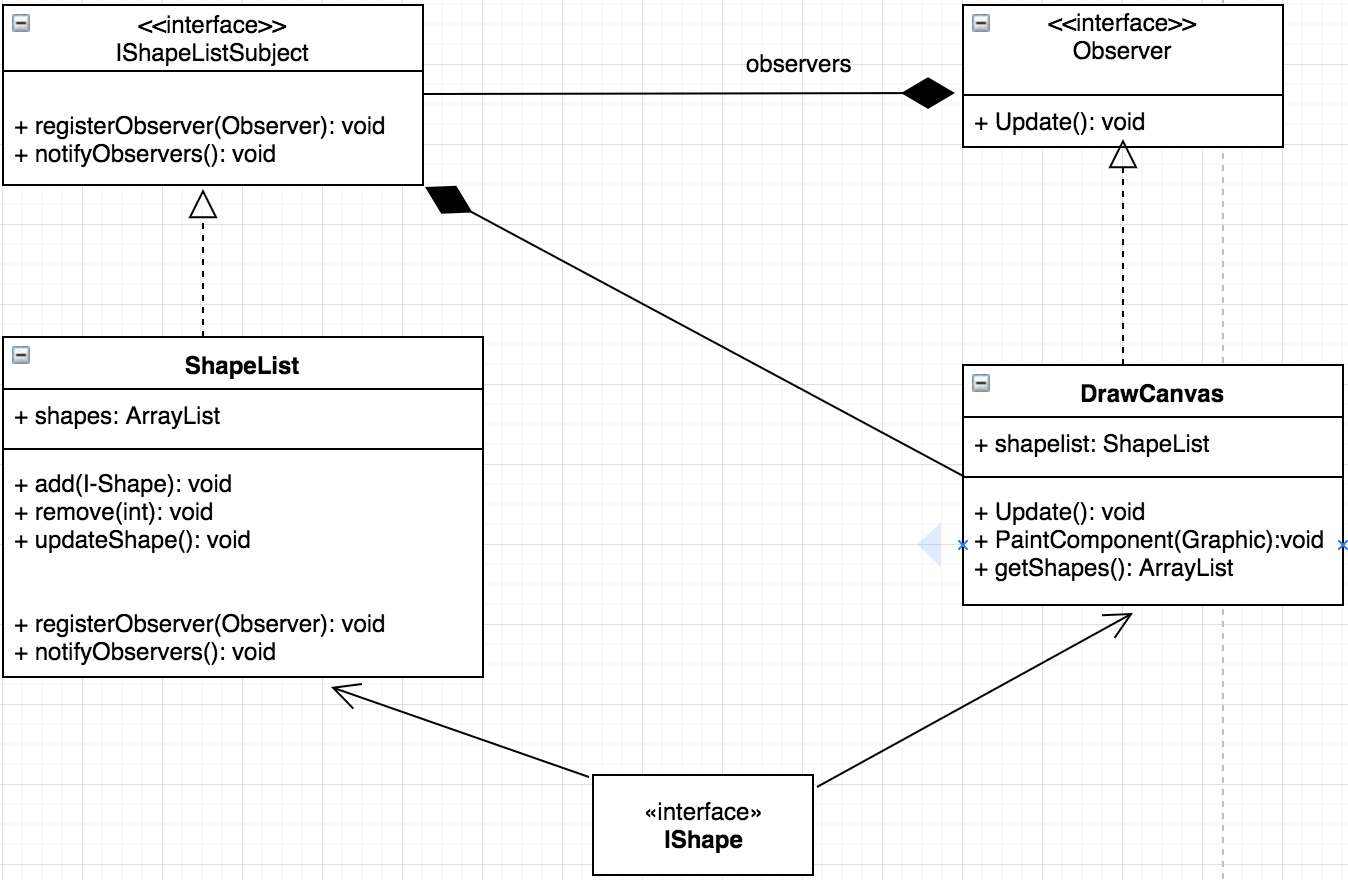


1. **Observer Pattern**

Observer are used extensively in the development, it sets up a communication channel between subject and subscriber, whenever certain function got called in subject, it notifies its subscriber to update their status.

In my application, there are three places used Observer Pattern.

1. Canvas subscribes to Shape-List Object, whenever new shape being added or removed to Array, Shape-List notifies Canvas to repaint shapes in the array so that new shape can show.
2. Main-Frame subscribes to Application-State, whenever mouse-mode changed, Application-State notifies Main-Frame to change its Mouse-Adaptor.
3. Shape-List listening to Application-State, whenever set-Active function got called, it notifies the Shapes in Shape-List changed its attributes. Since canvas are listening to shape list as well, the shape on the canvas will change it attributes accordingly.

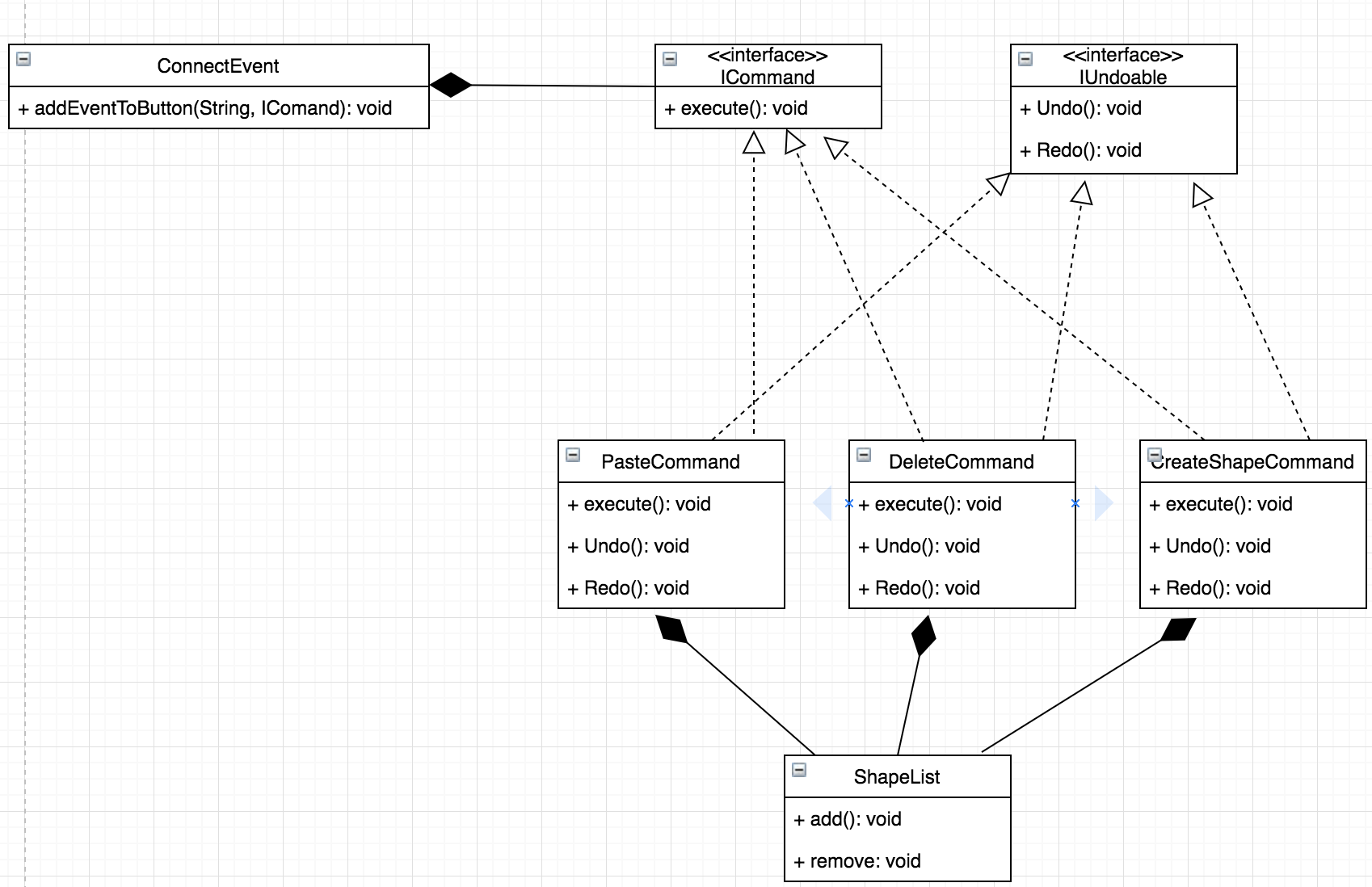


1. **Command Pattern**

Command Pattern are used for binding opening dialogs and action event to certain button on canvas. The command pattern decouples an object making a request from the one that knows how to perform it. Three essential class should be included in the design.

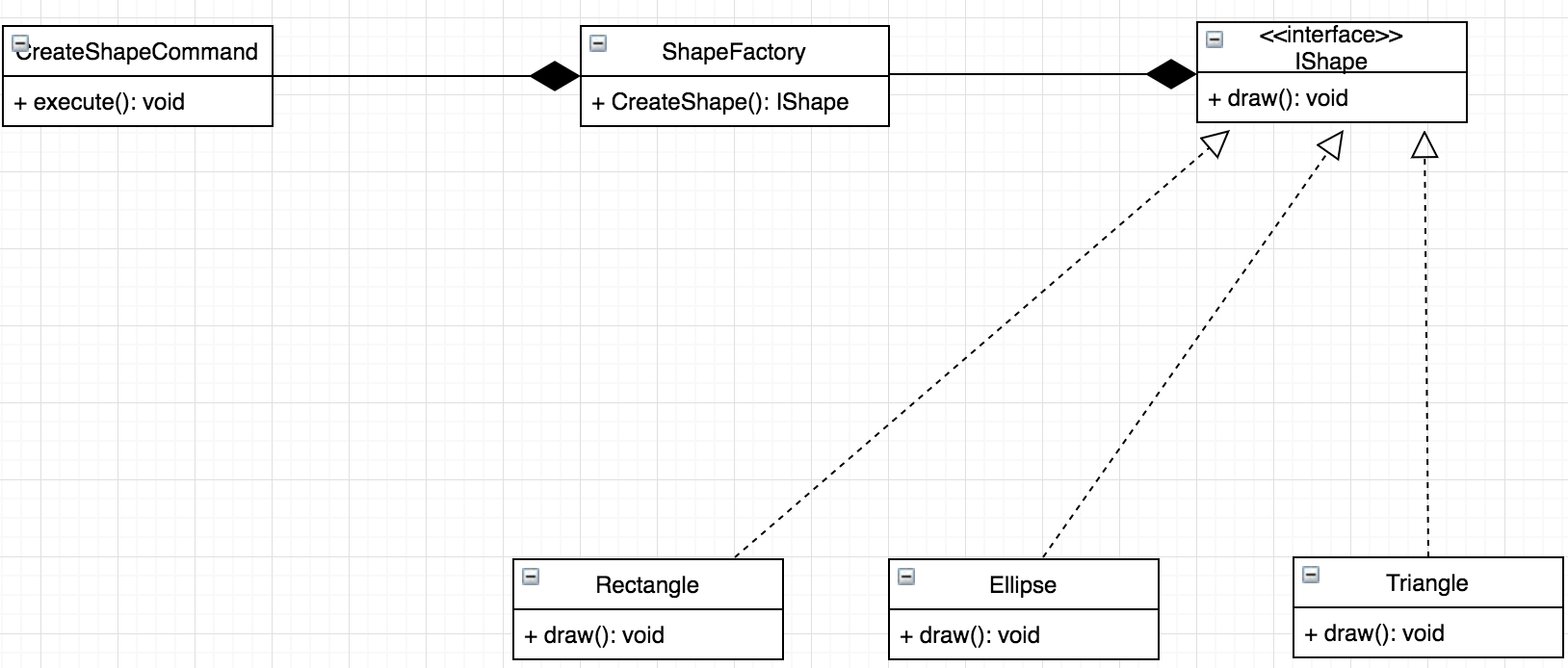
1. Command Object: This object is at the center of this decoupling and encapsulates a receiver with an action. In J-Paint, GUI class serves as the command object.
2. Invoker: An invoker makes a request of a Command object by calling its execute method, which invokes those actions on the receiver. In J-Paint, Connect-Event class serves as the invoker.
3. Receiver: Receivers are the actually object which implement the execute function. In J-Paint, 4 commands object include create, copy, paste, delete serve as receiver.

Besides those mentioned above, commands can also support undo and redo by implementing two method that restores the object to its previous state before the execute method was last called.



1. **Simple Factory Pattern**

Instead of flood the class with numerous if else statement, factory pattern was used to create shape object. Once we have a simple create-shape factory, any time client asks to create a shape it asks the factory to create one. Gone are the days when the create-Shape method needs to know whether the shapes are rectangle, ellipse or triangle. Factory method make sure the code open to extension which means when more types of shape coming in future, program will be easier to refactor.



**Part IV: Success and Failure.**

J-Paint application uses basic object-oriented techniques which include abstraction, encapsulation, polymorphism alone with 5 solid software design principles, several design patterns are well considered to choose to apply when developing the application. In the end, the application can compile and behavior correctly at run time.

There are still lots of improvement and refashion can be done of the code, for example, observer pattern might not be suitable for the shape changing its primary color and other attributes, instead, State pattern can be a good replacement of observer pattern in this case. There are serval parts of the code violate single responsibility principle, for example, my Main-Frame class should only handle the creation of GUI window, anything else like adding adaptor or other callback functions should be separated and encapsulated. Also, there are multiple classes are tightly coupled with each other which is not healthy for the future development.