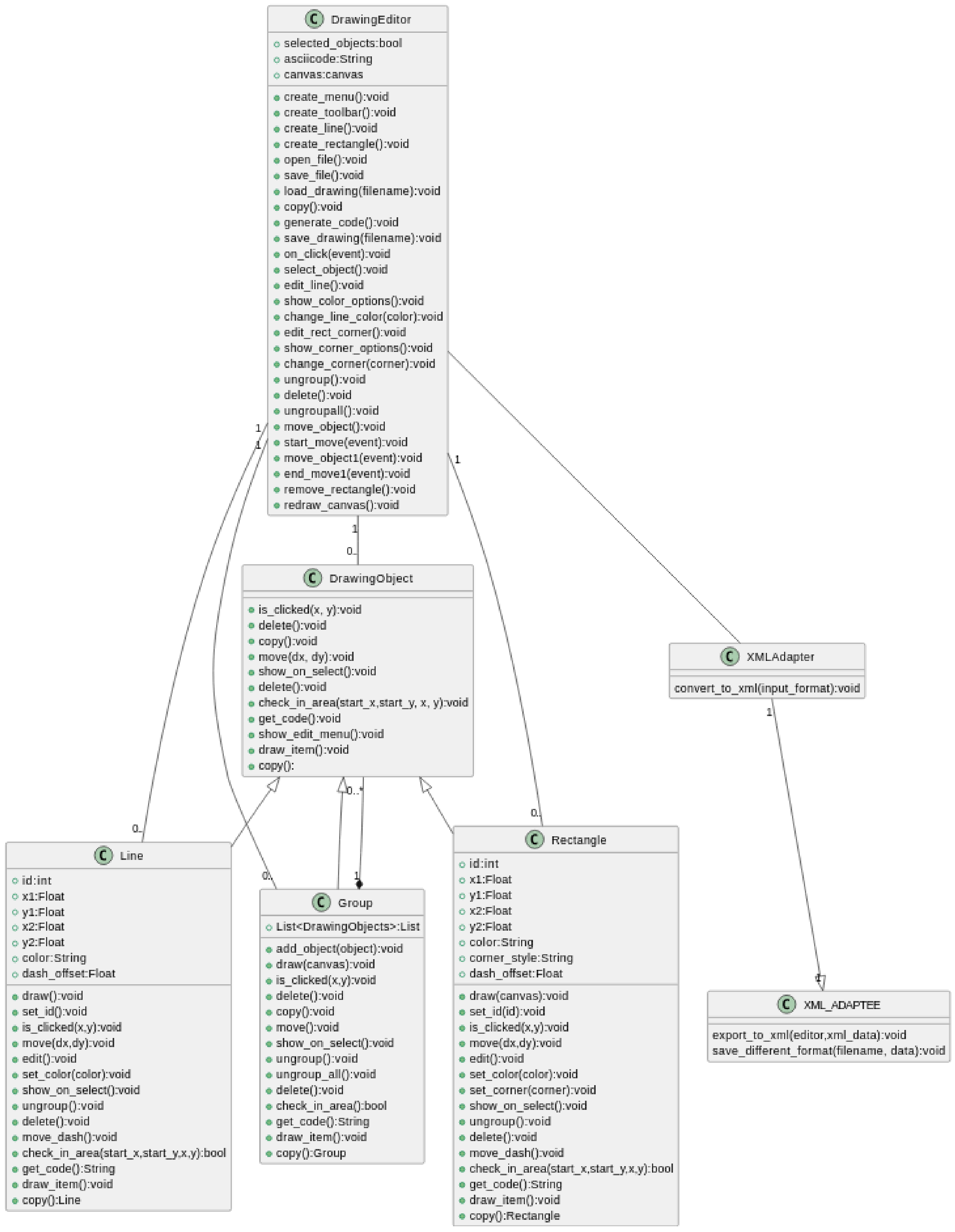
# DOCUMENTATION

Name of the team : **Deadly Designers**
Name of the product : **drawEase**
Date : **6th May, 2024**

Team Members :-
**Shivam Mittal (2022101105)**
**Mayank Mittal (2022101094)**
**Gopal Garg (2022101079)**
**Divyaraj Mahida (2022101085)**

# CLASS DIAGRAM



## Summary of classes

| Class | Responsibilities |
|---|---|
| Line | - Draw: Render the line on the canvas based on its properties such as coordinates, color, and dash offset.<br>- Set and Retrieve Properties: Allow setting and getting properties like ID, coordinates (start and end points), color, and dash offset.<br>- Handle Mouse Interactions: Respond to mouse events such as click, move, and delete to enable user interaction with the line object. |
| Rectangle | - Draw: Render the rectangle on the canvas based on its properties such as coordinates, |

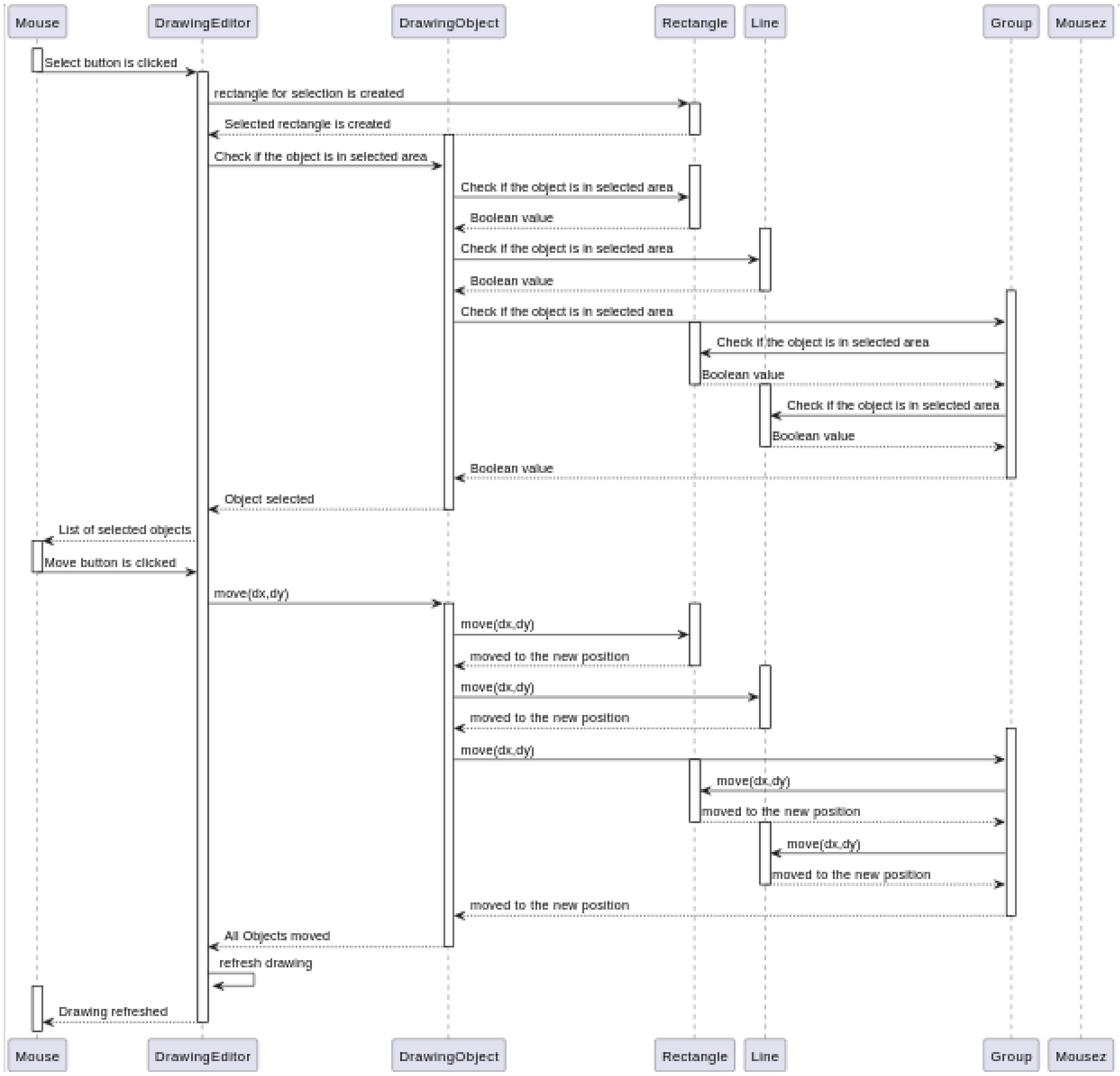
| | |
|---|---|
| | color, corner style, and dash offset.<br>- Set and Retrieve Properties: Allow setting and getting properties like ID, coordinates (top-left and bottom-right corners), color, corner style, and dash offset.<br>- Handle Mouse Interactions: Respond to mouse events such as click, move, and delete to enable user interaction with the rectangle object. |
| Drawing Editor | - Manage User Interface: Handle user interactions and manage UI components like menus, toolbars, and canvas.<br>- Create and Manipulate Objects: Provide methods for creating new drawing objects (lines, rectangles), editing existing objects, and managing object selection. |

| | |
|---|---|
| | - File Operations: Support functionalities for opening, saving, and loading drawing files.<br>- Generate Code: Generate code based on the current drawing for further processing or exporting.<br>- Handle Mouse Events: Respond to mouse events such as clicks, drags, and releases to enable interactive drawing and editing. |
| Group | - Manage Drawing Objects: Maintain a list of drawing objects within the group to organize and manipulate them collectively.<br>- Add, Draw, and Delete Objects: Provide functionality to add objects to the group, draw them on the canvas, and remove them from the group when needed.<br>- Handle Mouse Interactions: Respond to mouse events to enable selection, movement, and |

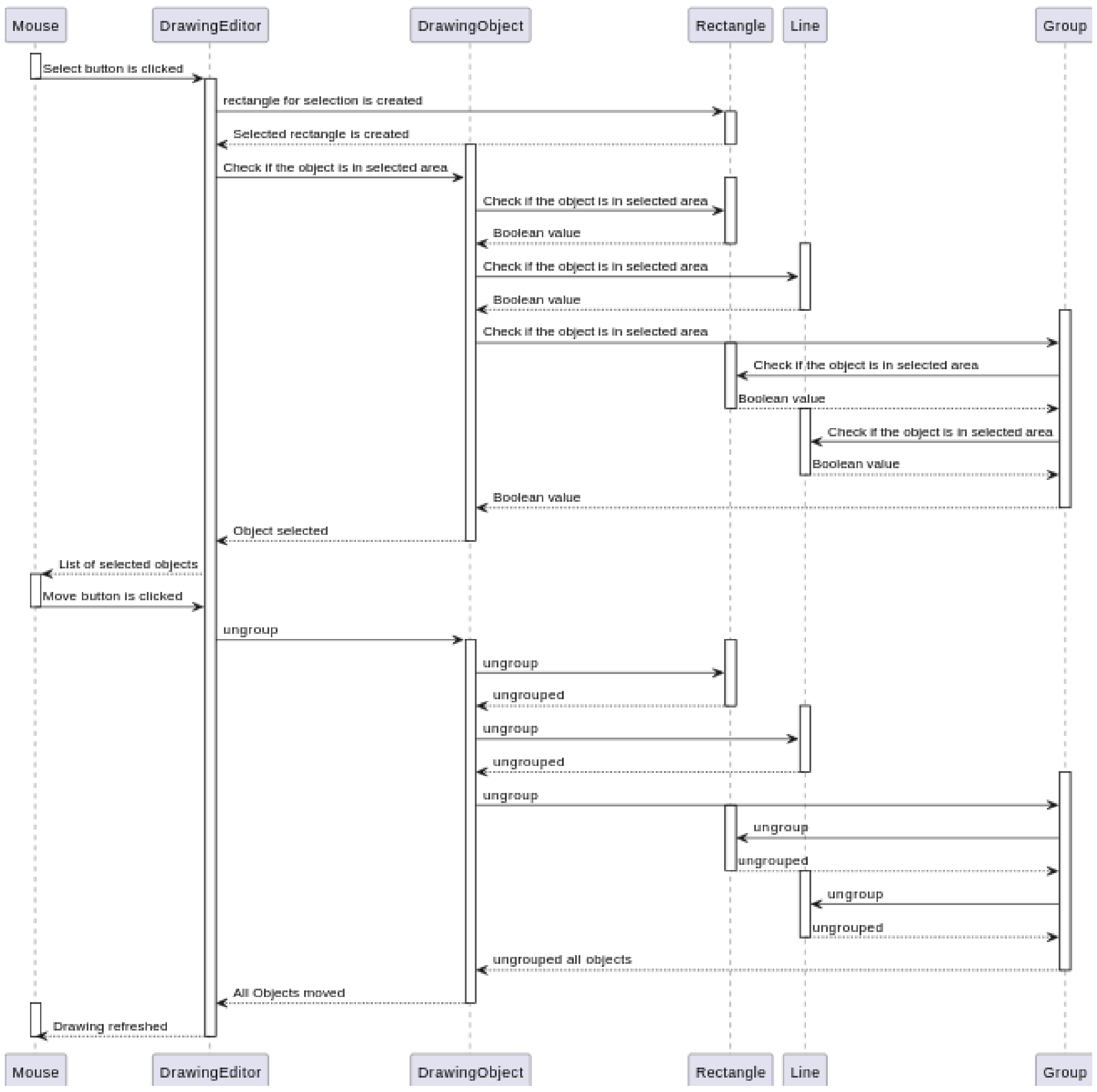
| | |
|---|---|
| | interaction with objects within the group.<br>- Ungroup Objects: Allow breaking the group apart to separate individual objects for independent manipulation. |
| Drawingobject | - Common Behaviors: Define common functionalities shared by all drawing objects, such as handling mouse clicks, moving objects, deleting objects, and copying objects.<br>- Draw: Implement the drawing method to render the object on the canvas with proper rendering.<br>- Show Proper Rendering: Ensure that the object is displayed correctly on the canvas according to its properties and state. |
| XMLAdapter | Converts standard ASCII code to XML format. |
| XML_ADAPTEE | Save or export the XML code. |

## SEQUENCE DIAGRAMS:-

1. Mouse selection of an element that is in a group, drag to a new position, update of the model, refresh of the drawing.


Mouse
DrawingEditor
DrawingObject
Rectangle
Line
Group
Mousez
Select button is clicked
rectangle for selection is created
Selected rectangle is created
Check if the object is in selected area
Check if the object is in selected area
Boolean value
Check if the object is in selected area
Boolean value
Check if the object is in selected area
Check if the object is in selected area
Boolean value
Check if the object is in selected area
Boolean value
Boolean value
Object selected
List of selected objects
Move button is clicked
move(dx,dy)
move(dx,dy)
moved to the new position
move(dx,dy)
moved to the new position
move(dx,dy)
move(dx,dy)
moved to the new position
move(dx,dy)
moved to the new position
moved to the new position
All Objects moved
refresh drawing
Drawing refreshed

2. Mouse selection of an element that is in a group, ungrouping of the elements in the group.



## Properties of the Design

1. In our design for the drawing editor application, we aimed for a balance among several key principles of good software design.

2. **Low Coupling:** We ensured that the classes in our design are loosely coupled. For example, the DrawingEditor class interacts with Line, Rectangle, and Group objects through their common interface DrawingObject. This loose coupling allows for easier modification and maintenance of the code.

3. **High Cohesion:** Each class in our design has a clear and single responsibility. For example, the Line class is responsible for

representing a line object and defining its behavior. This high cohesion improves the readability and maintainability of the code.

4. **Separation of Concerns:** We separated different concerns into different classes. For example, the DrawingObject class defines generic behavior common to all drawing objects, while the Line and Rectangle classes define behavior specific to lines and rectangles, respectively. This separation makes the code more modular and easier to understand.

5. **Information Hiding:** We encapsulated the internal state of objects and exposed only necessary interfaces. For example, the Line class encapsulates the coordinates of the line (x1, y1, x2, y2) and exposes methods to manipulate them (move, is_clicked, delete, etc.). This information hiding improves the maintainability of the code by preventing direct access to internal state.

6. **Law of Demeter:** We adhered to the Law of Demeter by limiting the interactions between objects to only immediate collaborators. For example, the DrawingEditor class interacts with Line, Rectangle, and Group objects through their common interface DrawingObject, rather than directly accessing their internal state. This reduces the coupling between classes and makes the code more maintainable.

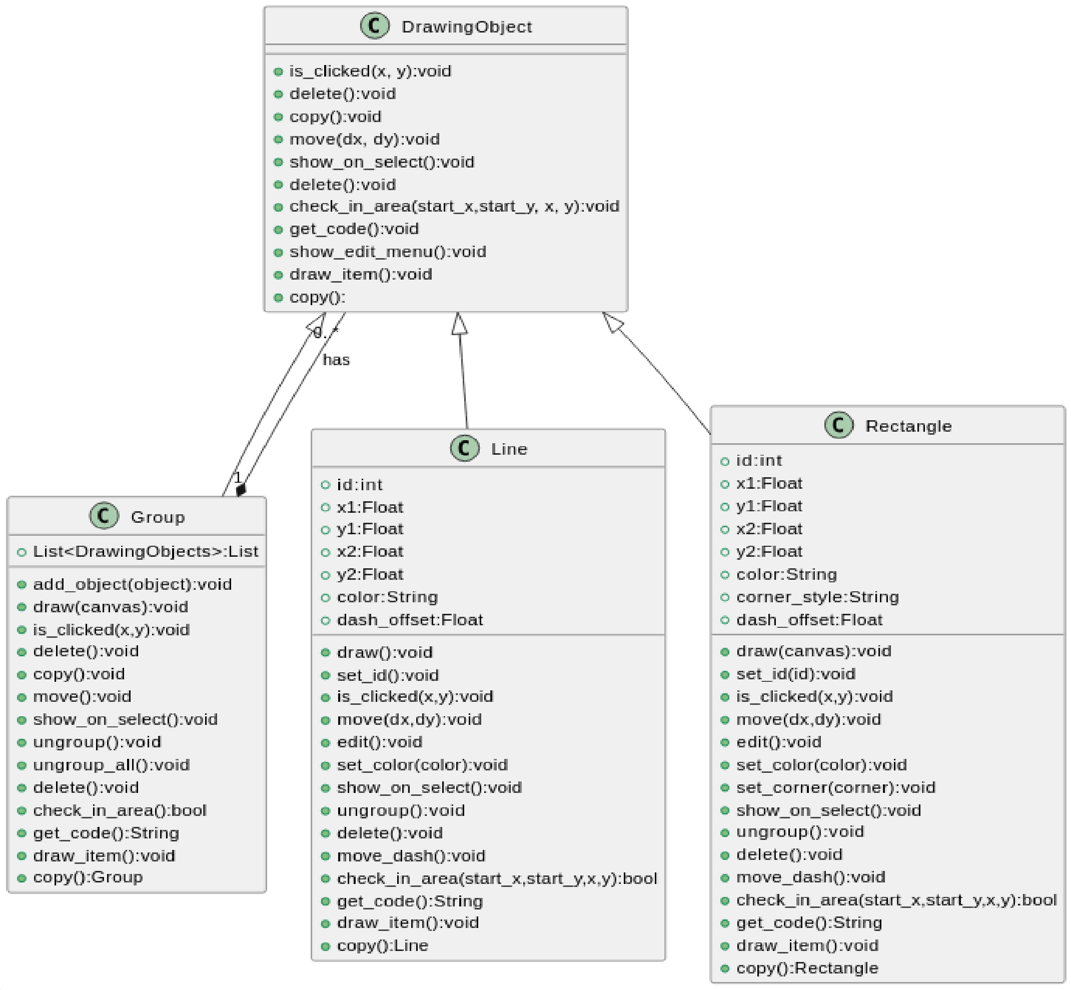
7. **Extensibility:** We ensured that our design allows for easy extension by adding new types of drawing objects or editor functionalities. For example, to add a new type of drawing object, you can simply create a new class that inherits from DrawingObject and implements its methods. This makes the code more flexible and adaptable to future changes.

8. **Reusability:** Our design promotes reusability by defining common interfaces (DrawingObject) and behaviors (move,

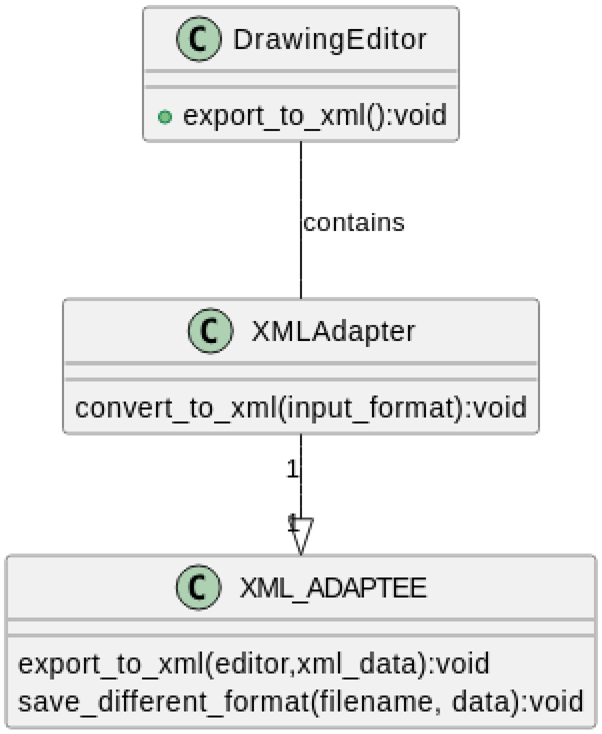
delete, etc.) that can be reused across different types of drawing objects. This reduces code duplication and improves maintainability.

## DESIGN PATTERNS USED:-

1. **Composite Structural Pattern:** We have used a composite Pattern for all Drawing objects in our design. The classes “Line” , “Rectangle” and “Group” are specialized classes of “DrawingObject” class.



2. **Adapter Structural Pattern:** For converting to XML we create XML Adapter class(that converts ASCII to XML ) and XML ADAPTEE class that exports the XML . Using this pattern any new format can easily be added just by writing the Adapter Class for the Format Conversion.


DrawingEditor
export_to_xml():void
contains
XMLAdapter
convert_to_xml(input_format):void
1
1
XML_ADAPTEE
export_to_xml(editor,xml_data):void
save_different_format(filename, data):void

## How we accommodated all points of Flexibility in our design

**1. New primitive drawing object (eg. ellipse):** For doing this we just need to create a new class and inherit the DrawingObject class in this class.This is because of the Composite Design Pattern used in our design.

**2. Adding new operations on the drawing objects (eg. rotate):** We just need to add declare functions in the DrawingObject class , and subsequently write these functions for all Objects in their respective inherited classes(Line, Rectangle,etc). If the implementation code is the same for all shapes , just write the function in Drawing object Class.

**3. Adding or replacing editors for new and existing object types (e.g., line style):**
New object types can easily be added in our design . This is because there is a separate button (hence separate function for click_handler in Drawing editor class) for each of the options.
Example:  line_color in our design has edit_line function in DrawingEditor Class. For adding new line_style just add a new function to the DrawingEditor and call these functions in the already existing functions in the Line Class

**4. New save or export file formats (e.g., JPEG):** New Save and Export formats can easily be added in the design because we have used Adapter Design Pattern. For adding a new export type , we just need to write a new Adapter and Adaptee class of the Format , and the Adapter class will convert the standard ASCII code to Format and the Adaptee class will export it.

Overall, We believe that our design demonstrates a good balance among these competing criteria, making it well-structured and able to accommodate future changes and product evolution.