

DS Term Project

Drawing Editor

Team 1

22102017 Hong Minji
23102009 Park Shinyung
23102025 Lee Haneol
23102031 Hwang Yuyoung

Table of Contents

1. Introduction
2. System Requirements
3. System Design
4. Team Roles & Responsibilities
5. Conclusion & Future Works
6. Appendices
7. References

1.Introduction

1)Project Overview

This project utilizes JavaFX to implement picture boards. This program allows users to add and edit various shapes. Specifically, we designed an intuitive and easy-to-use user interface (UI) using Scene Builder. This allows users to easily draw pictures without complicated manipulations.

2)Objectives

The main goal of this project is to allow the user to draw various shapes. To do this, we implement basic shapes such as squares, circles, and lines, and also provide detailed functions such as grouping, color change, copying/pasting, etc.

An additional goal of ours is to provide a user-friendly interface so that users can easily use the tools. We will provide an intuitive UI through icons, allowing users to quickly recognize and access the features they need. Finally, I wanted to write a detailed user manual so that users can easily understand and use the program.

3)Importance

The Paint Project is an important opportunity for us to effectively apply Java's data structure based on what we have learned.

1. Apply the basic principles of object-oriented programming, and visually represent the structure of the system through UML diagrams.

2. Effective data management can be achieved by defining each figure, color, tool, etc. as a class and utilizing various data structures such as arrangement, list, and hash map for figure list management.

3. This project is an overall process of software development

It provides an opportunity to experience planning, design, implementation, and testing, and helps develop problem-solving skills and learn the importance of teamwork in real-world development environments.

2. System Requirements

-mandatory shape and features

1) Shape:

Implemented Feature	Description	Used Methods
Show Shape Menu	When the user clicks the shape button, it switches to Shape mode and displays the shape selection menu at the mouse position.	showShapeMenu(MouseEvent event)
Start Drawing	Sets the clicked position as the starting point and corrects it to ensure it does not go outside the canvas boundaries.	startDrawing(MouseEvent event)
Draw Shape	When the user drags the mouse, it corrects the endpoint and draws the selected shape. Previous shapes are cleared, and existing shapes are redrawn.	drawShape(MouseEvent event)
Draw Line	It connects the start and end points to draw a line.	gc.strokeLine(startX, startY, endX, endY)

Draw Circle	It draws a circle based on the start and end points. * Always start from top left -> finish from bottom right	gc.strokeOval(startXC orrected,startYCorrect ed, size, size)
Draw Rectangle	It draws a rectangle based on the start and end points. * Always start from top left -> finish from bottom right	gc.strokeRect(startXC orrected,startYCorrect ed, rectWidth, rectHeight)
Finalize Shape	Sets the clicked position as the endpoint, draws the selected shape on the canvas -> adds the shape information to the shapes list.	finalizeShape(MouseE vent event)
Store Shape	It structures shape information, including type, coordinates, and color.	ShapeRecord()
Redraw Canvas	Clears the canvas and redraws all shapes stored in the shapes list. -> 새로운 도형을 그릴때 기존 도형이 지워지는것을 방지	redrawCanvas()

2)select

Implemented Feature	Description	Used Methods
Activate Select Mode	When the user clicks the Select Button, it switches to Select mode and resets event handlers to prepare for selecting shapes.	handleSelectButtonCli ck()
Single Selection	Allows the user to select a single shape by clicking on it. Only one shape is highlighted.	startSelection(MouseE vent event)
Multiple Selection	The user can drag to select multiple shapes within the rectangular selection area. All shapes in the range are highlighted.	dragSelection(Mouse Event event)
Highlight Selected Shapes	Highlights the selected shape(s) using a green dotted border to provide visual feedback to the user.	highlightShapes()

Exit Select Mode	Clicking anywhere on the blank canvas area deselects all currently selected shapes.	exitSelectionMode(MouseEvent event)
Save State	Before performing any selection or drag operation, the current state of the shapes is saved to enable Undo/Redo functionality.	updateSelectedShapes()

Team Required Features

1)move :

Implemented Feature	Description	Used Methods
Activate Move Mode	When the user clicks the move button, the Move mode is activated, and existing event handlers are reset.	handleMoveButtonClick(ActionEvent event)
Select Shape	When the user clicks on the canvas, it checks if there is a shape at the clicked position and selects it.	startMove(MouseEvent event)
Drag Movement	When the user drags the mouse, the selected shape moves according to the drag distance calculated from the starting position.	performMove(MouseEvent event)
Deselect Shape	When the user releases the mouse button, if it is considered a click rather than a drag, the selected shape is deselected.	endMove(MouseEvent event)
Highlight Shape	In move mode, the selected shape is highlighted in a green dotted line to provide visual feedback to the user.	highlightShapes()

2)copy/paste:

Implemented Feature	Description	Used Methods
Activate Copy Mode	When the user clicks the copy button, the Copy mode is activated, and the user can click on a shape to copy it to the clipboard.	handleCopyButton Click(ActionEvent event)
Copy Shape	When the user clicks on a shape, it checks if the click is within the shape's area and copies it to the clipboard.	java fx에서 지원하는 setOnMousePressed<-의 내장함수
Activate Paste Mode	When the user clicks the paste button, the Paste mode is activated, allowing the user to click on the canvas to place the copied shape.	handlePasteButton Click(ActionEvent event)
Paste Shape	When the user clicks on the canvas, it checks if the clipboard is not empty and pastes the previously copied shape at the clicked position.	java fx에서 지원하는 setOnMousePressed<-의 내장함수
Create New Shape	A new shape is created based on the copied shape's attributes, adjusted for the new position, and added to the shape list. 시작위치: 사용자가 클릭한 위치(offsetX, offsetY) 끝 위치: 시작위치로 부터 너비와 높이를 계산하여 위치 시킴	ShapeRecord()

3)change line color :

Implemented Feature	Description	Used Methods
Color Selection	When the user selects a color from the color picker(palette), the selected color is stored.	handleColorChange()
Show Color Context Menu	When the user clicks the color button, a color selection menu is displayed at the mouse position.	showColorContextMenu(MouseEvent event)
Change Color of Selected Shapes	If there are selected shapes, the color of all selected shapes is changed. *The line color changes in real time every time you press the color	Inside the loop: for (ShapeRecord shape : selectedShapes)

- Additional features:

1)File Save/Load:

Implemented Feature	Description	Used Methods
save file	If a current file is specified, it saves to that file; otherwise, it calls the "Save As" function.	handleSave()
Save As	Opens a file chooser dialog and saves shape information to the selected file.	handleSaveAs()
Save to File Method	Saves shape information in JSON format to the specified file.	saveToFile(File file)
Load File	Opens a file chooser dialog and loads shape information from the selected file.	handleLoad()
Load from File Method	Reads shape information stored in JSON format from the specified file into the shapes list.	loadFromFile(File file)

2)Undo/Redo:

Implemented Feature	Description	Used Methods
Save State	Saves the current state of the shapes list to allow undo and redo operations.	saveState()
Undo Operation	Reverts the canvas to the previous state by popping the last saved state from the undoStack.	undo()
Redo Operation	Reapplies the undone action by retrieving the state from the redoStack.	redo()
State Management	Manages two stacks (undoStack and redoStack) to keep track of states for undo and redo operations.	undoStack,redoStack
Redraw Canvas	Clears and redraws the canvas based on the shapes stored in the current state.	redrawCanvas()

3)Grouping:

Implemented Feature	Description	Used Methods
Group ID Generation	A variable used to generate a unique group ID for grouping shapes.	nextGroupId = 1;
Handle Group Button Click	Displays a context menu when the user clicks the group button, providing options for grouping and ungrouping.	handleGroupButton Click(MouseEvent event)
Grouping	If there are selected shapes, assigns a new group ID and updates the group ID of the shapes.	groupSelectedShapes()
Ungrouping	If there are selected shapes, sets their group ID to -1 to ungroup them.	ungroupSelectedShapes()

4)Rubber Band:

The main parts that implement the rubber band effect are the startDrawing and drawShape methods. The startDrawing() method sets the starting point, while the drawShape() method is called when the user drags the mouse, updating the endpoint of the shape based on the current mouse position. This results in the rubber band effect.

예시)

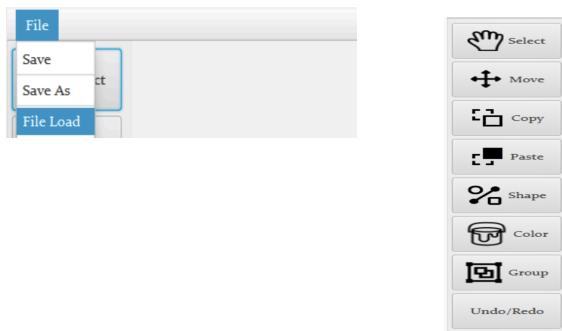
```
// rubber band 효과: 시작점과 현재 끝점을 연결하여 선을 그립니다.  
gc.strokeLine(startX, startY, endX, endY);
```

```
gc.strokeOval(startXCorrected, startYCorrected, size, size);
```

```
gc.strokeRect(startXCorrected, startYCorrected, rectWidth, rectHeight);
```











5)Menu bar:

We provide the following menu bar and toolbar functionalities



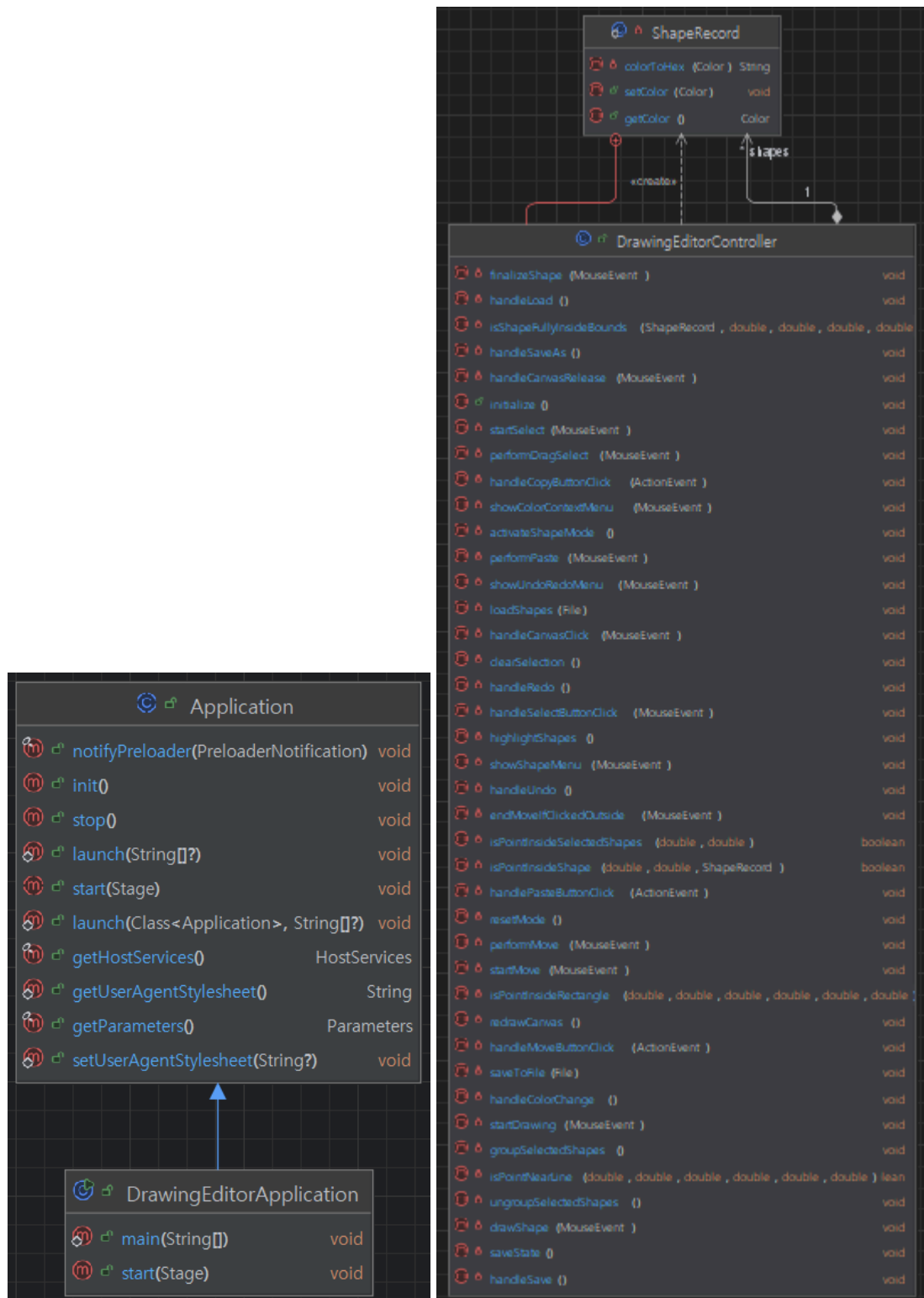
6)Icon:

We use 10 icons to create an appealing and user-friendly interface.

 move	 shape	 circle	 rectangle	 line
 copy	 paste	 paint	 group	 select

3.System Design

1)Class Diagram



<DrawingEditorApplication> class inherits <Application> class

<DrawingEditorController> class has one inner class <shapeRecord> class

In this project, the functionality of the Drawing Editor was implemented using a **method-centric approach** within a single controller class rather than a class-based modular structure. This decision was made to prioritize the rapid development and integration of key features within the limited project timeline.

The **DrawingEditorController** class serves as the central hub where all user inputs and events are handled, including shape creation, selection, movement, color changes, Undo/Redo, and grouping. While this approach allowed us to implement the required functionalities efficiently, it resulted in a monolithic structure with the following limitation/

Future Scope for Class Diagram Implementation

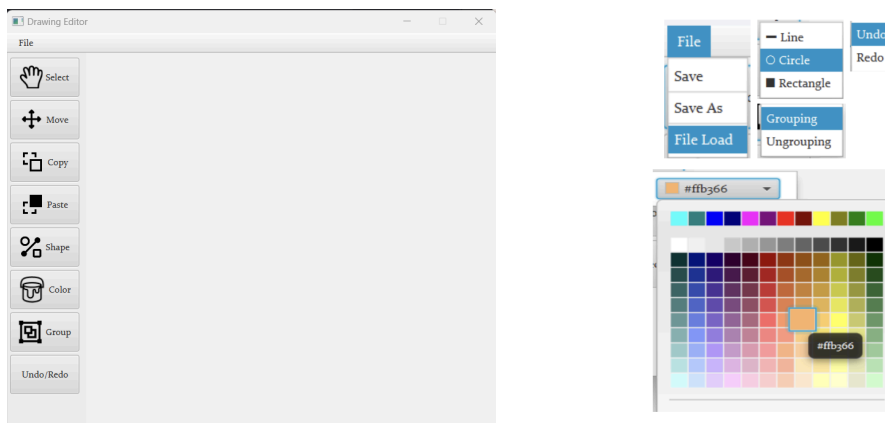
To address these limitations, the following modular structure is proposed for future improvements:

- ShapeManager
 - Responsible for creating, storing, and rendering shapes.
 - Manages the list of shapes on the canvas and provides methods to draw, move, or delete shapes.
- StateManager
 - Manages Undo/Redo functionality using stacks to save and restore the state of shapes.
 - Provides methods to save the current state and navigate between previous states.
- InputHandler
 - Handles user interactions such as mouse clicks, drags, and button events.
 - Separates input handling logic from shape and state management.
- ShapeRecord
 - A data class to represent individual shapes, storing properties like type, position, size, and color.

With this proposed structure, a **Class Diagram** can be created to represent the relationships and responsibilities of each component. This

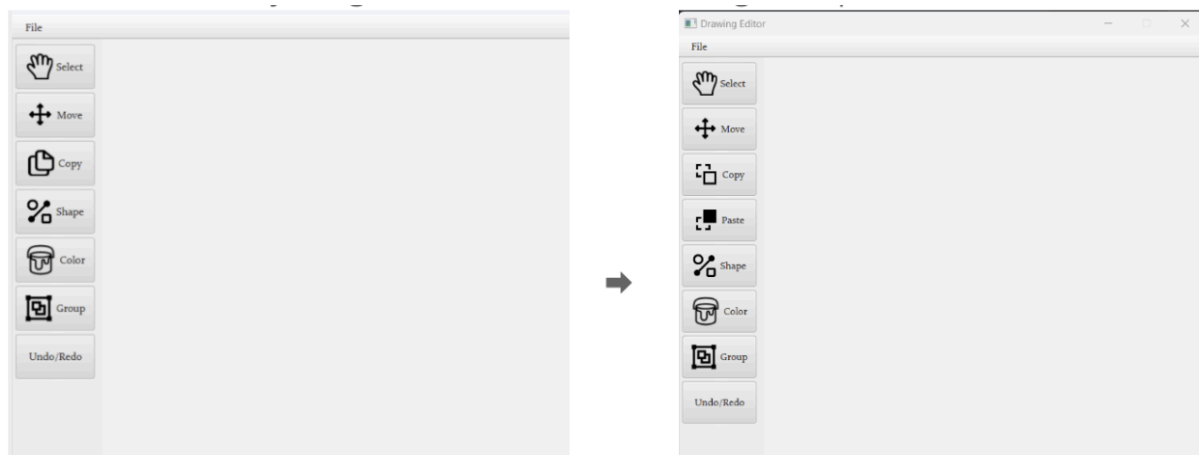
approach will enhance code maintainability, modularity, and scalability, making it easier to extend and debug the project in the future.

2)User Interface



File	Save	A menu for performing file-related tasks, such as saving or loading files.
	Save As	
	File Load	
Empty Canvas		The large central area serves as the workspace where users can draw and edit shapes. Users can add and edit various shapes in this space.
Select		A tool for selecting shapes or objects.
Move		Allows the user to move selected shapes or objects.
Copy		Copies the selected shapes or objects.
Paste		Pastes the copied shapes or objects.
Shape	Line	A tool for adding new shapes.
	Circle	
	Rectangle	

Color		Enables the selection or change of shape colors.
Group	Grouping	Groups selected shapes together for collective movement or manipulation.
	Ungrouping	
Undo/Redo	Undo	Allows the user to cancel previous actions or redo them.
	Redo	



We modified the interface based on our understanding of the required functionality assigned to T1. Initially, we interpreted "copy" as a standalone feature, assuming that we were only required to provide the copy function without the accompanying paste feature. Later, we realized that "copy" was intended to encompass both copy and paste functionalities. As a result, we updated the user interface to reflect this understanding.

4.Team Roles & Responsibilities

Minjl (22102017)	Undo/Redo, Grouping, Report
Shinhyung (23102009)	Color, File(Save, Load), Report
Haneol (23102025)	Copy/Paste, Move, Report
Yuyoung (23102031)	Shape(Draw), Select, Report

5.Conclusion & Future Works

1)Conclusion

The Drawing Editor project successfully achieved its goal of providing essential drawing functionalities alongside interactive features for user-friendly shape manipulation. Throughout the development, the following key functionalities were implemented:

- **Shape Drawing**

The editor supports creating lines, circles, and rectangles with precise positioning and scaling using mouse interactions. The `redrawCanvas()` method ensures that all existing shapes persist when a new shape is drawn.

- **Selection & Movement**

Users can select shapes using single or multiple selection modes and move them interactively. Visual feedback, such as highlighting selected shapes, enhances usability.

- **Copy/Paste**

Shapes can be copied and pasted at user-specified positions. This improves efficiency when working with repetitive shapes.

- **Change Line Color**

Users can dynamically update the line color of selected shapes or set the color for future shapes.

- **Undo/Redo**

By managing two stacks (`undoStack` and `redoStack`), the system enables users to revert or reapply changes efficiently. This feature improves usability by allowing error correction and experimentation without manual resets.

- **Grouping & Ungrouping**

Multiple shapes can be grouped and manipulated as a single entity. The `groupSelectedShapes()` and `ungroupSelectedShapes()` methods ensure seamless group management.

- **File Save/Load**

The application supports saving shapes to a file in JSON format and reloading them for further editing. This functionality ensures project persistence and reusability.

The project underwent iterative refinement. Initially, the **Copy** function was misunderstood as a standalone feature, but later, it was updated to include **Paste**, aligning with the project requirements.

However, the current implementation focuses heavily on **method-centric development**, where most functionalities are implemented within a single controller class (`DrawingEditorController`). While this approach enabled us to implement the features efficiently, it resulted in longer and less maintainable code.

2)Future Works

While the current version of the Drawing Editor satisfies essential requirements, several improvements and extensions can be made to enhance the system further:

- **Class Refactoring**

Separate functionalities into distinct classes to enhance modularity and maintainability:

- **ShapeTool** for creating and managing shapes.
- **StateTool** for managing undo/redo states.
- **EditTool** for handling user interactions like selecting, moving, and copying shapes.

By applying a class-based structure, the code can become cleaner, more modular, and easier to extend.

- **Advanced Shape Manipulation**

Implement features such as resizing and rotating shapes for more advanced editing capabilities. Provide snapping-to-grid

- **Improved File Management**

Introduce support for additional file formats, such as SVG or PNG, to enhance file export capabilities. Add an autosave feature to prevent data loss during long editing sessions.

- **User Interface Enhancements**

Provide a dynamic toolbar that adjusts based on the selected tool or mode. Add keyboard shortcuts for frequently used actions (e.g., Undo: Ctrl+Z, Redo: Ctrl+Y).

- **Performance Optimization**

Optimize the rendering process for large numbers of shapes to improve responsiveness. Implement lazy loading for complex files to manage system resources effectively.

- **Collaborative Editing**

Enable real-time collaboration where multiple users can edit the canvas simultaneously. Integrate cloud storage for shared project saving and loading.

3)Final Thoughts

The Drawing Editor project serves as a robust foundation for a functional and interactive drawing application. It demonstrates the effective use of JavaFX for GUI development and highlights the importance of structured design through concepts like state management, modular functionality, and intuitive user interface design.

While the current implementation focuses on functionality, future refinements—such as adopting a **class-based architecture**—will improve code maintainability and scalability. With these improvements, the Drawing Editor has the potential to evolve into a comprehensive tool for various design and educational purposes.

6.Appendices

https://github.com/shinh09/DS_project1.git

7.References

Configuration Settings:

<https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html>

<https://gluonhq.com/products/scene-builder/>

Develope

<https://www.youtube.com/watch?v=PgzC9yJt3lw>