Selected files

8 printable files

```
Week_5/5.3/DrawingProgram/Drawing.cs
Week_5/5.3/DrawingProgram/ExtensionMethods.cs
Week_5/5.3/DrawingProgram/MyCircle.cs
Week_5/5.3/DrawingProgram/MyLine.cs
Week_5/5.3/DrawingProgram/Program.cs
Week_5/5.3/DrawingProgram/MyRectangle.cs
Week_5/5.3/DrawingProgram/Shape.cs
Week_5/5.3/DrawingProgram/TestDrawing.txt
```

Week_5/5.3/DrawingProgram/Drawing.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using SplashKitSDK;
using System.Linq;
using System.Threading.Tasks;

namespace DrawingProgram
{
    public class Drawing
    {
        private readonly List<Shape> _shapes;
        private Color _background;

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

        public Drawing() : this(Color.White)
        {
        }

        // Properties
        public Color Background
        {
            get { return _background; }
            set { _background = value; }
        }

        public int ShapeCount
        {
            get { return _shapes.Count; }
        }

```

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

        // Methods
        public void AddShape(Shape s)
        {
            _shapes.Add(s);
        }

        public void RemoveShape(Shape s)
        {
            _ = _shapes.Remove(s);
        }

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

        public void SelectShapesAt(Point2D pt)
        {
            foreach (Shape s in _shapes)
            {
                s.Selected = s.IsAt(pt);
            }
        }

        public void RemoveSelectedShapes()
        {
            foreach (Shape s in SelectedShapes)
            {
                RemoveShape(s);
            }
        }

        public void Save(string filename)
        {
            StreamWriter writer = new StreamWriter(filename);

            try
            {
                writer.WriteColor(Background);
                writer.WriteLine(ShapeCount);

                foreach (Shape s in _shapes)
                {
                    s.SaveTo(writer);
                }
            }
            finally
            {
                writer.Close();
            }
        }

        public void Load(string filename)
        {
            StreamReader reader = new StreamReader(filename);
            try
            {

                Background = reader.ReadColor();
                int count = reader.ReadInteger();
                _shapes.Clear();

                for (int i = 0; i < count; i++)
                {
                    string kind = reader.ReadLine()!;
                    Shape s;

                    switch (kind)
                    {
                        case "Rectangle":
                            s = new MyRectangle();
                            break;
                        case "Circle":
                            s = new MyCircle();
                            break;
                        case "Line":
                            s = new MyLine();
                            break;
                        default:
                            throw new InvalidDataException("Unknown shape kind: " +
kind);
                    }
```

```csharp

                    s.LoadFrom(reader);
                    AddShape(s);
                }
            }
            finally
            {
                reader.Close();
            }
        }
    }
}
```

Week_5/5.3/DrawingProgram/ExtensionMethods.cs

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

namespace DrawingProgram
{
    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);
        }
    }
}
```

Week_5/5.3/DrawingProgram/MyCircle.cs

```csharp
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

using SplashKitSDK;

```

```csharp
namespace DrawingProgram
{
    public class MyCircle : Shape
    {
        private float _radius;

        public MyCircle() : this(Color.Blue, 0.0f, 0.0f, 129)
        {
        }

        public MyCircle(Color color, float x, float y, float radius) : base(color)
        {
            X = x;
            Y = y;
            Radius = radius;
        }

        public float Radius
        {
            get => _radius;
            set => _radius = value;
        }

        public override void Draw()
        {
            SplashKit.FillCircle(Color, X, Y, _radius);

            if (Selected)
            {
                DrawOutline();
            }
        }

        public override bool IsAt(Point2D pt)
        {
            Circle c = SplashKit.CircleAt(X, Y, _radius);
            return SplashKit.PointInCircle(pt, c);
        }

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

        public override void SaveTo(StreamWriter writer)
        {
            writer.WriteLine("Circle");
            base.SaveTo(writer);
            writer.WriteLine(Radius);
        }

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

Week_5/5.3/DrawingProgram/MyLine.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using SplashKitSDK;

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

        public MyLine() : this(Color.Red, 0.0f, 0.0f, 88, 88)
        {
        }

        public MyLine(Color color, float x, float y, float endX, float endY) : base(color)
        {
            X = x;
            Y = y;
            EndX = endX;
            EndY = endY;
        }

        public float EndX
        {
            get => _endX;
            set => _endX = value;
        }

        public float EndY
        {
            get => _endY;
            set => _endY = value;
        }

        public override void Draw()
        {
            SplashKit.DrawLine(Color, X, Y, EndX, EndY);

            if (Selected)
            {
                DrawOutline();
            }
        }

        public override bool IsAt(Point2D pt)
        {
            return SplashKit.PointOnLine(pt, SplashKit.LineFrom(X, Y, EndX, EndY));
        }

        public override void DrawOutline()
        {
            SplashKit.DrawCircle(Color.Black, X, Y, 5);
            SplashKit.DrawCircle(Color.Black, EndX, EndY, 5);
        }

        public override void SaveTo(StreamWriter writer)
        {
            writer.WriteLine("Line");
            base.SaveTo(writer);
            writer.WriteLine(EndX);
            writer.WriteLine(EndY);
        }

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

Week_5/5.3/DrawingProgram/Program.cs

```
using System;
using SplashKitSDK;

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

        public static void Main()
        {
```

```csharp
            Window window = new Window("Shape Drawer", 800, 600);
            Drawing myDrawing = new Drawing();

            ShapeKind kindToAdd = ShapeKind.Circle;

            // My ID: 104844794 => Last digit: 4
            // So I'm only able to draw a maximum of X lines within the timeframe
            int maxLines = 4;

            do
            {
                SplashKit.ProcessEvents();
                SplashKit.ClearScreen();

                if (maxLines <= 0 && kindToAdd == ShapeKind.Line)
                {
                    kindToAdd = ShapeKind.Circle;
                }

                // If the user presses the L key and has lines left to draw, they will draw lines
                if (SplashKit.KeyTyped(KeyCode.LKey) && maxLines > 0)
                {
                    kindToAdd = ShapeKind.Line;
                }

                if (SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }

                // If the user presses the C key or has run out of lines to draw, they will draw circles
                if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }

                if (SplashKit.KeyTyped(KeyCode.SKey))
                {
                    myDrawing.Save("TestDrawing.txt");
                }

                if (SplashKit.KeyTyped(KeyCode.OKey))
                {
                    try
                    {
                        myDrawing.Load("TestDrawing.txt");
                    }
                    catch (Exception e)
                    {
                        Console.Error.WriteLine("Error loading file: {0}", e.Message);
                    }
                }

                if (SplashKit.MouseClicked(MouseButton.LeftButton))
                {
                    Shape myShape;

                    switch (kindToAdd)
                    {
                        case ShapeKind.Circle:
                            myShape = new MyCircle();
                            break;
                        case ShapeKind.Line:
                            myShape = new MyLine();
                            maxLines--;
                            break;
                        default:
                            myShape = new MyRectangle();
                            break;
                    }

                    myShape.X = SplashKit.MouseX();
                    myShape.Y = SplashKit.MouseY();
                    myDrawing.AddShape(myShape);
                }

                if (SplashKit.KeyTyped(KeyCode.SpaceKey))
                {
                    myDrawing.Background = SplashKit.RandomRGBColor(255);
                }

                if (SplashKit.MouseClicked(MouseButton.RightButton))
                {
                    Point2D pt = new Point2D();
                    pt.X = SplashKit.MouseX();
                    pt.Y = SplashKit.MouseY();

                    myDrawing.SelectShapesAt(pt);
                }

                if (SplashKit.KeyTyped(KeyCode.DeleteKey) || SplashKit.KeyTyped(KeyCode.BackspaceKey))
                {
                    myDrawing.RemoveSelectedShapes();
                }

                myDrawing.Draw();
                SplashKit.RefreshScreen();
            } while (!window.CloseRequested);
```

```
        }
    }
}

```

Week_5/5.3/DrawingProgram/MyRectangle.cs

```
using System;
using System.IO;
using System.Collections.Generic;
using SplashKitSDK;
using System.Linq;
using System.Threading.Tasks;

namespace DrawingProgram
{
    public class MyRectangle : Shape
    {
        private float _width, _height;

        public MyRectangle() : this(Color.Green, 0.0f, 0.0f, 194, 194)
        {
        }

        public MyRectangle(Color color, float x, float y, float width, float height)
: base(color)
        {
            X = x;
            Y = y;
            Width = width;
            Height = height;
        }

        public float Width
        {
            get => _width;
            set => _width = value;
        }

        public float Height
        {
            get => _height;
            set => _height = value;
        }

        // Methods
        public override void Draw()
        {
            SplashKit.FillRectangle(Color, X, Y, Width, Height);

            if (Selected)
            {
```

```csharp
                DrawOutline();
            }
        }

        public override bool IsAt(Point2D pt)
        {
            return (pt.X >= X) && (pt.X <= (X + _width))
                && (pt.Y >= Y) && (pt.Y <= (Y + _height));
        }

        public override void DrawOutline()
        {
            SplashKit.DrawRectangle(Color.Black, X - 9, Y - 9, Width + 18, Height + 18);
        }

        public override void SaveTo(StreamWriter writer)
        {
            writer.WriteLine("Rectangle");
            base.SaveTo(writer);
            writer.WriteLine(Width);
            writer.WriteLine(Height);
        }

        public override void LoadFrom(StreamReader reader)
        {
            base.LoadFrom(reader);
            Width = reader.ReadInteger();
            Height = reader.ReadInteger();
        }
    }
}
```

Week_5/5.3/DrawingProgram/Shape.cs

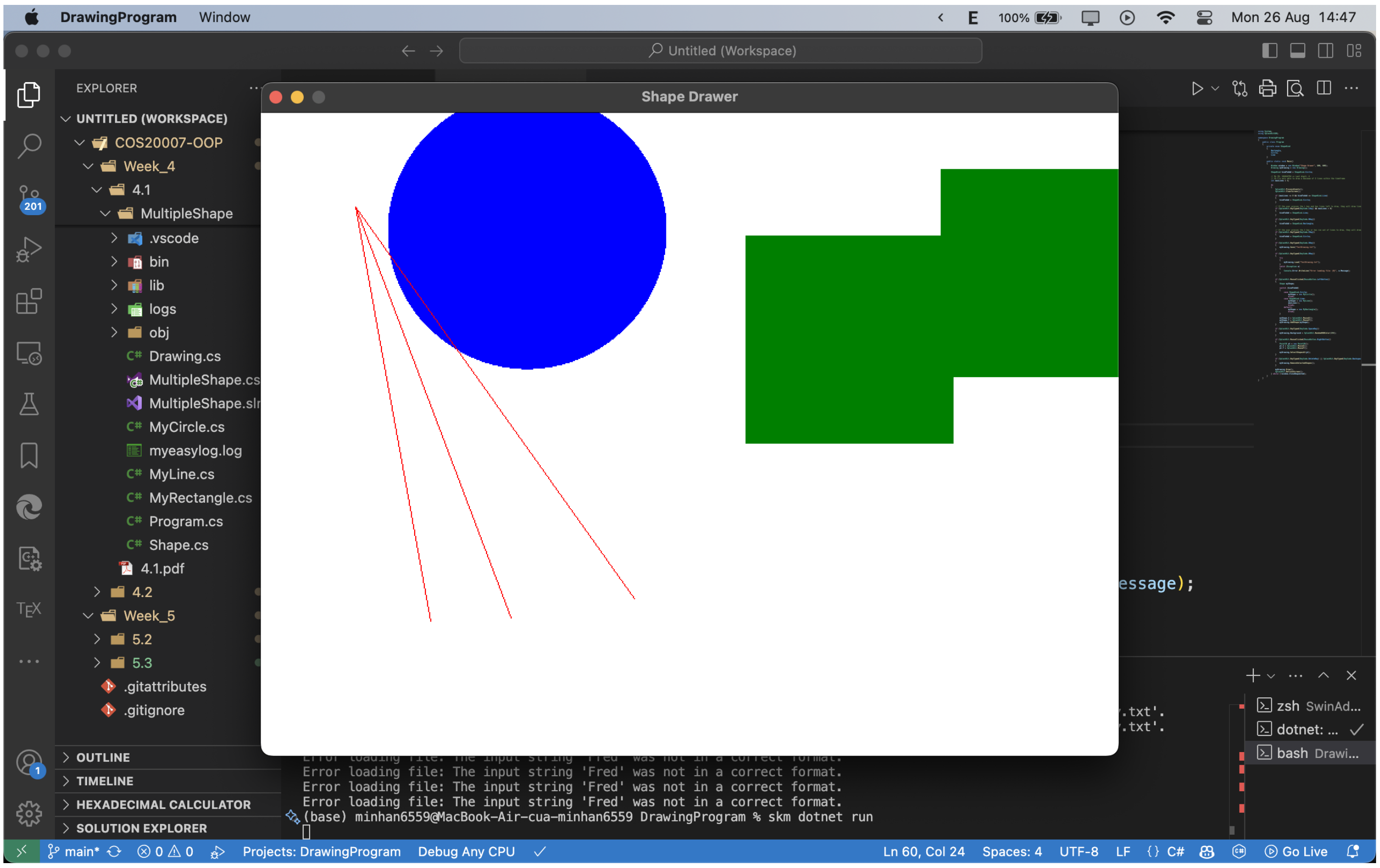
```csharp
using SplashKitSDK;

namespace DrawingProgram
{
    public abstract class Shape
    {
        // Fields
        private Color _color;
        private float _x, _y;
        private bool _selected;

        // Constructor
        public Shape() : this(Color.Yellow)
        {
        }

        public Shape(Color color)
        {
            _color = color;
            _x = 0.0f;
            _y = 0.0f;
            _selected = false;
        }

        // Properties
        public float X
        {
            get { return _x; }
            set { _x = value; }
        }

        public float Y
        {
            get { return _y; }
            set { _y = value; }
        }

        public Color Color
        {
            get { return _color; }
            set { _color = value; }
        }

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

        // Methods
        public abstract void Draw();
        public abstract void DrawOutline();
        public abstract bool IsAt(Point2D pt);
        public virtual void SaveTo(StreamWriter writer)
        {
            writer.WriteColor(Color);
            writer.WriteLine(X);
            writer.WriteLine(Y);
        }

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

    }
```

```
}
```

Week_5/5.3/DrawingProgram/TestDrawing.txt

```
1
1
1
6
Circle
0
0
1
248
110
129
Rectangle
0
0.5
0
452
115
194
194
Rectangle
0
0.5
0
634
53
194
194
Line
1
0
0
158
474
88
88
Line
1
0
0
233
471
88
88
Line
1
0
0
348
453
88
88
```

Screenshot of running program:



```
Error loading file: The input string 'Fred' was not in a correct format.
Error loading file: The input string 'Fred' was not in a correct format.
Error loading file: The input string 'Fred' was not in a correct format.
(base) minhan6559@MacBook-Air-cua-minhan6559 DrawingProgram % skm dotnet run
```