Drawing.cs

```
using System.Collections.Generic;
using SplashKitSDK;
using System.IO;
using System.IO.Enumeration;

namespace ShapeDrawer
{
    public class Drawing
    {
        private readonly List<Shape> _shapes;
        private Color _background;
        private StreamWriter writer;
        private Shape s;

        private StreamReader reader;
        private int count;
        private string kind;



        // Public property to access the background color
        public Color Background
        {
            get { return _background; }
            set { _background = value; }
        }

        public void Save(string filename)
        {
            // Step 1: Assign writer, a new StreamWriter passing in filename
```

```
        using (StreamWriter writer = new StreamWriter(filename))
        {
            // Step 2: Tell writer to WriteColor, passing in Background
            writer.WriteLine(Background.ToString());

            // Step 3: Tell writer to WriteLine, passing in ShapeCount
            writer.WriteLine(_shapes.Count);

            // Step 4: For each Shape s in _shapes
            foreach (Shape s in _shapes)
            {
                // Step 5: Tell s to SaveTo writer
                s.SaveTo(writer);
            }

            // Step 6: Tell writer to Close
            // This is automatically handled by the 'using' statement which disposes of the
StreamWriter
        }
    }

    public Drawing(Color background)
    {
        _shapes = new List<Shape>();
        _background = background;
    }

    public void Load(string filePath)
    {
        reader = new StreamReader(filePath);
        Background =  reader.ReadColor();
```

```
    count = reader.ReadInteger();
    _shapes.Clear();

    for (int i = 0;  i < count; i++)
    {
        reader.ReadLine();
        if (kind == "Rectangle")
        {
            s = new MyRectangle();
        }
        if (kind == "Circle")
        {
            s = new MyCircle();
        }
        else
        {
            continue;
        }

        s.LoadFrom(reader);
        _shapes.Add(s);

        reader.Close();
    }
}



public Drawing() : this(Color.White) { } // Default constructor

public void AddShape(Shape shape)
{
```

```
    _shapes.Add(shape);
}

public void RemoveShape(Shape shape)
{
    _shapes.Remove(shape);
}

public void Draw()
{
    SplashKit.ClearScreen(_background);
    foreach (Shape shape in _shapes)
    {
        shape.Draw();
    }
}

public void SelectShapesAt(Point2D pt)
{
    foreach (Shape s in _shapes)
    {
        if (s.IsAt(pt))
        {
            s.Selected = true;
        }
        else
        {
            s.Selected = false;
        }
    }
}
```

```
        public List<Shape> SelectedShapes
        {
            get
            {
                List<Shape> result = new List<Shape>();
                foreach (Shape s in _shapes)
                {
                    if (s.Selected)
                    {
                        result.Add(s);
                    }
                }
                return result;
            }
        }
    }
}
```

ExtensionMethod.cs

```
using System;
using System.IO;
using SplashKitSDK;

namespace ShapeDrawer
{
    public static class ExtensionMethods
    {
        public static int ReadInteger(this StreamReader reader)
        {
            return Convert.ToInt32(reader.ReadLine());
```

```
        }
        public static float ReadSingle(this StreamReader reader)
        {
            return Convert.ToSingle(reader.ReadLine());
        }




        public static Color ReadColor(this StreamReader reader)
        {
            return Color.RGBColor(reader.ReadSingle(), reader.ReadSingle(),
            reader.ReadSingle());
        }

        public static void WriteColor(this StreamWriter writer, Color clr)
        {
            writer.WriteLine("{0}\n{1}\n{2}", clr.R, clr.G,clr.B);
        }



    }
}
```

MyCircle.cs

```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class MyCircle : Shape
    {
        private int _radius;
```

```
// Property for Radius
public int Radius
{
    get { return _radius; }
    set { _radius = value; }
}

public override void LoadFrom(StreamReader reader)
{
    base.LoadFrom(reader);
    Radius = reader.ReadInteger();
}



// Default constructor initializes with Color.Blue and Radius 50
public MyCircle() : base(Color.Blue)
{
    _radius = 50;  // Default radius
}



// Constructor that accepts color and radius
public MyCircle(SplashKitSDK.Color color, int radius,float x, float y) : base(color)
{
    _radius = radius;
    X = x;
    Y = y;
}

// Override Draw method
public override void Draw()
```

```
    {
        SplashKit.FillCircle(Color, X, Y, _radius);

        if (Selected)
        {
            DrawOutline(); // Draw outline if selected
        }
    }

    // Override DrawOutline method
    public override void DrawOutline()
    {
        SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
    }

    public override void SaveTo(StreamWriter writer)
    {
        base.SaveTo(writer);
        writer.WriteLine("Circle");
        writer.WriteLine(Radius);
        writer.WriteLine($"{(int)(Color.R * 255)},{(int)(Color.G * 255)},{(int)(Color.B * 255)}");

    }

    // Override IsAt method to check if a point is within the circle
    public override bool IsAt(Point2D pt)
    {
        return SplashKit.PointInCircle(pt, SplashKit.CircleAt(X, Y, _radius));
    }
}
```

```
}
```

MyLine.cs

```
using SplashKitSDK;
using System;

namespace ShapeDrawer
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;




        // Constructor with parameters for color and coordinates
        public MyLine(Color color, float startX, float startY, float endX, float endY) : base(color)
        {
            X = startX;
            Y = startY;
            _endX = endX;
            _endY = endY;
        }
        // Default constructor initializing color and coordinates
        public MyLine() : this(Color.RandomRGB(255), 0.0f, 0.0f, 100, 100)
        {
        }

        // Property for EndX
        public float EndX
```

```
{
    get { return _endX; }
    set { _endX = value; }
}

// Property for EndY
public float EndY
{
    get { return _endY; }
    set { _endY = value; }
}

// Override Draw to actually draw the line
public override void Draw()
{
    SplashKit.DrawLine(Color, X, Y, _endX, _endY);
}

// Override DrawOutline for the selection
public override void DrawOutline()
{
    if (Selected)
    {
        SplashKit.DrawLine(Color.Black, X - 5, Y - 5, _endX + 5, _endY + 5);
    }
}

public override bool IsAt(Point2D pt)
{
    // Create a Line object from the start and end points
    Line line = SplashKit.LineFrom(X, Y, _endX, _endY);
```

```
            // Use the PointOnLine method, which expects a Line object
            return SplashKit.PointOnLine(pt, line);
        }
        public override void SaveTo(StreamWriter writer)
        {
            base.SaveTo(writer);
            writer.WriteLine("Line");
            writer.WriteLine(EndX);
            writer.WriteLine(EndY);
            writer.WriteLine($"{(int)(Color.R * 255)},{(int)(Color.G * 255)},{(int)(Color.B * 255)}");

        }
        public override void LoadFrom(StreamReader reader)
        {
            Color = reader.ReadColor();
            EndX = reader.ReadInteger();
            EndY = reader.ReadInteger();
        }
    }
}
```

MyRectangle.cs

```
using SplashKitSDK;

namespace ShapeDrawer
{
    public class MyRectangle : Shape
    {
        private int _width, _height;
```

```
// Property for Width
public int Width
{
    get { return _width; }
    set { _width = value; }
}

// Property for Height
public int Height
{
    get { return _height; }
    set { _height = value; }
}

// Default constructor initializes with Color.Green, 0.0f for x and y, and 100 for width and height
public MyRectangle() : this(Color.RandomRGB(255), 0.0f, 0.0f, 100, 100)
{
}

public override void SaveTo(StreamWriter writer)
{
    base.SaveTo(writer);
    writer.WriteLine("Rectangle");
    writer.WriteLine(Width);
    writer.WriteLine(Height);
    writer.WriteLine($"{(int)(Color.R * 255)},{(int)(Color.G * 255)},{(int)(Color.B * 255)}");

}

public override void LoadFrom(StreamReader reader)
{
```

```
    base.LoadFrom(reader);
    Width = reader.ReadInteger();
    Height = reader.ReadInteger();

}
// Constructor that accepts color, x, y, width, and height
public MyRectangle(Color color, float x, float y, int width, int height) : base(color)
{
    X = x;
    Y = y;
    _width = width;
    _height = height;
}

// Override Draw method
public override void Draw()
{
    SplashKit.FillRectangle(Color, X, Y, _width, _height);

    if (Selected)
    {
        DrawOutline(); // Draw outline if selected
    }
}

// Override DrawOutline method
public override void DrawOutline()
{
    SplashKit.DrawRectangle(Color.Black, X - 2, Y - 2, _width + 4, _height + 4);
}
```

```
        // Override IsAt method to check if a point is within the rectangle
        public override bool IsAt(Point2D pt)
        {
            return (pt.X >= X && pt.X <= X + _width && pt.Y >= Y && pt.Y <= Y + _height);
        }
    }
}
```

Program.cs

```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Program
    {
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }

        public static void Main()
        {
            // Default shape to add is Circle
            ShapeKind kindToAdd = ShapeKind.Circle;

            Drawing myDrawing = new Drawing(); // Create a Drawing object using the default constructor
            Window window = new Window("Shape Drawer", 800, 600);

            // Set up the file path for saving/loading
```

```
        string desktopPath = "C:\\Users\\joshu\\OneDrive\\Desktop\\COS20007
OOP\\ShapeDrawer";
        string filePath = System.IO.Path.Combine(desktopPath, "TestDrawing.txt");


        do
        {
            SplashKit.ProcessEvents();
            window.Clear(Color.White); // Clear with a temporary white color


            myDrawing.Draw(); // Draw all shapes


            if (SplashKit.MouseClicked(MouseButton.LeftButton))
            {
                // Create a new shape based on kindToAdd
                Shape newShape = null;  // Initialize as null to handle lines differently


                if (kindToAdd == ShapeKind.Rectangle)
                {
                    // Create a new rectangle
                    newShape = new MyRectangle();
                }
                else if (kindToAdd == ShapeKind.Circle)
                {
                    // Create a new circle
                    newShape = new MyCircle();
                }
                else if (kindToAdd == ShapeKind.Line)
                {
                    // Create a new line
                    float startX = SplashKit.MouseX();
                    float startY = SplashKit.MouseY();
```

```
            // Placeholder end points for the line, can be adjusted based on user input
            newShape = new MyLine(Color.Red, startX, startY, startX + 100, startY + 50);
        }

        if (newShape != null)
        {
            // Set the position of the new shape for non-line shapes
            if (kindToAdd != ShapeKind.Line)
            {
                newShape.X = SplashKit.MouseX();
                newShape.Y = SplashKit.MouseY();
            }

            myDrawing.AddShape(newShape); // Add the new shape to the drawing
        }
    }

    // Change shape to Rectangle if 'R' key is pressed
    if (SplashKit.KeyTyped(KeyCode.RKey))
    {
        kindToAdd = ShapeKind.Rectangle;
    }

    // Save the drawing if 'S' key is pressed
    if (SplashKit.KeyTyped(KeyCode.SKey))
    {
        // Call the Save method of Drawing class
        myDrawing.Save(filePath);
    }

    // Load the drawing if 'O' key is pressed
```

```
            if (SplashKit.KeyTyped(KeyCode.OKey))
            {
                myDrawing.Load(filePath);  // Load the drawing from the file
            }

            // Change shape to Circle if 'C' key is pressed
            if (SplashKit.KeyTyped(KeyCode.CKey))
            {
                kindToAdd = ShapeKind.Circle;
            }

            if (SplashKit.KeyTyped(KeyCode.LKey))
            {
                kindToAdd = ShapeKind.Line;
            }

            // Change the background color to a random color when the space key is pressed
            if (SplashKit.KeyTyped(KeyCode.SpaceKey))
            {
                myDrawing.Background = SplashKit.RandomColor();
            }

            // Select shapes at the current mouse pointer position when the right mouse button is
clicked
            if (SplashKit.MouseClicked(MouseButton.RightButton))
            {
                myDrawing.SelectShapesAt(SplashKit.MousePosition());
            }

            // Remove selected shapes if the delete or backspace key is pressed
            if (SplashKit.KeyTyped(KeyCode.DeleteKey) || SplashKit.KeyTyped(KeyCode.BackspaceKey))
```

```
            {
                foreach (var shape in myDrawing.SelectedShapes)
                {
                    myDrawing.RemoveShape(shape);
                }
            }

            window.Refresh(60);
        } while (!window.CloseRequested);

        window.Close();
    }
  }
}
```

Shape.cs

```
using SplashKitSDK;
using System.IO;
namespace ShapeDrawer
{
    public abstract class Shape
    {
        private Color _color;
        private float _x, _y,_width,_height;
        private bool _selected;

        public virtual void SaveTo(StreamWriter writer)
        {
            writer.WriteLine(_color);
            writer.WriteLine(X);
            writer.WriteLine(Y);
        }
```

```
public virtual void LoadFrom(StreamReader reader)
{
    Color = reader.ReadColor();
    X = reader.ReadInteger();
    Y = reader.ReadInteger();
}

// Property for Color
public Color Color
{
    get { return _color; }
    set { _color = value; }
}

// Property for X coordinate
public float X
{
    get { return _x; }
    set { _x = value; }
}

// Property for Y coordinate
public float Y
{
    get { return _y; }
    set { _y = value; }
}

public float Width //call and intialize the variable
{
```

```
    get { return _width; } //get and store the width
    set { _width = value; }
}
public float Height //call and intialize the variable
{
    get { return _height; } //get and store the height
    set { _height = value; }
}



// Property for Selected
public bool Selected
{
    get { return _selected; }
    set { _selected = value; }
}

// Default constructor that initializes with Color.Yellow
public Shape() : this(Color.Yellow)
{
}

// Constructor that accepts color as a parameter
public Shape(Color color)
{
    _color = color;
}

// Empty method bodies for Draw and DrawOutline
public virtual void Draw() { }
```

```
        public virtual void DrawOutline() { }

        // Virtual IsAt method returns false
        public virtual bool IsAt(Point2D pt)
        {
            return false;
        }
    }
}
```

testDrawing.txt

```
SplashKitSDK.Color
18
SplashKitSDK.Color
36
30
Circle
50
0,0,255
SplashKitSDK.Color
30
125
Circle
50
0,0,255
SplashKitSDK.Color
24
202
Circle
50
0,0,255
```

SplashKitSDK.Color

22

311

Circle

50

0,0,255

SplashKitSDK.Color

35

428

Circle

50

0,0,255

SplashKitSDK.Color

28

538

Circle

50

0,0,255

SplashKitSDK.Color

155

22

Rectangle

100

100

146,78,105

SplashKitSDK.Color

171

143

Rectangle

100

100

208,218,16

SplashKitSDK.Color

183

302

Rectangle

100

100

19,101,5

SplashKitSDK.Color

185

416

Rectangle

100

100

177,66,204

SplashKitSDK.Color

207

563

Rectangle

100

100

9,157,65

SplashKitSDK.Color

390

62

Line

490

112

255,0,0

SplashKitSDK.Color

396

181

Line

496

231

255,0,0

SplashKitSDK.Color

401

285

Line

501

335

255,0,0

SplashKitSDK.Color

406

397

Line

506

447

255,0,0

SplashKitSDK.Color

418

509

Line

518

559

255,0,0

SplashKitSDK.Color

429

461

Line

529

511

255,0,0

SplashKitSDK.Color

410

570

Line

510

620

255,0,0