ECE9020 Sketchpad Assignment

(Name arranged in alphabetical order)

Da(Clay) Cheng,	251350918
Junyan(Tristan) Huang,	251394173
Xiang Li,	251130633

Github: https://github.com/TristanWw/Sketchpad_JavaAWT

2024-08-08

1 Objectives

1.1 Basic Requirements

Design and implement a sketchpad project including the follow basic modes:

- A mode to draw sketches using scribbled freehand lines
- A mode to draw straight lines
- A mode to draw rectangles
- A mode to draw ellipses
- A mode to draw special cases of these (squares and circles)
- A mode to draw polygons (open and closed polygons)

In addition, the user should be able to select a **colour** for any of the graphical objects that are about to be drawn. Furthermore, the user should have a **selection** method so that they can identify an object that has already been drawn and perform the following operations:

- Move the selected object to a new location
- Cut the object from the drawing (delete a graphical object)
- Paste the selected object (copy and paste) to a new location

1.2 Advanced Requirements

- Group the object with another object (possibly creating groups of arbitrary objects)
- Ungroup a set of objects that have been grouped

1..3 Very Advanced Requirements

- Undo and Redo
- Save and Load a partially completed drawing, extending, modifying, moving, pasting parts of it.

2 Design Artefacts

Note:
All the diagrams used in the following chapters can be found in the "res" or "doc" folder of the project.

2.1 Prove of Concept Stage

Prior to the final design of the overall project, the three of us conducted a POC design to testify the object oriented design principles that need to be utilized in this project. This section will show the design of our POC stage.

Github: https://github.com/TristanWw/Sketchpad_JavaAWT/tree/dev-skeleton

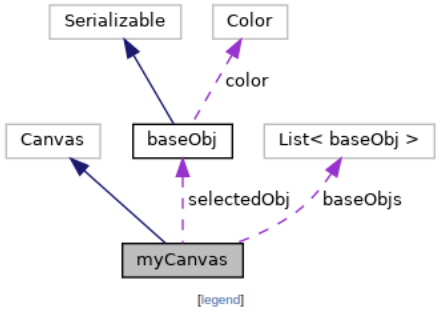
2.1.1 Object Diagrams

Note:
The diagrams used in this section are generated by a tool called "Doxygen" directly from the source code.

1. Define a **myCanvas** class that extends the Canvas class to act as the canvas. It has a **List<baseObj>** which is responsible for holding all the graphic objects.

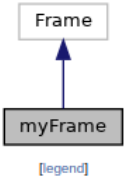
Inheritance diagram for myCanvas:

Collaboration diagram for myCanvas:

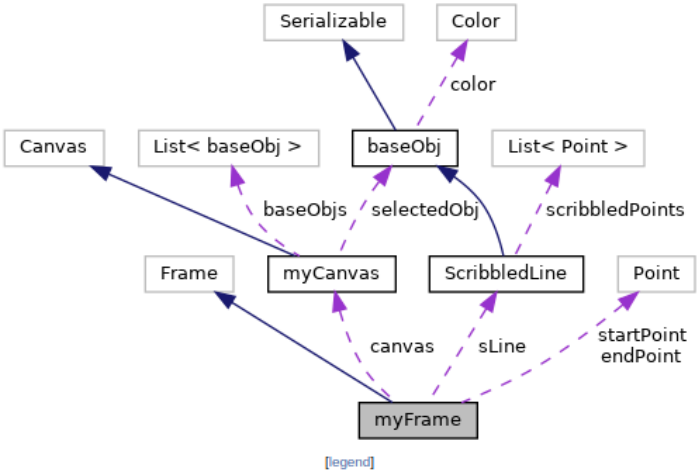


2. Define a **myFrame** class that extends the Frame class which will be responsible for instance the **window** and **buttons.** Also it will be responsible for handling the event handlers for different modes.

Inheritance diagram for myFrame:

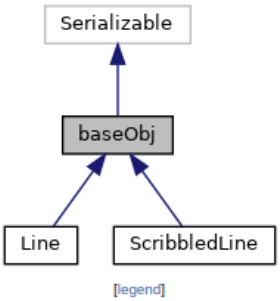


Collaboration diagram for myFrame:

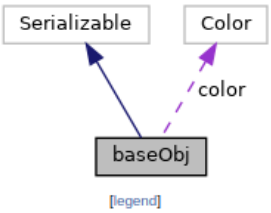


3. Class **baseObj** defines the abstract interfaces **copy, draw, translate, contains**, which will be implemented by the extended class for each mode.

Inheritance diagram for baseObj:



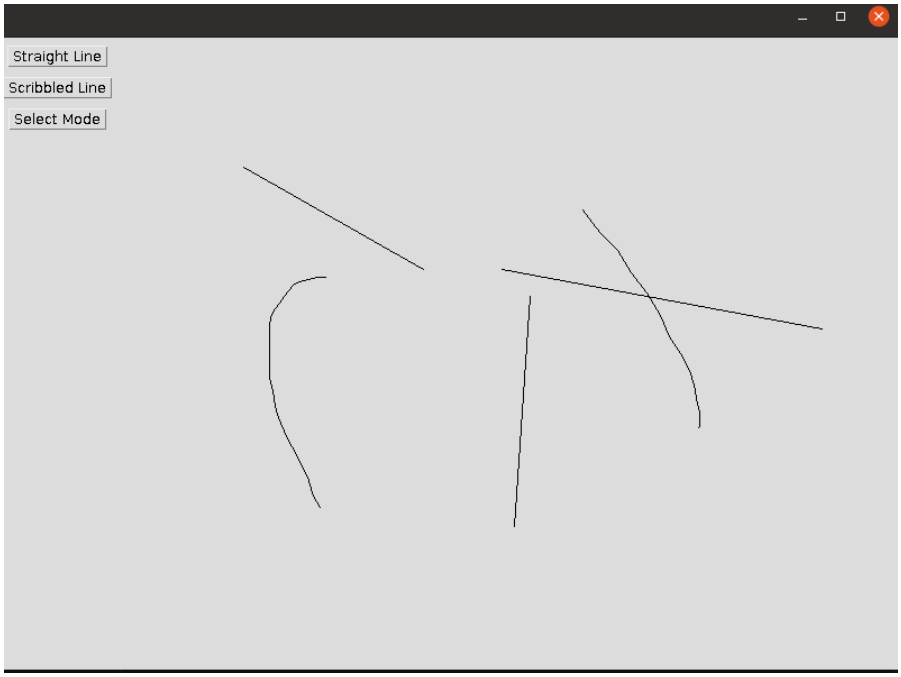
Collaboration diagram for baseObj:



2.1.2 Conclusion

The screenshot of the running POC program is shown in the below screenshot.

After we discussed this solution, we all agreed that we can follow this pattern and add drop down menus and put the mode selection beneath the menu line.



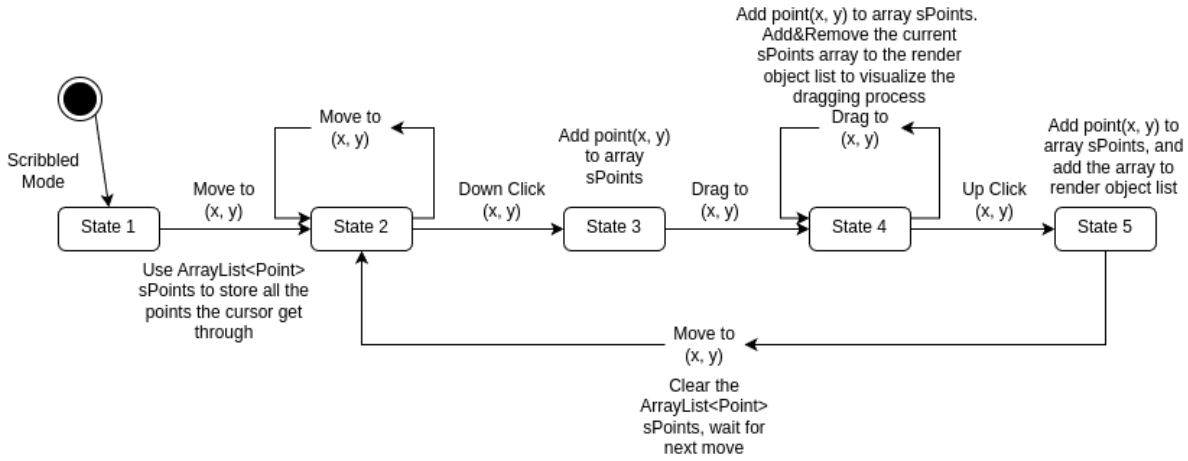
2.2 Refactored Design

After the design and discussion in the POC stage, we finally moved on to finalize the design of the sketchpad project. With some modification made, the detailed design diagram is shown in the following chapters.

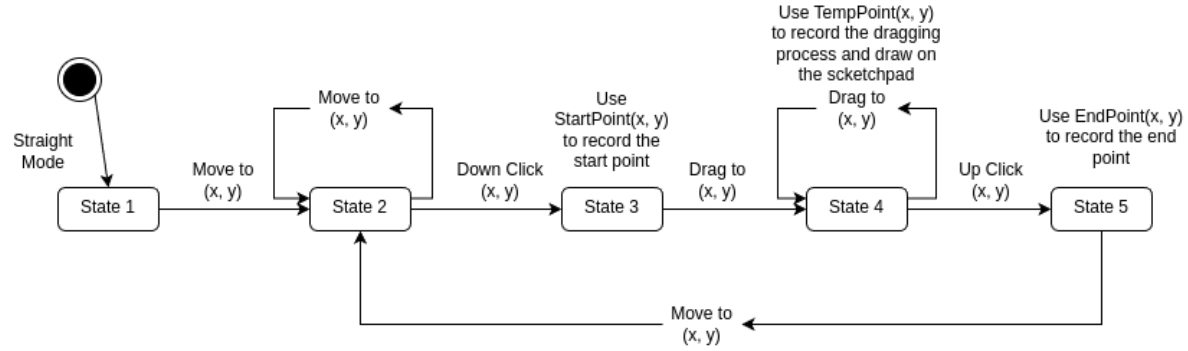
Github: https://github.com/TristanWw/Sketchpad_JavaAWT/tree/dev-refactored

2.2.1 State Chart Diagrams

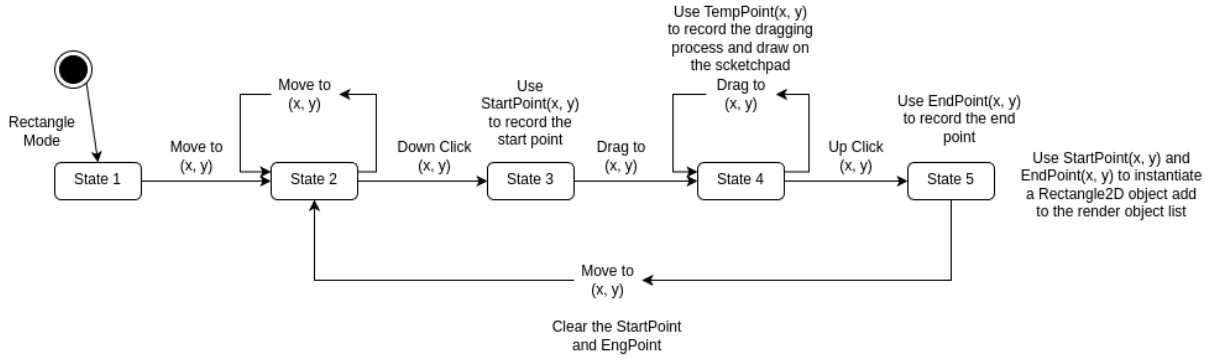
2.2.1.1 Scribbled freehand Line Mode



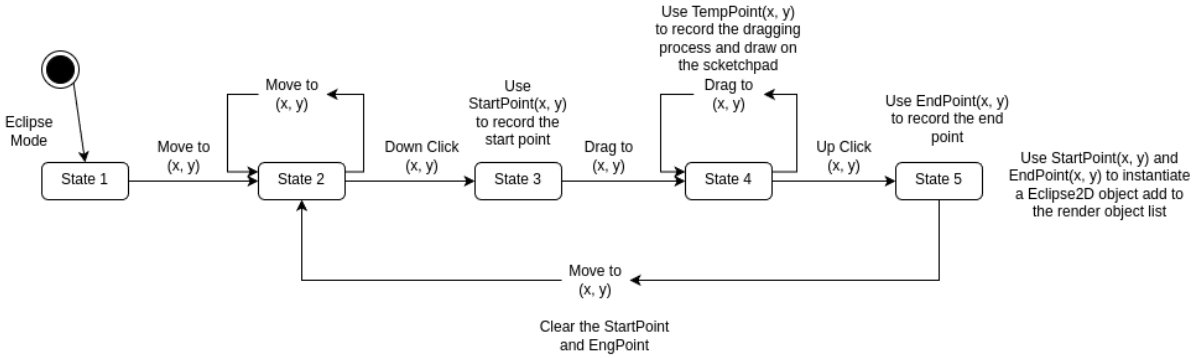
2.2.1.2 Straight Line Mode



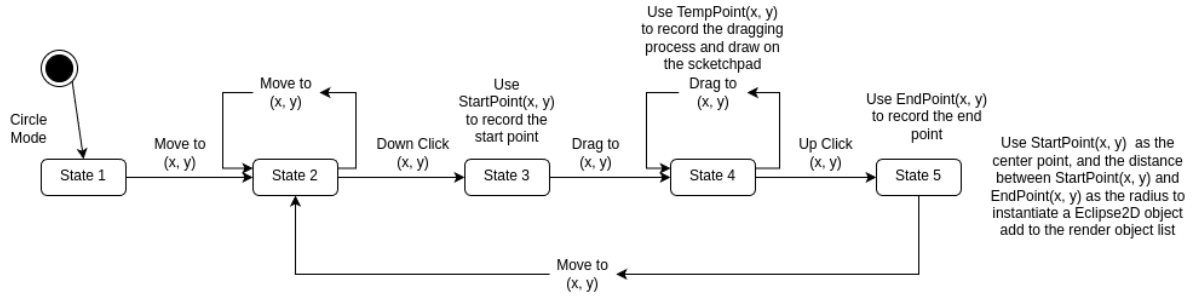
2.2.1.3 Rectangle Mode



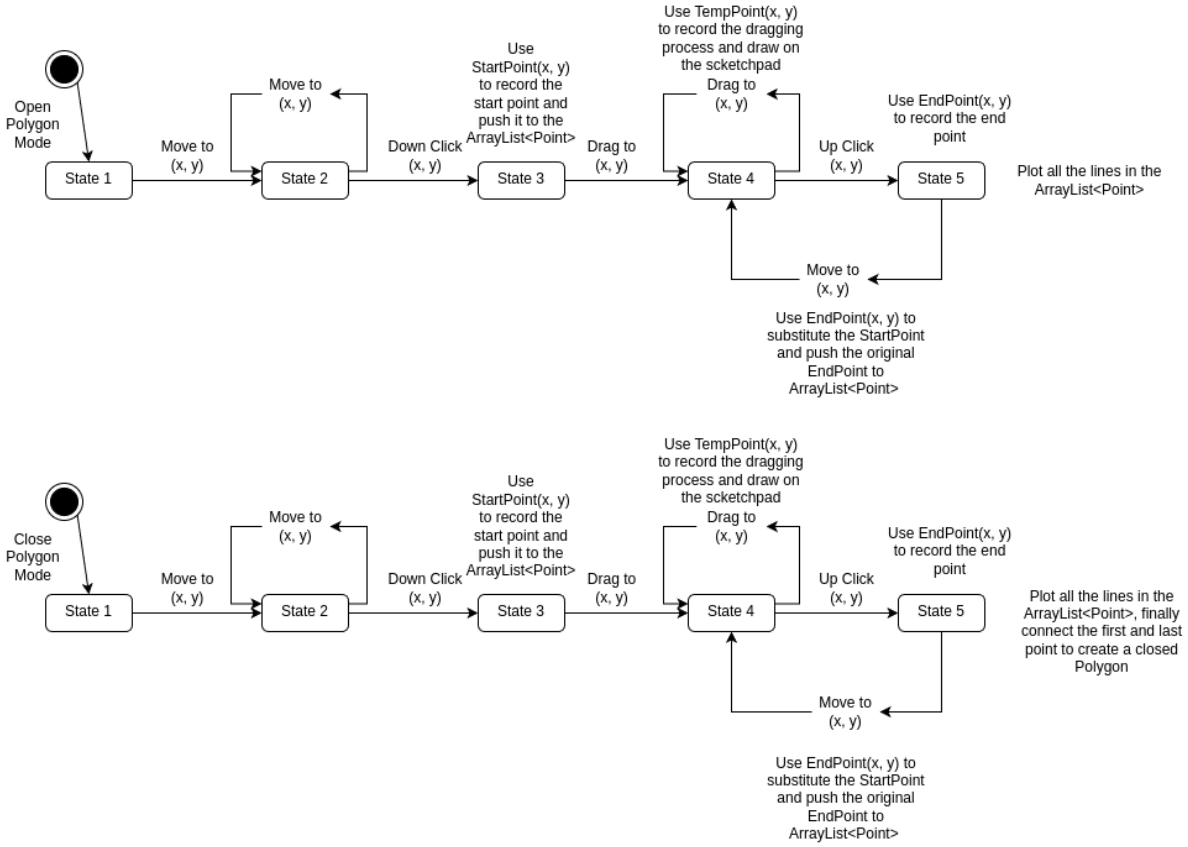
2.2.1.4 Eclipse Mode



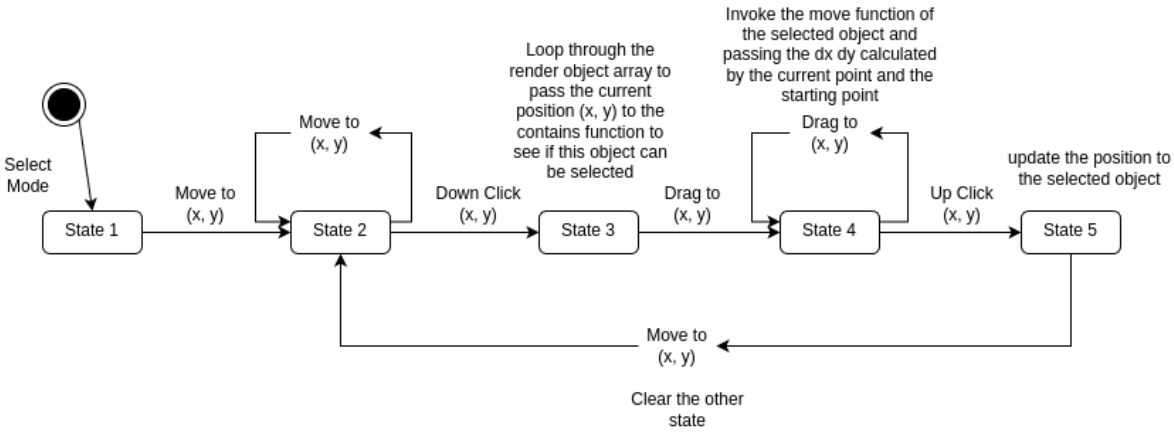
2.2.1.5 Circle Mode



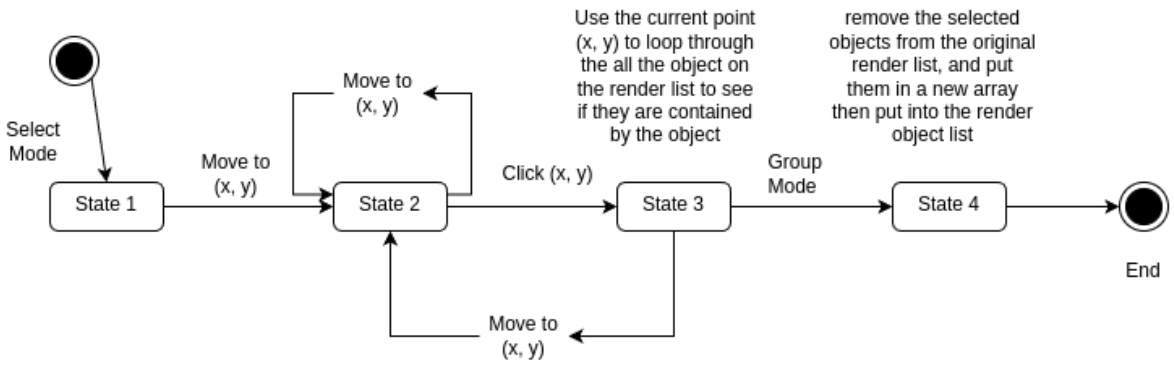
2.2.1.6 Polygon Mode(Open and Closed)

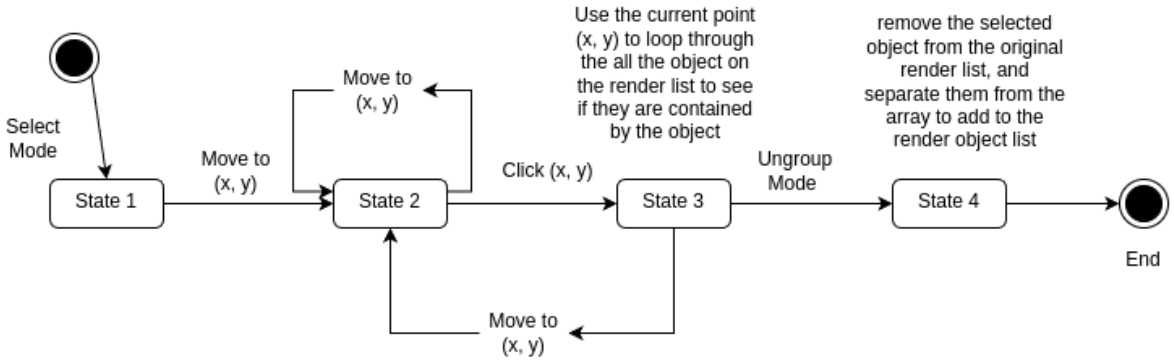


2.2.1.7 Select Mode

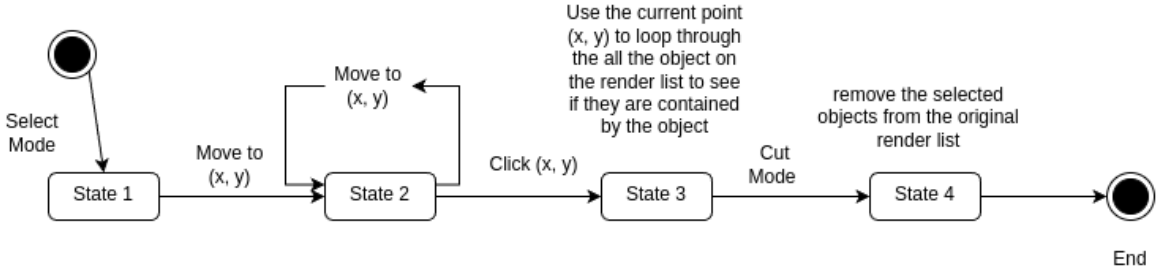


2.2.1.8 Group/Ungroup Mode





2.2.1.9 Cut Mode



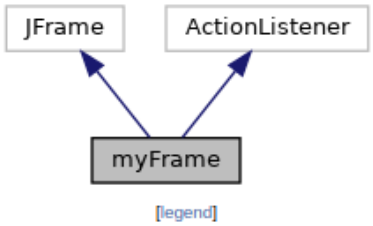
2.2.2 Object Diagrams

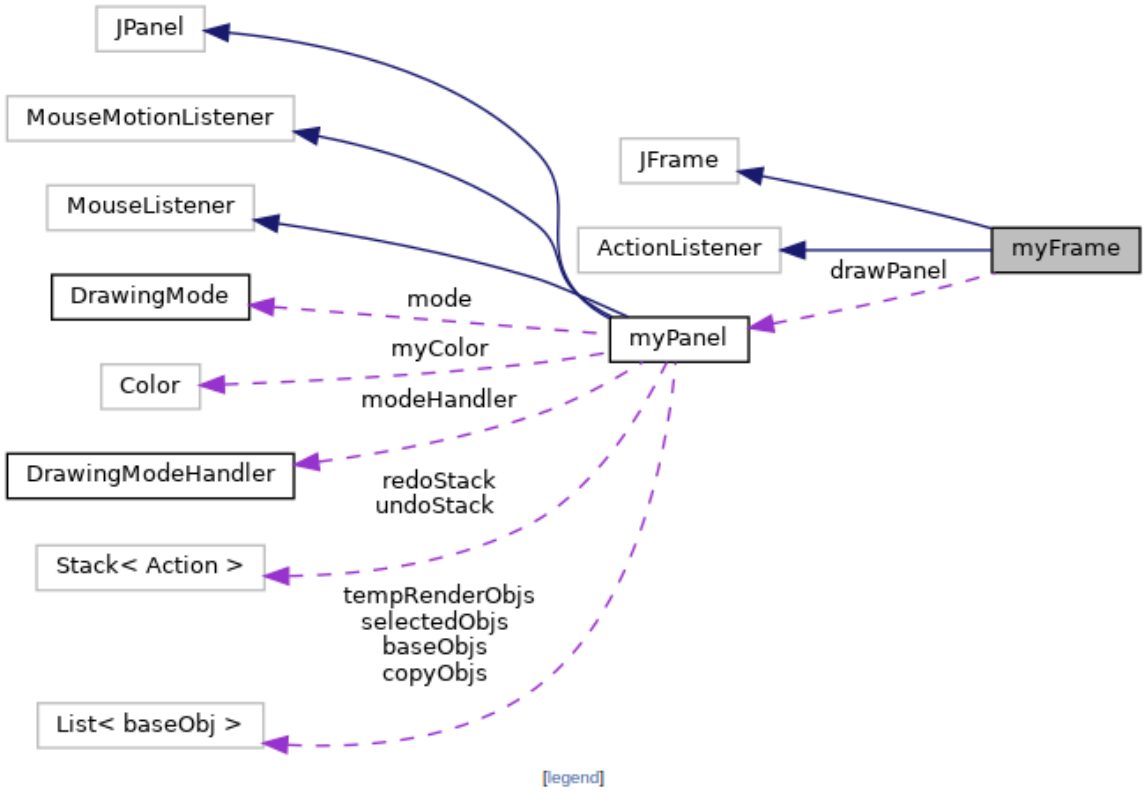
2.2.2.1 DrawingProgram class

- **DrawingProgram** is the entry point of the whole program, which is responsible for instance the **DrawingFrame**.

2.2.2.2 myFrame class

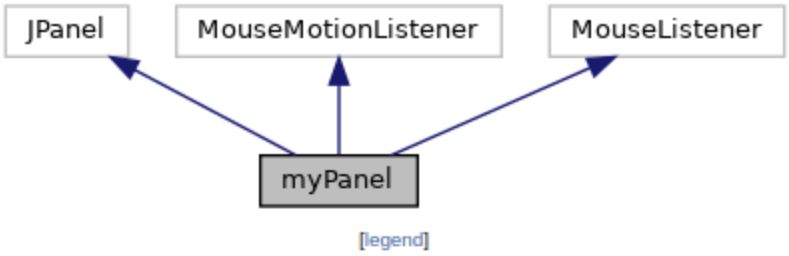
- As we can see from the following object diagram, **DrawingFrame** extends the **JFrame** and implements the **ActionListener** interface. The **DrawingFrame** object has a member variable **drawPanel** which is of type **DrawingPanel.**

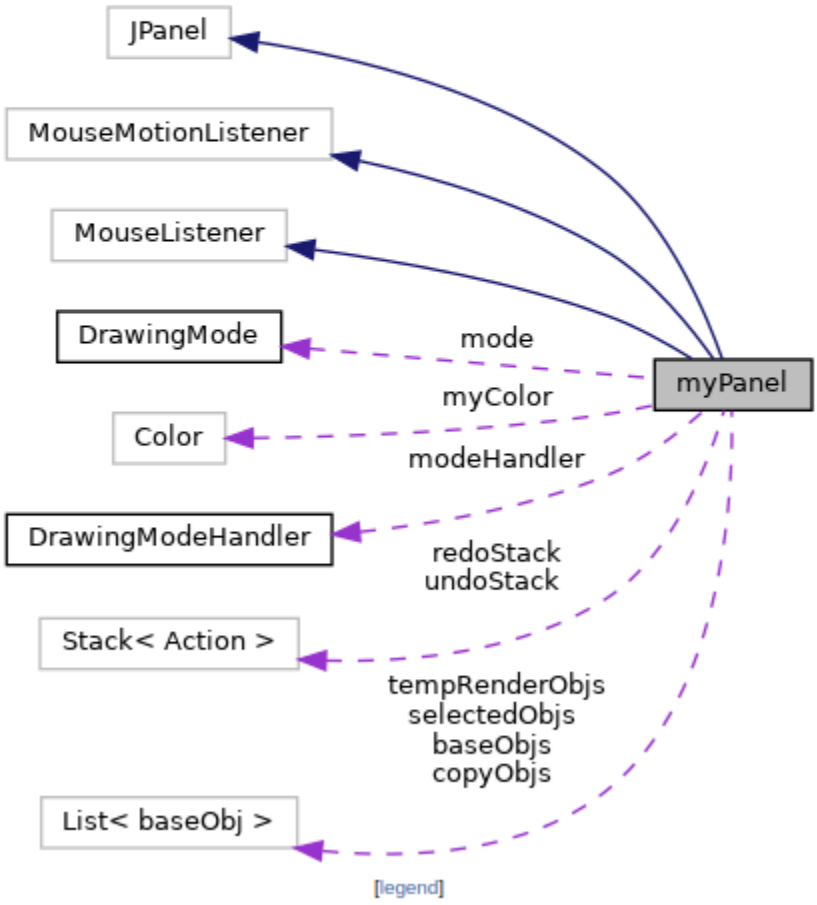




2.2.2.3 myPanel class

- The class **DrawingPanel** extends the **Jpanel** and implements the **MouseMotionListener, MouseListener, Serializable** interfaces. The **DrawingPanel** class has many members holding the information such as current **mode**, current drawing **shape,** current **clipboard** and current **selected** objects.

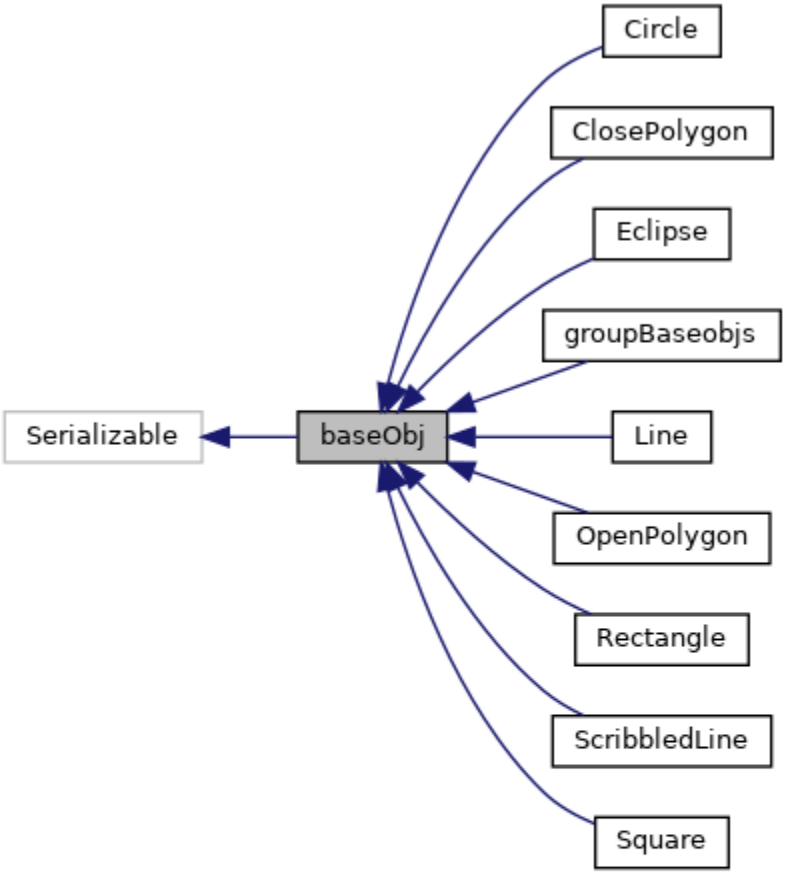


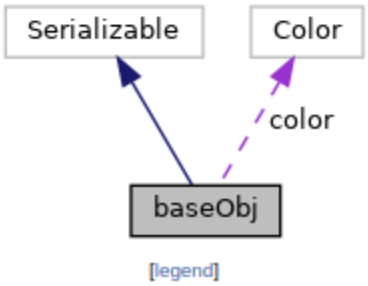


2.2.2.4 baseObj class and extended classes

- As we can see from the above diagram, there are multiple **List<baseObj>** which hold the **render**, **selected, copy** List.

- Below are the details about the base class **baseObj**, which implements the **Serializable** interface which will be used when saving the file.





- The design of **baseObj** is an abstraction of the operations of all objects that can be rendered on the sketchpad.

Public Member Functions

Color	**getColor** ()
void	**setColor** (Color c)

Package Functions

abstract boolean	**contains** (Point p, int offsetX, int offsetY)
abstract **baseObj**	**copy** ()
abstract void	**draw** (Graphics g, int offsetX, int offsetY)
abstract **Rectangle**	**getBounds** ()
abstract void	**gradient** (Graphics g, int offsetX, int offsetY)
abstract void	**translate** (double dx, double dy)

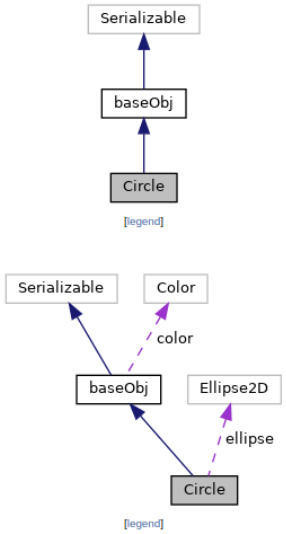
Private Attributes

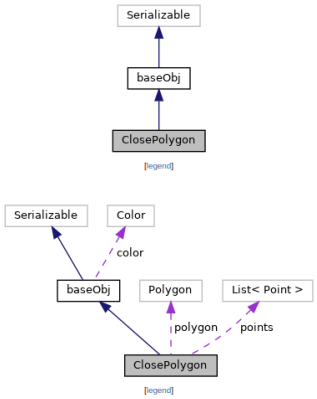
Color	**color**

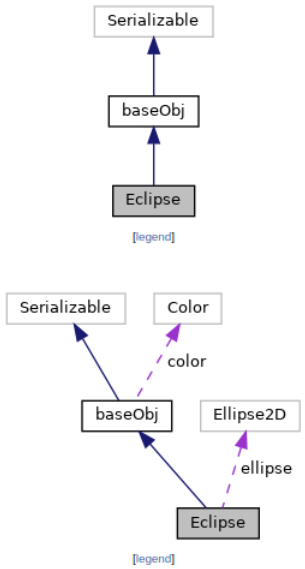
Static Private Attributes

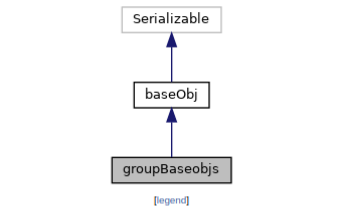
static final long	**serialVersionUID** = 1L

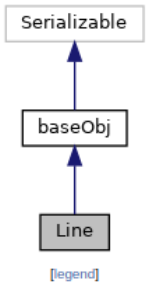
- When we implement a new drawing mode, we can **extend** the **baseObj** to make a customized class. Below are the extended classes.

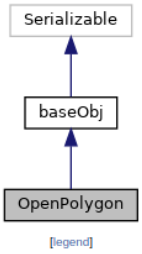


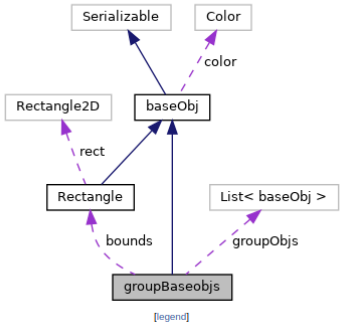


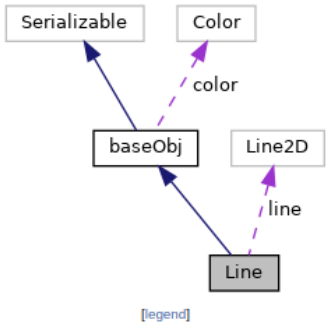


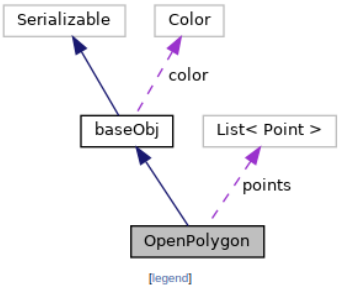


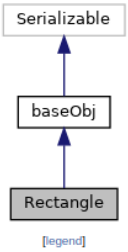


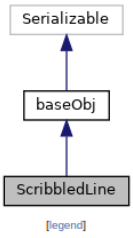


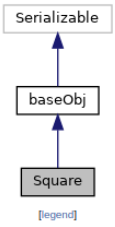


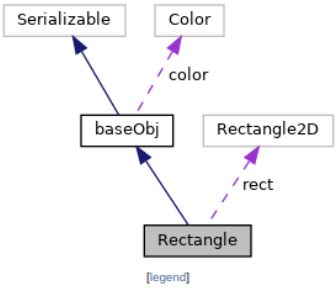


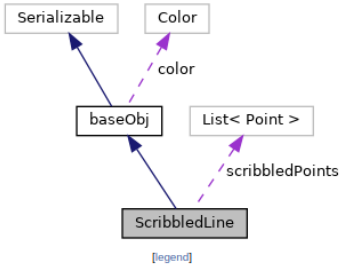


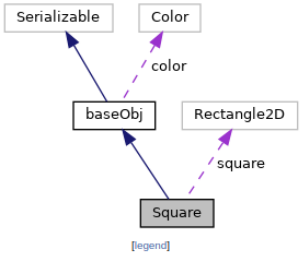










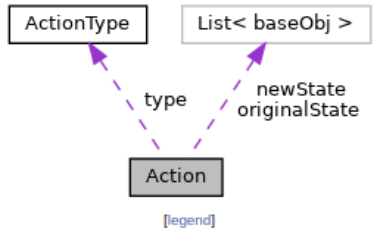


2.2.2.5 Action class for Undo/Redo

- There is an **Action** class which is very important when we try to implement the **Do** and **Undo** mechanisms. In the **Action** class we keep a copy of the List<baseObj> when **Cut, Copy, Paste, Group, Add, Remove, Group, Ungoup** happens.

Action Class Reference

Collaboration diagram for Action:



Package Functions

Action (List< **baseObj** > **originalState**, List< **baseObj** > **newState**, **ActionType** **type**)

Package Attributes

List< **baseObj** > **newState**

List< **baseObj** > **originalState**

ActionType **type**

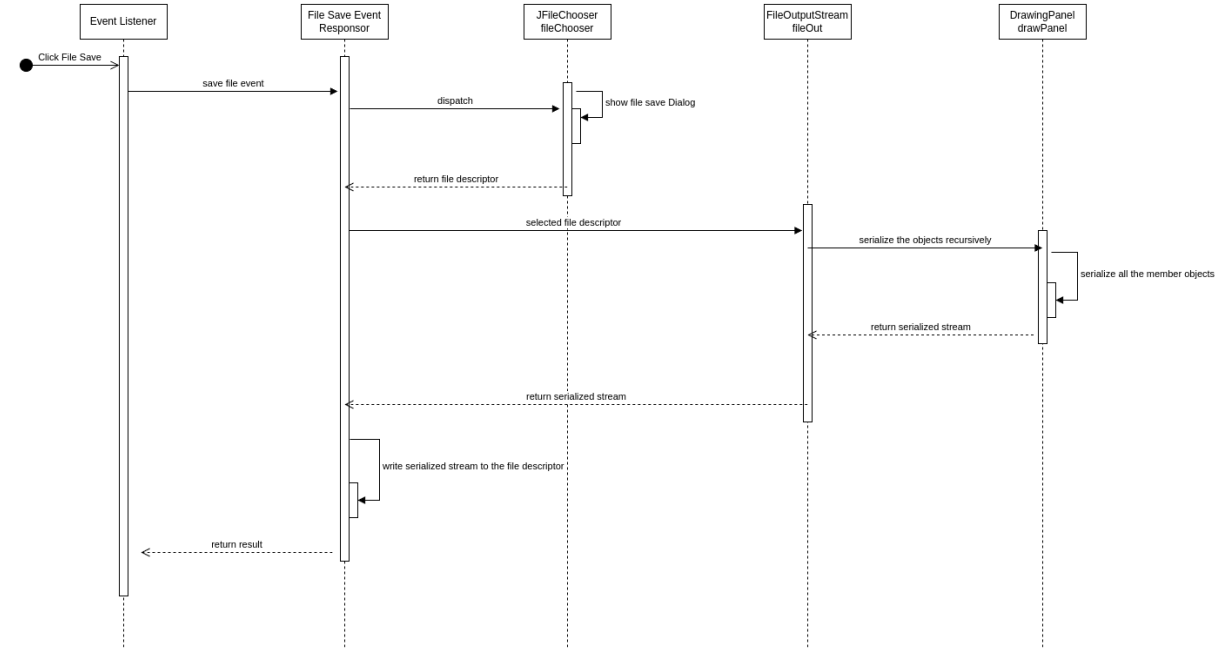
2.2.3 Data Flow Diagram

2.2.3.1 File Save

In this section, we will use the data flow diagram to explain the file save operation. There are other operations, we will not put too many diagrams here.

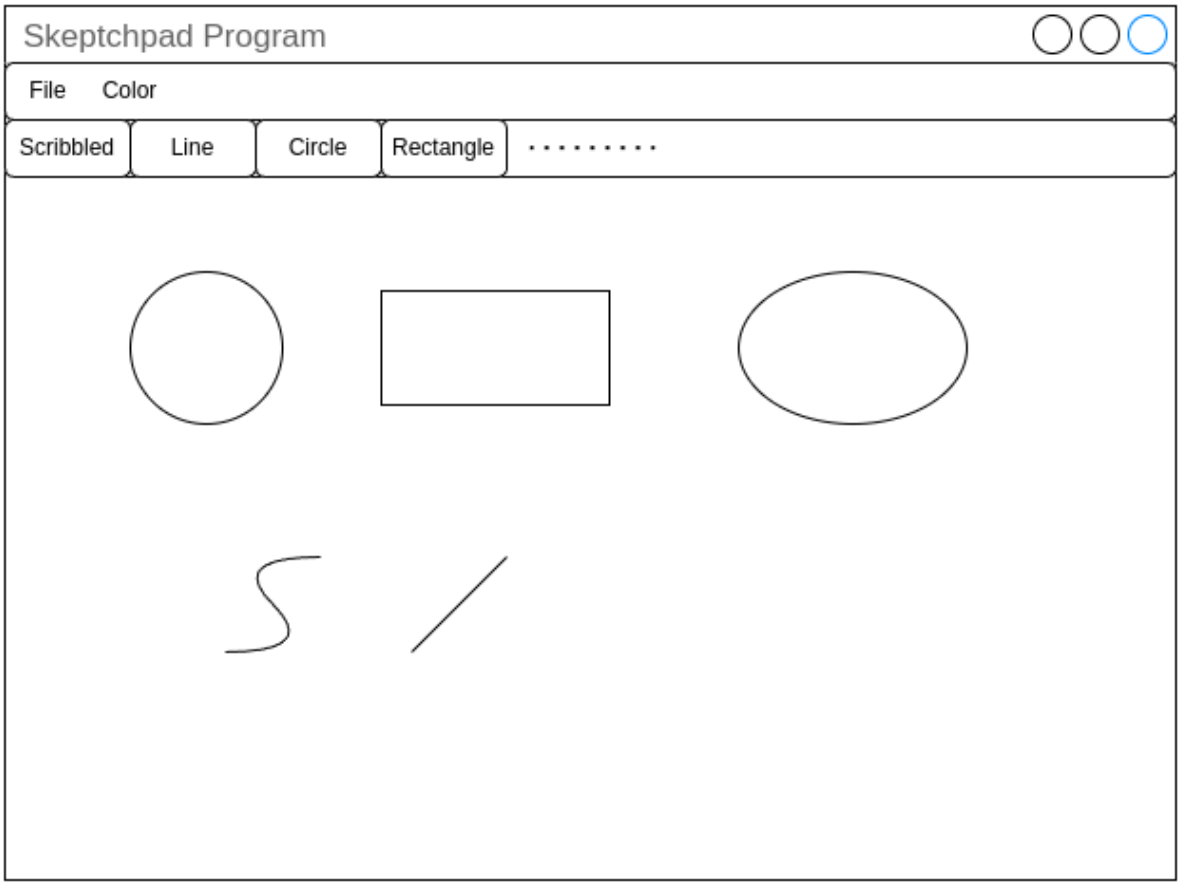
File Save Data Flow

In this diagram, we used the data flow diagram to show the operations of saving a file



2.2.4 UI Design

Below is the Design of the UI, for the drop boxes “File” and “Color”, we can choose from “load” “save” “Red” “Blue”.....



3 Implementation

3.1 Prove of Concept Stage

Using java AWT library to finish the POC.

Github: https://github.com/TristanWw/Sketchpad_JavaAWT/tree/dev-skeleton

3.2 Final Design

Using java AWT and Swing libraries together to implement the code. Source code can also be found in the source code folder.

Github: https://github.com/TristanWw/Sketchpad_JavaAWT/tree/dev-refactored

4. Demonstration

Below is a picture of the running program of the final design.
For detailed demonstration, please refer to the screen-capture videos.