Swinburne University of Technology

COS20007 Object Oriented Programming

Drawing Program - Saving and Loading

PDF generated at 14:25 on Monday 2nd October, 2023

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



namespace ShapeDrawer
{
    // the main program class

    public class Program
    {
        // enumerations for the different shapes

        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }

        // the main entry point of the the program

        public static void Main()
        {
            Window window = new Window("Shape Drawer:                                800,
↪   600);
            Drawing mydrawing = new Drawing();
            ShapeKind kindToAdd = ShapeKind.Circle;

            do
            {
                //Process user input

                SplashKit.ProcessEvents();

                // clear the screen

                SplashKit.ClearScreen();


                // check if the left mouse button was clicked

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


                    //Create a new shapes based on the selected kindToAdd

                    if (kindToAdd == ShapeKind.Circle)
                    {
                        MyCircle newCirc = new MyCircle();
```

```
                        newCirc.X = SplashKit.MouseX();
                        newCirc.Y = SplashKit.MouseY();
                        newShape = newCirc;

                    }

                    else if (kindToAdd == ShapeKind.Line)
                    {
                        MyLines newLine = new MyLines();
                        newLine.X = SplashKit.MouseX();
                        newLine.Y = SplashKit.MouseY();
                        newShape = newLine;
                    }
                    else
                    {
                        MyRectangle newRec = new MyRectangle();
                        newRec.X = SplashKit.MouseX();
                        newRec.Y = SplashKit.MouseY();
                        newShape = newRec;
                    }

                    //Add the shape to the drawing

                    mydrawing.AddShape(newShape);
                }


                // check for key events to change the kind of shape to add

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


                // Change the background color if the space key is pressed

                if (SplashKit.KeyDown(KeyCode.SpaceKey))
                {
                    mydrawing.Background = SplashKit.RandomRGBColor(255);
                }


                // Selected shapes at the current mouse position when the right mouse
↪  button is clicked
```

```
                if (SplashKit.MouseClicked(MouseButton.RightButton))
                {
                    mydrawing.SelectShapesAt(SplashKit.MousePosition());
                }


                // Remove selected shapes when the delete key is pressed

                if (SplashKit.KeyDown(KeyCode.DeleteKey) ||
↪   SplashKit.KeyDown(KeyCode.BackspaceKey))
                {
                    foreach (Shape shape in mydrawing.SelectedShapes)
                    {
                        mydrawing.RemoveShape(shape);
                    }
                }




                if (SplashKit.KeyDown(KeyCode.SKey))
                {
                    mydrawing.Save("C:/Users/aarya/Desktop/TestDrawing.txt");
                }


                if (SplashKit.KeyTyped(KeyCode.OKey))
                {
                    try
                    {
                        mydrawing.Load("C:/Users/aarya/Desktop/TestDrawing.txt");
                    }
                    catch (Exception e)
                    {
                        Console.Error.WriteLine("Error loading file: {0}",
↪   e.Message);
                    }
                }

                // draw the shapes and refresh the screen

                mydrawing.Draw();
                SplashKit.RefreshScreen();

                // Continue the loop until the user close the window

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

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


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 color)
        {
            writer.WriteLine("{0}\n{1}\n{2}", color.R, color.G, color.B);
        }
    }
}
```

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

namespace ShapeDrawer
{
    // represents a drawing
    public class Drawing
    {
        private readonly List<Shape> _shapes; // list to store shapes
        private Color _background; // background color of drawing


        // constructor to create a drawing with a specified background color
        public Drawing(Color background)
        {
            _shapes = new List<Shape>();
            _background = background;
        }


        // constructor to create a drawing with a default white background
        public Drawing() : this(Color.White)
        {

        }


        // gets and sets the background color of the drawing
        public Color Background
        {
            get { return _background; }
            set { _background = value; }
        }


        // gets the number of shapes in the drawing
        public int ShapeCount
        {
            get { return _shapes.Count; }
        }


        // gets a list of selected shapes in the drawing

        public List<Shape> SelectedShapes
        {
            get
            {
                List<Shape> result = new List<Shape>();
                foreach (Shape shapez in _shapes)
                {
```

```
                    if (shapez.Selected == true)
                    {
                        result.Add(shapez);
                    }
                }
                return result;
            }
        }


        // adds a shape to the drawing

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


        // clears the screen,sets the background color,and drawing all shapes in the drawing

        public void Draw()
        {
            SplashKit.ClearScreen(Background);

            foreach (Shape shapez in _shapes)
            {
                shapez.Draw();
            }
        }


        //selects shapes at a given point

        public void SelectShapesAt(Point2D pt)
        {
            foreach (Shape shapez in _shapes)
            {
                // check if shape is at point
                if (shapez.IsAt(pt))
                {
                    shapez.Selected = true;
                }
                else
                {
                    shapez.Selected = false;
                }
            }
        }


        // removes all selected shapes from the drawing

```

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

        // edit below


        public void Save(string filename)
        {
            StreamWriter writer = new(filename);
            try
            {
                writer.WriteColor(_background);
                writer.WriteLine(ShapeCount);
                foreach (Shape shape in _shapes)
                {
                    shape.SaveTo(writer);
                }
            }
            finally
            {
                writer.Close();
            }
        }


        public void Load(string filename)
        {
            StreamReader reader = new(filename);
            try
            {
                Shape shapez;
                string kind;
                Background = reader.ReadColor();
                int count = reader.ReadInteger();


                _shapes.Clear();

                for (int i = 0; i < count; i++)
                {
                    kind = reader.ReadLine();
                    switch (kind)
                    {
                        case "Rectangle":
                            shapez = new MyRectangle();
                            break;
                        case "Circle":
                            shapez = new MyCircle();
                            break;
                        case "Line":
                            shapez = new MyLines();
```

```
                            break;
                        default:
                            throw new InvalidDataException("Uknown: " + kind);
                    }
                    shapez.LoadFrom(reader);
                    AddShape(shapez);
                }
            }
            finally
            {
                reader.Close();
            }

        }
    }
}
```

```
using Shapedrawer;
using SplashKitSDK;
using System;
using System.Collections.Generic;

namespace ShapeDrawer
{
    // The abstract base class for all shapes

    public abstract class Shape
    {
        //color the shape

        private Color _color;

        //X-coordinate of shape

        private float _x;

        //Y-coordinate of shape

        private float _y;

        //whether the shape is selected

        private bool _selected;


        // Default constructor for a shape, sets the color to yellow

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

        //Constructor for a shape with a specified color

        public Shape(Color color)
        {
            _color = color;
        }


        // Gets and sets the color of the shape

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


        //gets and sets the X-coordinate of the shape

        public float X
        {
```

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


        // gets and sets the Y-coordinate of the shape

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

        // gets and sets whether the shape is selected

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


        // abstract method to draw the shape on the screen

        public abstract void Draw();

        // abstract method to draw the outline of the shape on the screen

        public abstract void DrawOutline();

        // abstract method for checking if a given point is inside the shape

        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();
        }



    }
```

```
}
```

```
using System;
using SplashKitSDK;
using System.Collections.Generic;
using Shapedrawer;




namespace ShapeDrawer
{
    // My rectangle class represents a rectangle

    public class MyRectangle : Shape
    {
        // Width and height of rectangle

        private int _width, _height;

        // constructor to create a rectangle with a specified width, height and color

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

        // constructor to create a rectangle with a specified width, height and color

        public MyRectangle() : this(Color.Green, 0, 0, 100, 100) { }

        // gets and sets the width of the rectangle

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


        // gets and sets the height of the rectangle

        public int Width
        {
            get { return _width; }
```

```
            set { _width = value; }
        }

        // draws the filled rectangle on the screen and outline if selected

        public override void Draw()
        {
            // draw the filled rectangle on the screen

            SplashKit.FillRectangle(color, X, Y, Width, Height);

            // if the rectangle is selected, draw the outline

            if (Selected)
            {
                DrawOutline();
            }

        }


        // draws the outline of the rectangle on the screen

        public override void DrawOutline()
        {
            // draw a sligly bigger black rectangle on the screen

            SplashKit.DrawRectangle(Color.Black, X - 2, Y - 2, Width + 4, Height + 4);
        }


        // checks if the rectangle is inside the rectangle

        public override bool IsAt(Point2D mouseLocation)
        {
            if (X < mouseLocation.X && mouseLocation.X < (X + Width) && Y < mouseLocation.Y && mouseLocation.Y < (Y + Height))
            {
                // if the point is within the rectangle, return true

                return true;
            }
            else
            {
                // if the point is not within the rectangle, return false

                return false;
            }
        }

        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();
        }

    }
}
```

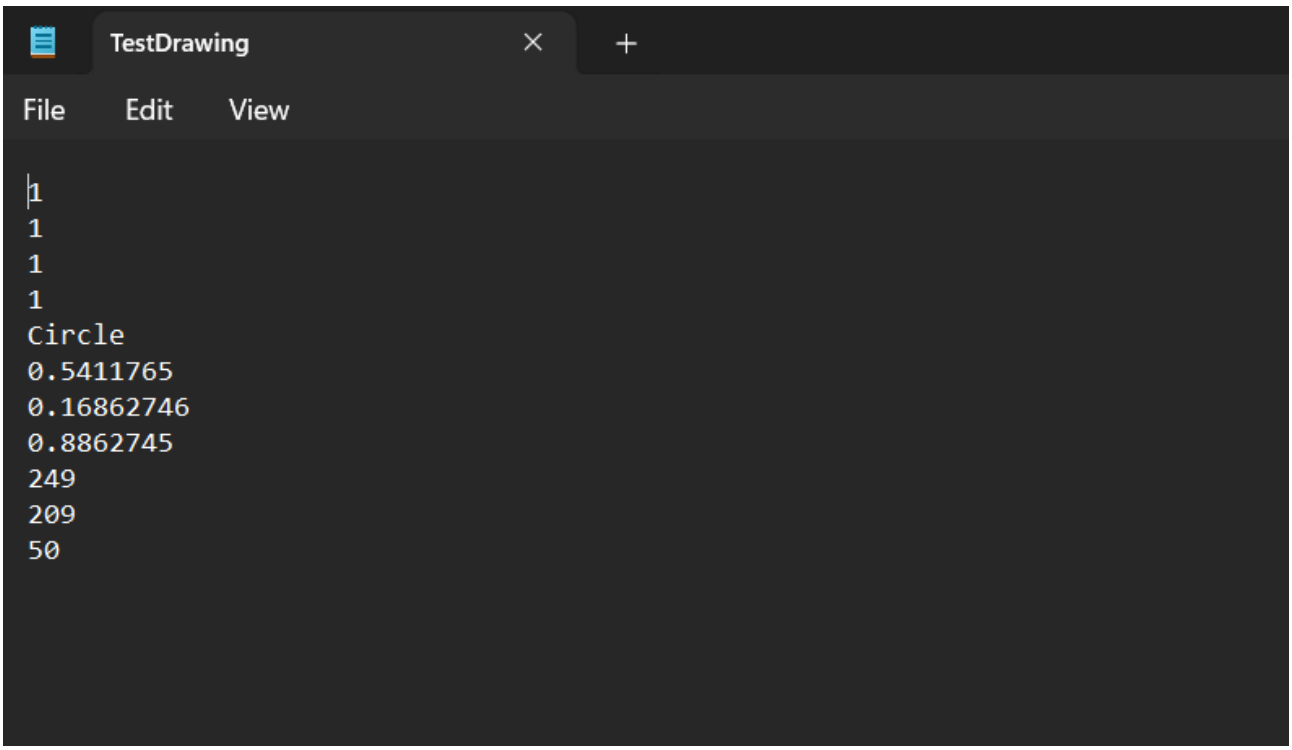
```
using System;
using SplashKitSDK;
using Shapedrawer;
using System.IO;

namespace ShapeDrawer
{
    // represents a circle

    public class MyCircle : Shape
    {
        private int _radius; // radius of circle

        // constructor to create a circle with a specified radius and color

        public MyCircle() : this(Color.BlueViolet, 50) { }

        // constructor to create a circle with a specified radius and color

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

        // gets and sets the radius of the circle

        public int Radius
        {
            get { return _radius; }
            set { _radius = value; }
        }


        // checks if the circle is at a specified point

        public override bool IsAt(Point2D pt)
        {
            // calculate the distance between the point the circle's center

            double hypotenuse = Math.Sqrt(Math.Pow(X - pt.X, 2) + Math.Pow(Y - pt.Y, 2));

            // if the distance is less than the radius, the point is inside the circle

            if (hypotenuse <= Radius)
            {
                return true; // point is inside circle
            }
            else
            {
                return false; // point is outside circle
            }
```

```
        }


        // draws the outline of the circle on the screen

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


        // draws the circle on the screen if selected

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

            // if selected, draw the outline of the circle

            if (Selected)
            {
                DrawOutline();
            }
        }


        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();
        }

    }
}
```

```
using System;
using SplashKitSDK;
using System.Collections.Generic;
using System.Linq;
using Shapedrawer;
//using System.IO;


namespace ShapeDrawer
{
    // MyLines class represents a line shapes

    public class MyLines : Shape
    {
        // coordinates of lines's end point

        private float _endX, _endY;

        // constructor to create a line with a specified end point and color

        public MyLines() : this(Color.Blue, 100, 100, 400, 300) { }

        // constructor to create a line with a specified end point and color

        public MyLines(Color color, float x, float y, float endx, float endy) : base(color)
        {
            X = x;
            Y = y;
            EndX = endx;
            EndY = endy;
        }

        // gets and sets the X-coordinate of the line's end point

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


        // gets and sets the Y-coordinate of the line's end point

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


        // checks if a given point is within of the line

```

```
        public override bool IsAt(Point2D pt)
        {
            // calculate the gradient and intercept of the line

            float gradient = (EndY - Y) / (EndX - X);
            float intercept = EndY - (gradient * EndX);

            // calculate the distance between the point and the line

            double margin = 10;

            // calculate the distance between the point and the line

            double distance = Math.Abs((gradient * pt.X) + intercept - pt.Y);

            // if the distance is less than the margin, the point is on the line

            return distance <= margin;
        }


        // draws the outline of the line on the screen

        public override void DrawOutline()
        {
            SplashKit.DrawCircle(Color.GhostWhite, X, Y, 4);
            SplashKit.DrawCircle(Color.GhostWhite, EndX, EndY, 4);
        }


        // draws the line on the screen if selected

        public override void Draw()
        {
            // draw the line

            SplashKit.DrawLine(color, X, Y, EndX, EndY);

            // if selected, draw the outline of the line

            if (Selected)
            {
                DrawOutline();
            }
        }


        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();
        }

    }
}
```


TestDrawing
File
Edit
View
1
1
1
1
Circle
0.5411765
0.16862746
0.8862745
249
209
50