Project Report

COMP2021 Object-Oriented Programming (Fall 2025)

Group 17

Members and contribution percentages:

Li Qicheng 24103417D 25%

Ruiyang Guo 24101223D 25%

Meng Xiangyu 24113644D 25%

Yu Chengjun 24110094D 25%

1. Introduction

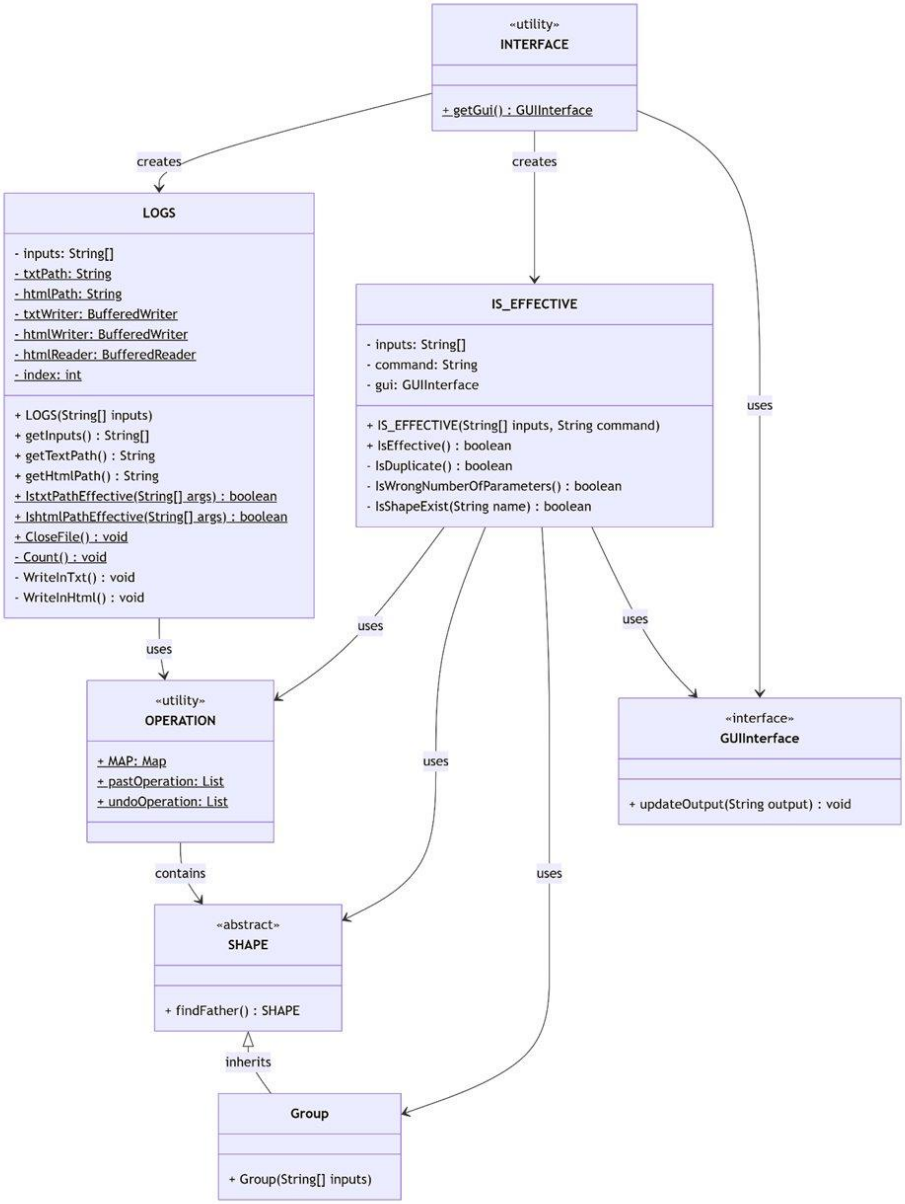
This report outlines the design and implementation of the Command Line Vector Graphics Software (Clevis), a project completed by Group 17. Clevis is a command-line-based vector graphics editor that supports creating, manipulating and managing geometric shapes such as rectangles, circles, lines and squares, as well as advanced operations like grouping, moving, deleting and querying spatial relationships.

The project emphasizes object-oriented design principles, modular architecture, and robust error handling, while implementing both compulsory requirements (REQ1–REQ14) and the BON1, 2 (i.e. GUI and undo/redo functionality).

2. The Command Line Vector Graphics Software (Clevis)

2.1 Design

Since each of us coded a different part of this project, we drew our UML separately. Moreover, if our UML were to combine, it wouldn't be able to display in a Word document.



UML by LI Qicheng:

IS_Effective is the class to examine whether user's input is effective.

It has boolean methods to detect different input error like duplicate-name shapes, wrong number of parameters, wrong operating object type and shapes not exist.

It is called by class INTERFACE, gets the data of shapes from SHAPE class, and can transmit data to GUIInterface.

LOGS is the class to log use's command into log.txt and log.html.

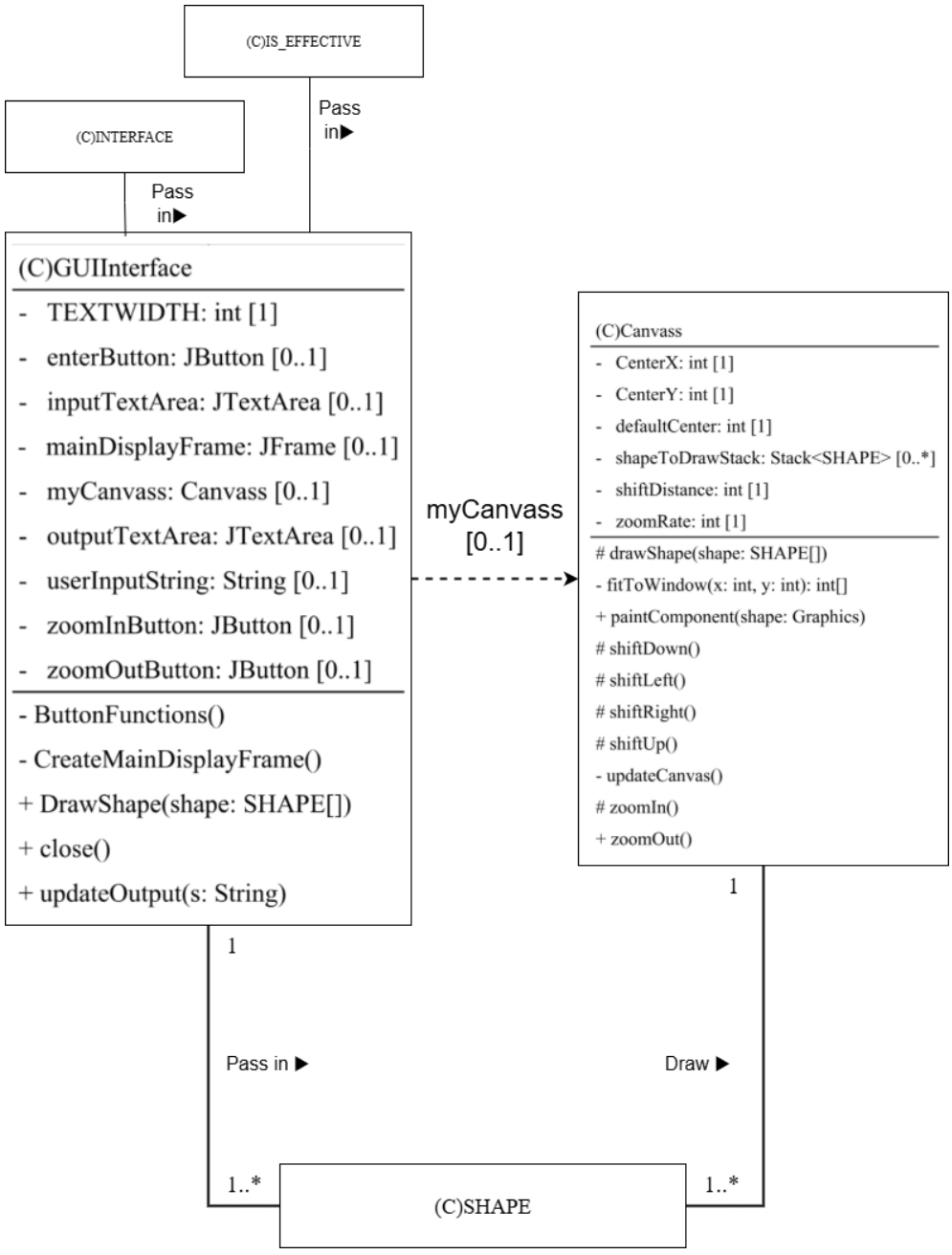
Two methods to detect potential error of the file path the user input, and will terminate the program

when detected.

Two methods for wring the command into the files, following the structure of .txt file and .html file.

One method to save and close the file user types in “quit” in INTERFACE.

After logging in, command will be passed to GUIInterface and OPERATION for further progress.



UML by Ruiyang:

My design of GUI is relatively simple. On GUIInterface class that setup and maintain the GUI. It originally had some buttons that zoom in and out, but then I removed them because our design idea changed. We only wanted GUI to display user's input result and without and interaction.

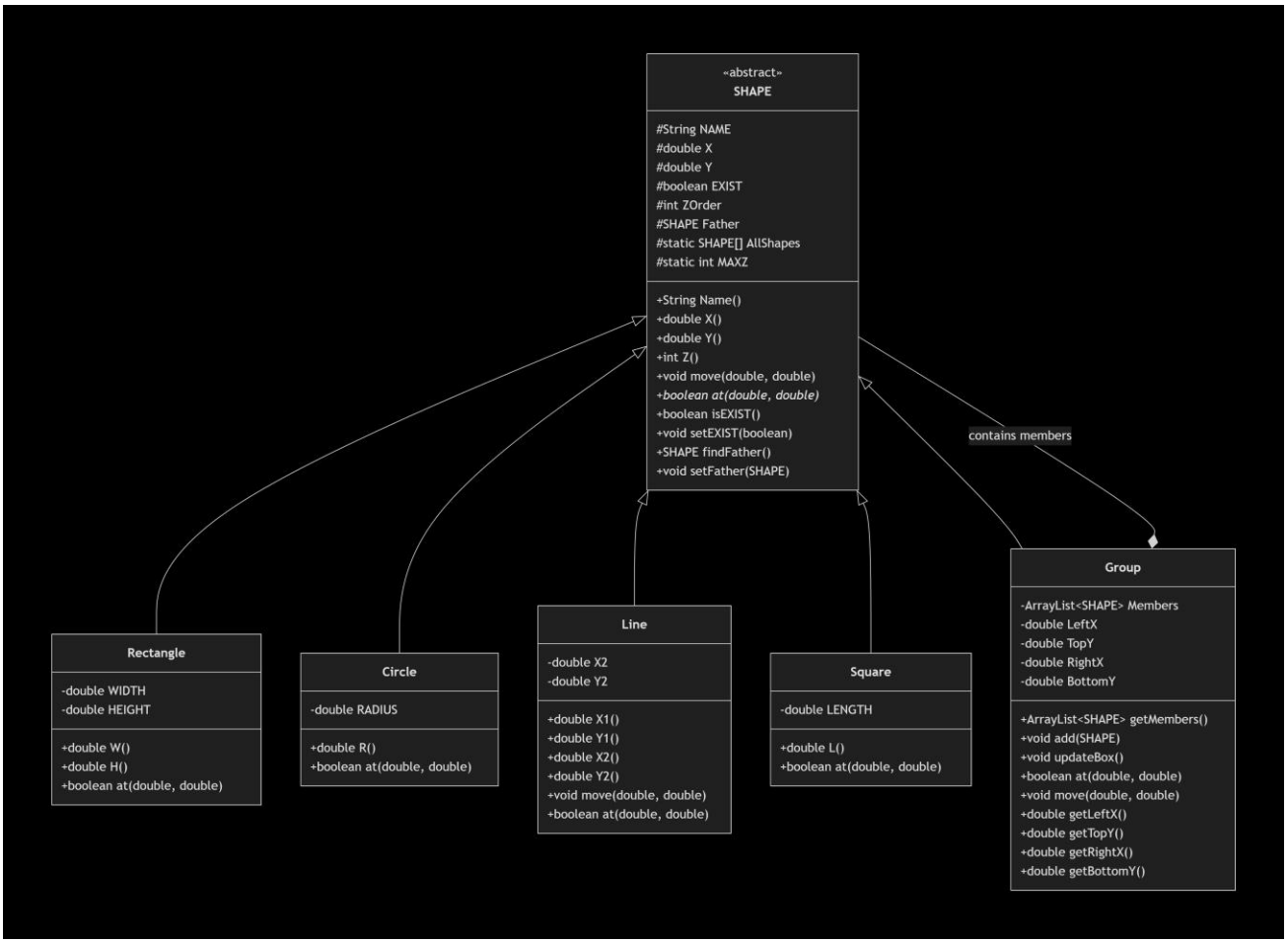
Canvass class is a class that extends JFrame to paint. It's protected zoom and shift functions are key binded with user's key input so that user doesn't have to directly interact with GUI (zoomOut should be protected but forgot to chang).

updataOutput is a function that can be call by other class to write output to GUI's console.

close closes the GUI thread when user enters quit in CLI.

drawShape draws the all user created shape on gui. SHAPE array is first passed into GUIInterface, then GUIInterface passes it into Canvass.

GUIInterface is only used by INTERFACE and IS_EFFECTIVE class. INTERFACE uses it for updating console (*updateOutput*) and close (*close*). IS_EFFECTIVE uses it for updating console (*updataOutput*) as well.



UML by MENG Xiangyu:

SHAPE is an abstract class. Rectangle, Circle, Line, Square and Group extend SHAPE.

SHAPE defines core properties including name, coordinates, existence flag, Z-order, father group reference, and a static array storing all shapes.

There are five concrete shape type, each has some unique attributes:

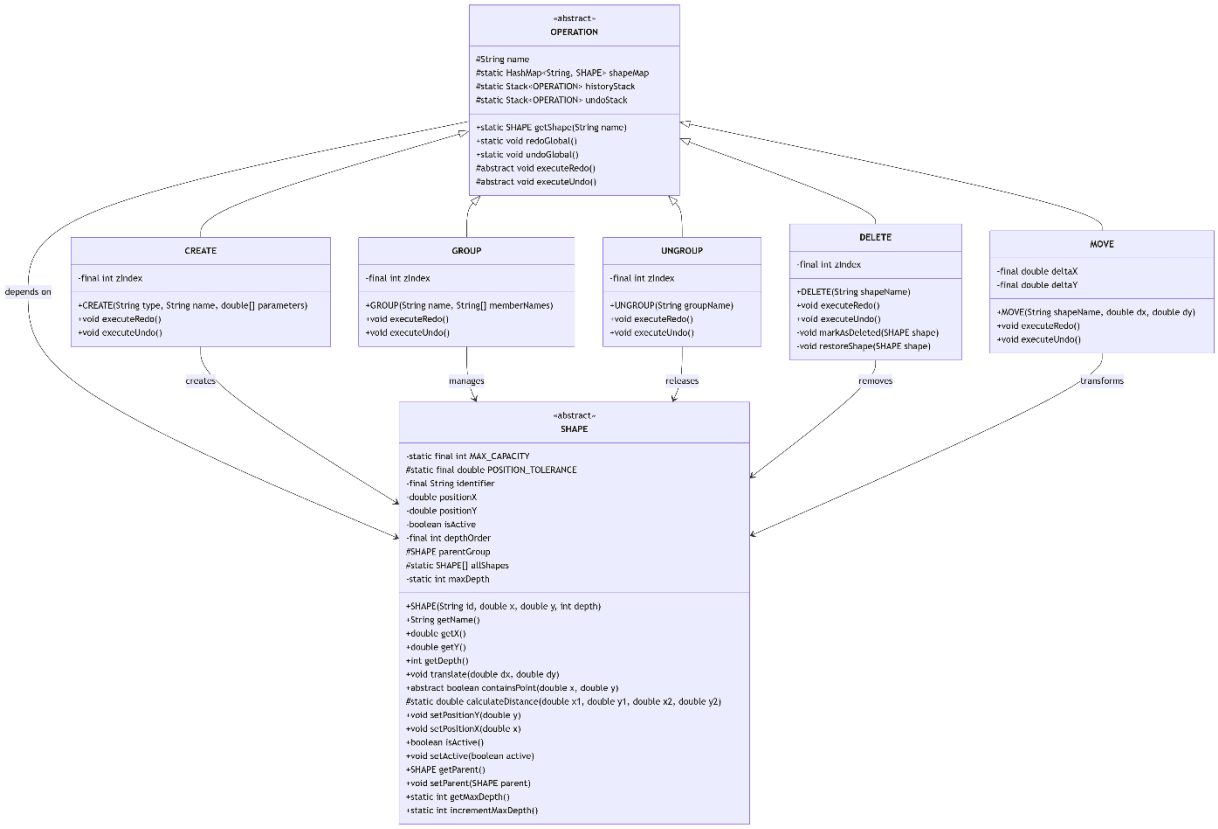
Rectangle with width and height dimensions

Circle with radius measurement

Line with dual endpoint coordinates

Square with uniform side length

Group with its members and bounding box.



UML by Yu Chengjun:

As illustrated in the class diagram, the abstract OPERATION class serves as the root of the command hierarchy, defining the template for all system actions. Concrete subclasses—CREATE, GROUP, UNGROUP, DELETE, and MOVE—extend this class to implement specific behaviors. This design choice allows each operation to maintain its own state.

To implement the Undo and Redo functionality (which is a bonus feature), I utilize two static Stack objects inside the abstract OPERATION class. The pastOperation stack (history) stores commands that have been successfully executed, while the undoOperation stack holds commands that have been reversed.

All the drawing and manipulation commands primarily interact with the abstract SHAPE objects. This design choice ensures a clear separation of responsibility: the OPERATION classes manage the flow and history of commands, while the SHAPE classes manage the actual geometric data and behavior.

2.2 Implementation of Requirements

[REQ1] 1) The requirement is implemented.

2) LOGS class is used to write users command into log files.

Constructor: LOGS(String[] inputs)

Public static method:

boolean IstxtPathEffective(String args[]) and boolean IshtmlPathEffective(String args[]): detect whether the file path is valid. If so, open the files and initialize them, otherwise return false and let INTERFACE termiate the program.

void CloseFile(): Save and close the log files when user type in ”quit”.

Private method:

void WriteInTxt() and void WriteInHtml(): write user’s command into log.txt and log.html, and only command pass the examination of IS_EFFECTIVE can be logged.

3) Empty file path: boolean IstxtPathEffective(String args[]) and boolean IshtmlPathEffective(String args[]) can get the string following “-txt” and “-html”, which is the path of the log file. if the path is empty, they will get nothing and return false. That will let INTERFACE terminate the program.

Invalid file path or no permission to open/write in the file: boolean IstxtPathEffective(String args[]), boolean IshtmlPathEffective(String args[]), void WriteInTxt() and void WriteInHtml() can catch IOException, and then these methods will print the exception, and methods in boolean type will return false to let INTERFACE terminate the program.

No permission to close the file: void CloseFile() can catch IOException, and print the exception to the user.

[REQ2] 1) The requirement is implemented.

2) Rectangle class extends SHAPE with additional width and height attributes

Constructor: Rectangle(String Name, double x, double y, double width, double height, int Zorder)

Coordinate system: (x,y) represents top-left corner

at() method implementation: Checks if point lies on any of the four boundary lines using Line class helper

Created via CREATE operation with type "rectangle"

3) Invalid params: $w \leq 0$, $h \leq 0$ → boolean IsEffective() method checks whether w or h is less than 0 and if it is less than 0 then prints: "Error! Width and height must be positive!" and return false.

Wrong number of parameters: the number of parameters is not equal to 5 (name, x, y, w, h) → boolean IsWrongNumberOfParameters() method whether the number of parameters is equal to 5, if not, print "Error! Wrong number of parameters!" and return false.

Shape with the same name: if (OPERATION.MAP.containsKey(inputs[1])]), then boolean IsDuplicate() prints: “Error! Duplicate Shape: ‘name’ ” and return true.

All the errors would lead to return in main method.

[REQ3] 1) The requirement is implemented.

2) Line class extends SHAPE with additional endpoints (X2, Y2)

Constructor: Line(String Name, double x, double y, double x2, double y2, int ZOrder)

at() method: Uses vector projection algorithm to determine if point lies on line segment within bias

Created via CREATE operation with type "line"

3) If the input provides fewer or more than the required number of arguments, the command should fail.

All coordinate parameters (x1, y1, x2, y2) must be valid double-precision floating-point numbers. Any non-numeric input for these values must lead to command rejection.

[REQ4] 1) The requirement is implemented.

2) Circle class extends SHAPE with radius attribute

Constructor: Circle(String Name, double x, double y, double radius, int Zorder)

Coordinate system: (x,y) represents center point

at() method: Checks if distance from point to center equals radius within ±0.05 bias

Created via CREATE operation with type "circle"

3) If the input provides an incorrect number of arguments (less or more than four), the command should fail.

The radius of a circle must be a positive value. If the user attempts to input zero or a negative number for the radius, the command must be deemed invalid and rejected.

[REQ5] 1) The requirement is implemented.

2) Square class extends SHAPE with side length attribute

Constructor: Square(String Name, double x, double y, double length, int Zorder)

Coordinate system: (x,y) represents top-left corner (same as rectangle)

at() method: Similar to rectangle, checks four boundary lines using equal side length

Created via CREATE operation with type "square"

3)If the input contains an incorrect number of arguments (i.e., not four), the command must fail.

Any input that is not a numeric value for these parameters will result in command rejection.

The side length of a square must be a positive value. If the user attempts to input zero or a negative number for the length, the command must be rejected as invalid.

[REQ6] 1) The requirement is implemented.

2) Group class extends SHAPE and contains ArrayList<SHAPE> Members

GROUP(name, members[])

- Adds members via add(shape)

```
while(shape.findFather()!=null)shape=shape.findFather();
if(shape==this)return;
Members.add(shape);
shape.setFather(this);
```

- When adding a shape, we find its father group recursively until its father==null, and add that into this group.

- If the shape is already in this group, skip it.

- Sets member.setFather(group)

- Update the bounding box after adding members.

3) Allow nested groups. When adding a shape, which has already in another group, into a new group, we will add its father group into the new group.

[REQ7] 1) The requirement is implemented.

2) UNGROUP operation class handles group dissolution

Removes group from MAP and sets EXIST=false

Restores member shapes by setting their Father=null

Handles nested groups properly through father references

Members become directly accessible for operations

3) When ungrouping a group, which is in another group, it will add its members into its father group and then delete this group.

```
Group Father=(Group)it.findFather();
if(Father!=null)Father.getMembers().remove(it);
for (SHAPE member : it.getMembers()) {
    member.setFather(null);
    if(Father!=null)Father.add(member);
}
```

[REQ8] 1) The requirement is implemented.

2) it.setEXIST(false) marks shapes and their members as non-existent.

3) Use recursive method to handle group type.

[REQ9] 1) The requirement is implemented.

2) Use a switch statement to get its concrete shape type.

3) Compute the bounding box in different ways according to its type.

[REQ10] 1) The requirement is implemented.

2) Each shape type implements an optimized move() method. For basic shapes, adjust primary coordinates only. For groups, recursively applies movement to all members

3) The command requires exactly three parameters: the target shape's name, the horizontal displacement (dx), and the vertical displacement (dy). If the input is missing any of these parameters, the command must be rejected.

The displacement values, dx and dy, must be provided as valid double-precision numbers. If any non-numeric characters are used for these values, the command should fail.

[REQ11] 1) The requirement is implemented.

2) Traverse the Allshape array from high index to low (i.e. from high Z order to low).

3) If it finds a shape that exists, not in a group, and is at (x,y), return that shape's name.

If not found, return null.

[REQ12] 1) The requirement is implemented.

2) Compute the bounding boxes of the two shapes.

3) If one vertex of one bounding box is in another bounding box, return true, otherwise return false.

[REQ13] 1) The requirement is implemented.

2) List the information about the target shape.

3) Use recursive method to handle group type, and use retract to express hierarchy.

[REQ14] 1) The requirement is implemented.

2) Traverse the Allshape array.

3) If that shape exists and is not in a group, list it.

[REQ15] 1) The requirement is implemented.

2) Say Goodbye and close the GUI.

3) return false to inform the interface stop the while loop.

[BON1] 1) The requirement is implemented.

2) The Gui interface is implemented through JSwing, it's able to display all drawn shapes, zoom in, zoom out, and move up, down, left, right.

3) Gui interface only serves as a display for command and shape, the interactions are all

complete through CLI

[BON2] 1) The requirement is implemented.

2) For redo() and undo() methods, we use two static stacks: pastOperation and undoOperation. Whenever a new command is executed, we clear the undo stack – this follows the classic linear undo/redo model.

3) The methods redo() and undo() check if the stack is empty and then apply to the operations.

3. Learning-to-Learn

Ruiyang Guo:

I consider the most important lesson of this group project is that setting up a git repository is of the utmost importance. Even if it means that some of us have to learn it from zero. During this project, we spent more than 4 hours dealing with version mismatch and synchronization of our code. In comparison, setting up a git repository only took less than half an hour.

Another lesson is that, when coding in a group, it's easier if we settle the behavior of our class and methods. Just like how in C you must declare a method signature in front of the main to let the compiler know the behavior of this method. It makes other coding easier, as they need not worry about your code implementation as long as the behavior is expected.

For future improvement of this project, I want to make it possible that the GUI and CLI could both input command. It requires the architecture of event-driven programming.

MENG Xiangyu:

While implementing the shapeAt functionality, I noticed it has a bias of 0.05, which causes a line segment's effective coverage area to a shape composed of two semicircles and a rectangle. To address this, I learned geometric principles and calculated the perpendicular distance from that point to the line segment. This allows me to determine the point's relative position to the line and ultimately compute the distance between them.

Furthermore, I came to appreciate the importance of encapsulation, which I have never used before. Through proper encapsulation, I enabled my team members to conveniently access and manipulate relevant objects while preventing unintended modifications.

In the future, I wish I could take part in more meaningful group projects. I find there are many differences between self-project and group projects, and it does improve me a lot in collaboration

ability and code readability.

LI Qicheng:

In this group project, I used BufferWriter to implement a logger that records commands in both TXT and HTML files. This material was not covered in lectures, and I learned how to structure HTML files and select appropriate tools for writing in different formats. As a result, I became adept at capturing user input and recognized the significance of program logs.

Additionally, I gained a clearer understanding of class hierarchies and data structures. The extensive use of inheritance and various data structures taught me how to optimize code for greater efficiency and conciseness, which are essential skills in programming.

In the future, I intend to expand my learning to enhance my self-directed approach. Gaining knowledge from diverse fields will broaden my perspective, enabling me to analyze problems more effectively and improve my self-learning efficiency.

Yu Chengjun:

I was responsible for building the Operation and redo/undo system in the Operation part. These operations need stacks to help with. During development, I mastered defining abstract classes and using HashMap. I also deeply understood the complexity of multiple groups problem, like grouping some shapes where there already exist some groups in it. So do the Ungroup operation.

My future learning plan focuses on using inheritance to better organize objects in my projects. I will study how to create more effective class hierarchies, learn when to use abstract classes versus interfaces, and practice identifying common behaviors that can be extracted into parent classes to make code more maintainable and extensible.