Team Name: Fantastic Five

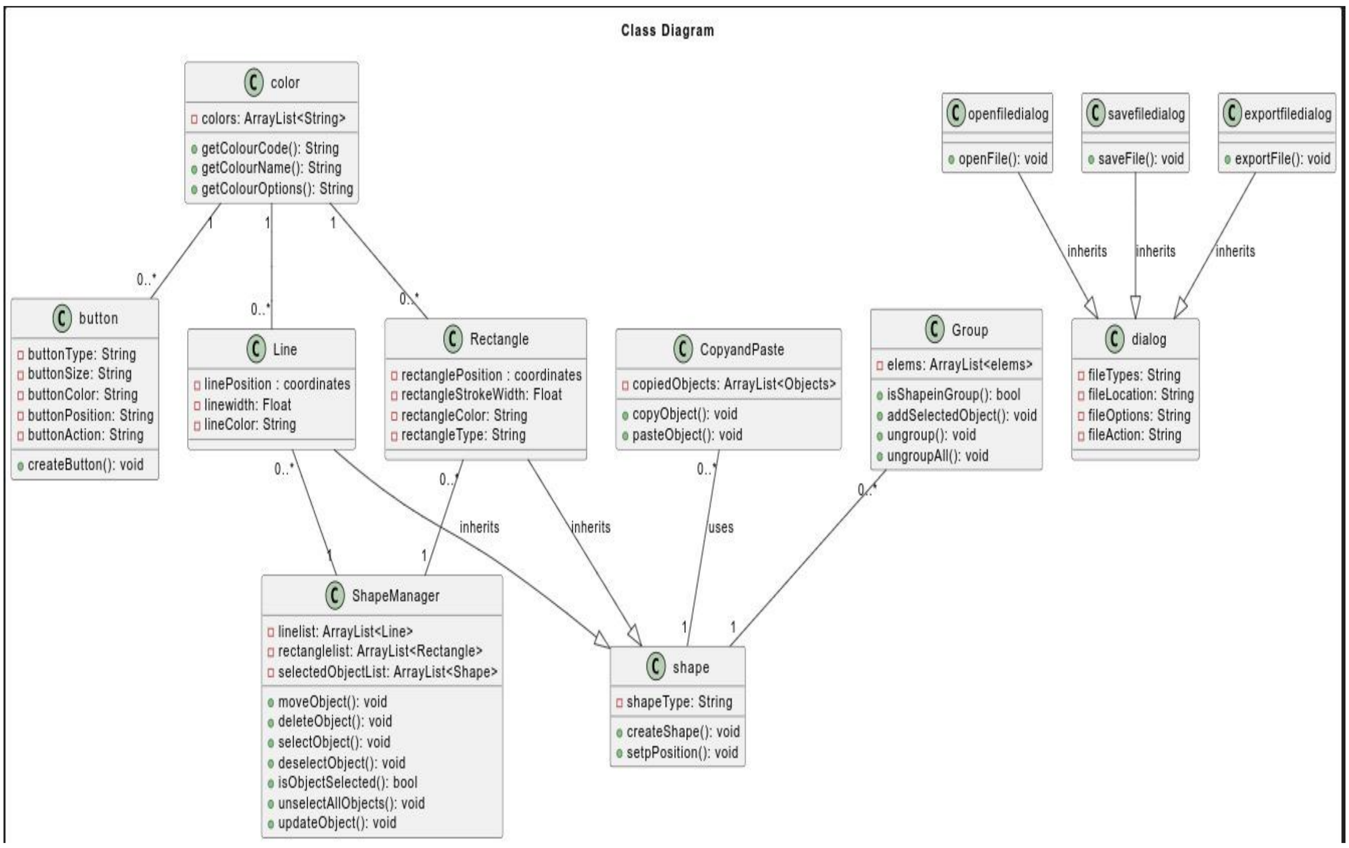
Product Name: SketchCraft

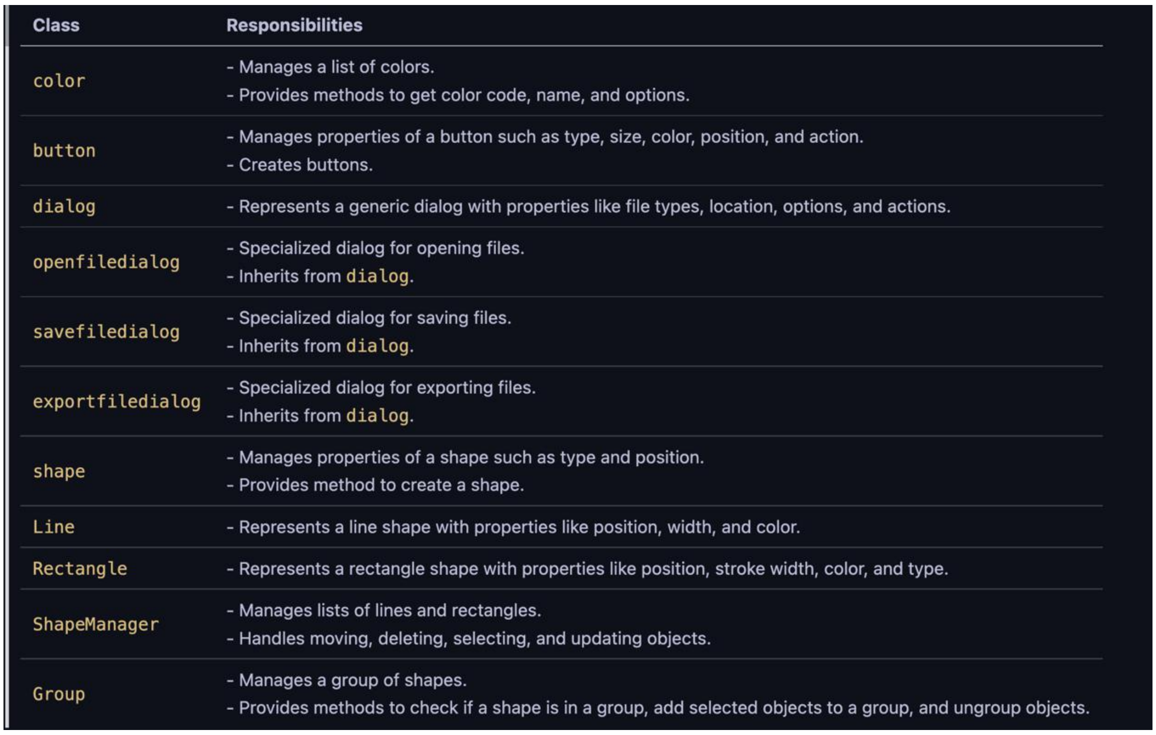
Date: 06/05/2024

Team Members:

Umang Patel(2022101037)
Kevin Thakkar(2022101064)
Archit Pethani(2022101098)
Neel Amrutia(2022101117)
Ansh Shah(2022111030)

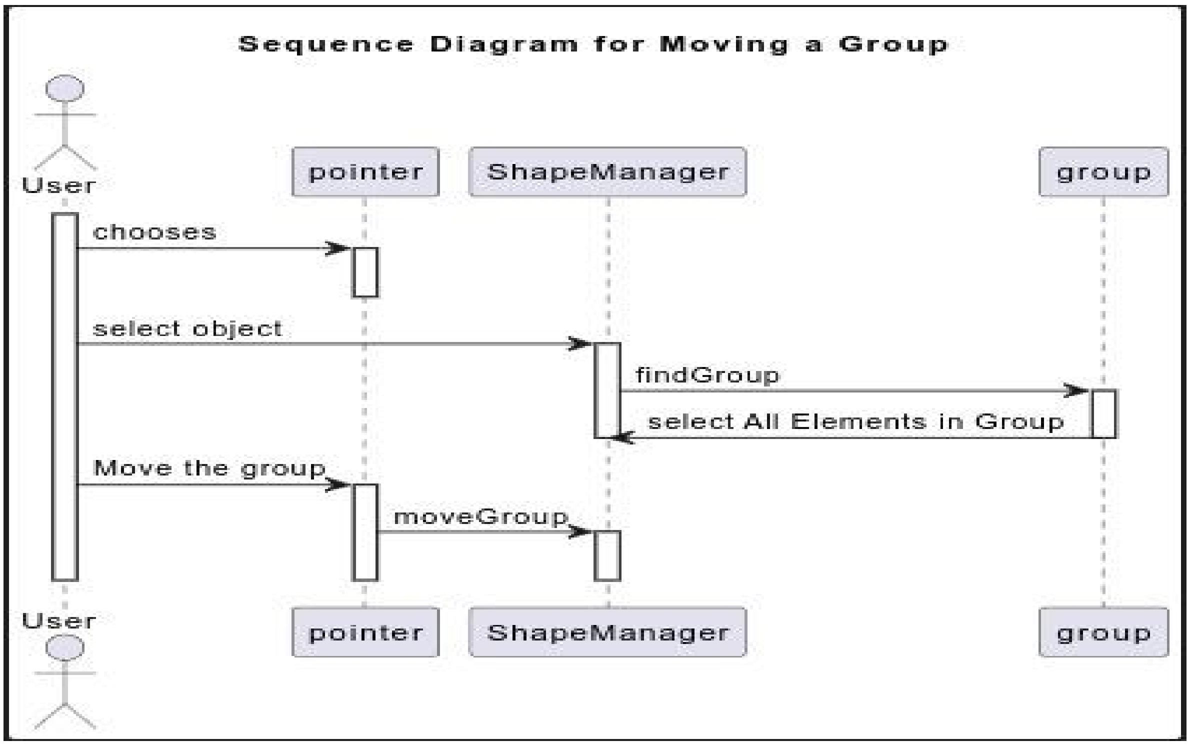
Class Diagram: -

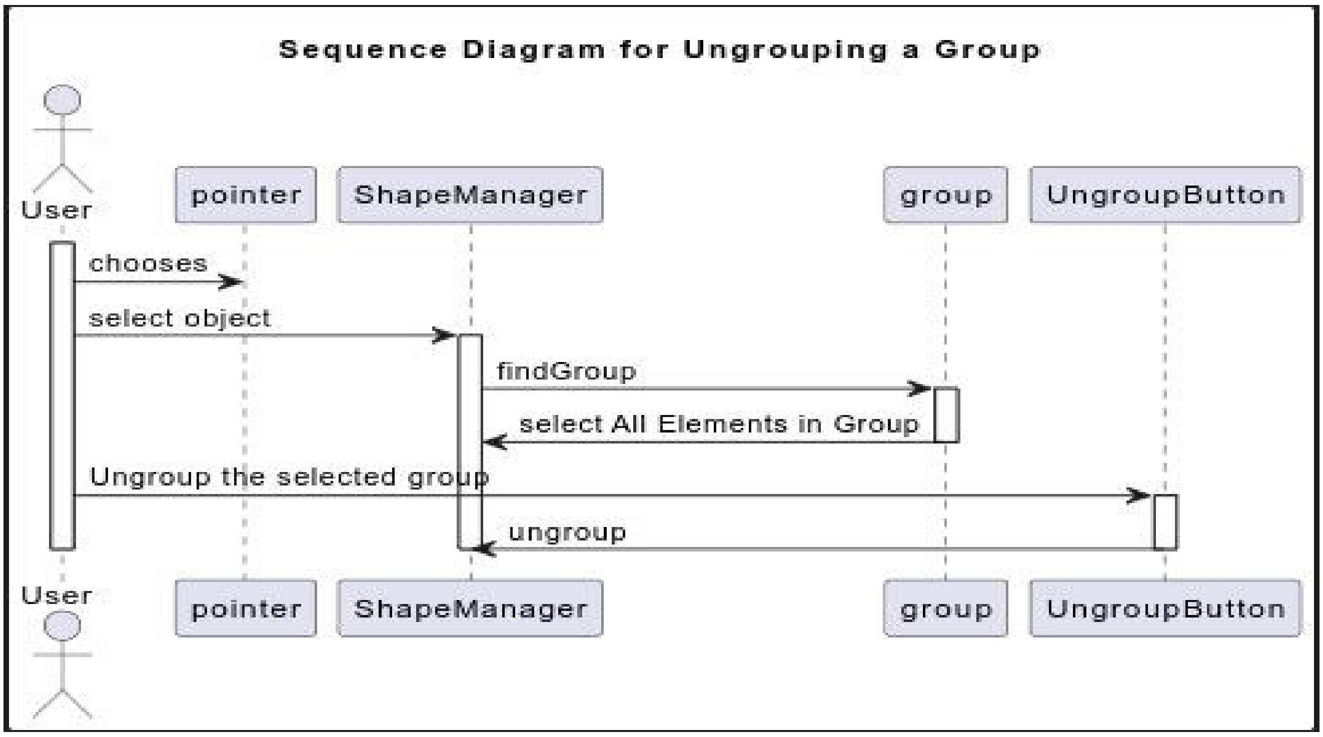


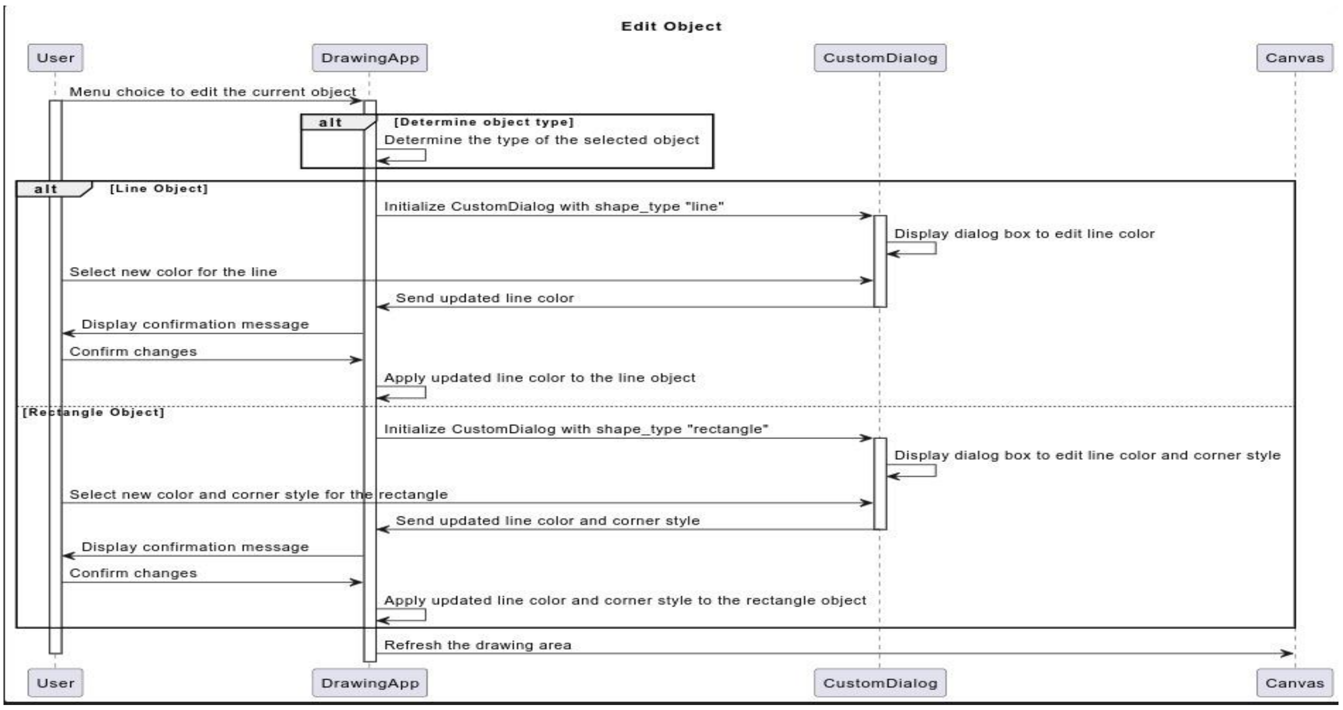


Class	Responsibilities
`color`	- Manages a list of colors. - Provides methods to get color code, name, and options.
`button`	- Manages properties of a button such as type, size, color, position, and action. - Creates buttons.
`dialog`	- Represents a generic dialog with properties like file types, location, options, and actions.
`openfiledialog`	- Specialized dialog for opening files. - Inherits from `dialog`.
`savefiledialog`	- Specialized dialog for saving files. - Inherits from `dialog`.
`exportfiledialog`	- Specialized dialog for exporting files. - Inherits from `dialog`.
`shape`	- Manages properties of a shape such as type and position. - Provides method to create a shape.
`Line`	- Represents a line shape with properties like position, width, and color.
`Rectangle`	- Represents a rectangle shape with properties like position, stroke width, color, and type.
`ShapeManager`	- Manages lists of lines and rectangles. - Handles moving, deleting, selecting, and updating objects.
`Group`	- Manages a group of shapes. - Provides methods to check if a shape is in a group, add selected objects to a group, and ungroup objects.

Sequence Diagram: -




Sequence Diagram for Ungrouping a Group
User
pointer
ShapeManager
group
UngroupButton
chooses
select object
findGroup
select All Elements in Group
Ungroup the selected group
ungroup
User
pointer
ShapeManager
group
UngroupButton


Edit Object
User
DrawingApp
CustomDialog
Canvas
Menu choice to edit the current object
alt
[Determine object type]
Determine the type of the selected object
alt
[Line Object]
Initialize CustomDialog with shape_type "line"
Display dialog box to edit line color
Select new color for the line
Send updated line color
Display confirmation message
Confirm changes
Apply updated line color to the line object
[Rectangle Object]
Initialize CustomDialog with shape_type "rectangle"
Display dialog box to edit line color and corner style
Select new color and corner style for the rectangle
Send updated line color and corner style
Display confirmation message
Confirm changes
Apply updated line color and corner style to the rectangle object
Refresh the drawing area
User
DrawingApp
CustomDialog
Canvas

Design Overview

The design of the drawing application aims to achieve a balance among various design principles such as low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, and reusability. Here's how the design reflects these principles:

1. **Low Coupling:** The classes in the system are designed to have minimal dependencies on each other. For example, the `Color` class is only coupled with the `Button` and `Line` classes through associations, allowing for flexibility in modifying or extending each class without impacting others.

2. **High Cohesion:** Each class is designed to have a single responsibility and encapsulate related functionality. For instance, the `Button` class handles button creation, while the `ShapeManager` class manages shapes, ensuring high cohesion within each class.

3. **Separation of Concerns:** The system is divided into classes that address specific concerns. For example, the `Dialog` class handles file-related dialog operations, while the `Shape` class encapsulates shape creation and manipulation functionality. This separation facilitates easier maintenance and modification of the codebase.

4. **Information Hiding:** The internal details of each class are hidden from other classes, exposing only necessary interfaces. For instance, the `Color` class exposes methods for getting color information without revealing its internal implementation details.

5. **Law of Demeter:** The classes interact with each other through well-defined interfaces, adhering to the principle of least knowledge. For example, the `ShapeManager` class manipulates shapes through methods like `moveObject()`, `deleteObject()`, and `selectObject()`, without directly accessing their internal state.

6. **Extensibility:** The design allows for easy extension by adding new classes or modifying existing ones. For example, new shape types can be added by extending the `Shape` class, and new dialog functionalities can be implemented by extending the `Dialog` class.

7. **Reusability:** The system promotes code reuse by encapsulating common functionalities into reusable components. For instance, the `Dialog` class provides a generic interface for file-related dialogs, which can be reused in different parts of the application.

Design Patterns

The design of the drawing application incorporates several design patterns to achieve the aforementioned balance:

1. **Factory Method Pattern:** The `Button`, `OpenFileDialog`, `SaveFileDialog`, and `ExportFileDialog` classes implement the factory method pattern to create instances of buttons and file dialogs, allowing for easy extension and customization.

2. **Singleton Pattern:** The `ShapeManager` class follows the singleton pattern to ensure that only one instance of the class exists throughout the application, providing a centralized management system for shapes.

3. **Composite Pattern:** The `Group` class implements the composite pattern to treat groups of shapes and individual shapes uniformly, enabling users to perform operations on both groups and individual shapes seamlessly.

How to Compile and Run the Code

1. **Compiling the Code:**
   - Ensure you have the necessary dependencies installed, including Python and any required libraries (e.g., tkinter).
   - Navigate to the directory containing the code files.
   - Run the command `python main.py` to compile and execute the code.

2. **Running the Application:**
   - After compiling the code, the drawing application window will open.
   - Use the menu options to perform various actions such as creating shapes, editing properties, and saving/loading files.

Assumptions :

- On Grouping the objects, on selecting any object from the group, all the elements of the group get selected.