Design Overview for Shape Drawer

Name: Vasudev Kallumada Vinod
Student ID: 104672319

Summary of Program

This application is a basic shape drawing application known as Shape Drawer. It enables users to draw, adjust, and save a drawing consisting of rectangles, circles, and lines. Users control the application using keyboard keys and clicking the mouse to add, highlight, move, and remove shapes. The application has the functionality to change the background color, draw random shapes, and save/load the drawing from a file. It also includes a functionality that writes the user's name on the screen with shapes.

Required Roles

Shape (abstract class) details

Responsibility	Type Details	Notes
Store common shape data	Color, float X, float Y, bool selected	All shapes inherit from this class
Abstract draw methods	Void Draw(), void DrawOutline(), bool isAt(Point2D)	Implement differently by each shape
Save/load data	SaveTo/LoadForm (StreamWriter/StreamReader)	Used for saving/loading shapes

MyRectangle details

Responsibility	Type Details	Notes
Represents rectangles	int Width, int Height	Inherits from Shape
Draws itself	Override Draw(), DrawOutline()	
Detects mouse clicks	Override IsAt(Point2D pt)	
Save/load data	Override SaveTo/LoadFrom	

MyCircle details

Responsibility	Type Details	Notes
Represents circles	Int Radius	Inherits from Shape
Draw itself	Override Draw(), DrawOutline()	
Detects mouse clicks	Override IsAt(Point2D pt)	
Save/load data	Override SaveTo/LoadFrom	

MyLine details

Responsibility	Type Details	Notes
Represents lines	float EndX, float EndY	Inherits from Shape
Draws itself	Override Draw(), DrawOutline()	
Detects mouse clicks	Override IsAt(Point2D pt)	

Drawing details

Responsibility	Type Details	Notes
Stores all shapes	List<Shape> _shapes	Manages adding/removing
Background color	Color _background	Can change color

Shape selection	SelectShapesAt(Point2D pt)	Used for selecting shapes
Save/load drawing	Save(string), Load(string)	All shapes/background
Draws everything	Draw()	Draws all shapes/background

ExtensionMethods details

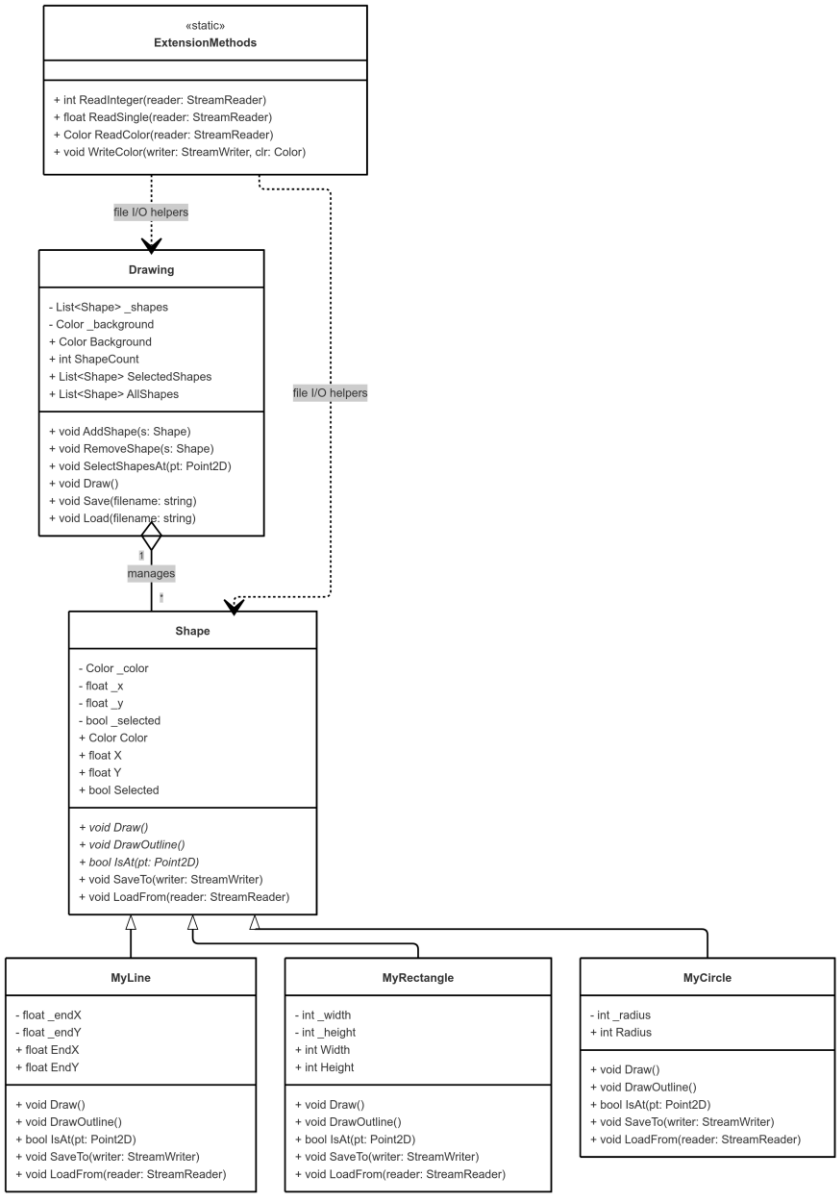
Responsibility	Type Details	Notes
Helper methods for files	ReadInteger, ReadSingle, ReadColor	Extension methods for Stream
	WriteColor	Used for color/numeric IO

Enum Details

ShapeKind (enum in Program.cs) details

Value	Notes
Rectangle	Used for choosing rectangle tool
Circle	Used for choosing circle tool
Line	Used for choosing line tool

Class Diagram

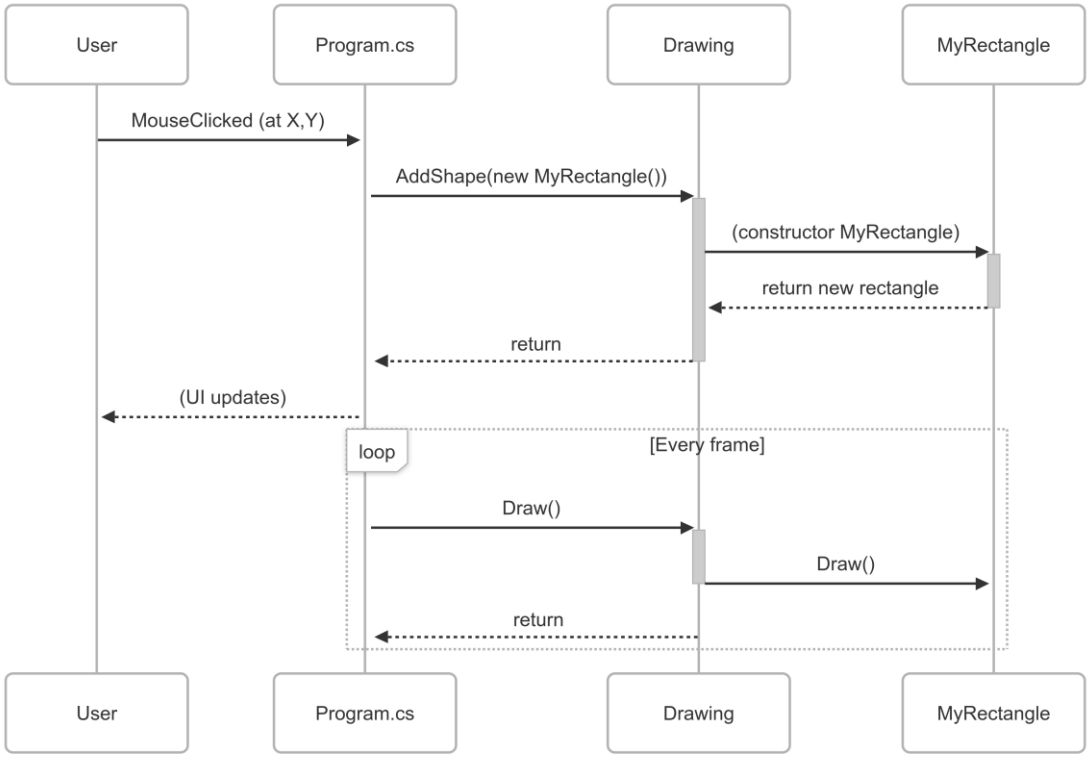


The class diagram for the Shape Drawer program is built around a main parent class called Shape. This class lays out the basic features that every shape in the program will have—like its color, where it sits on the canvas (its x and y position), and whether it's currently selected. It keeps these details tucked away in private fields but gives you easy access to them through public properties. The Shape class also sets out some rules by declaring abstract methods for drawing the shape, drawing an outline (for things like selection), and checking if a certain point is inside the shape. These methods are basically promises that any class inheriting from Shape will have to keep, by providing their own versions. On top of that, Shape offers built-in ways to save and load a shape's data to and from a file.
From this base, three specific shape classes branch out: MyRectangle, MyCircle, and MyLine. Each one manages its own unique details—rectangles keep track of their width and height, circles store their radius, and lines remember their end coordinates. These classes handle everything related to drawing themselves, showing outlines, selection, and saving or loading their own information. To keep things organized, the program uses a Drawing class as a sort of manager. It holds onto all the shapes, takes care of drawing everything on the screen,

handles selection, and manages file operations for the whole canvas. There's also an ExtensionMethods class, which groups together handy helper methods for reading and writing data, like numbers and colors, when saving or loading files. Altogether, this setup makes the program flexible, organized, and easy to work with.

Sequence Diagram

Adding a Shape :



The sequence diagram gives a step-by-step look at what happens behind the scenes when you use the Shape Drawer program. When you click on the canvas, the main program immediately picks up this action and starts the process of creating a new shape—like a rectangle. It asks the Drawing object to add this shape to its collection. The Drawing class takes care of building the new shape, stores it with the others, and then lets the main program know that it's done. As a result, the interface updates and the new shape appears right where you clicked.

From there, the program keeps everything up to date by constantly redrawing all the shapes on the screen. Every time the window refreshes (which happens many times per second), the main program tells the Drawing object to draw everything it manages. The Drawing object, in turn, calls the draw method on each individual shape, so all shapes get rendered and any changes are immediately visible. This back-and-forth makes the app feel interactive and responsive, giving you instant feedback as you add, select, or move shapes around.

Conclusion

All in all, the Shape Drawer program is a great example of how using object-oriented design can make a project both organized and easy to build on. By giving every shape a common “family” to belong to, and letting each type of shape handle its own details, the code stays neat and makes sense—even as you add more features. The Drawing class quietly takes care of managing all the shapes and making sure everything shows up properly for the user, while helper functions handle the nitty-gritty of saving and loading files. This way, the program isn’t just functional—it’s also flexible, letting you add new shapes or features without a headache. In the end, it gives users a smooth and simple way to play with shapes and create their own digital drawings.