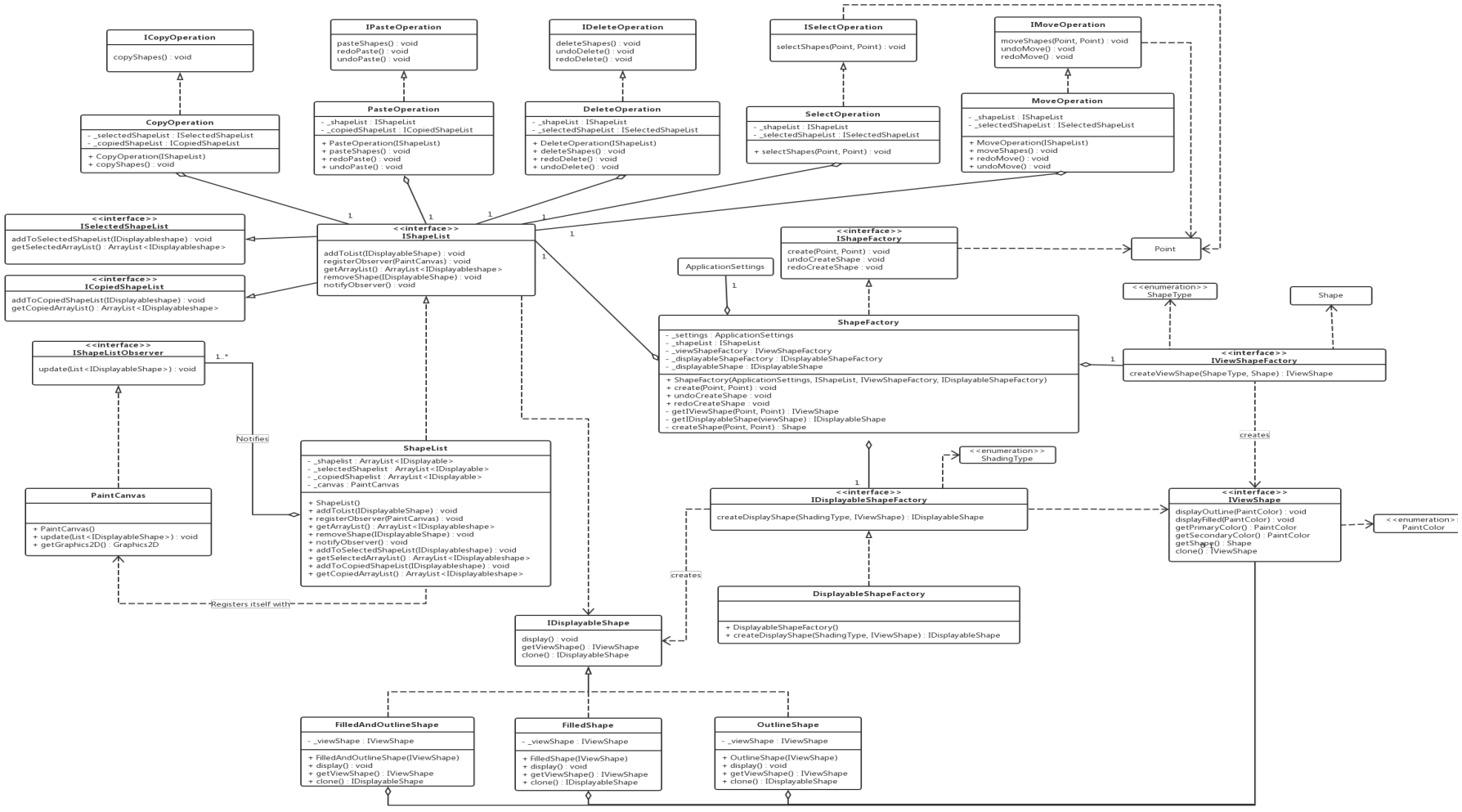
SE450 JPAINT – REPORT

Kaicheng Yan
1870810

1.Class Diagrams For The Model Classes



If the UML is not clear enough, please open the original picture or click: [Online View](#) .

2.Time Recording Journals

Date	Logs
Jul 17	Use Github as source control tool. Read start code, figure out how the start code works.
Jul 18	Add and implement events for the rest buttons (choose primary color, choose secondary color, choose shading type and choose mouse mode).
Jul 19 – Jul 20	Watch tutorial about MouseListener and write a simple demo about it, including add mouse and mouse motion event to canvas.
Jul 21	Try to pass the IPaintController to GuiWindow for getting the current values of settings.
Jul 24 - Jul 26	Implement the algorithms of drawing ellipse, rectangle and triangle.
Jul 27	Review the code that we told at lecture
Jul 28 – Jul 31	Refactor the code, including using observer pattern to draw mode, abstract factory pattern to create draw shapes, command pattern to create shape.
Aug 1 – Aug 3	Think about the pattern that uses in the mouse mode. Decide to use observer pattern and command pattern to the mouse mode.
Aug 4 – Aug 6	Work on the implements of select and move mode and fix several bugs.
Aug7 – Aug 9	Use command pattern to implement the undo and redo command. Using Singleton Pattern and abstract factory to refactor the draw and move mode's code, in order to make these modes undoable.
Aug 9 – Aug 11	Work on copy, paste and delete functionalities and make them undoable.

Aug 12 –Aug 13	Implement the keyboard shortcuts on copy, paste, delete, undo and redo.
Aug 14	Refactor the algorithm codes in each command's run method, encapsulate the concrete code.
Aug 15	Draw the class diagrams for model classes.
Aug 17 – Aug 19	Work on report of the project.

3.Time Summary

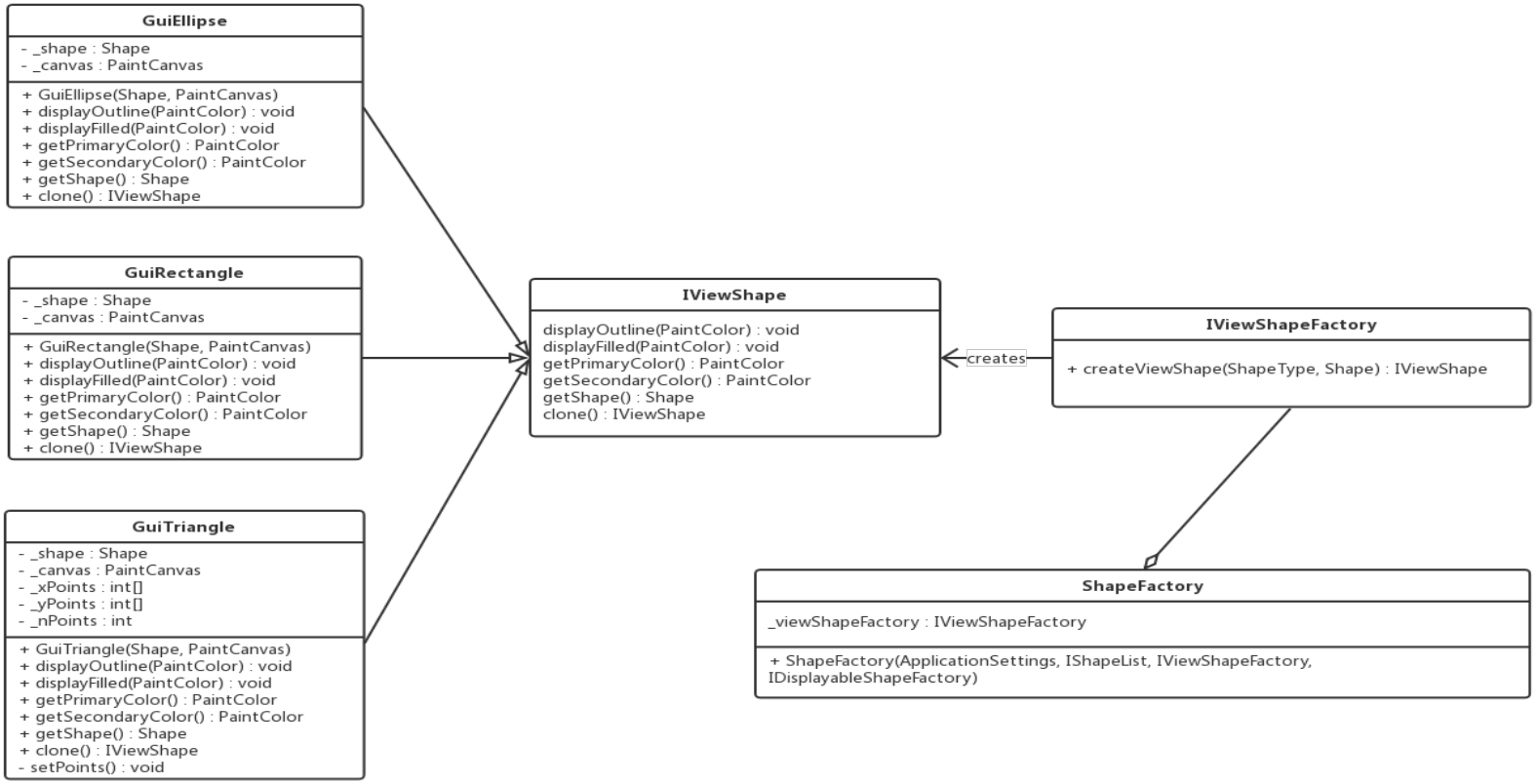
Week	1	2	3	4	5	Total(h)
Design(h)	7	7	7	5.5	4	30.5
Code(h)	4	8	9	7	6	34
Big bug(h)	1	2	2	1.5	4	10.5

*Not including the cost of report.

4.Notes On Patterns

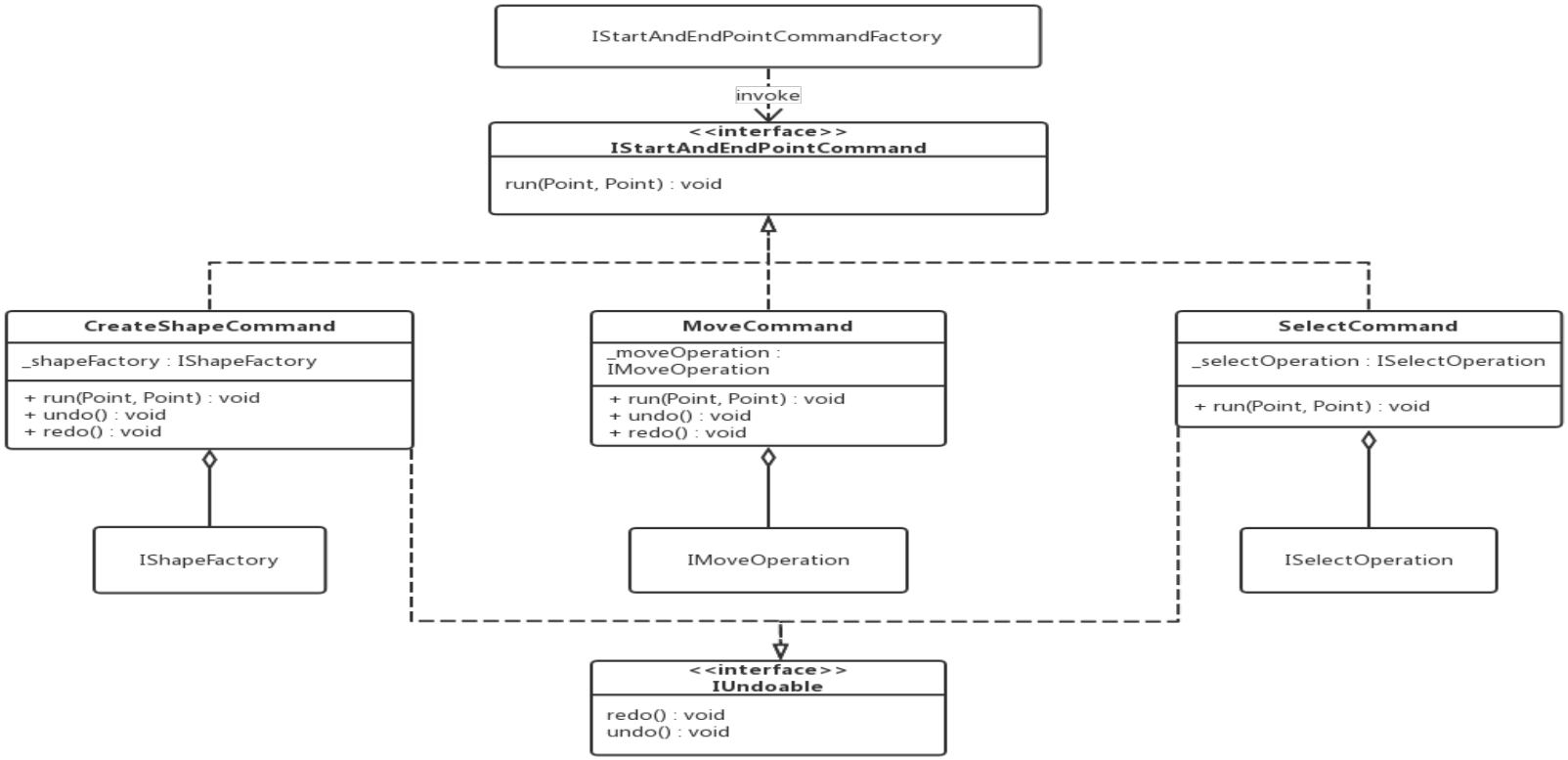
(a). Strategy Pattern

Through strategy pattern, I can encapsulate the algorithm and do part of the algorithm. In my ViewShapeFactory and displayableShapeFactory, they create different kind of shapes and shading types according to ShapeType or ShadingType. At the same time, the return type is the same for creating different shapes, because these concrete classes implement the same interface. When I am going to add a new shape, I only need to create the class that implement the interface and add the new shape type to the if…else… statement in creation. Strategy pattern makes the code more flexible and easy to extend new requirements.



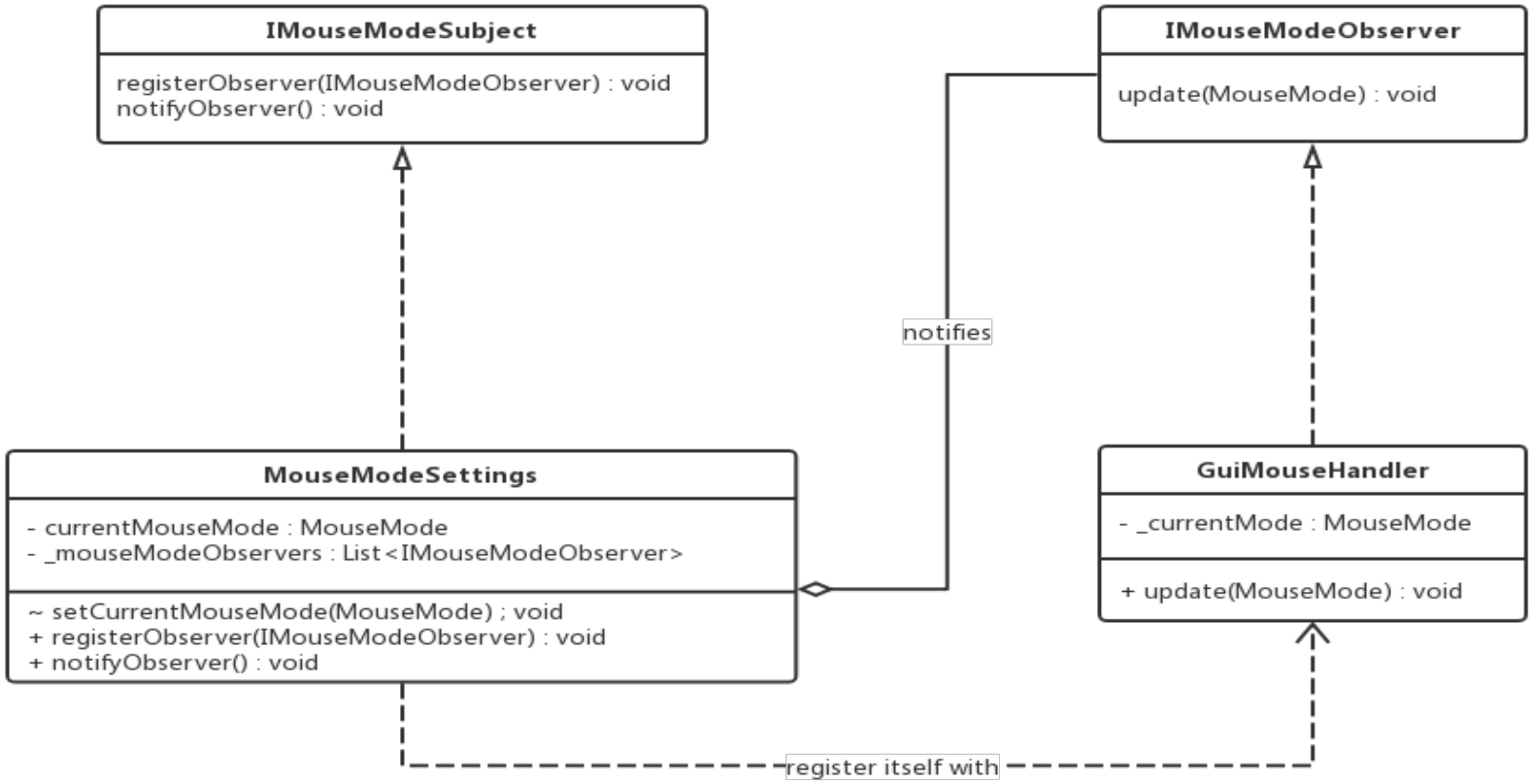
(b). Command Pattern

In the project, I use the command pattern in several functionalities. At first, I use it to refactor codes of adding events for each button. After Using the command pattern, I can pass a more abstract type(ICommand) which implemented by the concrete command classes into the addEvent() method. Inside the method, the button can listen the specific run method from different command classes. The command pattern encapsulates the messages, and makes the code more readable. The more important reason for using command pattern is it can add ability to UNDO operations and store the DO and UNDO operations in the same object. The draw, move, paste and delete operations should be undoable, so that I add these operations(commands) to the command history and I can track user's action track. One common use is using the command pattern to kick off the same action in many places, so I use the same command for both buttons' events and keyboard shortcuts, including copy, paste, delete, undo and redo command.



(c). Observer Pattern

I use observer pattern in two different operations. The first is used to implement displaying the shapes on the canvas. The observer is PaintCanvas and the subject is ShapeList. Once we create a new shape and add it to the ShapeList, the ShapeList will call the PaintCanvas updates the shapes on the canvas. The other functionality we use observer pattern is getting current mouse mode. The observer is GuiMouseHandler and the subject is MouseModeSettings. When we click the ok button in the dialog of “CHOOSE MOUSE MODE”, the new mouse will be set. Once we set the new mode, the subject will tell the observer to update the mode to current mode. Observer pattern is a subscribe pattern, the object subscribes to an object to be automatically notified when the event occurs.

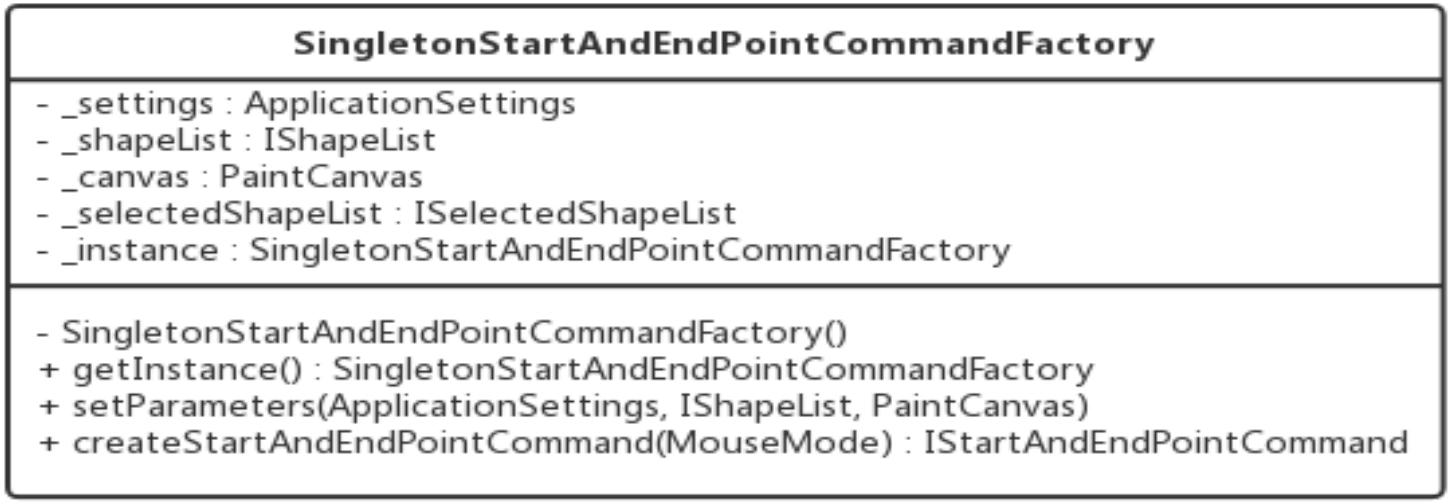


(d). Abstract Factory Pattern

Abstract factory pattern is a great way to decouple the relations from Model, Controller and View. In my project, IViewShapeFactory, IShapeFactory, IDisplayableShapeFactory, IStartAndEndPointCommandFactory and ICreateKeyboardCommandFactory use this design pattern, then I can create ViewShape, DisplayableShape and IStartAndEndCommand(draw, select and move) without coupling the Model and View. Without the abstract factory pattern, we are going to violate the MVC framework.

(e). Singleton Pattern

I combine abstract factory pattern, strategy pattern and singleton pattern together in SingletonStartAndEndPointCommandFactory which is used to create DrawShapeCommand, SelectCommand or MoveCommand. The StartAndEndPointCommand factory should be a single instance. I use lazy loading to implement the Singleton and make the constructor private. Inside the singleton pattern, I have a method called setParameters to set the parameters for create these commands. Have publicly accessible static method called getInstance for getting the object or calling methods on the singleton object.



5.Successes and Failures

During the whole project, when I work on how to change the mouse mode, at first time, I think about combining the state pattern and the observer pattern. I make GuiMouseHandler as the observer and MouseModeSettings as the subject. In the GuiMouseHandler, I use state pattern to handle different mouse mode. Once the mouse mode changes, it will automatically notify the mouse handler to change the current state. When I implement my idea, I find that the state pattern is not suitable for GuiMouseHandler. The state has exclusive reference and should be created inside the state pattern. If I create new state (DrawShapeCommand, selectCommand and MoveCommand) in the GuiMouseHandler, I have to pass ApplicationSettings, ShapeList and PaintCanvas into the GuiMouseHandler through the constructor. So, I abandon the state pattern and use command pattern instead of it.

When I work on Undo and Redo operations for draw and move shapes, I have to create multiple DrawShapeCommand and MoveCommand to track the user's action. According to the former design, I create and move shapes only through a single command. I decide to change the way to create these commands and refactor the creation codes. Factory pattern is a suitable way to create objects and I want to decouple the Model and View, so I choose abstract factory pattern to implement this operation. I also realize that SingletonStartAndEndPointCommandFactory only need one instance, and use the singleton pattern to it.

ShapeList has three fields whose type are both ArrayList: _shapelist which stores new shapes, _selectedShapelist which stores selected shapes and _copiedShapelist which stores copied shapes. ShapeList more likes a database and stores all shape data. ShapeList implements the IShapeList and IShapeList extends ISelectedShapeList and ICopiedShapeList. The advantage of this structure is when I need to use several lists in a class or method, I only pass the ShapeList(IShapeList) instead of several lists. It encapsulates the detail of lists. Selected shapes and copied shapes have own methods, if they both implements IShapeList, it will cause a fat interface problem. That's why IShapeList extends ISelectedShapeList and ICopiedShapeList.

When I finish the operations of keyboard shortcuts, I try the shortcuts several times. I find that after I click the button, the keyboard shortcuts do not work. If I do not click buttons, shortcuts work well. After debugging codes, I figure out the reason. I only add the keyboard listener to GuiWindow, when I click the button, it focuses on buttons. As a result, I add the keyboard listener to every button and fix this bug.