2023 ITM Data Structure

# Drawing Editor

## Delete, Copy, Line, Grouping

19102100 Jiwoo Choi
(Team Leader)
19102094 Hyunsoo Shin
20102107 Jihwan Kim
22101993 Eungyeol Kim
22101997 Seonghun Park
22102002 Juchan Yang

# Table of Contents

# 1. Introduction

Our project goal is making a drawing editor that can draw li ne and shapes with delete, copy, line coloring, Grouping functions. Plus, we added additional functions such as redo, undo, zoom in and out, and file opening and saving, in order to make the drawing app better.
This task reminded us of “Microsoft paint”. We tried to make our own “paint” or drawing app and add additional functions with a user-friendly interface.

# 2. System Requirements

This drawing tool consistes of four sections, and the interface with design will be demonstrated closely in the following chapter. In this part, it shows what functions the following program has, and how each works.

## I. Selection Tool

The very first section of our tool is Selection Tool. Though there is any visual menu on the user interface, it can be used anytime, helping the user to select the entities to manipulate them easily. Simply Dragging a certain area, user can select every entity located in the designated field. The selected area will be visualized in blue, and only entities completely covered in the area will be selected. User can select a single entity, or multiple entities at once. The selection area will be visualized in a rectangular form.

## II. Util Tool

A. Copy (ctrl + c)
B. Paste (ctrl + v)
C. Delete
D. Grouping

The second section of our tools is ‘Util Tool’. This part contains some practical functions to handle the selected entity, or entities on the canvas. If the button is pressed, the user can apply the corresponding functions while the user should press the button again to stop applying any more additional functions. From the leftmost, there are four buttons of Copy, Paste, Delete, and Grouping.
Now, let’s see how it works. With the selection tool, the user can select the entity, or entities to handle. When you press the copy button, entities selected will be copied. To paste them, the user can simply click the press button, and the copied entity set will visualized on the right downward of the original entity set. These functions are also available by pressing ctrl key and z, y key, each operates copy and paste, in short key. In the case of Delete, which is like the previous one, the user can remove any kind of selected entities by pressing the Delete button. The user can use this tool with undo/redo tools, which will be handled in the section 5. Finally, there is Grouping. This tool is to edit multiple entities easily by joining them into a one group. The user can copy, paste, and delete the multiple entities easily by grouping them.

## III. Drawing Tool

A. Line
B. Ellipse
C. Rectangle
D. Triangle
E. Pencil
F. Trapezoid
G. Pentagon
H. Hexagon

The third section of our tool is ‘Drawing Tool’. The user can draw many kinds of entities using this part. There are eight kinds of entities that the user can draw. Line, Ellipse, Rectangle, Triangle from the first raw, Pencil, Trapezoid, pentagon, hexagon from the second raw. To draw the shapes, the user should click the starting point, drag it to the ending point, then release the mouse. These two points can be anywhere in the canvas, allowing the user to freely utilize our tool. Unlike other seven tools, the Pencil, E, is not a shape. This is a tool that the user can draw straight or cured lines freely, just like drawing a line on the paper using a pencil. This tool works in a same way with other drawing tools, click-drag-and release. Each result will be treated as a entity, and the user can see the number of entities on the canvas in the status bar.

## IV. Line Coloring

### A. Palette

The Fourth section is about changing the color of lines. Using the palette, the user can change the outline of the entity. When the user selects entities and press the Line button on the right, a palette pops up. By choosing the color on the palette the user can change the visual of selected entities’ outlines. The palette has several options to choose the color whatever the user wants, such as RDB, HSL, or HWB.

## V. Undo/Redo

A. Undo (ctrl + z)
B. Redo (ctrl + y)

The fifth section is about Undo and Redo functions. This is for the case when the user accidently makes mistakes, or drawback any kind of result the user made. To use this tool, just click the undo or redo button on the right side of the bar. Clicking the button once will each restore the one most current action that the user has made. This is also available by pressing the short key, ctrl key and z key for undo, and y key for redo. If multiple entities were manipulated in a single action, this undo/redo function will influence to them all.

## VI. Zooming

The sixth section contains zooming. The user can zoom in or zoom out by dragging the slide bar located on the status bar. The user can adjust the scale of zooming by dragging the

slide bar. If the pointer is gets closer to 0, it zooms out. If the pointer gets closer to 1, it zooms in. the user can operate this menu by dragging with mouse.

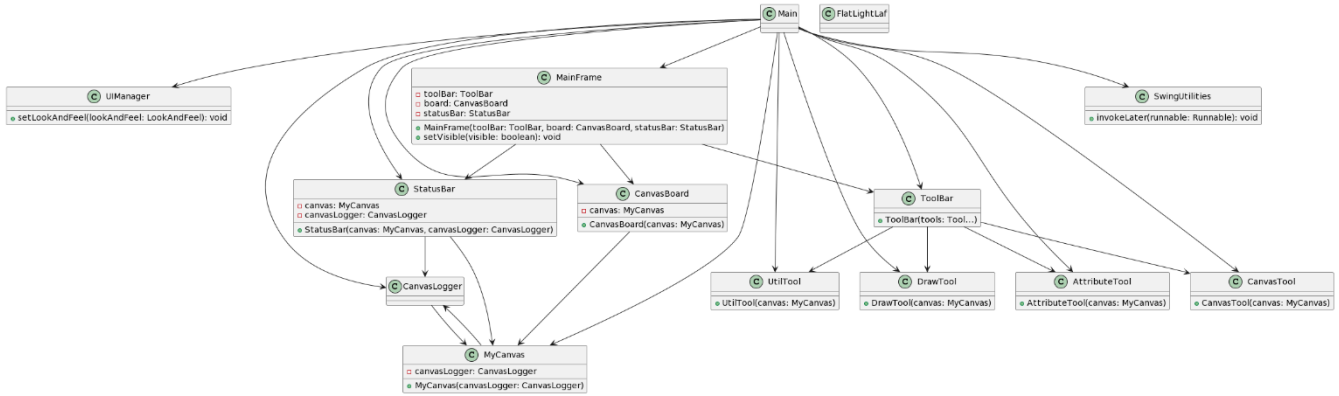
## VII. File Operation

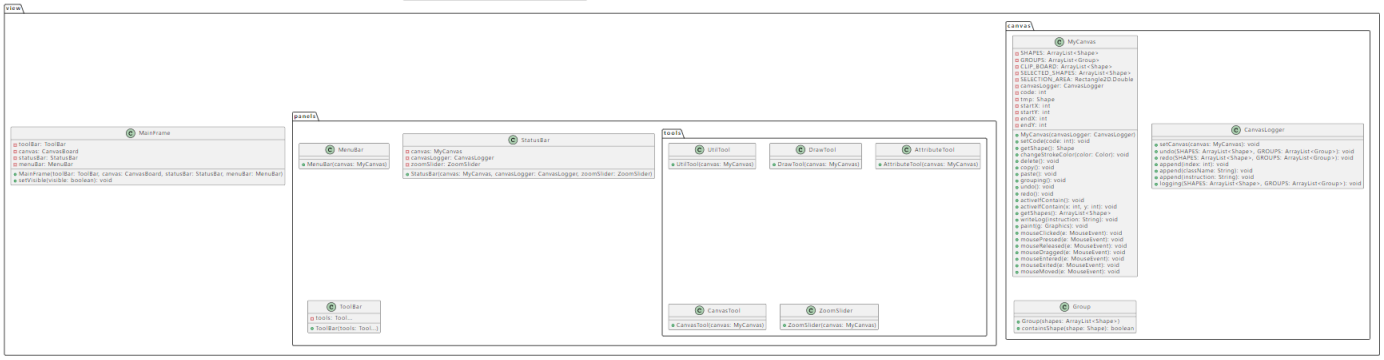
A. New File
B. Open
C. Save
D. Save as

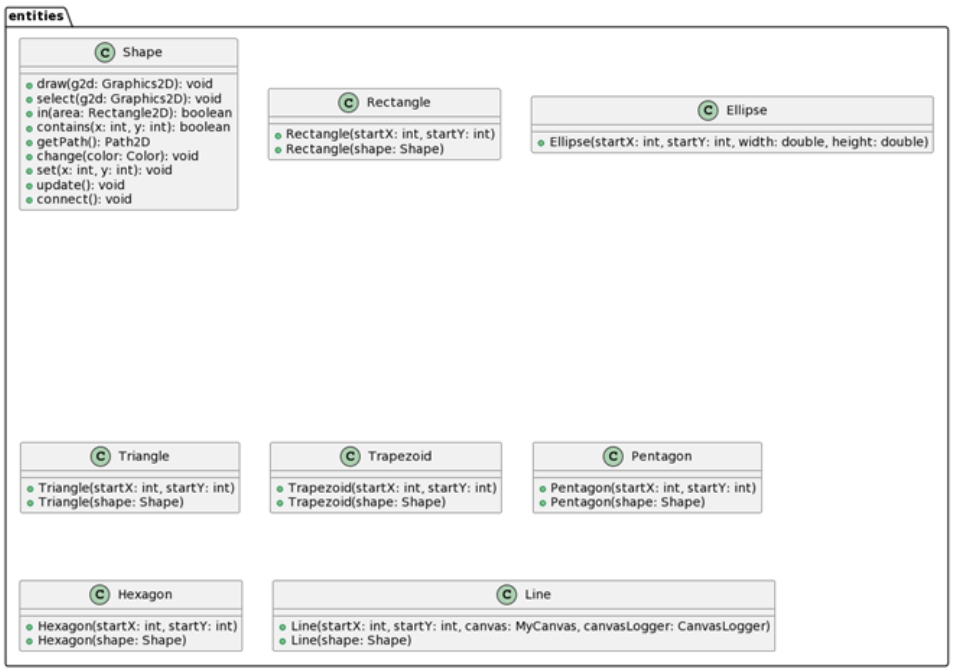
The last section is about file operation. This part is about opening and saving the canvas file. These functions are in the menu bar, on the top of our interface. There are four options. ‘New File’, ‘Open’, ‘Save’, ‘Save as’. Starting from the top, the ‘New File’ menu, the user can save the project as a new project. Next, the ‘Open’ menu, the user can bring the PNG formatted file to the canvas, allowing the user to draw continuously after with the previous project, or make projects with other pictures or photos. On the other hand, there are also some functions to save the project in canvas, using ‘Save’ menu or ‘Save as’ menu. Both saves the current version of canvas as PNG format to the designated location in the local system. The difference between these two are that ‘Save’ automatically save the file in storage with the same name with the previous version, while ‘Save as’ can specifically designate the name of the file, and the path where to save the project file. The user cannot use the Save menu if the user tries to save the project that hasn’t been saved. So this means to save the project for the first time, the user should use ‘Save as’ menu. If the user has saved the project previously, the user can use ‘Save’ menu. However, user can us ‘Save as’ for the pre-existing-but-has-saved-project, the user can use ‘New File’ menu.

# 3. System Design

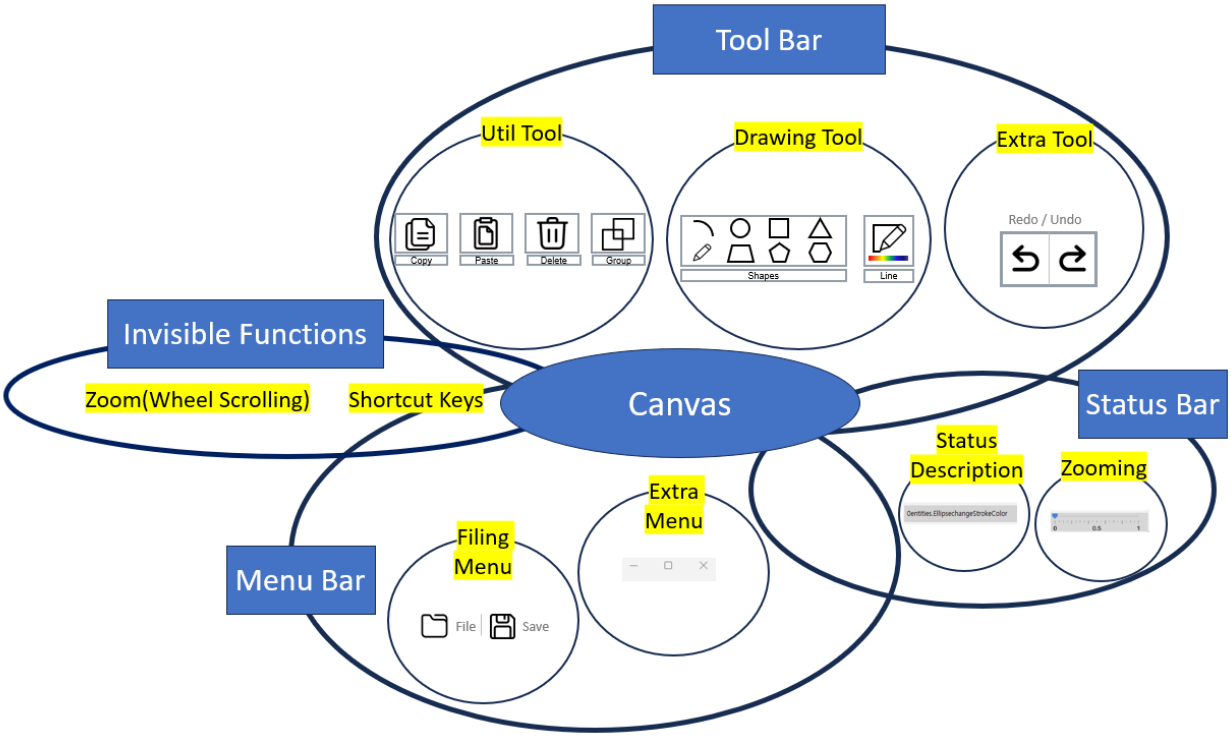
## - Class Diagram

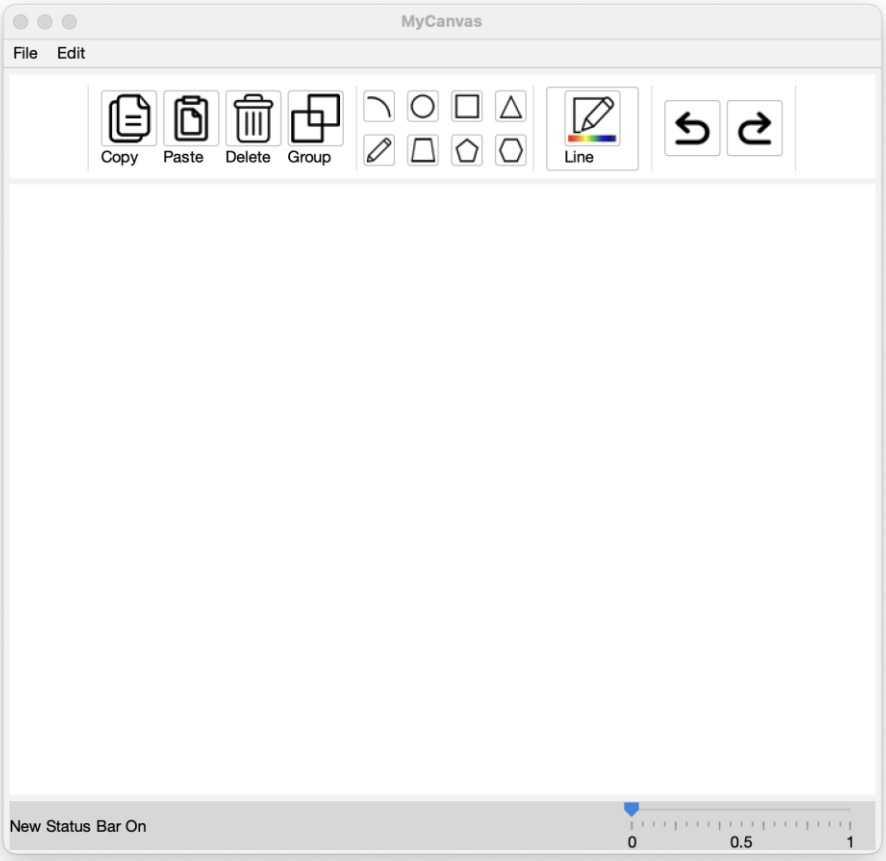



## - User Interface


Tool Bar
Util Tool
Copy
Paste
Delete
Group
Drawing Tool
Shapes
Line
Extra Tool
Redo / Undo
Invisible Functions
Zoom(Wheel Scrolling)
Shortcut Keys
Canvas
Status Bar
Status Description
Zooming
Extra Menu
Filing Menu
Menu Bar
File
Save




(Initailized on MacOS )

<Menu bar>

On the top of the frame, there is this menu bar.

The tool bar which is the main part for drawing and editing functions contains 3 kinds of tools; util tool, drawing tool, and extra tool. which consist of buttons and those buttons provide the appropriate functions for each tool.

The widest part is the canvas where we can see how shapes are drawn, just like a physical canvas for painters.

Lastly, there is a status bar, showing the current status. on the right edge, there's a line graphic which is the slicing for zooming.

<Menu bar>

File Management - In this menu bar, there are buttons for file level management such as saving and loading.

Extra Function – Just like most of app and web display, our drawing app offers screen minimizing, maximizing, and closing.

<Util Tool>

Delete - user can delete shapes by selecting entities and clicking ‘Delete’ button.

Copy and Paste - When the user selected shape and click the copy button, user can copy the shape. And we can paste the copied shape by clicking the paste button.

Group - User can make a group of objects by grouping them. We can define the group by dragging over entities and clicking the ‘Group’ button. Then, the objects are combined in a group that we can easily edit at once. In other words, we embodied the “Rubber Banding” function when we selected objects that we’ve made.

Drawing Tool - User can draw, insert entities and edit shapes and lines using this tool. There are given shapes in the shape box that user can use in this editor. User can choose between straight line, free line, circle, triangle, rectangle, trapezoid, pentagon and octagon. By clicking the decided shape and dragging our mouse, we pinpoint the first and the last edge and the shape is inserted.

Line Coloring –User can also change the color of lines to a certain color. When the user selects the shape and presses the line color button, you can see this color sample palate tab that you can choose the color you want.

<Extra Tool>

Redo/Undo - By clicking the arrow icons, you can undo or redo the actions so mistakes won’t hinder us.

Shortcut Keys - There are several shortcut keys for user’s convenience. For example, we can undo our activity by clicking the Control key and Z key or redo by Control key and Y key.

<Status Bar>

Current Status - This part shows the current status selected entities such as shape type, operation, and savings.

Zooming - User can zoom in the canvas with slicing bar or mouse wheel scrolling with ctrl key.

# 4. Team Roles & Responsibilities

**Jiwoo Choi**
I explained how Java's framework and GUI libraries(Awt, Swing) and their components are structured and implemented to my team members. Based on this, I established the initial framework with teammates. I designed the overall layout, applied LookAndFeel, and adjusted components to enhance the GUI apperance. I implemented the file-saving functionality and contributed to implementing the file-loading feature.

**Hyunsoo Shin**
Created a separate program to change colors that was able to be combined to our ‘DrawingEditor’ app with other teammates’ help / Wrote code to display buttons on MyCanvas / Implemented selection functions into our 'DrawingEditor’ package / Researched information to implement code to draw an ellipse / Implemented the function to zoom in/out the picture with a slider bar and added it to the Paint package. / Implemented ‘File Open’ function / Rewrite the shortcut code that caused the error to restore the function

**Jihwan Kim**
Researched for references and built codes about util parts. Built basic code for copy, paste and delete. Debugged selection and ellipse. Modified GUI presentation script.

**Eungyeol Kim**

Researched for codes to make ‘grouping’, ‘zooming’ and ‘pencil drawing’. Made separated programs having ‘grouping’, ‘zooming’, and ‘pencil drawing’ actions that worked properly on separated packages and were able to combined in our ‘Drawing App’ package with other teammates’ help. Designed GUI alone with teammates’ effective feedback. Summarized the expected functions and design of interface and wrote script for GUI presentation by myself and report with other members.

**Seonghun Park**

I've initially devised the overarching framework for a drawing tool application, harnessing the inherent capabilities of the javax.swing and java.awt packages. For the front-end, I've assigned a significant responsibility to the view package. For the back-end, encapsulated in the entities and enums packages, focuses on abstracting the objects being drawn or those recurring in nature. In line with Object-Oriented Programming (OOP) principles, I've tried to maximize the utilization of concepts such as Inheritance and Polymorphism in the implementation.

With this architectural foundation, the entities package encompasses class definitions pertinent to objects generated during drawing operations. Likewise, the enums package encapsulates specifications for properties that each button or component should possess, including images, text , with some getter functions. These properties are formally defined as enum classes. Furthermore, the view package serves as the visual interface for end-users, housing graphical elements such as buttons, canvases, and various tools.

To provide our team with a streamlined implementation process, within the entities package, I introduced an interface and abstract class for the Shape entity. This approach empowers our team members to leverage these classes for easier implementation. Taking charge of the implementation, I've successfully realized the majority of geometric entities, such as Ellipse, Group, Pentagon, Drawing Free Line(Pencil), etc.

In the enums package, we meticulously pre-designed enum classes (AttributeAuthority, CanvasAuthority, DrawAuthority, UtilAuthority) to categorize tools with similar characteristics. This design, facilitated through enumerations, served as a prelude to the implementation phase, providing valuable guidance to our team members.

Lastly, in the view package, a multitude of javax.swing and java.awt components were orchestrated to establish intricate inheritance relationships. By doing so, we pre-arranged and implemented a framework within which our team could comfortably navigate during the implementation phase, offering a structured foundation for the development process.

**Juchan Yang**

Researched for references and built codes about 'ToolBar', 'MenuBar', and entities including 'lines', 'rectangles', 'triangles', and 'ellipse'. Built codes and implemented functions for short key operation, such as 'ctrl+z', 'ctrl+y', Detected unexpected error operations that can possibly occur and analyzed the incident cause. Produced presentation resources, including building PPT slides, editing video and writing the final report,

## 5. Conclusion & Future Works

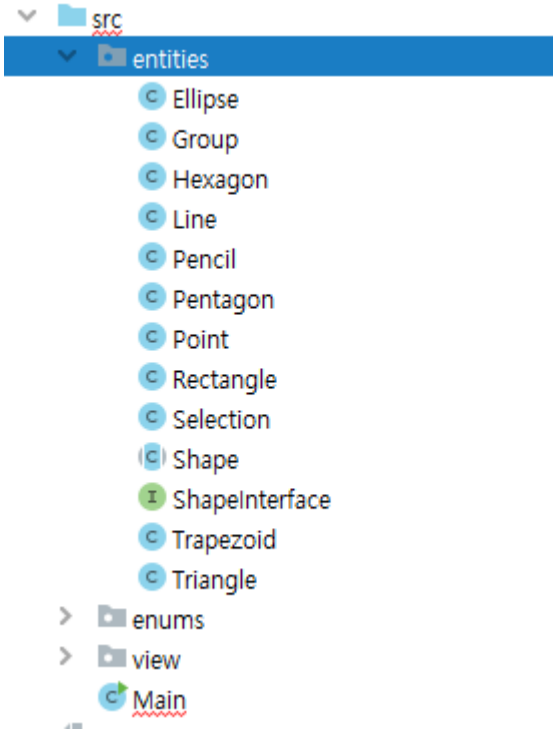
We made a drawing app that the user can draw straight lines, free lines and several kinds of shapes. It in consisted of four parts, the menu bar, tool bar, canvas, and status bar. It has copy

and paste function, which is also available in both using buttons and using ctrl + c, v short keys, delete function, and grouping function that binds several entities into a single group.
These operations can be held with the selection tool, which selects all the entities in the dragged area, visualized in blue. With drawing tool, the user can draw straight lines, ellipses, rectangles, triangles, free lines, trapezoid, pentagon, and hexagon. The user can change the format of entities' outlines, using the palette from the line button after selecting the intended entities. Also, the user can redo or undo the action with the button located in the right of the tool bar, or use 'ctrl + z, y' short key.

Additionally, there is zooming slide that can magnify or reduce the perspective scale by dragging the zoom slider point. To open or save the file, the user can use 'New File', 'Open', 'Save', 'Save as' menus to each make a new canvas, open new PNG file, rewrite the file, and save the project designating the file name and download path.

Furthermore, the drawing tool can be updated to support more various shapes such as arrows, cubes, or flowcharts, or supports functions to change the thickness or outlines. It can also updated to support online transaction for the simultaneous work of multiple users.

# 6. Appendices




(Structure of the program)

## [Entities]

### *Ellipse.java*

```
public class Ellipse extends Shape {
    public Ellipse(int x, int y) {
        super(x, y);
        points = zeroInitPoints(4);
        update();
        connect();
    }

    public Ellipse(Shape shape) {
        this(shape, true);
    }

    public Ellipse(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(4);
        update();
        connect();
    }

    @Override
```

```
    public void update() {
        points.get(0).set(leftUpperPoint);
        points.get(1).set(rightUpperPoint);
        points.get(2).set(rightLowerPoint);
        points.get(3).set(leftLowerPoint);
    }
}
```

*Group.java*

```
public class Group extends Shape{
    private final ArrayList<Shape> shapes = new ArrayList<>();

    public Group(ArrayList<Shape> shapes) {
        this.shapes.addAll(shapes);
        points = zeroInitPoints(4);
        update();
    }

    public Group(Group group, boolean offset) {
        points = zeroInitPoints(4);

        for(Shape shape: group.shapes) {
            String name = shape.getClass().getName();
            if (name.equals("entities.Rectangle")) {
                this.shapes.add(new Rectangle(shape, offset));
            } else if (name.equals("entities.Line")) {
                this.shapes.add(new Line(shape, offset));
            } else if (name.equals("entities.Triangle")) {
                this.shapes.add(new Triangle(shape, offset));
            } else if (name.equals("entities.Trapezoid")) {
                this.shapes.add(new Trapezoid(shape, offset));
            } else if (name.equals("entities.Pentagon")) {
                this.shapes.add(new Pentagon(shape, offset));
            } else if (name.equals("entities.Hexagon")) {
                this.shapes.add(new Hexagon(shape, offset));
            } else if (name.equals("entities.Ellipse")) {
                this.shapes.add(new Ellipse(shape, offset));
            } else if (name.equals("entities.Pencil")) {
                this.shapes.add(new Pencil(shape, offset));
            }
        }
        update();
    }

    public boolean containsShape(Shape shape) {
        return this.shapes.contains(shape);
    }
```

```
public int getSize() {
    return shapes.size();
}

@Override
public void update() {
    if(shapes.size()>0){
        this.leftUpperPoint.set(shapes.get(0).getLeftUpperPoint());
        this.rightUpperPoint.set(shapes.get(0).getRightUpperPoint());
        this.leftLowerPoint.set(shapes.get(0).getLeftLowerPoint());
        this.rightLowerPoint.set(shapes.get(0).getRightLowerPoint());
    }

    for(Shape shape: shapes){
        if(shape.leftUpperPoint.getX() < this.leftUpperPoint.getX())
            this.leftUpperPoint.setX(shape.leftUpperPoint.getX());
        if(shape.leftUpperPoint.getY() < this.leftUpperPoint.getY())
            this.leftUpperPoint.setY(shape.leftUpperPoint.getY());

        if(shape.rightUpperPoint.getX() > this.rightUpperPoint.getX())
            this.rightUpperPoint.setX(shape.rightUpperPoint.getX());
        if(shape.rightUpperPoint.getY() < this.rightUpperPoint.getY())
            this.rightUpperPoint.setY(shape.rightUpperPoint.getY());

        if(shape.leftLowerPoint.getX() < this.leftLowerPoint.getX())
            this.leftLowerPoint.setX(shape.leftLowerPoint.getX());
        if(shape.leftLowerPoint.getY() > this.leftLowerPoint.getY())
            this.leftLowerPoint.setY(shape.leftLowerPoint.getY());

        if(shape.rightLowerPoint.getX() > this.rightLowerPoint.getX())
            this.rightLowerPoint.setX(shape.rightLowerPoint.getX());
        if(shape.rightLowerPoint.getY() > this.rightLowerPoint.getY())
            this.rightLowerPoint.setY(shape.rightLowerPoint.getY());
    }

    points.get(0).set(leftUpperPoint);
    points.get(1).set(rightUpperPoint);
    points.get(2).set(rightLowerPoint);
    points.get(3).set(leftLowerPoint);
}

@Override
public void change(Color color) {
    for(Shape shape: shapes)
        shape.strokeColor = color;
}

@Override
public void draw(Graphics2D g2d) {
```

```
        for(Shape shape: shapes) {
            g2d.setStroke(shape.strokeSize);
            g2d.setColor(shape.strokeColor);
            g2d.draw(shape.path);
        }
    }

    public void remove(ArrayList<Shape> sh) {
        shapes.removeAll(sh);
        update();
    }

    public ArrayList<Shape> getShapes() {
        return shapes;
    }
}
```

*Hexagon.java*

```
public class Hexagon extends Shape{
    public Hexagon(int x, int y) {
        super(x, y);
        points = zeroInitPoints(6);
        update();
    }

    public Hexagon(Shape shape) {
        this(shape, true);
    }

    public Hexagon(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(6);
        update();
        connect();
    }

    public void update() {
        points.get(0).set(new Point(
                (int) (leftUpperPoint.getX()*0.8 + rightUpperPoint.getX()*0.2),
                leftUpperPoint.getY()
        ));
        points.get(1).set(new Point(
                (int) (rightUpperPoint.getX()*0.8 + leftUpperPoint.getX()*0.2),
                rightUpperPoint.getY()
        ));
        points.get(2).set(new Point(
                rightUpperPoint.getX(),
                (rightUpperPoint.getY() + rightLowerPoint.getY())/2
        ));
```

```
            points.get(3).set(new Point(
                    (int) (rightUpperPoint.getX()*0.8 + leftUpperPoint.getX()*0.2),
                    rightLowerPoint.getY()
            ));
            points.get(4).set(new Point(
                    (int) (leftUpperPoint.getX()*0.8 + rightUpperPoint.getX()*0.2),
                    leftLowerPoint.getY()
            ));
            points.get(5).set(new Point(
                    leftUpperPoint.getX(),
                    (rightUpperPoint.getY() + rightLowerPoint.getY())/2
            ));
        }
}
```

*Line.java*

```
public class Line extends Shape{

    public Line(int x, int y) {
        super(x, y);
        points = zeroInitPoints(2);
        update();
    }

    public Line(Shape shape) {
        this(shape, true);
    }

    public Line(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(2);
        update();
        connect();
    }

    @Override
    public void update() {
        points.get(0).set(startPoint);
        points.get(1).set(endPoint);
    }
}
```

*Pencil.java*

```
public class Pencil extends Shape{

    public Pencil(int x, int y) {
        super(x, y);
```

```
        points = new ArrayList<>();
    }

    public Pencil(Shape shape) {
        this(shape, true);
    }

    public Pencil(Shape shape, boolean offset) {
        super(shape, offset);
        points = new ArrayList<>();
        for(Point p : shape.points)
            points.add(new Point(p, offset));
        connect();
    }

    @Override
    public void set(int endX, int endY) {
        points.add(new Point(endX, endY));

        leftUpperPoint.set(
                Math.min(leftUpperPoint.getX(), endX),
                Math.min(leftUpperPoint.getY(), endY)
        );
        rightUpperPoint.set(
                Math.max(rightUpperPoint.getX(), endX),
                Math.min(rightUpperPoint.getY(), endY)
        );
        leftLowerPoint.set(
                Math.min(leftLowerPoint.getX(), endX),
                Math.max(leftLowerPoint.getY(), endY)
        );
        rightLowerPoint.set(
                Math.max(rightLowerPoint.getX(), endX),
                Math.max(rightLowerPoint.getY(), endY)
        );
    }

    @Override
    public void connect() {
        path.reset();
        if (points.size() > 1) {
            path.moveTo(points.get(0).getX(), points.get(0).getY());
            for (int i = 1; i < points.size(); i++)
                path.lineTo(points.get(i).getX(), points.get(i).getY());
        }
    }

    @Override
    public void update() {}
```

```
}
```

*Pentagon.java*

```
package entities;

public class Pentagon extends Shape{

    public Pentagon(int x, int y) {
        super(x, y);
        points = zeroInitPoints(5);
        update();
    }

    public Pentagon(Shape shape) {
        this(shape, true);
    }

    public Pentagon(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(5);
        update();
        connect();
    }

    public void update() {
        double halfUpperEdge = Math.abs((rightUpperPoint.getX() -
leftUpperPoint.getX())/2);

        points.get(0).set(new Point(
                (int) (leftUpperPoint.getX() + halfUpperEdge),
                leftUpperPoint.getY()
        ));
        points.get(1).set(new Point(
                rightUpperPoint.getX(),
                (int)(rightUpperPoint.getY()*0.65 + rightLowerPoint.getY()*0.35)
        ));
        points.get(2).set(new Point(
                (int) (rightLowerPoint.getX()*0.8 + leftLowerPoint.getX()*0.2),
                rightLowerPoint.getY()
        ));
        points.get(3).set(new Point(
                (int) (leftLowerPoint.getX()*0.8 + rightLowerPoint.getX()*0.2),
                leftLowerPoint.getY()
        ));
        points.get(4).set(new Point(
                leftUpperPoint.getX(),
                (int)(leftUpperPoint.getY()*0.65 + leftLowerPoint.getY()*0.35)
        ));
```

```
    }
}
```

*Point.java*

```
package entities;

import java.awt.Graphics2D;
import java.awt.Color;



public class Point {
    private int x;
    private int y;
    private int radius = 6;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public Point(Point point) {
        this(point, true);
    }

    public Point(Point point, boolean offset) {
        if(offset) {
            this.x = point.x + 10;
            this.y = point.y + 10;
        } else {
            this.x = point.x;
            this.y = point.y;
        }
    }

    void draw(Graphics2D g2d) {
        g2d.setColor(Color.RED);
        g2d.fillOval(x-radius/2, y-radius/2, radius, radius);
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }
```

```
    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public void set(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void set(Point point) {
        this.x = point.x;
        this.y = point.y;
    }

}
```

*Rectangle.java*

```
package entities;


public class Rectangle extends Shape{


    public Rectangle(int x, int y) {
        super(x, y);


        points = zeroInitPoints(4);
        update();
    }


    public Rectangle(Shape shape) {
        this(shape, true);
    }


    public Rectangle(Shape shape, boolean offset) {
```

```
        super(shape, offset);
        points = zeroInitPoints(4);
        update();
        connect();
    }

    @Override
    public void update() {
        points.get(0).set(leftUpperPoint);
        points.get(1).set(rightUpperPoint);
        points.get(2).set(rightLowerPoint);
        points.get(3).set(leftLowerPoint);
    }
}
```

*Seleciton.java*

```
package entities;

import java.awt.*;

public class Selection extends Shape{
    private Color fillColor = new Color(0, 0, 255, 100);

    public Selection(int x, int y) {
        super(x, y);
        points = zeroInitPoints(4);
        update();
    }
```

```
    @Override
    public void update() {
        points.get(0).set(leftUpperPoint);
        points.get(1).set(rightUpperPoint);
        points.get(2).set(rightLowerPoint);
        points.get(3).set(leftLowerPoint);
    }

    @Override
    public void draw(Graphics2D g2d) {
        g2d.setStroke(strokeSize);
        g2d.setColor(Color.BLUE);
        g2d.draw(path);
        g2d.setColor(fillColor);
        g2d.fill(path);
    }
}
```

*Shape.java*

```
package entities;

import java.awt.*;
import java.awt.geom.*;
import java.util.*;



public abstract class Shape implements ShapeInterface{
    ArrayList<Point> points;
    Point startPoint;
```

```
Point endPoint;
Point leftUpperPoint;
Point rightUpperPoint;
Point leftLowerPoint;
Point rightLowerPoint;
Path2D path = new Path2D.Double();
BasicStroke strokeSize = new BasicStroke(2.0f);
Color strokeColor = Color.BLACK;

public Shape() {
    this(0, 0);
}

public Shape(int x, int y) {
    startPoint = new Point(x, y);
    endPoint = new Point(x, y);
    leftUpperPoint = new Point(x, y);
    rightUpperPoint = new Point(x, y);
    leftLowerPoint = new Point(x, y);
    rightLowerPoint = new Point(x, y);
}

public Shape(Shape shape, boolean offset) {
    this(shape);
    startPoint = new Point(shape.startPoint, offset);
    endPoint = new Point(shape.endPoint, offset);
    leftUpperPoint = new Point(shape.leftUpperPoint, offset);
    rightUpperPoint = new Point(shape.rightUpperPoint, offset);
    leftLowerPoint = new Point(shape.leftLowerPoint, offset);
    rightLowerPoint = new Point(shape.rightLowerPoint, offset);
```

```
}

public Shape(Shape shape) {
    strokeSize = shape.strokeSize;
    strokeColor = shape.strokeColor;
}

public void set(int endX, int endY) {
    endPoint = new Point(endX, endY);
    int x = startPoint.getX();
    int y = startPoint.getY();

    leftUpperPoint.set(
            Math.min(x, endX),
            Math.min(y, endY)
    );
    rightUpperPoint.set(
            Math.max(x, endX),
            Math.min(y, endY)
    );
    leftLowerPoint.set(
            Math.min(x, endX),
            Math.max(y, endY)
    );
    rightLowerPoint.set(
            Math.max(x, endX),
            Math.max(y, endY)
    );
}
```

```
public boolean contains(int x, int y) {
    return path.contains(x, y);
}

public boolean in(Rectangle2D rect) {
    return rect.contains(path.getBounds2D());
}

@Override
public void change(Color color) {
    strokeColor = color;
}

@Override
public void select(Graphics2D g2d) {
    for(Point point: points) point.draw(g2d);
}

@Override
public void draw(Graphics2D g2d) {
    g2d.setStroke(strokeSize);
    g2d.setColor(strokeColor);
    if(this.getClass().getName().equals("entities.Ellipse")) {
        g2d.drawOval(
                leftUpperPoint.getX(), leftUpperPoint.getY(),
                rightLowerPoint.getX() - leftUpperPoint.getX(),
                rightLowerPoint.getY() - leftUpperPoint.getY()
        );
        return;
    }
```

```
        g2d.draw(path);
    }

    @Override
    public void connect() {
        path.reset();
        if (points.size() > 1) {
            path.moveTo(points.get(0).getX(), points.get(0).getY());
            for (int i = 1; i < points.size(); i++)
                path.lineTo(points.get(i).getX(), points.get(i).getY());
            path.closePath();
        }
    }

    protected ArrayList<Point> zeroInitPoints(int n) {
        ArrayList<Point> tmp = new ArrayList<>(n);
        for(int i=0; i<n; i++) tmp.add(new Point(0, 0));
        return tmp;
    }

    public Point getLeftUpperPoint() {
        return leftUpperPoint;
    }

    public Point getLeftLowerPoint() {
        return leftLowerPoint;
    }

    public Point getRightUpperPoint() {
        return rightUpperPoint;
```

```
    }

    public Point getRightLowerPoint() {
        return rightLowerPoint;
    }
}

// entities ShapeInterface
package entities;

import java.awt.*;

public interface ShapeInterface {
    public void draw(Graphics2D g2d);
    public void select(Graphics2D g2d);
    public void change(Color color);
    public void connect();
    public void update();
}

// entities.Trapezoid
package entities;

public class Trapezoid extends Shape{

    public Trapezoid(int x, int y) {
        super(x, y);
        points = zeroInitPoints(4);
        update();
    }
```

```
    public Trapezoid(Shape shape) {
        this(shape, true);
    }

    public Trapezoid(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(4);
        update();
        connect();
    }

    @Override
    public void update() {
        points.get(0).set(new Point(
                (int) (leftUpperPoint.getX()*0.8 + rightUpperPoint.getX()*0.2),
                leftUpperPoint.getY()
        ));
        points.get(1).set(
                new Point((int) (leftUpperPoint.getX()*0.2 + rightUpperPoint.getX()*0.8),
                        leftUpperPoint.getY())
        );
        points.get(2).set(rightLowerPoint);
        points.get(3).set(leftLowerPoint);
    }
}

// entities.Triangle
package entities;
```

```
public class Triangle extends Shape{

    public Triangle(int x, int y) {
        super(x, y);
        points = zeroInitPoints(3);
        update();
    }

    public Triangle(Shape shape) {
        this(shape, true);
    }

    public Triangle(Shape shape, boolean offset) {
        super(shape, offset);
        points = zeroInitPoints(3);
        update();
        connect();
    }

    @Override
    public void update() {
        points.get(0).set(new Point((leftUpperPoint.getX() + rightUpperPoint.getX())/2,
                leftUpperPoint.getY()));
        points.get(1).set(rightLowerPoint);
        points.get(2).set(leftLowerPoint);
    }
}
// enums.AttributeAuthority
package enums;
```

```
import javax.swing.ImageIcon;



public enum AttributeAuthority {
    LINE_COLOR(new ImageIcon("images/color.png"), "Line", 20);

    private final ImageIcon icon;
    private final String name;
    private final int code;

    AttributeAuthority(ImageIcon icon, String name, int code) {
        this.icon = icon;
        this.name = name;
        this.code = code;
    }

    public ImageIcon getIcon() {
        return icon;
    }

    public String getName() {
        return name;
    }

    public int getCode() {
        return code;
    }
}
```

```
// enums.CanvasAuthority
package enums;

import javax.swing.ImageIcon;



public enum CanvasAuthority {
    UNDO(new ImageIcon("images/undo.png"), 30),
    REDO(new ImageIcon("images/redo.png"), 31);



    private final ImageIcon icon;
    private final int code;

    CanvasAuthority(ImageIcon icon, int code) {
        this.icon = icon;
        this.code = code;
    }

    public ImageIcon getIcon() {
        return icon;
    }

    public int getCode() {
        return code;
    }
}

// enums.DrawAuthority
```

```
package enums;

import javax.swing.ImageIcon;



public enum DrawAuthority {
    LINE(new ImageIcon("images/draws/line.png"), 10),
    CIRCLE(new ImageIcon("images/draws/ellipse.png"), 11),
    RECTANGLE(new ImageIcon("images/draws/rectangle.png"), 12),
    TRIANGLE(new ImageIcon("images/draws/triangle.png"), 13),
    PENCIL(new ImageIcon("images/draws/pencil.png"), 14),
    TRAPEZOID(new ImageIcon("images/draws/trapezoid.png"), 15),
    PENTAGON(new ImageIcon("images/draws/pentagon.png"), 16),
    HEXAGON(new ImageIcon("images/draws/hexagon.png"), 17);



    private final ImageIcon icon;
    private final int code;

    DrawAuthority(ImageIcon icon, int code) {
        this.icon = icon;
        this.code = code;
    }

    public ImageIcon getIcon() {
        return icon;
    }

    public int getCode() {
```

```
        return code;
    }
}

// enums.UtilAuthority
package enums;

import javax.swing.ImageIcon;



/* 그루핑, 색변경, 복사/붙여넣기 상수 */
public enum UtilAuthority {
    COPY(new ImageIcon("images/copy.png"), "Copy", 0),
    PASTE(new ImageIcon("images/paste.png"), "Paste", 1),
    DELETE(new ImageIcon("images/delete.png"), "Delete", 2),
    GROUP(new ImageIcon("images/group.png"), "Group", 3);

    private final ImageIcon icon;
    private final String name;
    private final int code;



    UtilAuthority(ImageIcon icon, String name, int code) {
        this.icon = icon;
        this.name = name;
        this.code = code;
    }

    public String getName() {
```

```
        return name;
    }

    public ImageIcon getIcon() {
        return icon;
    }

    public int getCode() {
        return code;
    }
}

// view.MainFrame
package view;

import view.panels.*;
import view.panels.MenuBar;

import javax.swing.*;
import java.awt.*;



public class MainFrame extends JFrame {
    public MainFrame(ToolBar toolBar, CanvasBoard canvas, StatusBar statusBar, MenuBar
menuBar) {
        setTitle("MyCanvas");
        setLayout(new FlowLayout(FlowLayout.LEFT));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setResizable(false);
        setSize(720, 695);
```

```
        setJMenuBar(menuBar);
        add(toolBar);
        add(canvas);
        add(statusBar);
    }
}

// view.canvas.CanvasLogger
package view.canvas;

import entities.*;

import javax.swing.*;
import java.util.*;



public class CanvasLogger extends JLabel {
    private String TEMP = "";
    private static Stack<String> MSG_STACK;
    private static Stack<ArrayList<Shape>> LOG_SHAPE_STACK, REDO_SHAPE_STACK;
    private static Stack<ArrayList<Group>> LOG_GROUP_STACK, REDO_GROUP_STACK;
    private MyCanvas canvas; // DO NOT REMOVE THIS CODE


    public CanvasLogger() {
        super();
        init();
    }
```

```
public void setCanvas(MyCanvas canvas) {
    this.canvas = canvas;
}

public void init() {
    TEMP = "";
    MSG_STACK = new Stack<>();
    LOG_SHAPE_STACK = new Stack<>();
    LOG_GROUP_STACK = new Stack<>();
    REDO_SHAPE_STACK = new Stack<>();
    REDO_GROUP_STACK = new Stack<>();

    setText("New Status Bar On");
    setSize(570, 20);
}

public void logging(ArrayList<Shape> shapes, ArrayList<Group> groups) {
    setText(TEMP);
    ArrayList<Shape> tmp1 = new ArrayList<>();
    for(Shape shape: shapes) {
        String name = shape.getClass().getName();
        switch (name) {
            case "entities.Rectangle" -> tmp1.add(new Rectangle(shape, false));
            case "entities.Line" -> tmp1.add(new Line(shape, false));
            case "entities.Triangle" -> tmp1.add(new Triangle(shape, false));
            case "entities.Trapezoid" -> tmp1.add(new Trapezoid(shape, false));
            case "entities.Pentagon" -> tmp1.add(new Pentagon(shape, false));
            case "entities.Hexagon" -> tmp1.add(new Hexagon(shape, false));
            case "entities.Ellipse" -> tmp1.add(new Ellipse(shape, false));
            case "entities.Pencil" -> tmp1.add(new Pencil(shape, false));
```

```
            }
        }
        LOG_SHAPE_STACK.add(tmp1);

        ArrayList<Group> tmp2 = new ArrayList<>(groups);
        for(Group group: groups)
            tmp2.add(new Group(group, false));
        LOG_GROUP_STACK.add(tmp2);
        TEMP = "";
    }

    public void undo(ArrayList<Shape> shapes, ArrayList<Group> groups) {
        if(LOG_SHAPE_STACK.size() == 0) return;
        REDO_SHAPE_STACK.add(LOG_SHAPE_STACK.pop());
        REDO_GROUP_STACK.add(LOG_GROUP_STACK.pop());
        apply(shapes, groups);
    }

    public void redo(ArrayList<Shape> shapes, ArrayList<Group> groups) {
        if(REDO_SHAPE_STACK.size() == 0) return;
        LOG_SHAPE_STACK.add(REDO_SHAPE_STACK.pop());
        LOG_GROUP_STACK.add(REDO_GROUP_STACK.pop());
        apply(shapes, groups);
    }

    private void apply(ArrayList<Shape> shapes, ArrayList<Group> groups) {
        shapes.clear(); groups.clear();
        if(LOG_SHAPE_STACK.size() == 0) return;
        shapes.addAll(LOG_SHAPE_STACK.get(LOG_SHAPE_STACK.size()-1));
        groups.addAll(LOG_GROUP_STACK.get(LOG_GROUP_STACK.size()-1));
```

```
    }

    public void append(String logMsg) {
        TEMP += "\t" + logMsg;
    }

    public void append(int i) {
        TEMP += "\t" + String.format("%d", i);
    }

    public void update() {
        setText(MSG_STACK.get(MSG_STACK.size()-1));
    }
}

// view.canvas.MyCanvas
package view.canvas;

import entities.Rectangle;
import entities.Shape;
import entities.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.awt.image.*;
import java.util.*;
```

```
public class MyCanvas extends JPanel implements MouseListener, MouseMotionListener {
    private static ArrayList<Shape> SHAPES = new ArrayList<>();
    private static ArrayList<Group> GROUPS = new ArrayList<>();
    private static ArrayList<Shape> CLIP_BOARD = new ArrayList<>();
    private static ArrayList<Shape> SELECTED_SHAPES = new ArrayList<>();
    private static Rectangle2D SELECTION_AREA = new Rectangle2D.Double();
    private CanvasLogger canvasLogger;
    private int code;
    private Shape tmp;
    private int startX, startY, endX, endY;
    private double zoomFactor = 1.0;
    private Selection selection;
    private BufferedImage loadedImage;

    public MyCanvas(CanvasLogger canvasLogger) {
        this.canvasLogger = canvasLogger;
        setPreferredSize(new Dimension(700, 490));
        setBackground(Color.WHITE);
        addMouseListener(this);
        addMouseMotionListener(this);
        addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {handleKeyPress(e);}
        });

        InputMap inputMap = getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
        ActionMap actionMap = getActionMap();
```

```
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_Z,
InputEvent.CTRL_DOWN_MASK), "undoAction");
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_Y,
InputEvent.CTRL_DOWN_MASK), "redoAction");
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_C,
InputEvent.CTRL_DOWN_MASK), "copyAction");
        inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_V,
InputEvent.CTRL_DOWN_MASK), "pasteAction");


        actionMap.put("undoAction", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                undo();
            }
        });
        actionMap.put("redoAction", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                redo();
            }
        });
        actionMap.put("copyAction", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                copy();
            }
        });
        actionMap.put("pasteAction", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                paste();
            }
```

```
        });
        setFocusable(true);
        requestFocus();
    }

    public void init() {
        canvasLogger.init();
        tmp = null;
        SHAPES = new ArrayList<>();
        GROUPS = new ArrayList<>();
        CLIP_BOARD = new ArrayList<>();
        SELECTED_SHAPES = new ArrayList<>();
        SELECTION_AREA = new Rectangle2D.Double();
        zoomFactor = 1.0;
        setPreferredSize(new Dimension(700, 490));
        setBackground(Color.WHITE);
        addMouseListener(this);
        addMouseMotionListener(this);
        addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) { handleKeyPress(e); }
        });
        setFocusable(true);
        requestFocus();
    }

    public void handleKeyPress(KeyEvent e) {
        if (e.isControlDown()) {
            if (e.getKeyCode() == KeyEvent.VK_Z) {
                undo();
```

```
            } else if (e.getKeyCode() == KeyEvent.VK_Y) {
                redo();
            } else if (e.getKeyCode() == KeyEvent.VK_C) {
                copy();
            } else if (e.getKeyCode() == KeyEvent.VK_V) {
                paste();
            }
        }
    }

    public void setCode(int code) {
        this.code = code!= this.code ? code:-1;
    }

    public Shape getShape() {
        if(code == 10) {
            return new Line(startX, startY);
        } else if(code == 11) {
            return new Ellipse(startX, startY);
        } else if(code == 12) {
            return new Rectangle(startX, startY);
        } else if(code == 13) {
            return new Triangle(startX, startY);
        } else if(code == 14) {
            return new Pencil(startX, startY);
        } else if(code == 15) {
            return new Trapezoid(startX, startY);
        } else if(code == 16) {
            return new Pentagon(startX, startY);
        } else if(code == 17) {
```

```
                return new Hexagon(startX, startY);
            }
            return null;
        }

        public void changeStrokeColor(Color color) {
            for(Shape shape : SELECTED_SHAPES)
                shape.change(color);
            writeLog("changeStrokeColor");
            repaint();
        }

        public void delete() {
            tmp = null;
            for(Shape s : SELECTED_SHAPES) {
                if (s.getClass().getName().equals("entities.Group")) SHAPES.removeAll(((Group)
s).getShapes());
            }
            for(int i=GROUPS.size()-1; i>=0; i--) {
                GROUPS.get(i).remove(SELECTED_SHAPES);
                if(GROUPS.get(i).getSize() <= 1) GROUPS.remove(i);
            }
            SHAPES.removeAll(SELECTED_SHAPES);
            SELECTED_SHAPES.clear();
            writeLog("delete");
            repaint();
        }

        public void copy() {
            CLIP_BOARD.clear();
```

```
        if(SELECTED_SHAPES.get(0).getClass().getName().equals("entities.Group"))
            CLIP_BOARD.addAll(((Group) SELECTED_SHAPES.get(0)).getShapes());
        else
            CLIP_BOARD.addAll(SELECTED_SHAPES);
    }

    public void paste() {
        SELECTED_SHAPES.clear();
        for(Shape shape: CLIP_BOARD) {
            String name = shape.getClass().getName();
            if (name.equals("entities.Rectangle")) {
                tmp = new Rectangle(shape);
            } else if (name.equals("entities.Line")) {
                tmp = new Line(shape);
            } else if (name.equals("entities.Triangle")) {
                tmp = new Triangle(shape);
            } else if (name.equals("entities.Trapezoid")) {
                tmp = new Trapezoid(shape);
            } else if (name.equals("entities.Pentagon")) {
                tmp = new Pentagon(shape);
            } else if (name.equals("entities.Hexagon")) {
                tmp = new Hexagon(shape);
            } else if (name.equals("entities.Ellipse")) {
                tmp = new Ellipse(shape);
            } else if (name.equals("entities.Pencil")) {
                tmp = new Pencil(shape);
            }
            SHAPES.add(tmp);
            SELECTED_SHAPES.add(tmp);
        }
```

```
        copy();
        writeLog("paste");
        repaint();
    }

    public void grouping() {
        if(SELECTED_SHAPES.size() <= 1) return;
        Group group = new Group(SELECTED_SHAPES);
        SELECTED_SHAPES.clear();
        GROUPS.add(group);
        SELECTED_SHAPES.add(group);
        writeLog("grouping");
        repaint();
    }

    public void undo() {
        SELECTED_SHAPES.clear();
        tmp = null;
        canvasLogger.undo(SHAPES, GROUPS);
        repaint();
    }

    public void redo() {
        SELECTED_SHAPES.clear();
        tmp = null;
        canvasLogger.redo(SHAPES, GROUPS);
        repaint();
    }
```

```
    public void activeIfContain() {
        for(int i = SHAPES.size()-1; i >= 0; i--)
            if(SHAPES.get(i).in(SELECTION_AREA))
SELECTED_SHAPES.add(SHAPES.get(i));
    }

    public void activeIfContain(int x, int y) {
        for(int i = SHAPES.size()-1; i >= 0; i--) {
            if (SHAPES.get(i).contains(x, y)) {
                for(int j = GROUPS.size()-1; j>=0; j--) {
                    if (GROUPS.get(j).containsShape(SHAPES.get(i))) {
                        SELECTED_SHAPES.add(GROUPS.get(j));
                        return;
                    }
                }
                SELECTED_SHAPES.add(SHAPES.get(i));
                return;
            }
        }
    }

    public void setZoomFactor(double factor) {
        zoomFactor = factor;
        repaint();
    }

    private void writeLog(String instruction) {
        for(Shape shape: SELECTED_SHAPES) {
            if(SHAPES.contains(shape)){
                canvasLogger.append(SHAPES.indexOf(shape));
```

```
                canvasLogger.append(shape.getClass().getName());
            }
        }
        canvasLogger.append(instruction);
        canvasLogger.logging(SHAPES, GROUPS);
    }

    public void loadImage(BufferedImage image) {
        loadedImage = image;
        repaint();
    }

    @Override
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D) g;
        g2d.scale(zoomFactor, zoomFactor);
        super.paint(g);
        if(tmp != null) tmp.draw(g2d);
        for(Shape shape: SHAPES) shape.draw(g2d);
        for(Shape shape: SELECTED_SHAPES) shape.select(g2d);
        if(this.selection != null) selection.draw(g2d);
    }

    @Override
    public void mouseClicked(MouseEvent e) {
        SELECTED_SHAPES.clear();
        activeIfContain(e.getX(), e.getY());
        repaint();
    }
```

```
// 마우스 누를때
@Override
public void mousePressed(MouseEvent e) {
    SELECTED_SHAPES.clear();
    startX = (int) (e.getX() / zoomFactor);
    startY = (int) (e.getY() / zoomFactor);
    endX = (int) (e.getX() / zoomFactor);
    endY = (int) (e.getY() / zoomFactor);
    tmp = getShape();
    repaint();
}

// 마우스 땔때
@Override
public void mouseReleased(MouseEvent e) {
    endX = (int) (e.getX() / zoomFactor);
    endY = (int) (e.getY() / zoomFactor);

    if(Math.abs(endX - startX) < 10 && Math.abs(endY - startY) < 10) {
        endX = startX + 10;
        endY = startY + 10;
    }
    if(tmp != null) {
        tmp.set(endX, endY);
        tmp.update();
        tmp.connect();
        SHAPES.add(tmp);
        SELECTED_SHAPES.add(tmp);
        writeLog("create");
    } else {
```

```
            SELECTION_AREA.setRect(
                    Math.min(startX, endX),
                    Math.min(startY, endY),
                    Math.abs(endX - startX),
                    Math.abs(endY - startY)
            );
            activeIfContain();
        }
        this.selection = null;
        repaint();
    }

    @Override
    public void mouseDragged(MouseEvent e) {
        endX = (int) (e.getX() / zoomFactor);
        endY = (int) (e.getY() / zoomFactor);

        if(tmp != null) {
            tmp.set(endX, endY);
            tmp.update();
            tmp.connect();
        } else {
            this.selection = new Selection(startX, startY);
            this.selection.set(endX, endY);
            this.selection.update();
            this.selection.connect();
            SELECTION_AREA.setRect(
                    Math.min(startX, endX),
                    Math.min(startY, endY),
                    Math.abs(endX - startX),
```

```
                        Math.abs(endY - startY)
                );
            }
            repaint();
        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2d = (Graphics2D) g;
            if (loadedImage != null) {
                g2d.drawImage(loadedImage, 0, 0, this);
            }
        }

        @Override
        public void mouseEntered(MouseEvent e) {}
        @Override
        public void mouseExited(MouseEvent e) {}
        @Override
        public void mouseMoved(MouseEvent e) {}
}

// view.panels.CanvasBoard
package view.panels;

import view.canvas.*;

import javax.swing.*;
import java.awt.*;
```

```
public class CanvasBoard extends JPanel   {
      private MyCanvas canvas;

      public CanvasBoard(MyCanvas canvas) {
            this.canvas = canvas;
            setBackground(Color.WHITE);
            add(canvas);
      }

      public MyCanvas getCanvas() {
            return canvas;
      }
}

// view.panels.MenuBar
package view.panels;

import view.canvas.*;
import view.panels.menus.*;

import javax.swing.*;
import java.awt.*;

public class MenuBar extends JMenuBar {
      public MenuBar(MyCanvas canvas) {
            this.setBackground(Color.WHITE);
            add(new FileMenu(canvas));
            add(new EditMenu());
```

```
    }
}

// view.panels.StatusBar
package view.panels;

import view.canvas.*;
import view.panels.tools.*;

import javax.swing.*;
import java.awt.*;



public class StatusBar extends JPanel{
    MyCanvas canvas;
    CanvasLogger canvasLogger;


    public StatusBar(MyCanvas canvas, CanvasLogger canvasLogger, ZoomSlider zoomSlider) {
        setBackground(new Color(211, 211, 211));
        setLayout(new BorderLayout());
        this.canvasLogger = canvasLogger;
        this.canvas = canvas;
        setPreferredSize(new Dimension(710, 40));
        add(canvasLogger, BorderLayout.WEST);
        add(zoomSlider, BorderLayout.EAST);
    }
}

// view.panels.ToolBar
```

```
package view.panels;


import view.panels.tools.*;


import javax.swing.*;
import java.awt.*;


public class ToolBar extends JPanel {
    public ToolBar(UtilTool utilTool, DrawTool drawTool, AttributeTool attributeTool, CanvasTool canvasTool) {
        add(new JLabel(new ImageIcon("images/partition.png")));
        add(utilTool);
        add(new JLabel(new ImageIcon("images/partition.png")));
        add(drawTool);
        add(new JLabel(new ImageIcon("images/partition.png")));
        add(attributeTool);
        add(new JLabel(new ImageIcon("images/partition.png")));
        add(canvasTool);
        add(new JLabel(new ImageIcon("images/partition.png")));
        setBackground(Color.WHITE);
        setPreferredSize(new Dimension(710, 85));
    }
}

// view.panels.menus.EditMenu
package view.panels.menus;


import javax.swing.*;


public class EditMenu extends JMenu {
```

```
    public EditMenu() {
        super("Edit");
    }
}

// view.panels.menus.FileMenu
package view.panels.menus;

import view.canvas.*;
import view.panels.menus.menuItems.*;

import javax.swing.*;


public class FileMenu extends JMenu {
    public FileMenu(MyCanvas canvas) {
        super("File");
        add(new NewFile(canvas));
        add(new Open(canvas));
        add(new Save(canvas));
        add(new SaveAs(canvas));
    }
}

// view.panels.menus.menuItems.NewFile
package view.panels.menus.menuItems;

import view.canvas.*;
```

```
import javax.swing.*;
import java.awt.event.*;

public class NewFile extends JMenuItem{
    public NewFile(MyCanvas canvas) {
        super("New File", new ImageIcon("images/file.png"));
        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                SaveAs.name = null;
                canvas.init();
                canvas.loadImage(null);
                canvas.repaint();
            }
        });
    }
}

// view.panels.menus.menuItems.Open
package view.panels.menus.menuItems;

import view.canvas.*;

import javax.imageio.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;

public class Open extends JMenuItem {
```

```
    private final MyCanvas canvas;

    public Open(MyCanvas canvas) {
        super("Open", new ImageIcon("images/file.png"));
        this.canvas = canvas;

        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFileChooser fileChooser = new JFileChooser();
                int result = fileChooser.showOpenDialog(null);

                if (result == JFileChooser.APPROVE_OPTION) {
                    File selectedFile = fileChooser.getSelectedFile();
                    loadImage(selectedFile);
                }
            }
        });
    }

    private void loadImage(File selectedFile) {
        try {
            BufferedImage image = ImageIO.read(selectedFile);
            canvas.loadImage(image);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
// view.panels.menus.menuItems.Save
import javax.swing.*;
import javax.swing.filechooser.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;


public class Save extends JMenuItem {
    public Save(MyCanvas canvas) {
        super("Save", new ImageIcon("images/save.png"));
        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(SaveAs.name != null) {
                    try {
                        ImageIO.write(SaveAs.getBufferedImageFromPanel(canvas),   "png",
new File(SaveAs.name));
                        System.out.println("saved Correctly");
                    } catch (IOException e1) {
                        System.out.println("Failed to save image");
                    }
                } else {
                    saveAsImage(canvas);
                }
            }
        });
    }


    private void saveAsImage(MyCanvas canvas) {
        BufferedImage bufferedImage = SaveAs.getBufferedImageFromPanel(canvas);
        JFileChooser jFileChooser = new JFileChooser();
```

```
            jFileChooser.setFileFilter(new FileNameExtensionFilter("*.png", "png"));

            int rVal = jFileChooser.showSaveDialog(null);
            if (rVal == JFileChooser.APPROVE_OPTION) {
                File file = jFileChooser.getSelectedFile();
                SaveAs.name = file.getAbsolutePath();
                SaveAs.name = (SaveAs.name.endsWith(".png")) ? SaveAs.name : SaveAs.name
+ ".png";
                try {
                    ImageIO.write(bufferedImage, "png", new File(SaveAs.name));
                    System.out.println("Saved correctly: " + file.getAbsolutePath());
                } catch (IOException e1) {
                    System.out.println("Failed to save image");
                }
            }
            if (rVal == JFileChooser.CANCEL_OPTION)
                System.out.println("No file chosen");
        }
}

// view.panels.menus.menuItems.SaveAs
package view.panels.menus.menuItems;

import view.canvas.*;

import javax.imageio.*;
package view.panels.menus.menuItems;

import view.canvas.*;
```

```
import javax.imageio.*;
import javax.swing.*;
import javax.swing.filechooser.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;

public class SaveAs extends JMenuItem {
    public static String name;
    BufferedImage bufferedImage;

    public SaveAs(MyCanvas canvas) {
        super("Save as...", new ImageIcon("images/save.png"));

        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                bufferedImage = getBufferedImageFromPanel(canvas);
                JFileChooser jFileChooser = new JFileChooser();
                jFileChooser.setFileFilter(new FileNameExtensionFilter("*.png", "png"));
                int rVal = jFileChooser.showSaveDialog(null);
                System.out.println(rVal);
                if (rVal == JFileChooser.APPROVE_OPTION) {
                    File file = jFileChooser.getSelectedFile();
                    name = file.getAbsolutePath();
                    name = (name.endsWith(".png")) ? name : name+".png";
                    try {
                        ImageIO.write(bufferedImage, "png", new File(name));
                        System.out.println("saved Correctly " + file.getAbsolutePath());
```

```
                    } catch (IOException e1) {
                        System.out.println("Failed to save image");
                    }
                }
                if (rVal == JFileChooser.CANCEL_OPTION)
                    System.out.println("No file chosen");
            }
        });
    }

    public static BufferedImage getBufferedImageFromPanel(MyCanvas canvas) {
        BufferedImage image = new BufferedImage(canvas.getWidth(), canvas.getHeight(),
BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();
        g.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS,
RenderingHints.VALUE_FRACTIONALMETRICS_ON);
        g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
RenderingHints.VALUE_TEXT_ANTIALIAS_ON);
        canvas.paint(g);
        g.dispose();
        return image;
    }

    public String getName() {
        return name;
    }
}

// view.panels.tools.AttributeTool
package view.panels.tools;
```

```
import enums.*;
import view.canvas.*;
import view.panels.tools.buttons.*;

import javax.swing.*;
import java.awt.*;



public class AttributeTool extends JPanel{
    public AttributeTool(MyCanvas canvas) {
        for(AttributeAuthority attributeAuthority: AttributeAuthority.values())
            add(new AttributeButton(attributeAuthority, canvas));
        setBackground(Color.WHITE);
    }
}

// view.panels.tools.CanvasTool
package view.panels.tools;

import enums.*;
import view.canvas.*;
import view.panels.tools.buttons.*;

import javax.swing.*;
import java.awt.*;



public class CanvasTool extends JPanel {
    public CanvasTool(MyCanvas canvas) {
```

```
        for(CanvasAuthority canvasAuthority: CanvasAuthority.values())
            add(new CanvasButton(canvasAuthority, canvas));
        setBackground(Color.WHITE);
    }
}

// view.panels.tools.UtilTool
package view.panels.tools;

import enums.*;
import view.canvas.*;
import view.panels.tools.buttons.*;

import javax.swing.*;
import java.awt.*;



public class UtilTool extends JPanel{
    public UtilTool(MyCanvas canvas) {
        for (UtilAuthority utilAuthority : UtilAuthority.values())
            add(new UtilButton(utilAuthority, canvas));
        setBackground(Color.WHITE);
    }
}

// view.panels.tools.ZoomSlider
package view.panels.tools;

import view.canvas.*;
```

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.util.*;

public class ZoomSlider extends JSlider {
    public ZoomSlider(MyCanvas canvas) {
        super(JSlider.HORIZONTAL, 100, 200, 100);
        setMajorTickSpacing(20);
        setMinorTickSpacing(5);
        setPreferredSize(new Dimension(200, 30));
        setPaintTicks(true);
        setPaintLabels(true);

        addChangeListener(new ChangeListener() {
            public void stateChanged(ChangeEvent e) {
                if (canvas != null)
                    canvas.setZoomFactor(getValue() / 100.0);
                repaint();
            }
        });

        Hashtable<Integer, JLabel> labelTable = new Hashtable<>();
        labelTable.put(100, new JLabel("0"));
        labelTable.put(150, new JLabel("0.5"));
        labelTable.put(200, new JLabel("1"));
        setLabelTable(labelTable);
    }
}
```

```
// view.panels.tools.buttons
package view.panels.tools.buttons;
 CanvasBoard

import enums.*;
import view.canvas.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;



public class AttributeButton extends JButton {
    public AttributeButton(AttributeAuthority attributeAuthority, MyCanvas canvas) {
        setLayout(new BorderLayout());
        JButton button = new JButton(attributeAuthority.getIcon());
        JLabel label = new JLabel(attributeAuthority.getName());
        setBackground(Color.WHITE);
        button.setFocusPainted(false);
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                Color selectedColor = JColorChooser.showDialog(null, "Choose a Color", Color.YELLOW);
                if (selectedColor != null)
                    canvas.changeStrokeColor(selectedColor);
            }
        });
        add(button, BorderLayout.CENTER);
```

```
            add(label, BorderLayout.SOUTH);
        }
}

// view.panels.tools.CanvasButton
package view.panels.tools.buttons;

import enums.*;
import view.canvas.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;



public class CanvasButton extends JButton{
    public CanvasButton(CanvasAuthority canvasAuthority, MyCanvas canvas) {
        super(canvasAuthority.getIcon());
        setBackground(Color.WHITE);
        setFocusPainted(false);

        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if(canvasAuthority.getCode() == 30)
                    canvas.undo();
                if(canvasAuthority.getCode() == 31)
                    canvas.redo();
            }
        });
```

```
    }
}

// view.panels.tools.DrawButton
package view.panels.tools.buttons;

import enums.*;
import view.canvas.*;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;



public class DrawButton extends JButton{
    public static ArrayList<DrawButton> db = new ArrayList<>();
    private static final Color ACTIVE_COLOR = Color.decode("#D3D3D3");

    public DrawButton(DrawAuthority drawAuthority, MyCanvas canvas) {
        super(drawAuthority.getIcon());
        db.add(this);
        setBackground(Color.WHITE);
        setFocusPainted(false);
        addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                canvas.setCode(drawAuthority.getCode());
                if (DrawButton.this.getBackground() == ACTIVE_COLOR) {
                    DrawButton.this.setBackground(Color.WHITE);
```

```
                } else {
                    for(DrawButton d : db) d.setBackground(Color.WHITE);
                    DrawButton.this.setBackground(ACTIVE_COLOR);
                }
                SwingUtilities.updateComponentTreeUI(DrawButton.this);
            }
        });
    }
}

//tools.buttons.UtilButton
package view.panels;

import view.canvas.*;
import view.panels.tools.*;

import javax.swing.*;
import java.awt.*;

public class StatusBar extends JPanel {
    MyCanvas canvas;
    CanvasLogger canvasLogger;

    public StatusBar(MyCanvas canvas, CanvasLogger canvasLogger, ZoomSlider zoomSlider) {
        Color color = new Color(215, 215, 215);
        setBackground(color);
        setLayout(new BorderLayout());
```

```
        this.canvasLogger = canvasLogger;
        this.canvas = canvas;
        setPreferredSize(new Dimension(710, 40));

        JPanel zoomSliderPanel = new JPanel(new BorderLayout());
        zoomSliderPanel.add(zoomSlider, BorderLayout.EAST);
        zoomSliderPanel.setBackground(color);
        zoomSliderPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 10));

        add(canvasLogger, BorderLayout.WEST);
        add(zoomSliderPanel, BorderLayout.EAST);
    }
}
```

## 7. References


- "Java AWT Geom Package." Java Platform, Standard Edition 8, 중부 대학교, 연도. http://cris.joongbu.ac.kr/course/java/api/java/awt/geom/package-summary.html
- https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=javaking75&logNo=140189860985 (마우스 이벤트)
- https://edu-coding.tistory.com/110 (이벤트 처리)
- https://young0105.tistory.com/78 (엔티티 참고)
- https://blog.aspose.com/ko/imaging/draw-shapes-in-java/ (엔티티 참고)
- https://m.blog.naver.com/tkddlf4209/220599753497 (키보드 리스너)
- https://yooniron.tistory.com/13 (리스너 처리)
- https://mainia.tistory.com/1514 (캔버스 참고)
- https://wickies.tistory.com/18 (AWT 참고)

- https://stackoverflow.com/questions/6575578/convert-a-graphics2d-to-an-image-or-bufferedimage(파일 저장)
- https://coding-factory.tistory.com/261 (GUI 참고)
- https://marine1188.tistory.com/entry/JAVA%EC%9E%90%EB%B0%94Interface-%EB%A7%8C%EB%93%A4%EA%B8%B0-implements%EA%B5%AC%ED%98%84-%ED%95%98%EA%B8%B0 (인터페이스 참고)
- lhttps://gigglehd.com/gg/soft/10731147 (그림판 참고)
- http://cris.joongbu.ac.kr/course/2018-1/jcp/api/java/awt/class-use/Color.html (awt 참고)
- https://stickode.tistory.com/804 (줌인아웃 참고)
- https://blog.naver.com/highkrs/220549262072 (버튼 참고)
- https://hbase.tistory.com/153 (입출력 참고)
- https://lasbe.tistory.com/65 (파일 참고)
- https://mangkyu.tistory.com/73 (ENUM 참고)