Drawing Editor - PaintWizard

TEAM *ArtistryDevs*
Priet Ukani(2022111039),
Sahil Patel(2022101046),
Sparsh Goel(2022101051),
Garvit Gupta(2022101113),
Tanishq Agarwal(2022101060)

—

Date: 06-05-2024

—

Design and Analysis of Software Systems

PaintWizard App

This is a Python application that provides a simple paint program with various drawing and editing features. The app is built using the tkinter library, which is the standard GUI (Graphical User Interface) library for Python.

The main features of the app include:

1. **Drawing Tools**: Users can draw lines and rectangles on the canvas using the provided toolbar buttons.
2. **Selection and Editing**: Users can select one or more shapes on the canvas and perform various operations such as moving, copying, deleting, and editing the properties (color, width, corner style) of the selected shapes.
3. **Grouping and Ungrouping**: Users can group and ungroup multiple shapes together, allowing them to manipulate the grouped shapes as a single entity.
4. **Open and Save Files**: Users can open existing files in either TXT or XML format, which contain information about the drawn shapes and their properties. The app can load and display the shapes from these files.
5. **Copy and Paste**: Users can copy and paste shapes within the canvas.

The app uses various UI components such as buttons, toolbars, dialogs, and color choosers to provide an interactive user experience. The code also includes custom functions for drawing shapes with rounded corners and handling user input events.

Overall, this Python application provides a basic paint program interface with essential drawing and editing capabilities, allowing users to create, manipulate, and manage simple geometric shapes on a canvas.

Language/Module Used

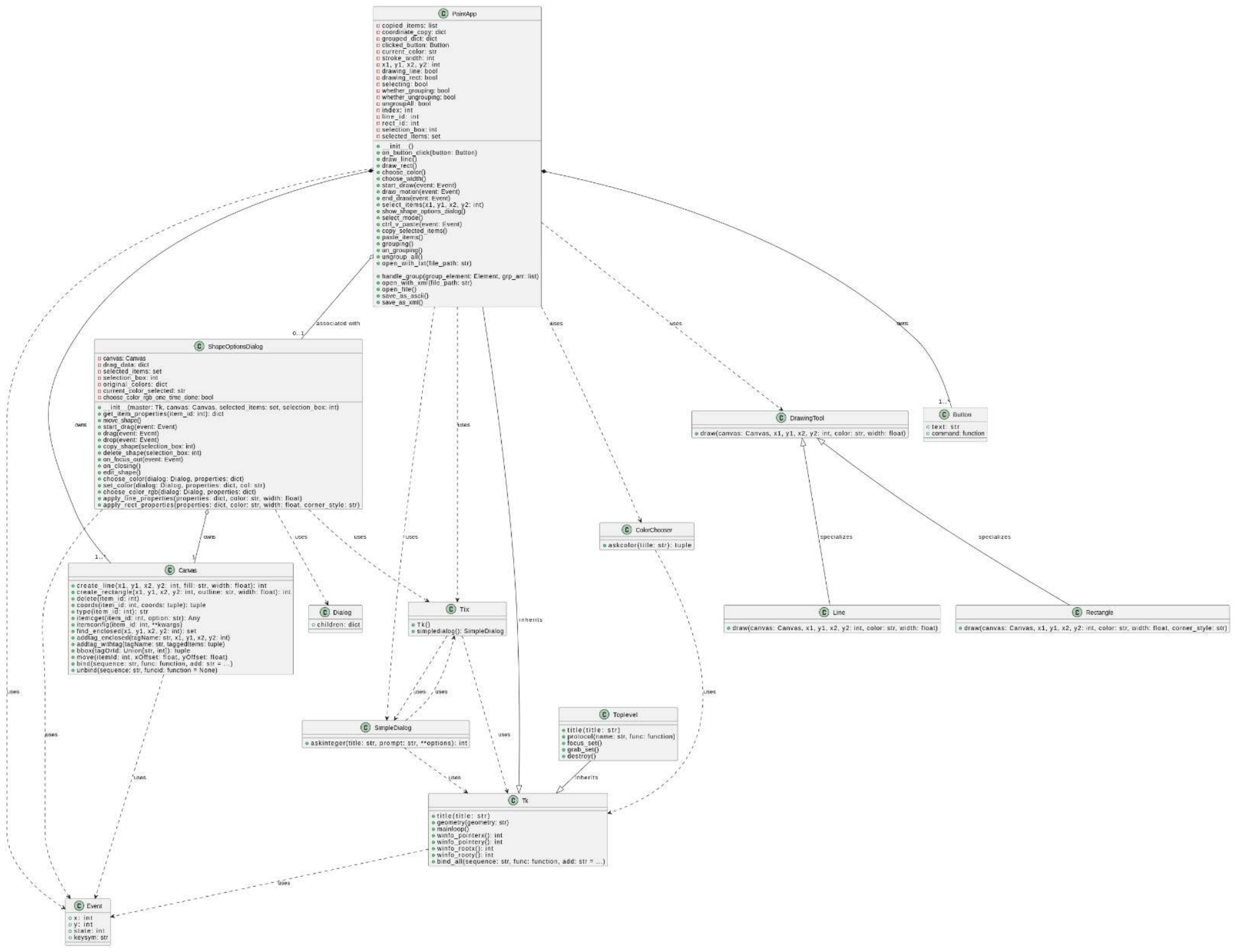
Our app is completely based on Python and we have used the module Tkinter for GUI in our application.

File Structure

The files in the ZIP File include-

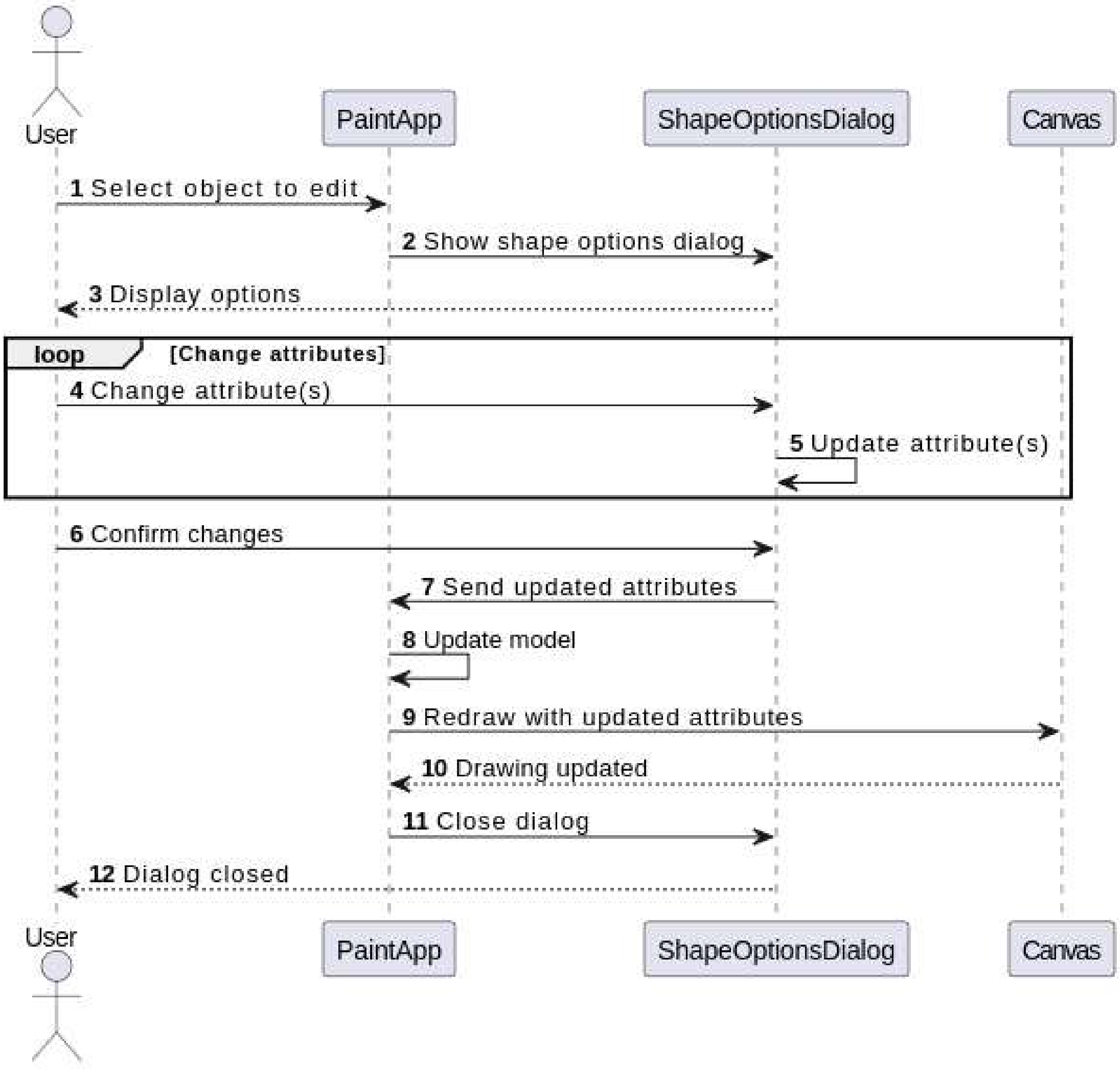
1. Main.py – The source code for the application
2. ClassDiagram.png
3. SeqDiagram1.png
4. SeqDiagram2.png
5. SeqDiagram3.png
6. README.txt – Instructions to Use
7. Report.pdf – Detailed explanation and diagrams.

CLASS DIAGRAM


PaintApp
copied_items: list
coordinate_copy: dict
grouped_dict: dict
clicked_button: Button
current_color: str
stroke_width: int
x1, y1, x2, y2: int
drawing_line: bool
drawing_rect: bool
selecting: bool
whether_grouping: bool
whether_ungrouping: bool
ungroupAll: bool
index: int
line_id: int
rect_id: int
selection_box: int
selected_items: set
__init__()
on_button_click(button: Button)
draw_line()
draw_rect()
choose_color()
choose_width()
start_draw(event: Event)
draw_motion(event: Event)
end_draw(event: Event)
select_items(x1, y1, x2, y2: int)
show_shape_options_dialog()
select_mode()
ctrl_v_paste(event: Event)
copy_selected_items()
paste_items()
grouping()
un_grouping()
ungroup_all()
open_with_txt(file_path: str)
handle_group(group_element: Element, grp_arr: list)
open_with_xml(file_path: str)
open_file()
save_as_ascii()
save_as_xml()
associated with
0..1
ShapeOptionsDialog
canvas: Canvas
drag_data: dict
selected_items: set
selection_box: int
original_colors: dict
current_color_selected: str
choose_color_rgb_one_time_done: bool
__init__(master: Tk, canvas: Canvas, selected_items: set, selection_box: int)
get_item_properties(item_id: int): dict
move_shape()
start_drag(event: Event)
drag(event: Event)
drop(event: Event)
copy_shape(selection_box: int)
delete_shape(selection_box: int)
on_focus_out(event: Event)
on_closing()
edit_shape()
choose_color(dialog: Dialog, properties: dict)
set_color(dialog: Dialog, properties: dict, col: str)
choose_color_rgb(dialog: Dialog, properties: dict)
apply_line_properties(properties: dict, color: str, width: float)
apply_rect_properties(properties: dict, color: str, width: float, corner_style: str)
owns
owns
1..*
1
Canvas
create_line(x1, y1, x2, y2: int, fill: str, width: float): int
create_rectangle(x1, y1, x2, y2: int, outline: str, width: float): int
delete(item_id: int)
coords(item_id: int, coords: tuple): tuple
type(item_id: int): str
itemcget(item_id: int, option: str): Any
itemconfig(item_id: int, **kwargs)
find_enclosed(x1, y1, x2, y2: int): set
addtag_enclosed(tagName: str, x1, y1, x2, y2: int)
addtag_withtag(tagName: str, taggedItems: tuple)
bbox(tagOrId: Union[str, int]): tuple
move(itemId: int, xOffset: float, yOffset: float)
bind(sequence: str, func: function, add: str = ...)
unbind(sequence: str, funcid: function = None)
uses
uses
uses
uses
uses
uses
Dialog
children: dict
Tix
Tk()
simpledialog(): SimpleDialog
uses
uses
uses
inherits
SimpleDialog
askinteger(title: str, prompt: str, **options): int
uses
Toplevel
title(title: str)
protocol(name: str, func: function)
focus_set()
grab_set()
destroy()
inherits
Tk
title(title: str)
geometry(geometry: str)
mainloop()
winfo_pointerx(): int
winfo_pointery(): int
winfo_rootx(): int
winfo_rooty(): int
bind_all(sequence: str, func: function, add: str = ...)
uses
Event
x: int
y: int
state: int
keysym: str
uses
uses
ColorChooser
askcolor(title: str): tuple
uses
uses
DrawingTool
draw(canvas: Canvas, x1, y1, x2, y2: int, color: str, width: float)
owns
1..*
Button
text: str
command: function
specializes
specializes
Line
draw(canvas: Canvas, x1, y1, x2, y2: int, color: str, width: float)
Rectangle
draw(canvas: Canvas, x1, y1, x2, y2: int, color: str, width: float, corner_style: str)

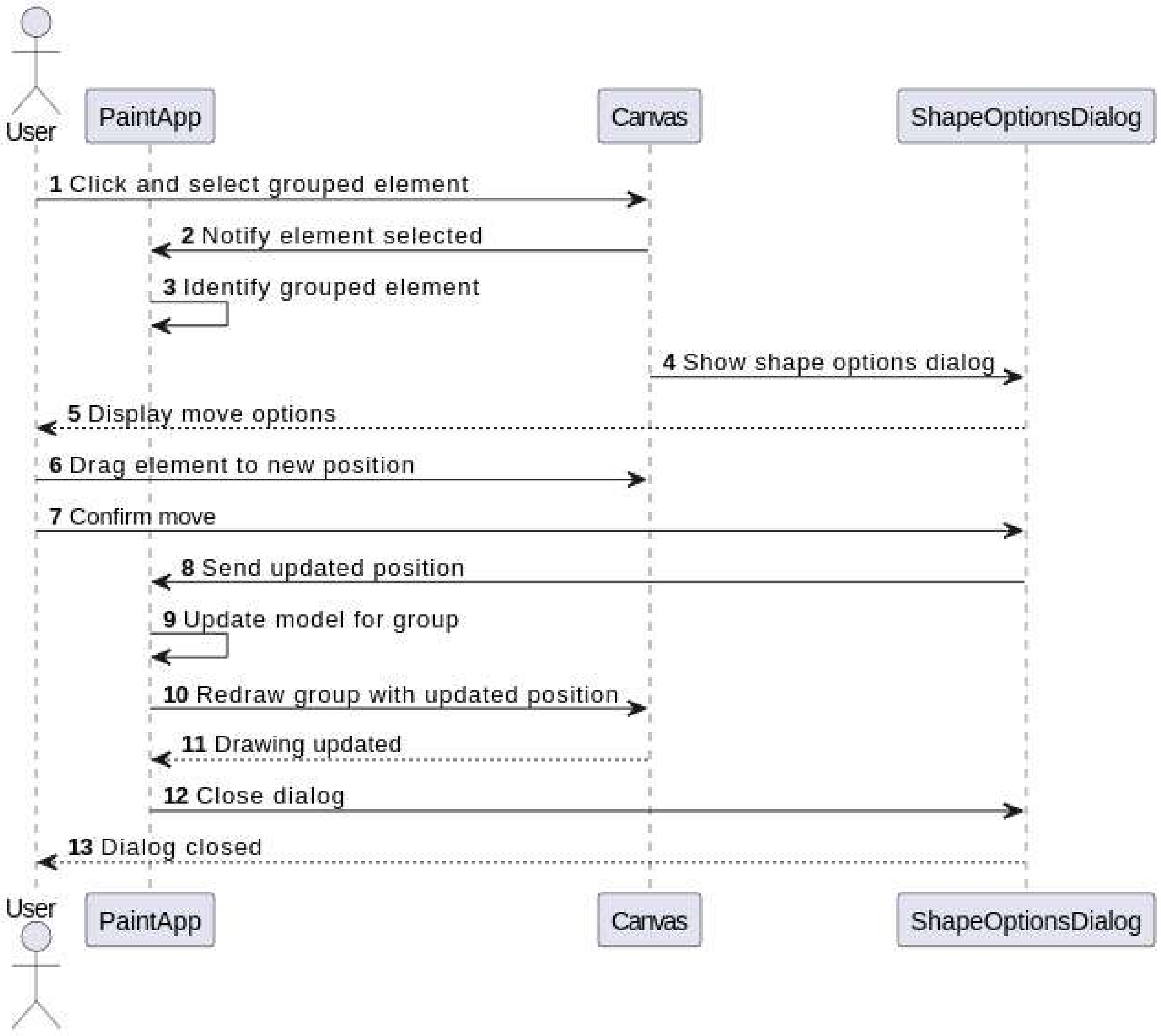
SEQUENCE DIAGRAM 1

Menu choice to edit the current object, user change of attributes, user confirmation of change, update of element in the model, refresh of the drawing.



SEQUENCE DIAGRAM 2

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



CLASS RESPONSIBILITIES

Class	Responsibilities
`PaintApp`	The main application class that handles the overall functionality of the paint app, including drawing lines and rectangles, selecting items, copying/pasting, grouping/ungrouping, color selection, stroke width selection, file I/O operations, and event handling.
`ShapeOptionsDialog`	A dialog box that allows the user to manipulate selected shapes, such as moving, copying, deleting, and editing properties like color and line width.
`DrawingTool`	An abstract base class for drawing tools like `Line` and `Rectangle`.
`Line`	A concrete class that inherits from `DrawingTool` and is responsible for drawing lines on the canvas.
`Rectangle`	A concrete class that inherits from `DrawingTool` and is responsible for drawing rectangles on the canvas.
`Button`	Represents a button UI element with a text label and a command to execute when clicked.
`Canvas`	Provides a drawing surface where shapes can be created, manipulated, and displayed. It offers various methods for creating and modifying shapes, as well as handling events.
`Dialog`	A base class for dialogs, likely used by `ShapeOptionsDialog`.
`ColorChooser`	A utility class that provides a dialog for selecting colors.
`SimpleDialog`	A dialog box for obtaining simple input from the user, such as integers.
`Event`	Represents an event object containing information about user interactions, like mouse clicks and key presses.
`Tix`	A library that provides utility classes like `Tk` and `SimpleDialog`.
`Toplevel`	A top-level window, likely used for dialogs like `ShapeOptionsDialog`.
`Tk`	The main window of the application, providing methods for setting up the window and handling events.

DESIGN PARAMETERS

The UML diagram provided outlines the design of a PaintApp application, showcasing a balance among various design principles such as low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, and reusability. Let's discuss how these principles are reflected in the design and how they contribute to the expected product evolution:

1. **Low Coupling and High Cohesion**:
    - Low coupling is achieved by having classes communicate through well-defined interfaces and minimizing direct dependencies between them. For example, the `PaintApp` class interacts with other classes like `Button`, `Canvas`, and `ShapeOptionsDialog` through clearly defined methods, promoting encapsulation and reducing interdependence.
    - High cohesion is ensured by organizing related functionalities within each class. For instance, methods like `draw_line()`, `draw_rect()`, and `select_items()` are encapsulated within the `PaintApp` class, focusing on a specific aspect of the application's behavior.
2. **Separation of Concerns**:
    - The design separates different concerns into distinct classes. For example, the `PaintApp` class is responsible for managing the overall application state and user interactions, while the `DrawingTool`, `Line`, and `Rectangle` classes handle specific drawing functionalities. This separation allows for easier maintenance and modification of individual components without affecting the entire system.
3. **Information Hiding**:
    - The internal details of each class are hidden from external entities, promoting encapsulation and abstraction. For instance, the `ShapeOptionsDialog` class encapsulates the logic for managing shape properties and interactions with the canvas, hiding the implementation details from the `PaintApp` class.
4. **Law of Demeter**:
    - The design adheres to the Law of Demeter by limiting the interaction between objects to only immediate collaborators. For example, the `PaintApp` class interacts with the `Canvas` class to perform drawing operations, without directly accessing the internal details of other classes like `DrawingTool` or `ShapeOptionsDialog`.
5. **Extensibility and Reusability**:
    - The design allows for easy extension and reuse of components. For instance, new drawing tools or shapes can be added by creating subclasses of `DrawingTool` and implementing specific drawing logic without modifying existing code. Similarly, the `ShapeOptionsDialog` class can be reused in other applications that require shape manipulation functionalities.
6. **Design Patterns**:
    - The design employs various design patterns to achieve its objectives. For example, the Command pattern is used to encapsulate user actions within `Button` objects, allowing for easy undo/redo functionality and decoupling user interface elements from application logic. Additionally, the Observer pattern can be inferred from the event handling mechanisms, where objects like `Canvas` and `ShapeOptionsDialog` observe user events and respond accordingly.

Overall, the design of the PaintApp application demonstrates a thoughtful balance among competing criteria, ensuring flexibility, maintainability, and scalability in the face of evolving product requirements. By adhering to established design principles and leveraging design patterns effectively, the design lays a solid foundation for future enhancements and iterations of the product.