DRAWING APP

Team Members-
Aadi Prasad (2022115005)
Aisha Harris (2022101012)
Hari Shankar (2022114008)
Aaditya Vardhan Narain (2022111038)

Product Name-
Drawing App

Date-
May 6th,2024

Abstract-
The application is a graphical drawing tool designed to enable users to create and manipulate various shapes and objects on a digital canvas. Users can interact with the interface to draw lines, rectangles, and other geometric shapes, adjust their properties such as color and size, and organize them into groups for easier management. The tool provides a user-friendly menu and toolbar interface for selecting drawing tools, changing colors, saving and opening drawings, and exporting drawings to XML format. Additionally, users can perform actions such as moving, copying, deleting, and grouping objects to customize their compositions. The application's intuitive design and functionality cater to both casual users looking to unleash their creativity and professionals seeking a versatile tool for digital illustration and design.

A Table Summarizing Major Role of Each Class

DrawingApp	Manages the main loop of the application. Handles user input events and updates the display accordingly.
Canvas	Represents the drawing area where objects are displayed. Responsible for drawing objects on the screen.
Toolbar	Provides tools for the user to interact with the canvas (e.g., drawing tools, selecting objects, saving, etc.).

Menu	Displays menu options (e.g., save, open, export to XML). Handles menu-related events.
Object	Abstract base class for drawable objects on the canvas. Defines common properties and methods for objects.
Line	Represents a line object. Inherits from Object.
Rectangle	Represents a rectangle object. Inherits from Object.
GroupedObject	Represents a group of objects. Can contain multiple objects or subgroups. Provides methods for managing groups.

Narrative of Design Principles

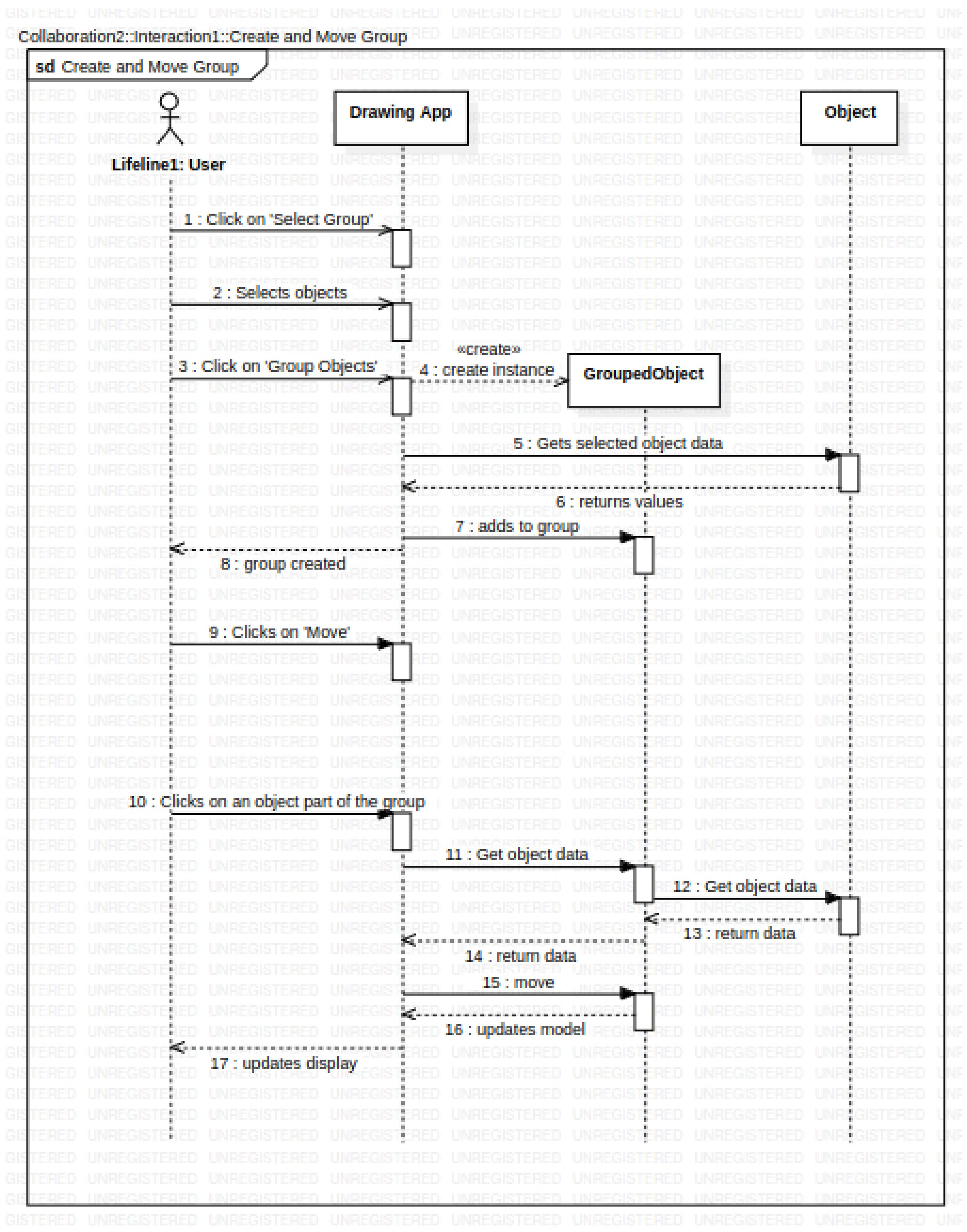
The design of the application reflects a balance among various design principles and criteria to ensure a robust, maintainable, and extensible architecture.

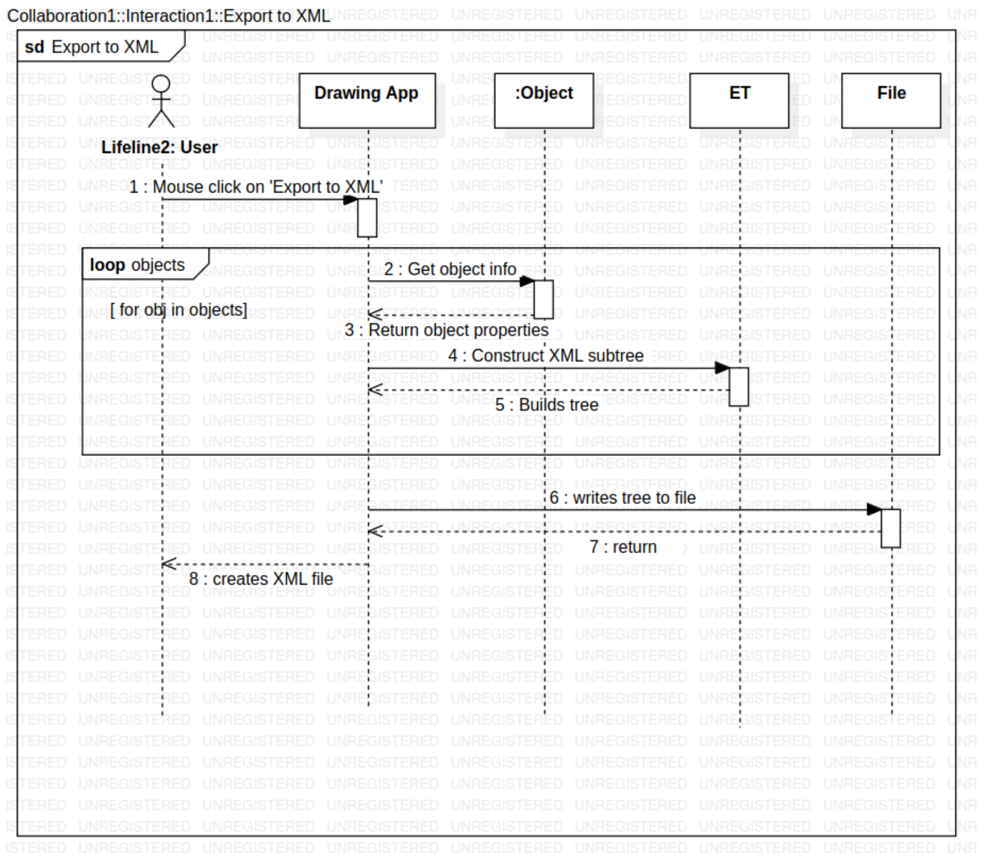
1. **Low Coupling and High Cohesion**: The classes in the application are designed with low coupling and high cohesion. Each class has a specific responsibility and interacts with other classes through well-defined interfaces. For example, the `DrawingApp` class manages the main loop and user input events, while the `Canvas` class handles drawing objects on the screen. This division of responsibilities ensures that changes in one part of the system have minimal impact on other parts, promoting maintainability and scalability.
2. **Separation of Concerns**: The design separates different concerns into distinct classes. For example, the `Menu` class is responsible for displaying menu options and handling menu-related events, while the `Toolbar` class provides tools for interacting with the canvas. This separation allows for better organization of code and makes it easier to understand and maintain.
3. **Information Hiding**: Encapsulation is used to hide the internal implementation details of each class and expose only the necessary interfaces. For instance, the `Object` class encapsulates common properties and methods for drawable objects, abstracting away the specific details of individual objects such as lines or rectangles. This allows for easier maintenance and modification of the codebase without affecting other parts of the system.
4. **Law of Demeter**: The design adheres to the Law of Demeter by limiting the interactions between objects to only direct neighbors. For example, the `Toolbar` class interacts with

the `Canvas` class to perform drawing operations, rather than directly accessing the objects on the canvas. This helps to reduce the complexity of the system and improves its modularity.

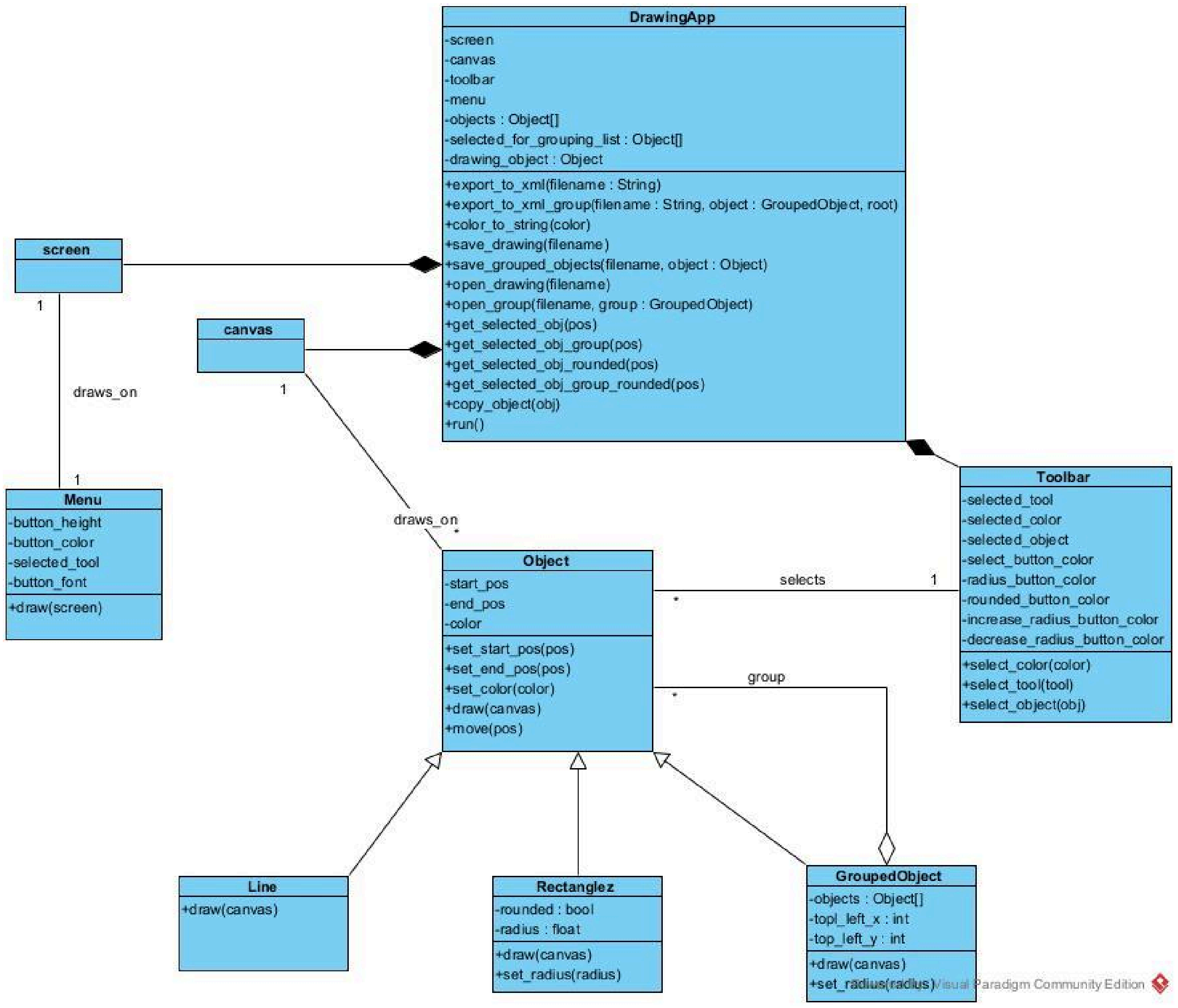
5. **Extensibility and Reusability**: The design is built with extensibility and reusability in mind. For example, the `Object` class serves as a base class for drawable objects, allowing for the easy addition of new object types in the future. Similarly, the use of inheritance and composition enables code reuse across different parts of the application.
6. **Design Patterns**: Several design patterns are used to achieve the balance among competing criteria. For example, the Composite pattern is employed to represent groups of objects in the `GroupedObject` class, allowing for the treatment of individual objects and groups in a uniform manner. Additionally, the Observer pattern may be used to decouple the user interface from underlying data changes, enhancing maintainability and scalability.

Sequence Diagrams





Class Diagram


DrawingApp
-screen
-canvas
-toolbar
-menu
-objects : Object[]
-selected_for_grouping_list : Object[]
-drawing_object : Object
+export_to_xml(filename : String)
+export_to_xml_group(filename : String, object : GroupedObject, root)
+color_to_string(color)
+save_drawing(filename)
+save_grouped_objects(filename, object : Object)
+open_drawing(filename)
+open_group(filename, group : GroupedObject)
+get_selected_obj(pos)
+get_selected_obj_group(pos)
+get_selected_obj_rounded(pos)
+get_selected_obj_group_rounded(pos)
+copy_object(obj)
+run()
screen
1
draws_on
1
Menu
-button_height
-button_color
-selected_tool
-button_font
+draw(screen)
canvas
1
draws_on
*
Object
-start_pos
-end_pos
-color
+set_start_pos(pos)
+set_end_pos(pos)
+set_color(color)
+draw(canvas)
+move(pos)
selects
*
1
group
*
Toolbar
-selected_tool
-selected_color
-selected_object
-select_button_color
-radius_button_color
-rounded_button_color
-increase_radius_button_color
-decrease_radius_button_color
+select_color(color)
+select_tool(tool)
+select_object(obj)
Line
+draw(canvas)
Rectanglez
-rounded : bool
-radius : float
+draw(canvas)
+set_radius(radius)
GroupedObject
-objects : Object[]
-topl_left_x : int
-top_left_y : int
+draw(canvas)
+set_radius(radius)
Visual Paradigm Community Edition