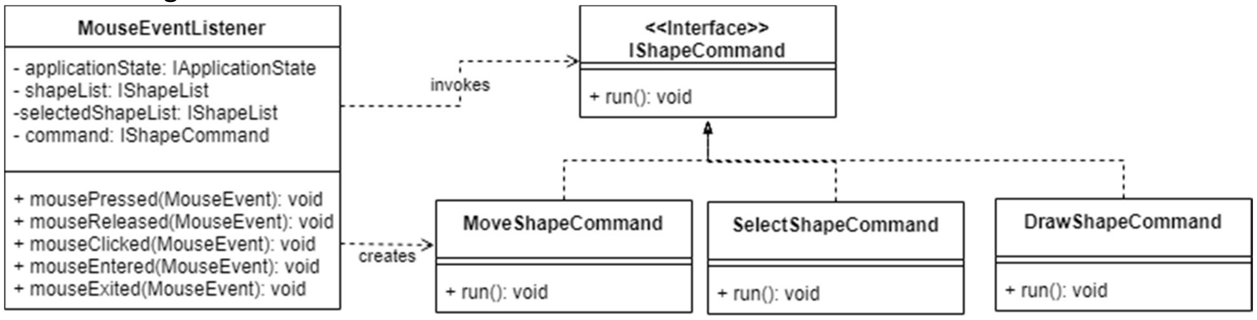
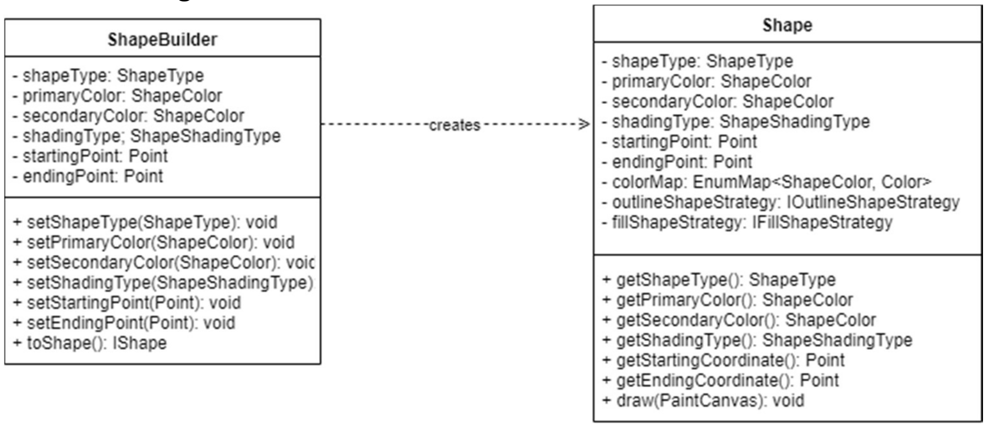
Kendall Zettlmeier
SE 450
Final Project Report

For my implementation of JPaint I don't have any missing features and I don't have any extra credit implemented. One thing to note is that when you resize the canvas, my shapes disappear but will reappear after you take an action on the screen. For example if you move the screen and then add another shape, you will see the previous shapes reappear along with the new shape you just added.

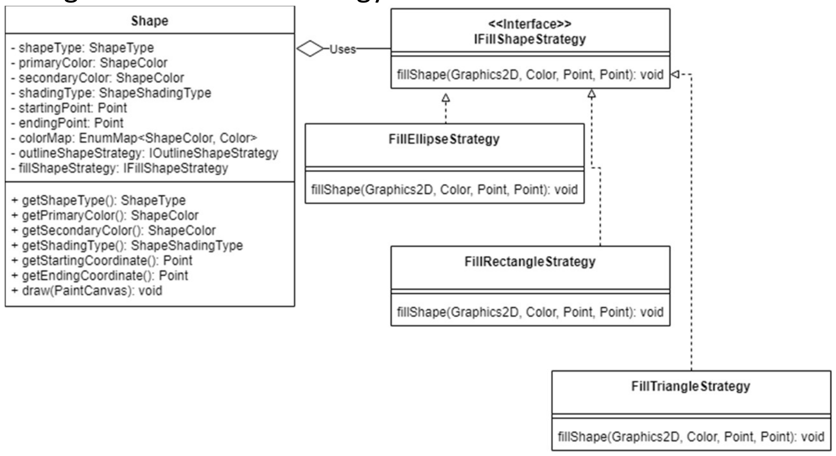
The first pattern that I used was the Command Pattern. I used the command pattern for the different actions you can take on the canvas when clicking and dragging the mouse. In my MouseEventListener on the mouseReleased() function, I check which StartAndEndPointMode the user is in and then decide which command to instantiate. Afterwards we call the run() function which will either execute run on CreateShapeCommand, SelectShapeCommand, or MoveShapeCommand. The reason I chose this pattern because each of the commands had a similar interface for each implementation so it made sense to just have a run() function to use after the mouse is released and it goes off and does it's corresponding command. Please find the class diagram for the Command Pattern below.


MouseEventListener
- applicationState: IApplicationState
- shapeList: IShapeList
-selectedShapeList: IShapeList
- command: IShapeCommand
+ mousePressed(MouseEvent): void
+ mouseReleased(MouseEvent): void
+ mouseClicked(MouseEvent): void
+ mouseEntered(MouseEvent): void
+ mouseExited(MouseEvent): void
invokes
creates
<<Interface>>
IShapeCommand
+ run(): void
MoveShapeCommand
+ run(): void
SelectShapeCommand
+ run(): void
DrawShapeCommand
+ run(): void

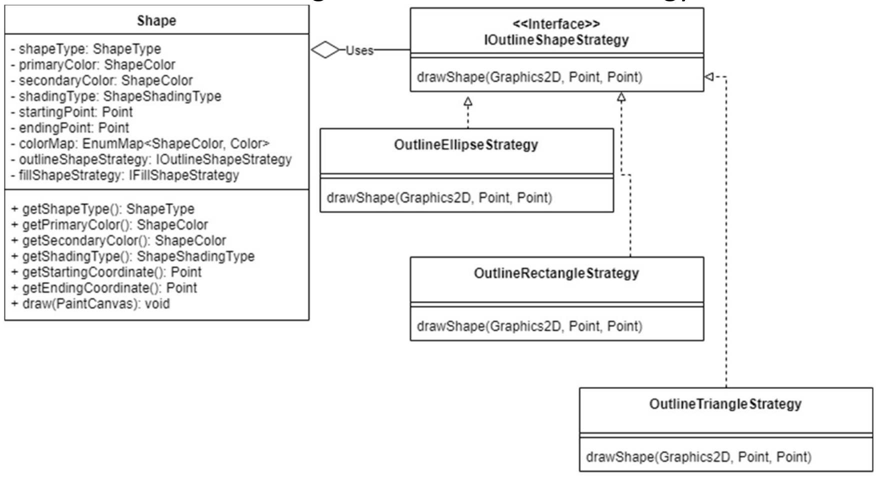
The second pattern that I used was the builder pattern. I created the ShapeBuilder in order to create an IShape object. It made sense to create the ShapeBuilder as we needed to create a lot of shapes in different locations and of different shape types (Ellipse, Rectangle, or Triangle). Please find the class diagram below for the Builder Pattern.


ShapeBuilder
- shapeType: ShapeType
- primaryColor: ShapeColor
- secondaryColor: ShapeColor
- shadingType; ShapeShadingType
- startingPoint: Point
- endingPoint: Point
+ setShapeType(ShapeType): void
+ setPrimaryColor(ShapeColor): void
+ setSecondaryColor(ShapeColor): voic
+ setShadingType(ShapeShadingType)
+ setStartingPoint(Point): void
+ setEndingPoint(Point): void
+ toShape(): IShape
creates
Shape
- shapeType: ShapeType
- primaryColor: ShapeColor
- secondaryColor: ShapeColor
- shadingType: ShapeShadingType
- startingPoint: Point
- endingPoint: Point
- colorMap: EnumMap<ShapeColor, Color>
- outlineShapeStrategy: IOutlineShapeStrategy
- fillShapeStrategy: IFillShapeStrategy
+ getShapeType(): ShapeType
+ getPrimaryColor(): ShapeColor
+ getSecondaryColor(): ShapeColor
+ getShadingType(): ShapeShadingType
+ getStartingCoordinate(): Point
+ getEndingCoordinate(): Point
+ draw(PaintCanvas): void

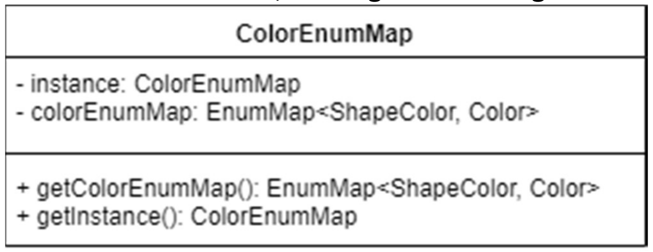
The third pattern that I used was the Strategy pattern and I used it for two instances. The first instance I used it in was for the fill type on the object. So I created the IFillShapeStrategy and I have the FillEllipseStrategy, FillRectangleStrategy, and FillTriangleStrategy. Using the strategy pattern helped me on draw just make a quick check to see how the shape should be filled. You can find the class diagram for the Fill Strategy Pattern below.


Shape
- shapeType: ShapeType
- primaryColor: ShapeColor
- secondaryColor: ShapeColor
- shadingType: ShapeShadingType
- startingPoint: Point
- endingPoint: Point
- colorMap: EnumMap<ShapeColor, Color>
- outlineShapeStrategy: IOutlineShapeStrategy
- fillShapeStrategy: IFillShapeStrategy
+ getShapeType(): ShapeType
+ getPrimaryColor(): ShapeColor
+ getSecondaryColor(): ShapeColor
+ getShadingType(): ShapeShadingType
+ getStartingCoordinate(): Point
+ getEndingCoordinate(): Point
+ draw(PaintCanvas): void
Uses
<<Interface>>
IFillShapeStrategy
fillShape(Graphics2D, Color, Point, Point): void
FillEllipseStrategy
fillShape(Graphics2D, Color, Point, Point): void
FillRectangleStrategy
fillShape(Graphics2D, Color, Point, Point): void
FillTriangleStrategy
fillShape(Graphics2D, Color, Point, Point): void

The second strategy pattern that I used was for the IOutlineStrategy with the implementations of OutlineEllipseStrategy, OutlineRectangleStrategy, and OutlineTriangleStrategy. Using the strategy pattern here helped me on drawn just make a quick check to see if the shape should be outlined. You can find the class diagram for the Outline Strategy Pattern below.


Shape
- shapeType: ShapeType
- primaryColor: ShapeColor
- secondaryColor: ShapeColor
- shadingType: ShapeShadingType
- startingPoint: Point
- endingPoint: Point
- colorMap: EnumMap<ShapeColor, Color>
- outlineShapeStrategy: IOutlineShapeStrategy
- fillShapeStrategy: IFillShapeStrategy
+ getShapeType(): ShapeType
+ getPrimaryColor(): ShapeColor
+ getSecondaryColor(): ShapeColor
+ getShadingType(): ShapeShadingType
+ getStartingCoordinate(): Point
+ getEndingCoordinate(): Point
+ draw(PaintCanvas): void
Uses
<<Interface>>
IOutlineShapeStrategy
drawShape(Graphics2D, Point, Point)
OutlineEllipseStrategy
drawShape(Graphics2D, Point, Point)
OutlineRectangleStrategy
drawShape(Graphics2D, Point, Point)
OutlineTriangleStrategy
drawShape(Graphics2D, Point, Point)

The fourth pattern that I used was the Singleton pattern. I used the Singleton pattern to create my ColorEnumMap for mapping ShapeColor and Java Color. The reason that I used a singleton was because I only ever needed a single instance of the mapper to be used whenever I needed it. You can find my class diagram for the Singleton Pattern below. (Note for some reason when I tried to underline my static member instance, it underlined the whole thing and couldn't figure out why so I just left it not underlined, same goes for the getInstance() function).


ColorEnumMap
- instance: ColorEnumMap
- colorEnumMap: EnumMap<ShapeColor, Color>
+ getColorEnumMap(): EnumMap<ShapeColor, Color>
+ getInstance(): ColorEnumMap

Overall for the project, I didn’t have too much of an issue. I was able to figure out the best patterns that suited each situation after you went over them in class. The only issue I ended up having was when I was working with the lists and trying to remove shapes. The issue I was having was concurrent editing of a single instance. The reason that I was having the issue was when I was editing the list I was setting up draw at the same time as adding/deleting from the list. What I should have done was cloned the actual list that I was trying to add and update to when I was looping through the lists. I decided against it just because of the time it would have taken to create the clone implementation. Otherwise I didn’t have any other issues with the implementation.