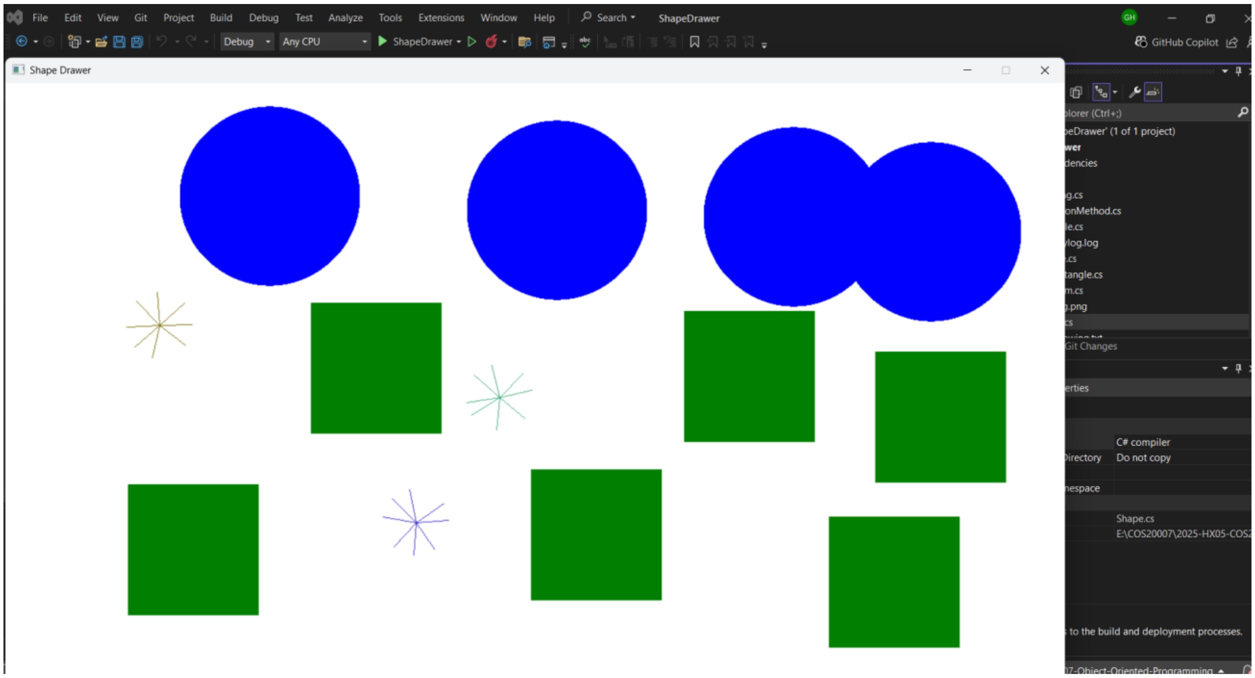

1

1

1

34

Circle

0

0

1

319

136

108

Circle

0

0

1

666

153

108

Circle

0

0

1

952

161

108

Circle

0

0

1

1118

179

108

Rectangle

0

0.49803922

0

1051

324

158

158

Rectangle

0

0.49803922

0

635

466

158

158

Rectangle

0

0.49803922

0

369

265

158

158

Rectangle

0

0.49803922

0

148

484

158

158

Rectangle

0

0.49803922

0

995

523

158

158

Rectangle

0

0.49803922

0

820

275

158

158

Line

0.31202126

0.69472337

0.53532517

597

379

636.0218

370.20804

Line

0.31202126

0.69472337

0.53532517

597

379

627.44183

404.94794

Line

0.31202126

0.69472337

0.53532517

597

379

593.8657

418.877

Line

0.31202126

0.69472337

0.53532517

597

379

563.58136

400.98163

Line

0.31202126

0.69472337

0.53532517

597

379

557.54236

385.56464

Line

0.31202126

0.69472337

0.53532517

597

379

566.9644

352.5829

Line

0.31202126

0.69472337

0.53532517

597

379

587.84485

340.0618

Line

0.31202126

0.69472337

0.53532517

597

379

625.3233

350.75482

Line

0.52394176

0.46763512

0.09219642

186

292

225.99149

291.17422

Line

0.52394176

0.46763512

0.09219642

186

292

217.64127

316.471

Line

0.52394176

0.46763512

0.09219642

186

292

177.89682

331.17062

Line

0.52394176

0.46763512

0.09219642

186

292

156.36446

318.8651

Line

0.52394176

0.46763512

0.09219642

186

292

146.08049

294.53635

Line

0.52394176

0.46763512

0.09219642

186

292

158.43642

263.01294

Line

0.52394176

0.46763512

0.09219642

186

292

183.84325

252.0582

Line

0.52394176

0.46763512

0.09219642

186

292

216.06694

265.6186

Line

0.29389325

0.22861415

0.80684835

496

530

535.9013

527.1915

Line

0.29389325

0.22861415

0.80684835

496

530

518.7751

562.88306

Line

0.29389325

0.22861415

0.80684835

496

530

493.47842

569.9204

Line

0.29389325

0.22861415

0.80684835

496

530

469.14694

559.6465

Line

0.29389325

0.22861415

0.80684835

496

530

456.61722

523.0003

Line

```
0.29389325
0.22861415
0.80684835
496
530
467.99164
501.44247
Line
0.29389325
0.22861415
0.80684835
496
530
488.0253
490.803
Line
0.29389325
0.22861415
0.80684835
496
530
529.20776
507.701
```

```
// ===== Shape.cs =====
using SplashKitSDK;
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using MyGame;

namespace ShapeDrawer
{
    public abstract class Shape
    {
        private Color _color;
        private float _x;
        private float _y;
        private bool _selected;

        public Shape()
        {
            this._color = Color.Yellow;
            this._x = 0.0f;
            this._y = 0.0f;
        }

        public Shape(Color color)
        {
            this._color = color;
            this._x = 0.0f;

            this._y = 0.0f;

        }

        public Color Color

        {

            get { return _color; }

            set { _color = value; }

        }

        public float X

        {

            get { return _x; }

            set { _x = value; }

        }

        public float Y

        {

            get { return _y; }

            set { _y = value; }

        }

        public bool Selected

        {

            get { return _selected; }

            set { _selected = value; }
```

```
        }

        public abstract void DrawOutline();

        public abstract void Draw();

        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.ReadSingle();
            Y = reader.ReadSingle();
        }
    }
}

// ===== MyLine.cs =====
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;
using MyGame;

namespace ShapeDrawer
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;
        private const int ENDPOINT_RADIUS = 5;
        private const float LINE_LENGTH = 40.0f;
        public const int NUM_LINES = 8; // Easy to change number of lines
        private static Random _random = new Random();
        private float _angle; // Store the unique angle for this line

        public float EndX
        {
            get { return _endX; }
            set { _endX = value; }
        }
        public float EndY
        {
            get { return _endY; }
            set { _endY = value; }
        }

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

        public MyLine() : this(Color.Red, 0.0f, 0.0f, 0)
        {
        }

        public MyLine(Color color, float startX, float startY, int lineIndex) : base(color)
        {
            this.X = startX;
            this.Y = startY;

            // Calculate base angle for this line (in radians)
            float baseAngle = (float)(2 * Math.PI * lineIndex / NUM_LINES);

            // Add some random variation to the angle
            this._angle = baseAngle + (float)(_random.NextDouble() * 0.5 - 0.25); // ±0.25 radians variation
```

```
  // Calculate end point using trigonometry
  this._endX = startX + LINE_LENGTH * (float)Math.Cos(_angle);
  this._endY = startY + LINE_LENGTH * (float)Math.Sin(_angle);
}

public override void DrawOutline()
{
  // Draw circles around start and end points
  SplashKit.DrawCircle(Color.Black, X, Y, ENDPOINT_RADIUS);
  SplashKit.DrawCircle(Color.Black, _endX, _endY, ENDPOINT_RADIUS);
}

public override void Draw()
{
  if (Selected)
  {
    DrawOutline();
  }
  SplashKit.DrawLine(Color, X, Y, _endX, _endY);
}

public override bool IsAt(Point2D pt)
{
  Line line = SplashKit.LineFrom(X, Y, _endX, _endY);
  return SplashKit.PointOnLine(pt, line, 10);
}
```

```
        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.ReadSingle();
            _endY = reader.ReadSingle();
        }
    }
}

// ===== Drawing.cs =====
using SplashKitSDK;
using System;
using System.Collections.Generic;
using System.IO;
using MyGame;

namespace ShapeDrawer
```

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

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

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

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

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

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

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

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

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

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

public void Save(string filename)
{
    Console.WriteLine($"Saving drawing to {filename}");
    Console.WriteLine($"Background color: {Background}");
    Console.WriteLine($"Number of shapes: {ShapeCount}");

    StreamWriter writer = new StreamWriter(filename);
    try
```

```
        {
            writer.WriteColor(Background);
            writer.WriteLine(ShapeCount);
            foreach (Shape s in _shapes)
            {
                s.SaveTo(writer);
            }
        }
        finally
        {
            writer.Close();
        }

        Console.WriteLine("Save completed successfully");
    }

    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 = null;

    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);
    }

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

```
            }
        }
    }
}

// ===== Program.cs =====
using SplashKitSDK;
using System;
using System.Threading;

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

        public static void Main()
        {
            Window window = new Window("Shape Drawer", 1280, 720);
            Drawing myDrawing = new Drawing();
            ShapeKind kindToAdd = ShapeKind.Circle;
```

```
Random random = new Random();

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

    if (SplashKit.KeyTyped(KeyCode.RKey))
    {
        kindToAdd = ShapeKind.Rectangle;
    }
    if (SplashKit.KeyTyped(KeyCode.CKey))
    {
        kindToAdd = ShapeKind.Circle;
    }
    if (SplashKit.KeyTyped(KeyCode.LKey))
    {
        kindToAdd = ShapeKind.Line;

    }

    if (SplashKit.MouseClicked(MouseButton.RightButton))
    {
        Shape newShape;
        switch (kindToAdd)
        {
            case ShapeKind.Circle:
              newShape = new MyCircle();
              newShape.X = SplashKit.MouseX();
              newShape.Y = SplashKit.MouseY();
              myDrawing.AddShape(newShape);
              break;
            case ShapeKind.Line:
              // Create all lines at once with random color
              Color lineColor = SplashKit.RandomColor();
              for (int i = 0; i < MyLine.NUM_LINES; i++)
              {
                newShape = new MyLine(lineColor, SplashKit.MouseX(),
SplashKit.MouseY(), i);
                myDrawing.AddShape(newShape);

              }
              //SplashKit.DrawLine(Color.Red, SplashKit.MouseX(), SplashKit.MouseY(),
1280, 720);
              break;
            default:
              newShape = new MyRectangle();
              newShape.X = SplashKit.MouseX() - 79;
              newShape.Y = SplashKit.MouseY() - 79;
              myDrawing.AddShape(newShape);
              break;
          }
        }
        if (SplashKit.MouseClicked(MouseButton.LeftButton))
        {
          myDrawing.SelectShapesAt(SplashKit.MousePosition());
        }

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

        if (SplashKit.KeyTyped(KeyCode.DeleteKey) ||
SplashKit.KeyTyped(KeyCode.BackspaceKey))
        {
          List<Shape> selectedShapes = myDrawing.SelectedShapes;
          foreach (Shape shape in selectedShapes)
          {
            myDrawing.RemoveShape(shape);
          }
        }

        if (SplashKit.KeyTyped(KeyCode.SKey))
        {
          myDrawing.Save("E:/COS20007/2025-HX05-COS20007-Object-Oriented-
Programming/ShapeDrawer_5.3/TestDrawing.txt");
          Console.WriteLine("Drawing saved to TestDrawing.txt");
        }
```

```
                if (SplashKit.KeyTyped(KeyCode.OKey))
                {
                    try
                    {
                        myDrawing.Load("E:/COS20007/2025-HX05-COS20007-Object-Oriented-Programming/ShapeDrawer_5.3/TestDrawing.txt");
                    }
                    catch (Exception e)
                    {
                        Console.Error.WriteLine("Error loading file: {0}", e.Message);
                    }
                }

                myDrawing.Draw();
                SplashKit.RefreshScreen();

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

// ===== MyCircle.cs =====
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;
using MyGame;

namespace ShapeDrawer
{
    public class MyCircle : Shape
    {
        private int _radius;
        public int Radius
        {
            get { return _radius; }
            set { _radius = value; }
        }

        public MyCircle() : this(Color.Blue, 50 + 58)
        {
        }

        public MyCircle(Color color, int radius) : base(color)
        {
            _radius = radius;
        }
```

```
public override void DrawOutline()
{
    SplashKit.DrawCircle(Color.Black, X, Y, _radius + 10);
}

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

public override bool IsAt(Point2D pt)
{
    double dx = pt.X - X;
    double dy = pt.Y - Y;
    return (dx * dx + dy * dy) <= (_radius * _radius);
}

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();
        }
    }
}

// ===== MyRectangle.cs =====
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
using System.IO;
using MyGame;

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

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

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

public MyRectangle() : this(Color.Green, 0.0f, 0.0f, 100 + 58, 100 + 58)
{
}

public MyRectangle(Color color, float x, float y, int width, int height) : base(color)
{
   this._width = width;
   this._height = height;
   this.X = x;
   this.Y = y;
}

public override void DrawOutline()
```

```
{
  SplashKit.DrawRectangle(Color.Black, X - 5, Y - 5, _width + 10, _height + 10);
}

public override void Draw()
{
  if (Selected)
  {
    DrawOutline();
  }
  SplashKit.FillRectangle(Color, X, Y, _width, _height);
}

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

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();
        }
    }
}

// ===== ExtnetionMethod.cs =====
using System;
using System.IO;
using SplashKitSDK;
namespace MyGame
{
    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(clr.R);
      writer.WriteLine(clr.G);
      writer.WriteLine(clr.B);
    }
  }
}
```