Dhruv Kore
SE450
Summer 2018

1) List of Features:
    a. Pick a shape
    b. Pick a primary color
    c. Pick a secondary color
    d. Click + drag to draw a shape
    e. shading type (outline only, filled-in, outline and filled-in)
    f. Select shapes
    g. Click and drag select shapes
        i. Note: There is no transparent box indicating what is being selected
    h. Copy
    i. Paste
    j. Delete shape
    k. Undo
        i. Note: Undoing a delete command works once but there after does not
    l. Redo
        i. Note: Redoing a delete command works once but there after does not
    m. Move shape

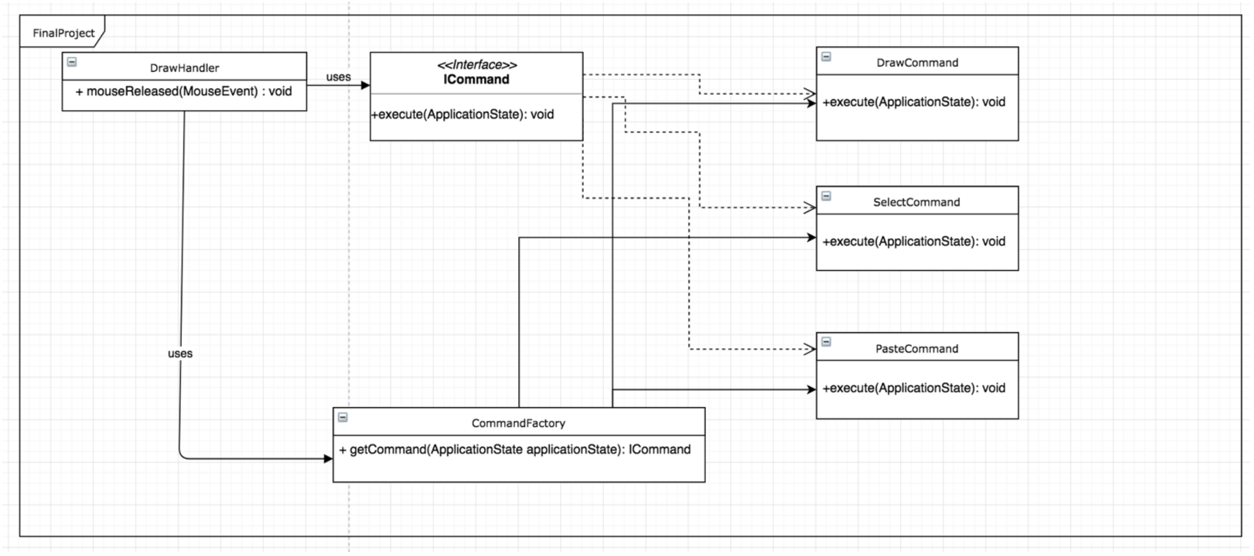
Dhruv Kore
SE450
Summer 2018

2) Notes on Design

Dependent on the activeStartAndEndPointMode, the mouse click handler needs to behave differently. Instead of a switch statement, the Command design pattern is used so that the Mouse Handler does not need to know what type of command to execute. Commands can also be more easily be added to this application because of the ICommand abstraction.

The mouseReleased event gets the appropriate instance required and refers to it as ICommand. DrawCommand, SelectCommand, and PasteCommand inherit from ICommand and are created by the CommandFactory. The mouseReleased function just needs to call execute and pass in the ApplicationState required.

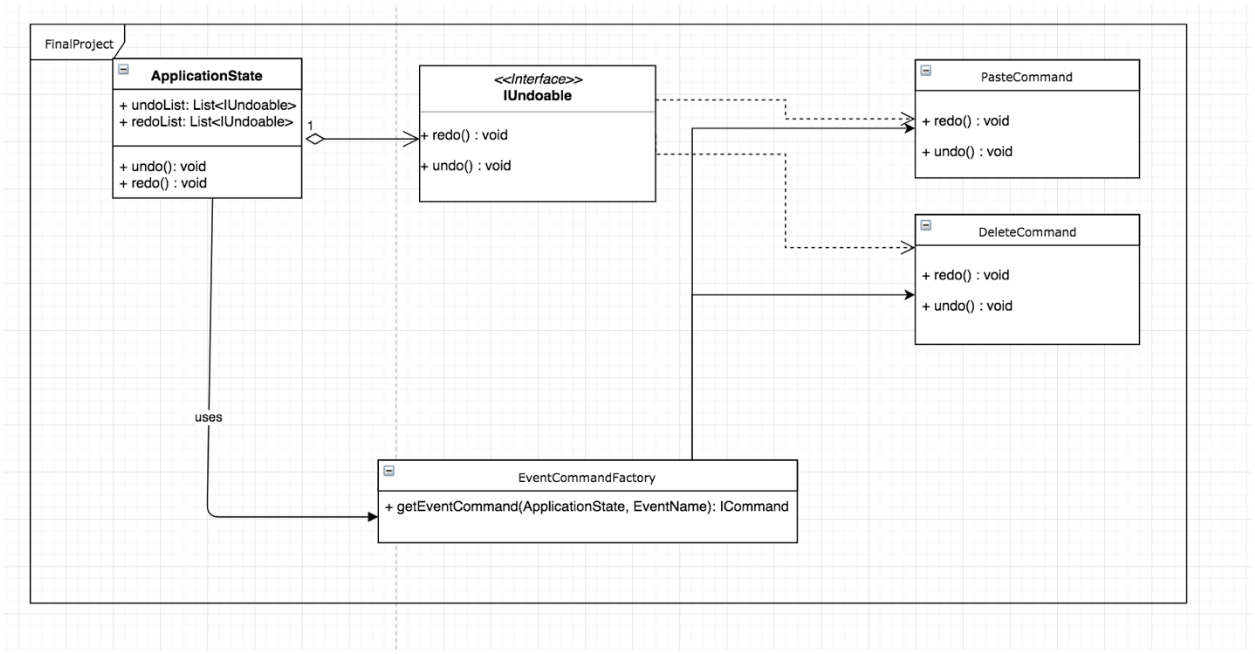
Command Design Pattern for MouseEvent Commands. DrawHandler Shown


FinalProject
DrawHandler
+ mouseReleased(MouseEvent) : void
uses
<<Interface>>
ICommand
+execute(ApplicationState): void
DrawCommand
+execute(ApplicationState): void
SelectCommand
+execute(ApplicationState): void
PasteCommand
+execute(ApplicationState): void
uses
CommandFactory
+ getCommand(ApplicationState applicationState): ICommand

Dhruv Kore
SE450
Summer 2018

Dependent on the history of commands run, undo and redo should be able to undo and redo commands. To put commands in the same list and be able to undo and redo in the correct order, the PasteCommand and DeleteCommand inherit from IUndoable. The ApplicationState gets the instance of IUndoable which gets stored in a list within the undoList and redoList Objects. With this design pattern, the undo and redo commands can be run without knowing what command is being undone and redone. More commands can also be implemented to be IUndoable without having to change the undo() and redo() implementation within ApplicationState by very much.

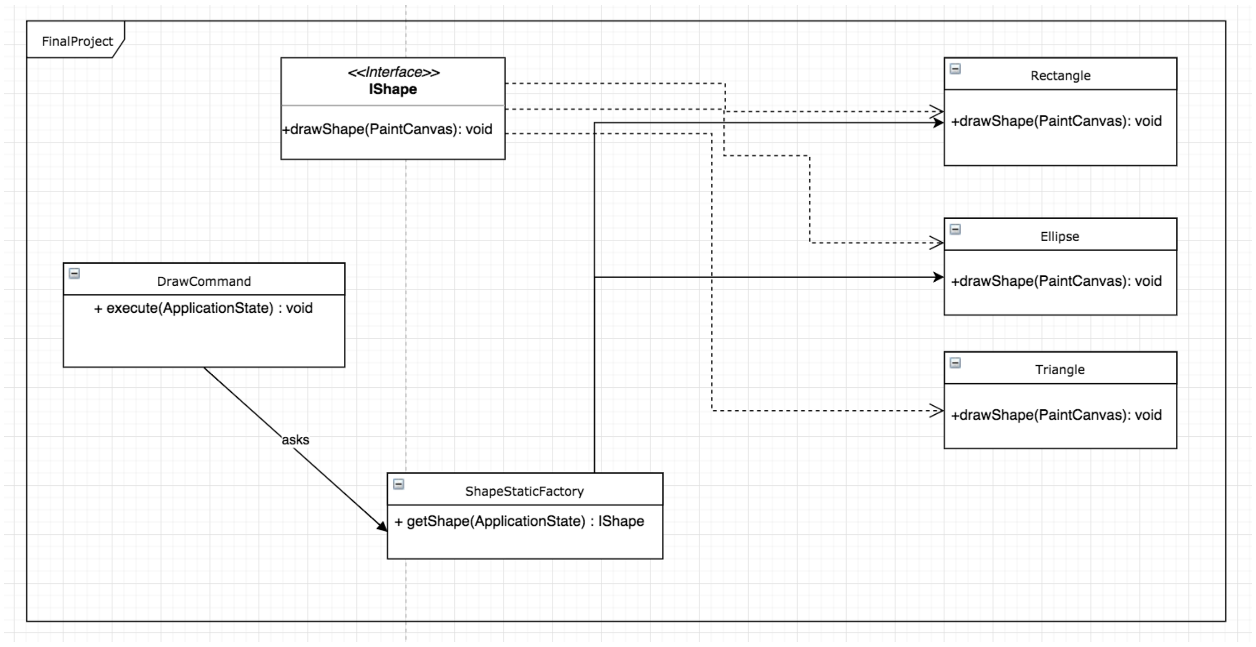
Command Pattern for ButtonEvent Commands Shown



Dhruv Kore
SE450
Summer 2018

The DrawCommand needs to draw shapes when the draw mode is active. In order to reduce tight coupling, the IShape interface is created. The ShapeStaticFactory is used by DrawCommand to create the correct shape instance. Thereafter, it only needs to call drawShape while passing in the PaintCanvas as an argument. The static factory contains all the switch statements so that it is easier to implement a new shape if in the future it is required. The new shape would be created by the ShapeStaticFactory and also inherit from IShape. It would be referred to as IShape every time after creation.

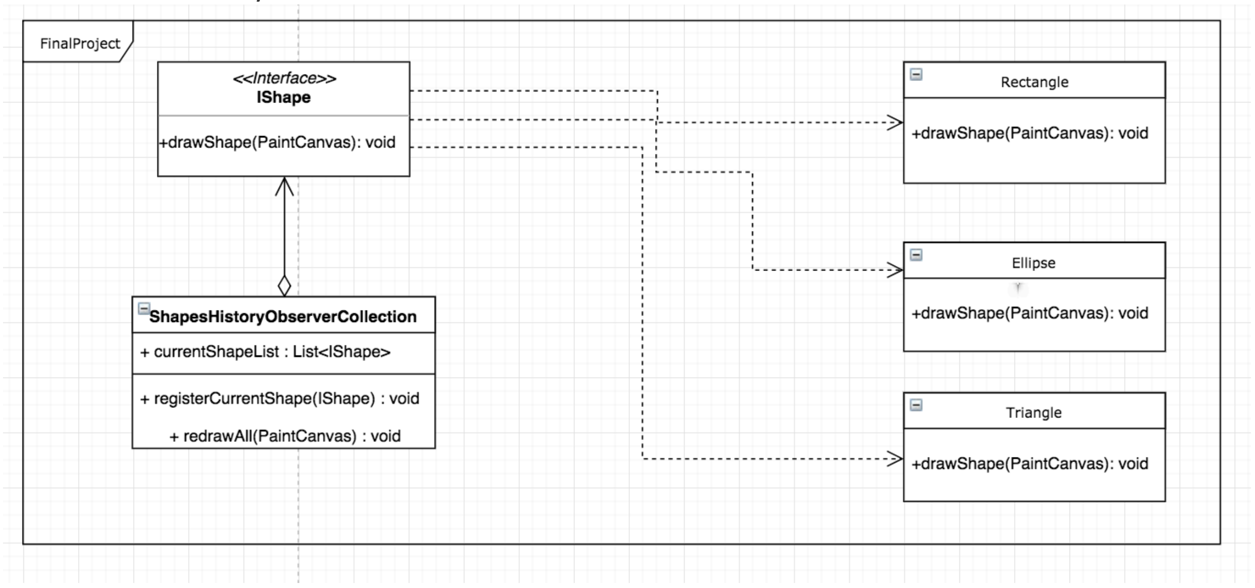
Static Factory for Shapes


FinalProject
<<Interface>>
IShape
+drawShape(PaintCanvas): void
Rectangle
+drawShape(PaintCanvas): void
Ellipse
+drawShape(PaintCanvas): void
Triangle
+drawShape(PaintCanvas): void
DrawCommand
+ execute(ApplicationState) : void
asks
ShapeStaticFactory
+ getShape(ApplicationState) : IShape

Dhruv Kore
SE450
Summer 2018

The observer pattern makes it easy to inform all the shapes created to be re-drawn. When there is a modification made, the ShapesHistoryObserverCollection has a redrawAll method that clears the PaintCanvas and loops through all the current shapes. The drawShape is then called on all the IShape references contained within currentShapeList.

Observer for History and Redraw


FinalProject
<<Interface>>
IShape
+drawShape(PaintCanvas): void
ShapesHistoryObserverCollection
+ currentShapeList : List<IShape>
+ registerCurrentShape(IShape) : void
+ redrawAll(PaintCanvas) : void
Rectangle
+drawShape(PaintCanvas): void
Ellipse
+drawShape(PaintCanvas): void
Triangle
+drawShape(PaintCanvas): void

Dhruv Kore
SE450
Summer 2018

3) Successes and Failures
    a. Successes
        i. Almost all of the design patterns in the class were revisited to see if they would fit into the requirements
        ii. The implementation of the Command pattern helped very much with the undo and redo. It also helped very much with the different logic associated with the mouse click events.
        iii. The cloning of shapes was required for the copy and paste. This was integrated correctly with the undo and redo buttons.
        iv. Deleting selected shapes was simple enough but the undoing and redoing of the shapes became easy due to the ICommand interface. The ICommand instance stored for DeleteCommand, would store the deleted shapes. These shapes would then later be added back to the currentShapes list when the Redo is invoked.
        v. Due to the enormity of the project, a lot of fundamentals of OOP were used and refreshed. It is something that can be used as a portfolio piece.
    b. Failures
        i. There are some minor bugs that could not be implement in the given amount of time
        ii. For the shapes, the initial idea was to implement a flyweight but this would not work. The pattern had to be adjusted to use Static Factory instead
        iii. Incorrect implementation of moving contributed to a lot of time in this project. To correct it was troublesome. In the end, it correctly works.
        iv. This project was long and took a lot of time. This is as much of a failure due to a bit of procrastination as a success due to the amount learned.
        v. Deeper into the project, it was apparent that dependency inversion was important. Tight coupling prevented the addition of additional features in the features list mentioned above. After the dependency inversion principal was exercised, everything flowed into place and things were more easily extendable.