CSCI 4448 – Final Report

Group 21: Paint

Jonathan Vu, Robert Allen

1. Features Implemented

User Requirements		
ID	**Description**	Priority
US-01	As a user I can save my image.	High
US-02	As a user I can load an image.	High
US-03	The user should be able to rotate the canvas.	Medium
US-04	The user should be able to select a brush size and color for all drawing actions.	High
US-05	The user should be able to draw straight lines.	Critical
US-06	The user should be able to draw polygons.	Critical
US-07	The user should be able to draw ellipses.	Critical
US-09	The user should be able to draw paths.	High
US-10	The user should be able to fill enclosed areas with color. Enclosed areas being a 2D shape completely enclosed by a line.	Medium
US-13	The user should be able to resize the canvas	Medium

Functional Requirements		
ID	Description	Priority
FR-03	All drawn shapes will be stored as vectors rather than bitmaps.	Critical
FR-04	Saved images will be stored on the local filesystem.	Critical

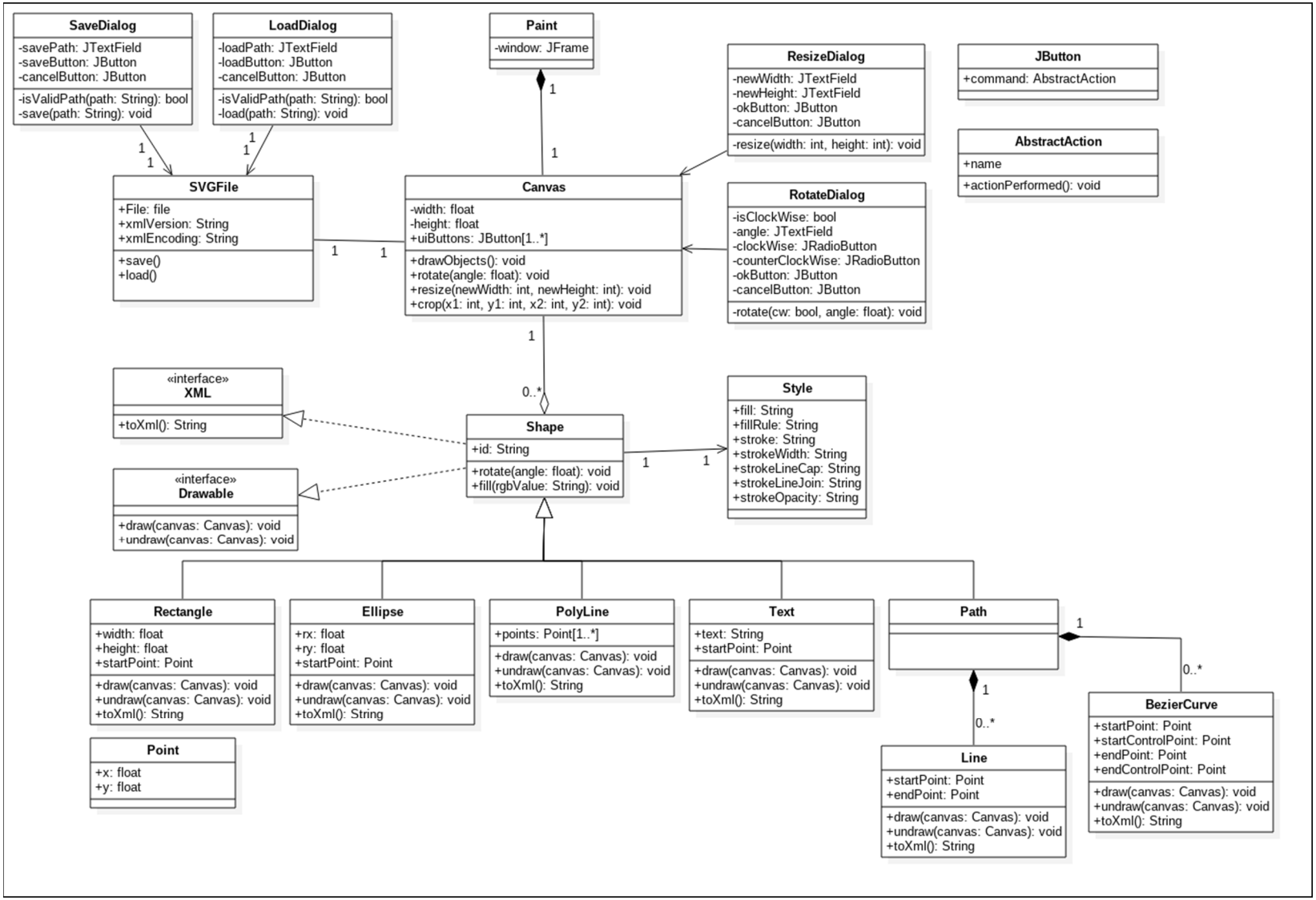
Non-Functional Requirements		
ID	**Description**	Priority
NFR-01	For usability, the images should be saved in the standard SVG format to allow cross-compatibility with other vector image editors.	Nice-to-have

2. Features Not Implemented

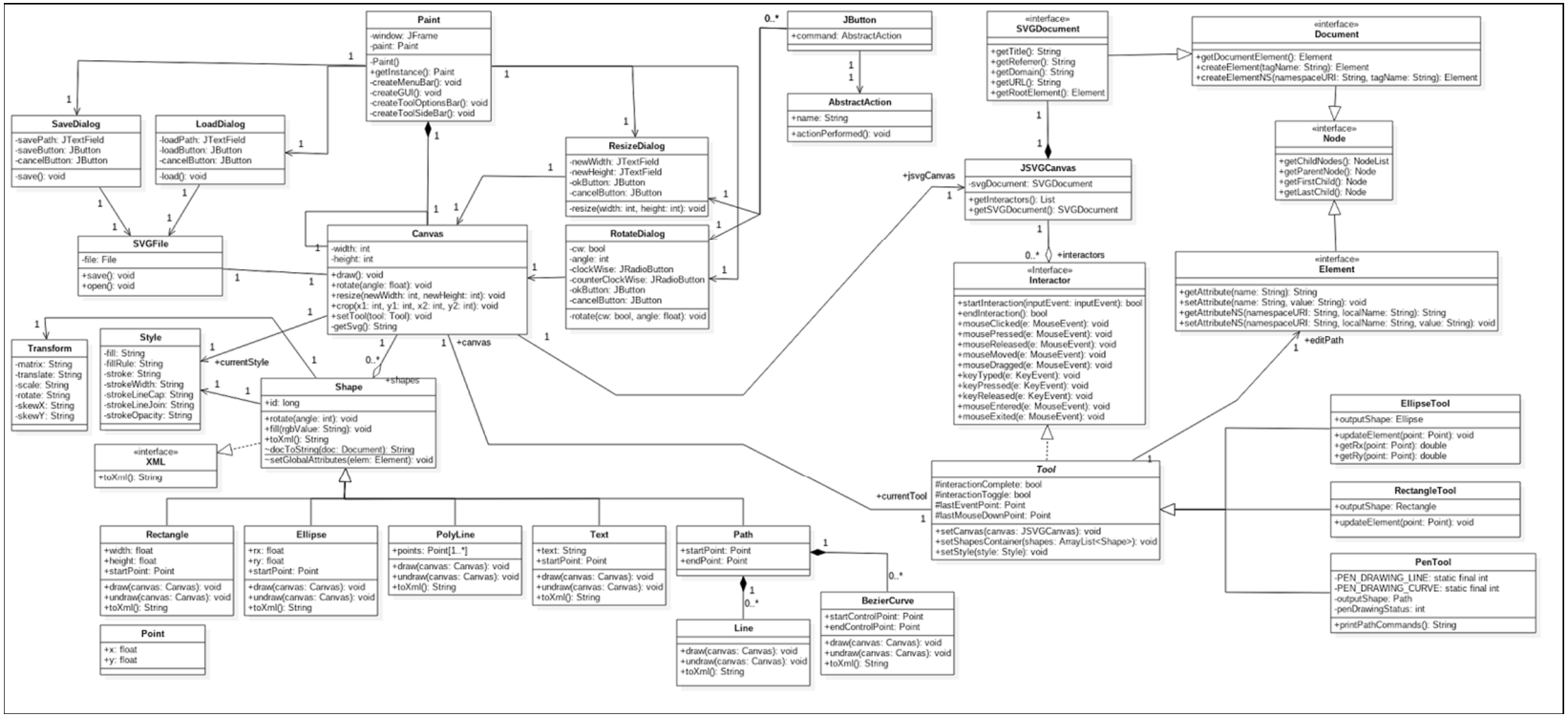
User Requirements		
ID	**Description**	**Priority**
US-08	The user should be able to create textboxes.	Low
US-11	User should be able to select and rotate or resize previously drawn shapes without distortion.	Nice-to-have
US-12	The user should be able to draw using tablet pen input.	Nice-to-have
US-14	The user should be able to crop the canvas	Medium

Functional Requirements		
ID	**Description**	**Priority**
FR-01	When selecting a color, the user should be presented with a color palette.	High
FR-02	When selecting brush size, the user should be presented with a list of numbers beside sample brush lines.	High
FR-05	The program should add visible anchor points to drawn shapes that can be used to modify the shapes.	Medium

3. Old Class Diagram



4. Final Class Diagram



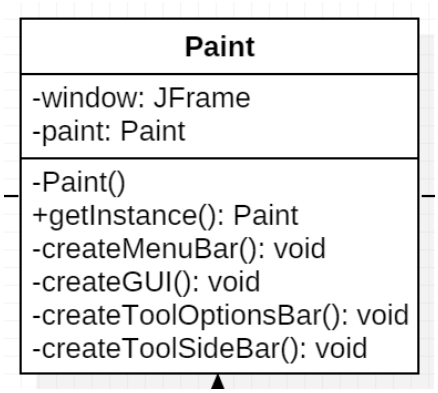
5. Design Changes

The core component of our code stayed relatively the same. The main challenge with our program was how we were going to represent the different SVG shapes in our program. The biggest thing that changed was that we added a library, apache batik, which would handle the rendering of the SVG for us. This is what caused the giant branching structure over to the right. While adding the library did make it easier to code it also made it harder to represent everything in a class diagram.

I think it was helpful to diagram out how we were going to go about the main functionality in our code. It left no questions when adding functionality relating to a particular shape of where that functionality should go. The struggles with class diagrams come from the fact that they don't have to deal with the hurdles that actual coding often introduces and although we may have tried to think of every class we were going to need in building the program, it's really hard to tell you're missing something until you've sat down and tried to code it.

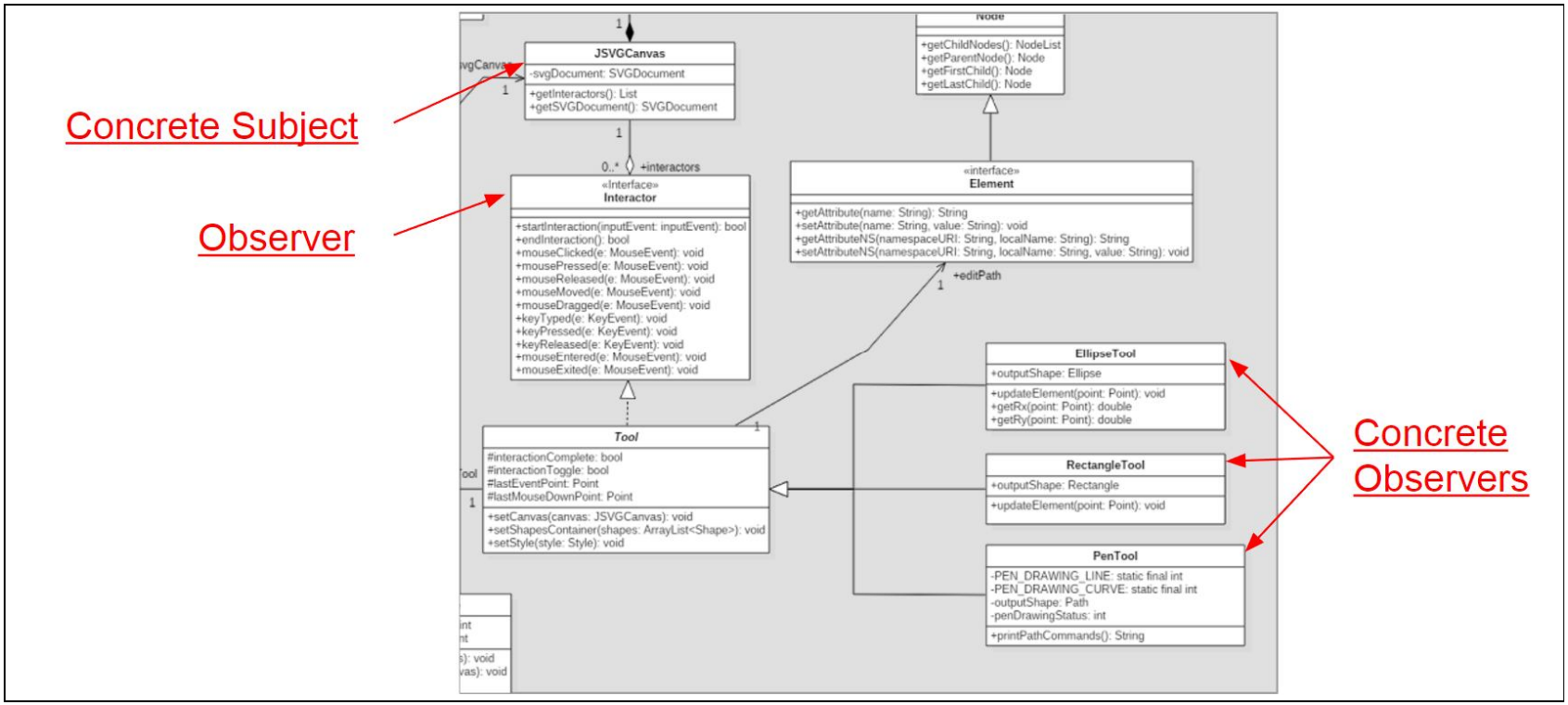
6. Design Patterns

A. Singleton



We implemented the singleton pattern into our main Paint class. Each instance of paint represents a full running instance of the program. This includes things like the GUI and the database connection, things we did not want to have duplications of. This doesn't disallow the user from opening two different copies of Paint but it serves as a safeguard in the code to prevent us from creating more than the necessary instance.

B. Observer

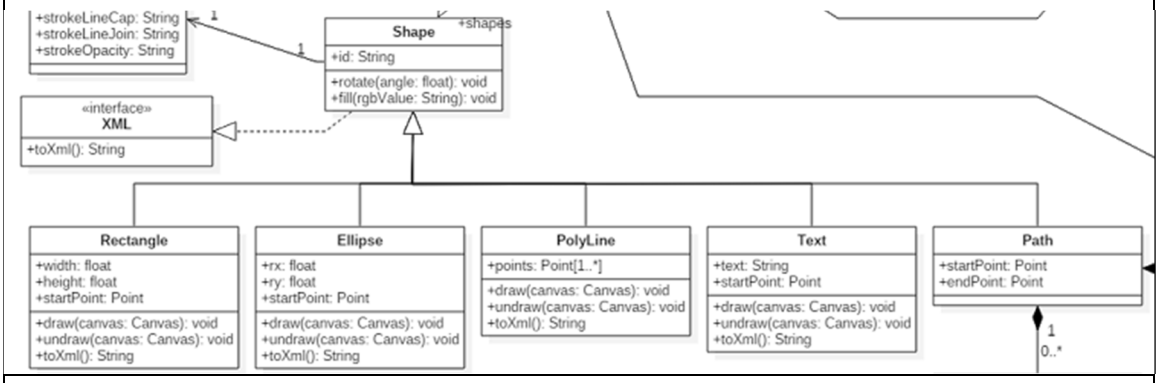


To implement drawing tools, we needed a way to capture user interactions with the batik canvas. Fortunately, the batik canvas implements the observer pattern using the Interactor interface. By implementing the interactor interface in our base class Tool, we are able to subscribe to and be notified only when user input events occur in our subject.

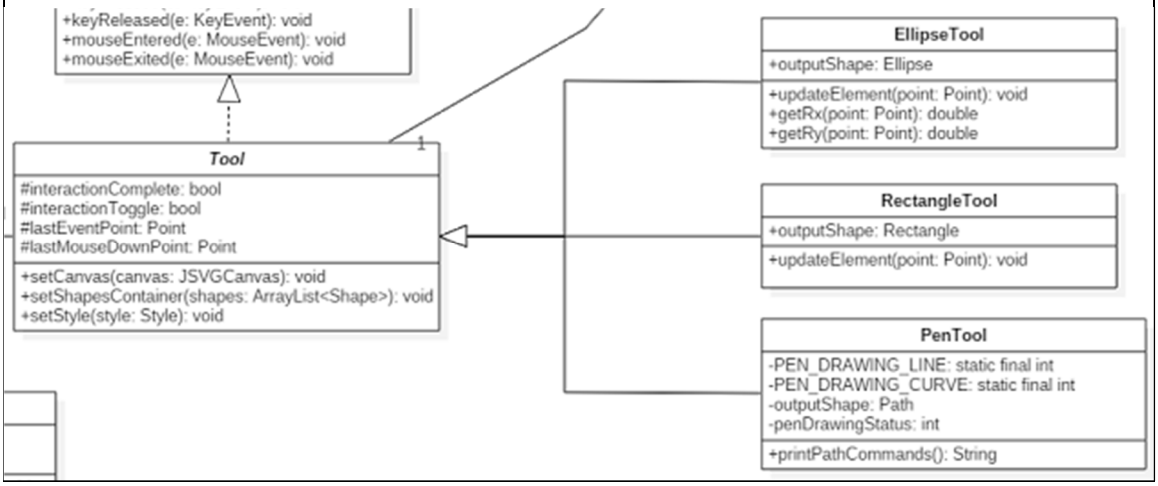
The observer pattern is especially useful in this case because the canvas would not be able to repaint and provide a responsive preview of what the user is drawing if the input handler polled constantly without yielding control.

C. Composition

Shapes Composition



Tools Composition

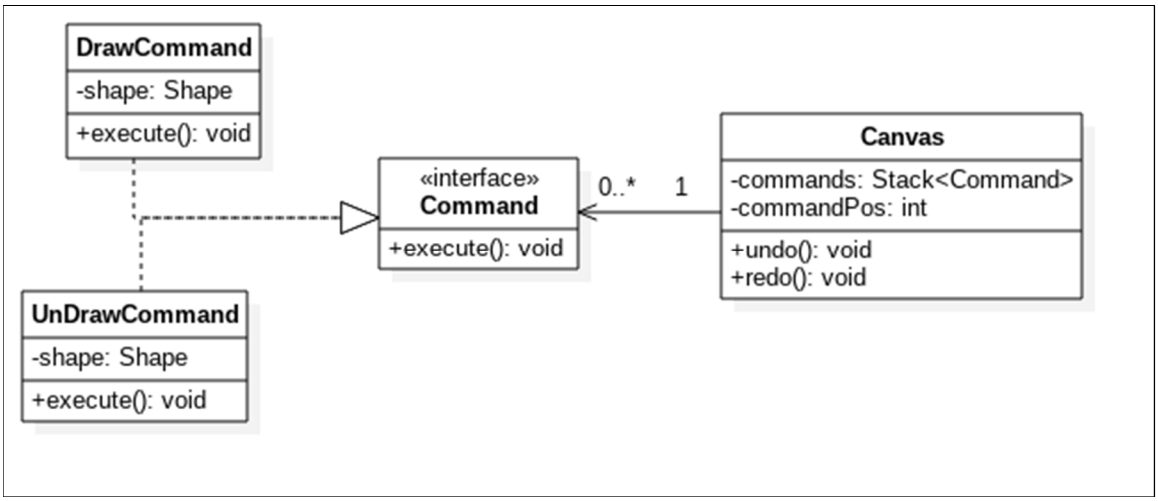


Composition is applied in two important parts of our system: Shapes and Tools. All types of shapes derive from the Shape class. This allows the canvas to store and track all shapes as Shape objects. Every Shape object implements the XML interface and is associated with a Style object. Style dictates how the shape is drawn and some shapes may not use all of Style's attributes so their relationship is implemented as an association. Output to an SVG file is a requirement for all shapes, so the related methods are represented as the XML interface.

The drawing tool implementation takes advantage of composition to allow all tools to share the same observer interface for handling user input as well as the same interface for changing settings such as Style, output Shape storage, and the target JSVGCanvas.

7. Possible Design Pattern Additions

A. Command



One design pattern that we could have implemented was the command pattern. We would utilize the command pattern to enable actions to be undone and redone, a feature highly necessary in an image editor. We could create a Command interface which would have an execute method. The implementations of this interface (DrawCommand, and UnDrawCommand) could store an instance of the shape they modify. The canvas class could then keep a Stack of these commands as well as an integer indicating our current position in the stack. Then when someone selects Edit->Undo or Edit->Redo have the Canvas execute the command in the stack.

This would make allow us to track the changes to the image that occur when we edit it. A feature that we currently do not would be useful.

8. Lessons Learned

We've learned that creating a class diagram and plotting out a plan of how we're going to implement a design beforehand can be very helpful. The ability to refer to a class diagram to see what classes needed to be created first or which had to be implement to satisfy user requirements was very useful. Using the requirement specifications and UML diagrams together also allowed us to see any holes in the system and definitely saved us from rewrites.

However, in the future, we would probably take a more blended approach to designing the system by doing only a preliminary design of the system in UML before testing our assumptions in code. This experimentation and iteration would have helped us arrive at implementations more quickly than abstractly reasoning in UML did. For example, the major changes in our class diagram would not have occurred if we had begun coding earlier in the

design phase and discovered the Apache Batik SVG toolkit. Although, this may have been our own lack of design intuition and framework knowledge rather than a problem with the approach taught in this course.