GraphiFlow

A Unified Modeling Language (UML) Tool for Streamlined Software Design

2022101106, 2022101099, 2022101043, 2022113010, 2022101030

UML Class Diagram

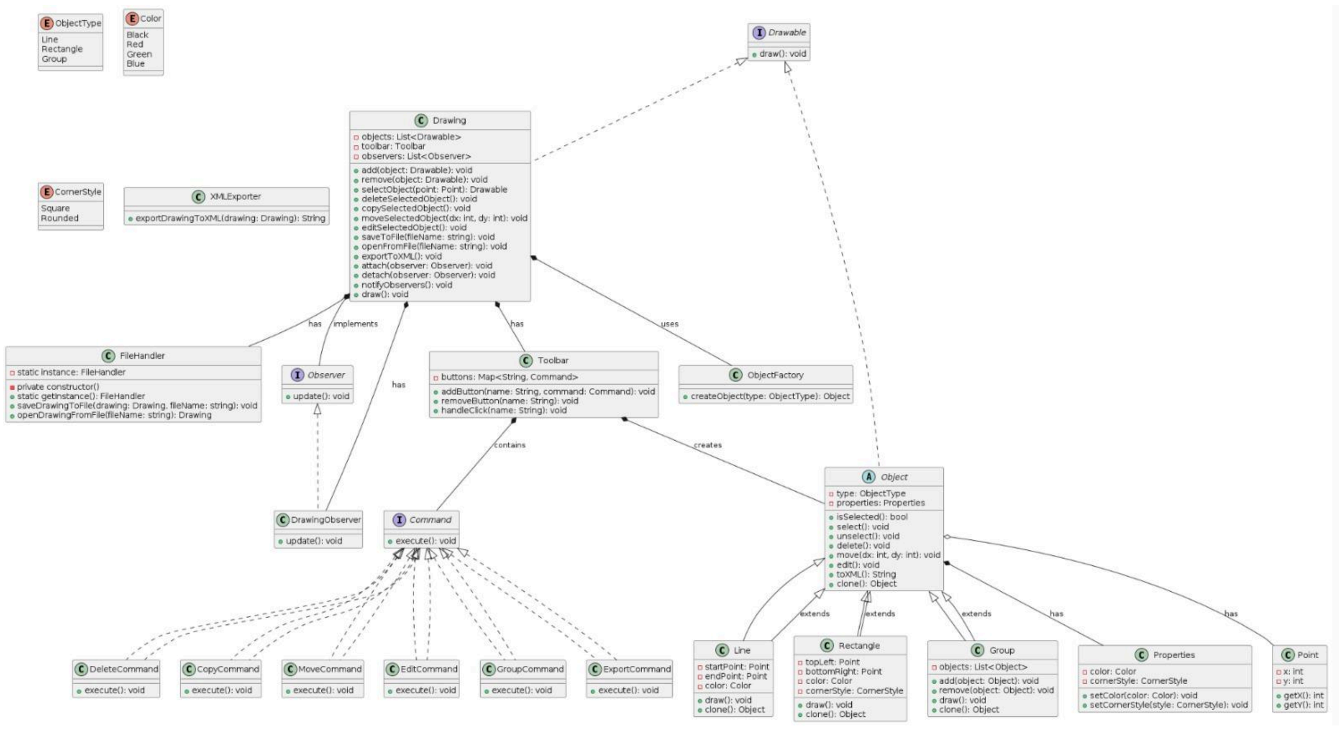
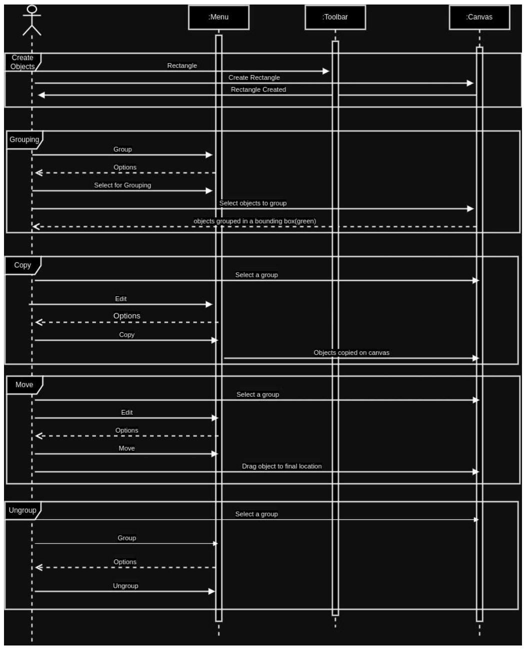


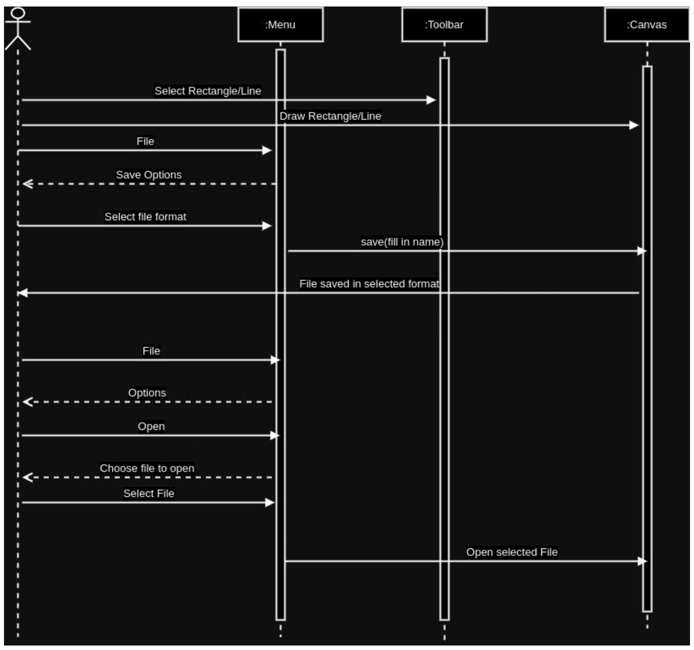
Table summarising the responsibilities of each major class

Drawable	Defines a generic drawable object with methods for drawing and manipulating its state.
Drawing	Manages a collection of drawable objects, including adding, selecting and managing them while observing changes in their state.
Toolbar	Contains a set of commands associated with UI buttons to manipulate the drawing, including adding and removing buttons linked to commands.
Point	Represents a coordinate point with x and y values used for defining positions of drawable objects.
Command	Abstract class for commands that can be executed, such as draw, copy, delete and move commands.

ObjectFactory	Creates drawable objects of specific types based on the input object type.
Object	Represents a drawable object with properties like type, colour, selection status and methods for moving and editing. Extends to specific shapes like Line and Rectangle.
Properties	Manages properties of drawable objects such as colour and corner style applicable to various shapes.
Observer	An interface for observing changes within objects, requiring an update method.
DrawingObserver	Implements Observer to react to updates in the drawing state.
FileHandler	Manages file operations for drawings such as saving to or loading from files, utilising a singleton pattern for its instance.
XMLExporter	Handles exporting drawings to a XML format, converting a drawing's data into XML string format.
DeleteCommand, CopyCommand, MoveCommand, EditCommand, GroupCommand, ExportCommand	Concrete implementations of the Command interface, defining specific actions for manipulating the drawing's state.

Sequence Diagrams





Low Coupling and High Cohesion

Low Coupling: The system is structured such that each class has a well-defined purpose with minimal dependencies on other classes. For example, the FileHandler manages file operations independently from the UI classes like Menu and Toolbar. This separation ensures changes in file handling logic require minimal changes to other system parts.

High Cohesion: Each class focuses on a single responsibility. Drawing manages a collection of drawable objects, Toolbar handles UI elements associated with commands, and ObjectFactory solely creates objects. This focus enhances maintainability and understanding of the system.

Separation of Concerns and Information Hiding

Separation of Concerns: The system divides functionality among classes based on distinct features: drawing operations, UI management, and file handling. This separation ensures that implementing changes or fixing bugs in one module doesn't affect others unnecessarily.

Information Hiding: Classes encapsulate their data and expose only necessary methods to other parts of the system. For instance, the Drawable class exposes a draw() method, but internal states like position or colour are managed internally and modified through controlled methods.

Law of Demeter

The classes adhere to the Law of Demeter, often communicating only with directly related classes. For example, Command classes do not directly manipulate properties of the canvas but instead call methods of Drawing, which in turn manages the drawable objects.

Extensibility and Reusability

Extensibility: The use of the Command pattern for operations like move, delete, and copy makes the system extensible. New commands can be added with minimal changes to existing code, adhering to the open-closed principle.

Reusability: Components such as Drawable, Command, and ObjectFactory are designed to be reused. For example, ObjectFactory can be extended to support more object types without modifying existing code.

Design Patterns

Factory Pattern: Used in ObjectFactory to create instances of drawable objects. This pattern supports the easy addition of new drawable types.

Command Pattern: Applied in managing user actions like moving and copying objects. It encapsulates a request as an object, allowing flexible parameterization of GUI components with different requests.

Observer Pattern: Employed to update various components about state changes in the drawing, ensuring that the UI is always in sync with the backend state.

Singleton Pattern: The FileHandler class is implemented as a singleton, ensuring that there is a single instance managing file operations, which is crucial for consistency and effective resource management.

Conclusion

The application's architecture demonstrates a well-thought-out design that successfully balances flexibility, efficiency, and robustness, anticipating future product evolution needs. These principles and patterns not only ensure the software is robust and maintainable but also facilitate future enhancements and integrations.