Introduction

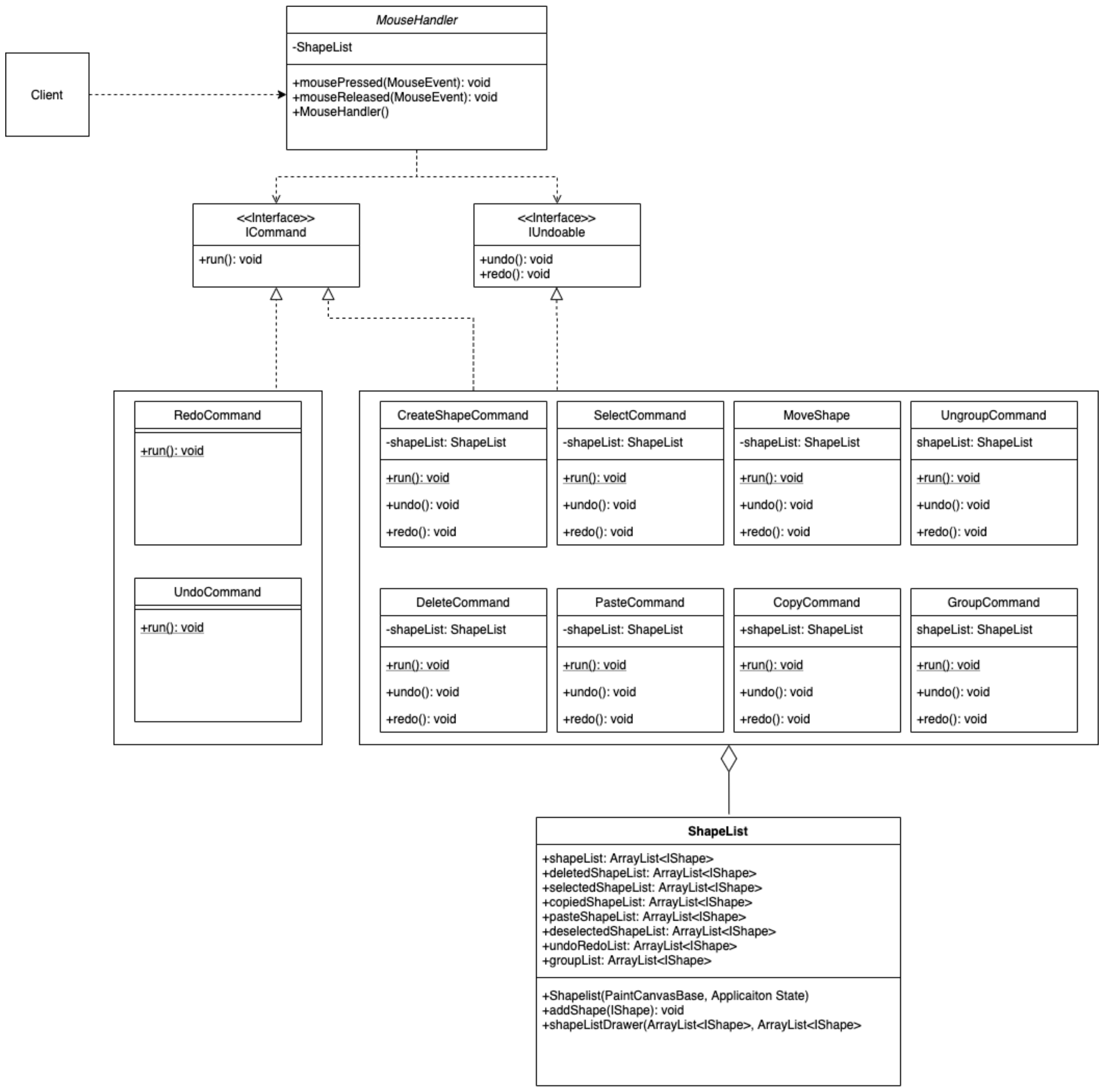
The project for this course was to develop a “MS Paint”-like application in Java called JPaint, which allowed the students to implement various design patterns and object-oriented programming concepts such as interfaces, abstract classes, and more throughout the course. This course was challenging, but also very rewarding as I feel the knowledge gained throughout this course will translate well into other software engineering courses and further build my growth as a software engineer.

List of missing features, bugs, extra credit, and miscellaneous notes.

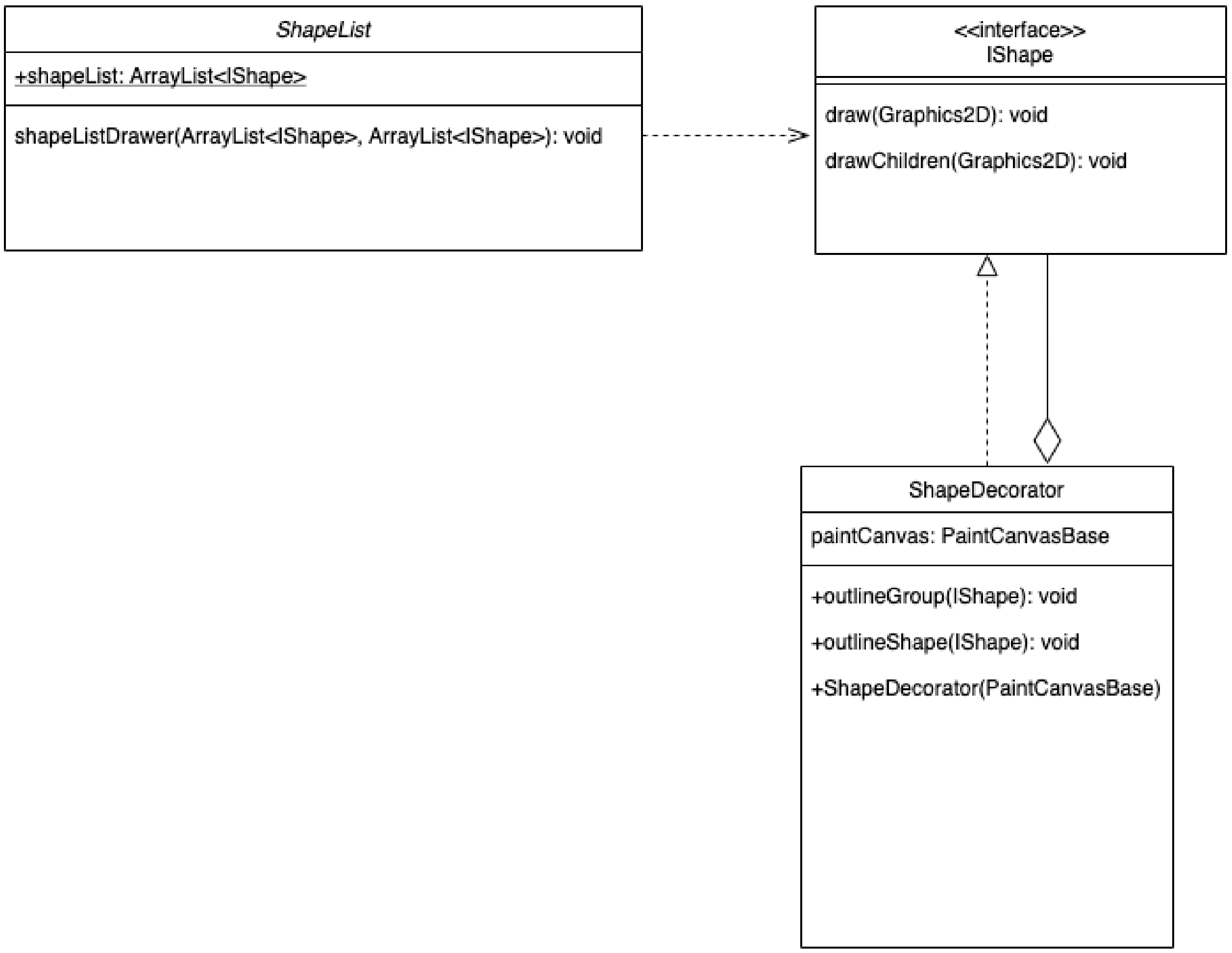
- When selecting a triangle, the dashed outline does not include a 5-10pt buffer, while the rectangle and ellipse shapes do.
- After moving a shape several times, invoking the undo command will only undo the last move. Every subsequent undo afterwards will perform the same move sequence.
- After grouping 2 groups together, then clicking undo>redo>undo, the second undo command will ungroup all shapes into individual shapes, as opposed to undoing back to 2 groups of shapes.
- When in draw mode and the user clicks the mouse without dragging and drawing a shape, an object is still created. This can be seen when in select mode and dragging over the area that was clicked – you'll see a small highlighted section on the clicked location.

Notes on design

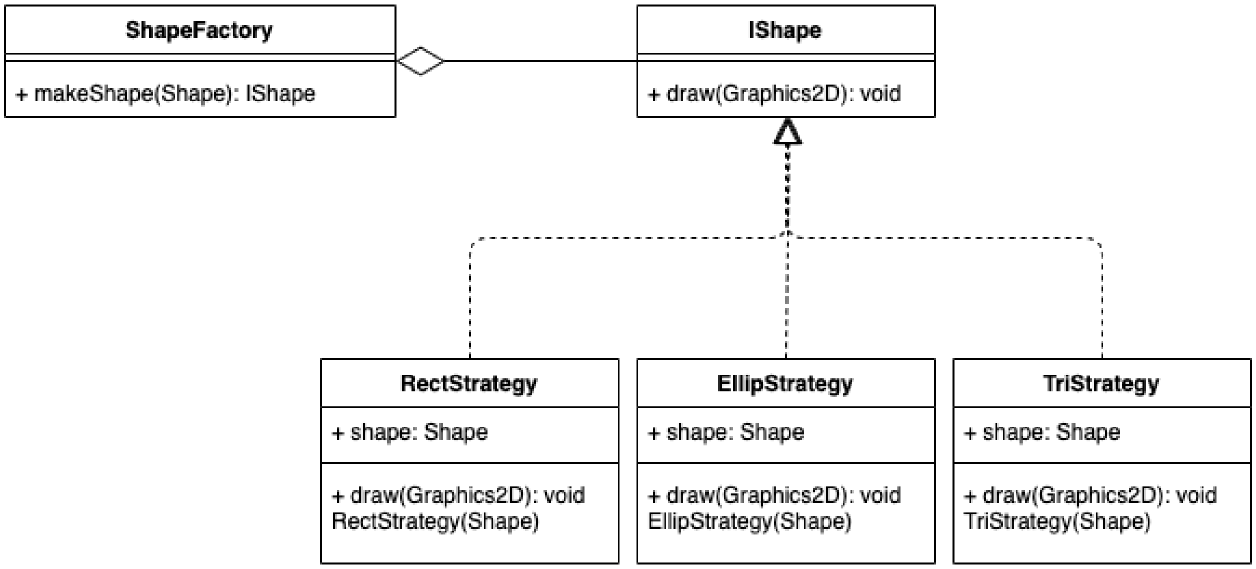
1. Command Pattern
    a. The command pattern was used in order to execute various commands in the JPaint application. This was my most widespread design pattern since it was used across all commands for instantiating shapes. The classes used for this pattern were MouseHandler, CreateShapeCommand, MoveShape, SelectShape, UndoCommand, RedoCommand, DeleteCommand, CopyCommand, PasteCommand, GroupCommand, UngroupCommand, and SelectCommand.



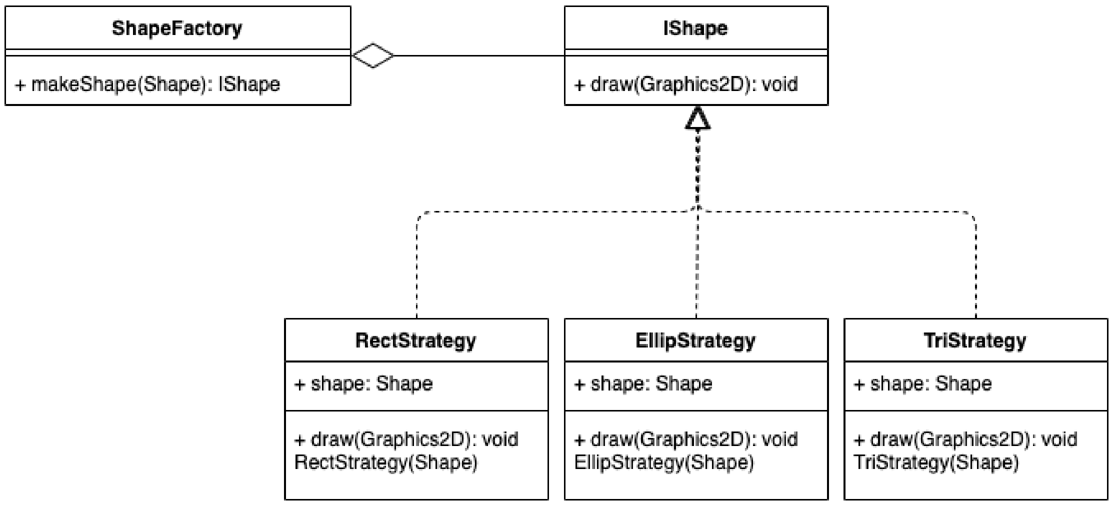
2. Decorator Pattern
    a. The decorator pattern was used to draw the dotted outlines for shapes and groups of shapes when selected. This design pattern was useful because it enabled additional behavior to be added to objects without affecting the other objects assigned to the same class. The ShapeDecortor was the class that used this pattern, while inheriting from the IShape interface. This was beneficial so I didn't have to create one class to contain the logic for all 3 shapes.


ShapeList
+shapeList: ArrayList<IShape>
shapeListDrawer(ArrayList<IShape>, ArrayList<IShape>): void
<<interface>>
IShape
draw(Graphics2D): void
drawChildren(Graphics2D): void
ShapeDecorator
paintCanvas: PaintCanvasBase
+outlineGroup(IShape): void
+outlineShape(IShape): void
+ShapeDecorator(PaintCanvasBase)

3. Factory Pattern
    a. The factory pattern was used to create instances of different shape objects at runtime. This was used because it was a cleaner, more efficient way to create different instances of shapes. A ShapeFactory class was created, while the subclasses RectStrategy, EllipStrategy, and TriStrategy contained the logic to create the shape objects themselves.

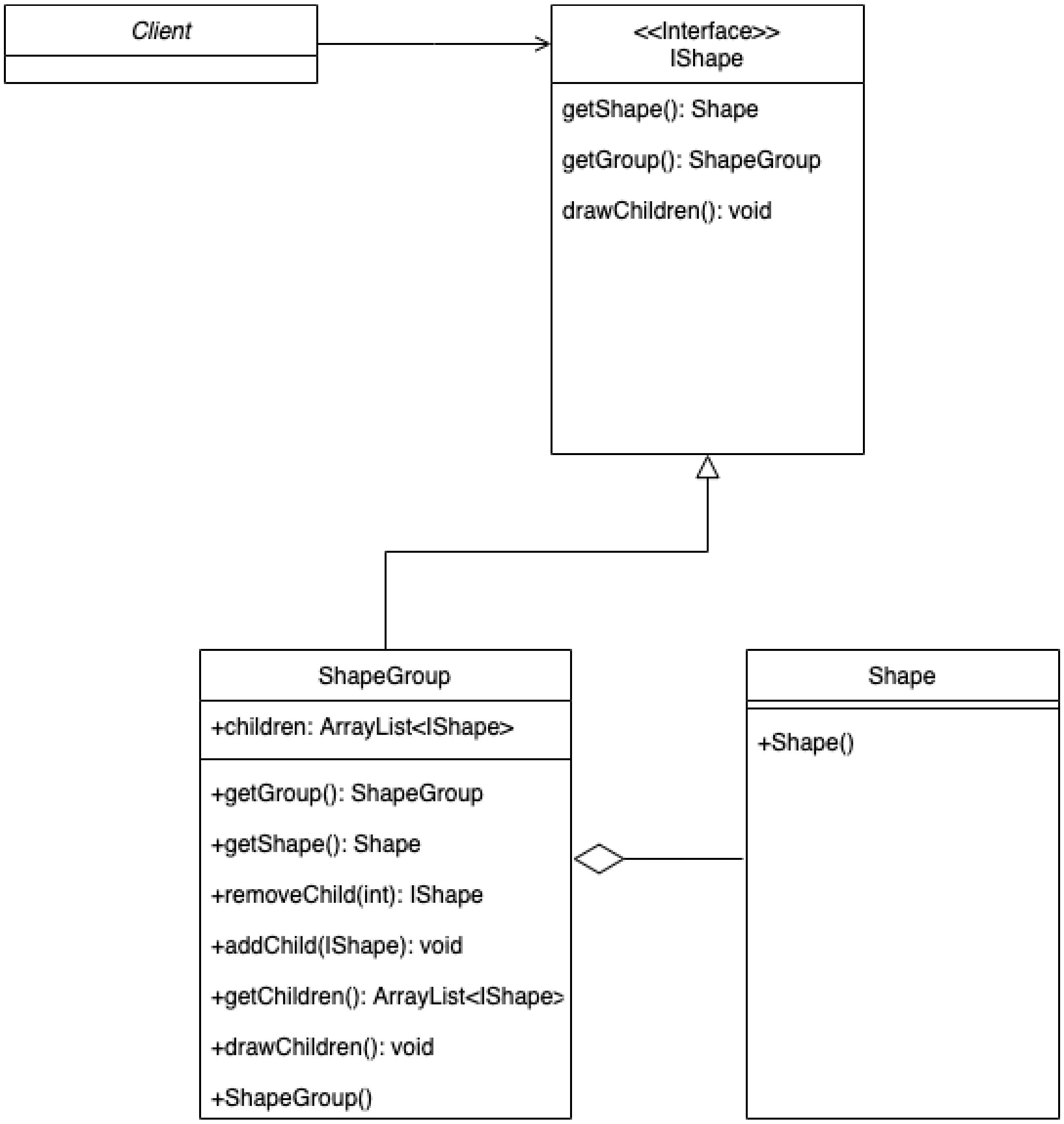


4. Strategy Pattern
    a. The strategy pattern allowed me to encapsulate different logic for each shape. Similar to the factory pattern, this was beneficial in separating the logic for different shapes. This pattern was captured in the classes RectStrategy, EllipStrategy, and TriStrategy.



5. Composite Pattern

a. I used the composite pattern for grouping and ungrouping shapes. This was chosen due to its efficiency in grouping objects together and treating them as a single instance, thus resembling a tree-structure for shape/group objects. It allowed. This was very helpful when combining shapes into groups.



Successes and Failures:

The most challenging part of the project was implementing groups and the functionality to group and ungroup shapes. This was mostly because I wasn't using an interface for shapes up until this point. I was initially using a shape class with no interface. I quickly realized that in order to implement this functionality, I was going to need a way to be able to pass both groups of shapes and individual shapes to the shapelist and the shapelist drawer. Once implemented via the composite pattern, things became much more simple.

Also, though in hindsight it seems ridiculous, I struggled to get the MouseHandler working at first. However, this was very beneficial as it gave me more insights into how to investigate, scan, and read through online documentation in order to implement a technical solution.