Swinburne University of Technology

COS20007 Object Oriented Programming

Drawing Program - Multiple Shape Kinds

PDF generated at 22:30 on Tuesday 19th September, 2023

Produced by OOP Submission Formatter

```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public class Program
    {
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }
        public static void Main()
        {
            Drawing myDrawing = new Drawing();
            ShapeKind kindToAdd = ShapeKind.Circle;

            new Window("Drawing Shape", 800, 600);
            do
            {
                SplashKit.ProcessEvents();
                SplashKit.ClearScreen();
                if (SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }
                if (SplashKit.KeyTyped(KeyCode.LKey))
                {
                    kindToAdd = ShapeKind.Line;
                }
                if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }
                if (SplashKit.MouseClicked(MouseButton.LeftButton))
                {
                    Shape newShape;
                    if (kindToAdd == ShapeKind.Circle)
                    {
                        MyCircle newCircle = new MyCircle();

                        newShape = newCircle;

                    }
                    else if (kindToAdd == ShapeKind.Rectangle)
                    {
                        MyRectangle newRect = new MyRectangle();

                        newShape = newRect;
                    }
                    else
                    {
                        MyLine newLine = new MyLine();

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

                if (SplashKit.MouseClicked(MouseButton.RightButton))
                {
                    myDrawing.SelectedShapeAt(SplashKit.MousePosition());
                }

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

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

                myDrawing.Draw();

                SplashKit.RefreshScreen();

            } while (!SplashKit.WindowCloseRequested("Drawing Shape"));
        }
    }
}
```

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

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()
        {
            List<Shape> selectedShapes = new List<Shape>();
            foreach (Shape s in _shapes)
            {
                if (s.Selected)
                {
                    selectedShapes.Add(s);
                }
            }
            return selectedShapes;
        }
        public int ShapeCount
        {
            get
            {
                return _shapes.Count;
            }
        }

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

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

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

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

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

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

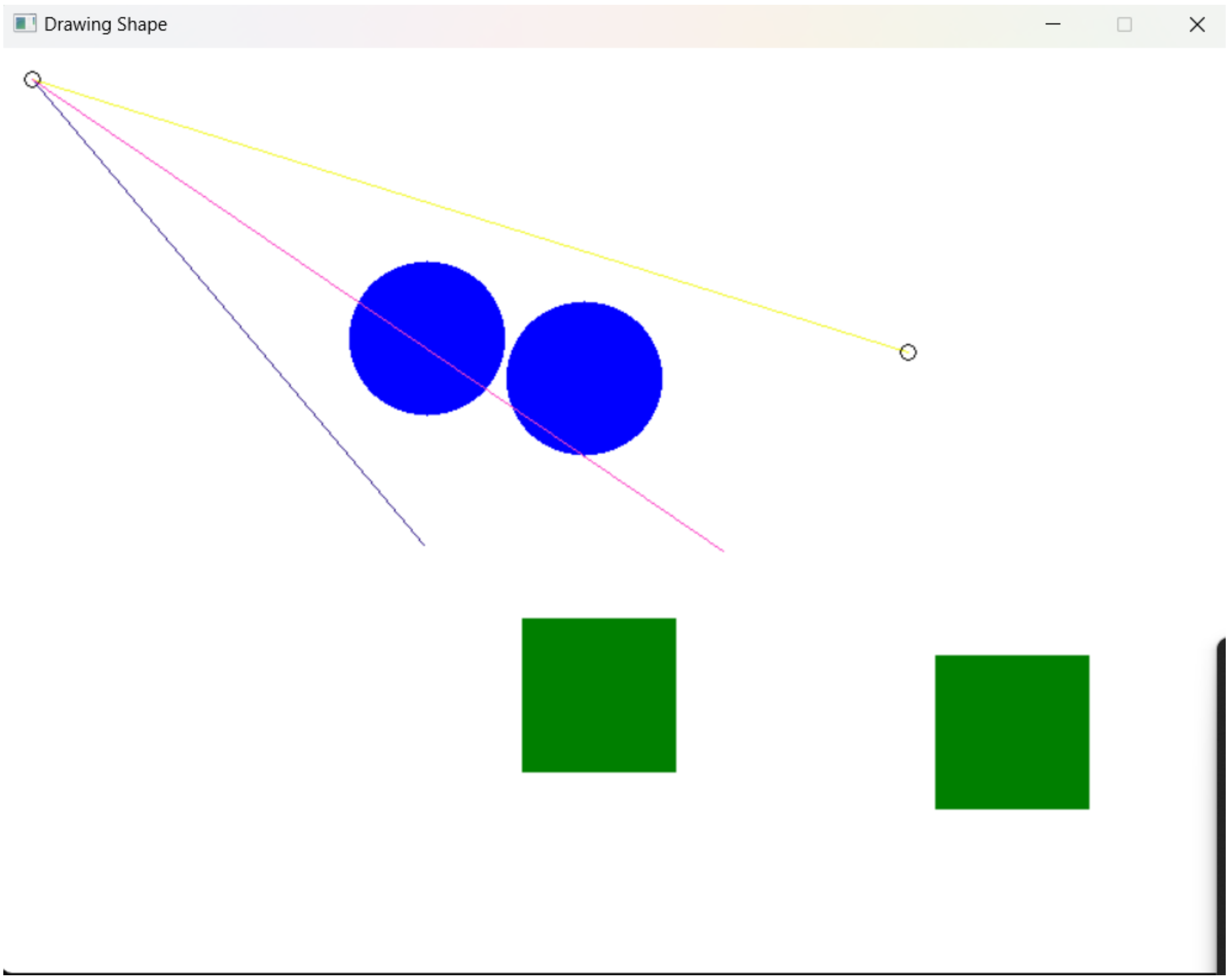
```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public abstract class Shape
    {
        private Color _color;
        private float _x, _y;
        private bool _selected;
        private int _witdh, _height;
        public Shape(Color clr)
        {
            _color = clr;
        }

        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 abstract void Draw();
        public abstract bool IsAt(Point2D p);


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

        public abstract void DrawOutline();


    }
}
```

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

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

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

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

        public int Width // Corrected typo
        {
            get { return _width; }
            set { _width = value; }
        }
        public int Height
        {
            get { return _height; }
            set { _height = value; }
        }
        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.FillRectangle(Color, X, Y, Width, Height);
        }
        public override void DrawOutline()
        {
            SplashKit.FillRectangle(Color, X - 2, Y - 2, Width + 4, Height + 4);
        }
        public override bool IsAt(Point2D p)
        {
            if ((p.X > X) && (p.X < (X + _width)))
            {
                if ((p.Y > Y) && (p.Y < (Y + _height)))
                {
                    return true;
                }
            }
            return false;
        }
    }
}
```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SplashKitSDK;
namespace ShapeDrawer
{
    public class MyCircle : Shape
    {
        private int _radius;

        public MyCircle(Color clr,float x, float y, int radius) : base(clr)
        {
            X = x;
            Y = y;
            _radius = radius;
        }
        public MyCircle() : this(Color.Blue,0,0, 50) { }
        public int Radius { get { return _radius; } }
        public override void Draw()
        {
            if (Selected)
                DrawOutline();
            SplashKit.FillCircle(Color, X, Y, _radius);
        }
        public override void DrawOutline()
        {
            SplashKit.FillCircle(Color, X - 2, Y - 2, _radius + 2);
        }
        public override bool IsAt(Point2D p)
        {
            double a = (double)(p.X - X);
            double b = (double)(p.Y - Y);
            if (Math.Sqrt(a * a + b * b) < _radius)
            {
                return true;
            }
            return false;
        }
    }
}
```

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

namespace ShapeDrawer
{
    public class MyLine : Shape
    {
        private float _endX;
        private float _endY;
        public MyLine(Color clr,float startX, float startY, float endX, float endY) : base(clr)
        {
            _endX = endX;
            _endY = endY;
            X = startX;
            Y = startY;

        }
        public MyLine() : this(Color.RandomRGB(255), 0, 0, 20, 20) { }

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

        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.DrawLine(Color, X, Y, _endX, _endY);
        }
        public override void DrawOutline()
        {
            SplashKit.DrawCircle(Color.Black, X, Y, 5);
            SplashKit.DrawCircle(Color.Black, _endX, _endY, 5);
        }
        public override bool IsAt(Point2D p)
        {
            // Calculate the distance from the point to the line
            double distance = Math.Abs((EndY - Y) * p.X - (EndX - X) * p.Y + EndX * Y - EndY * X)
```

```
                              / Math.Sqrt(Math.Pow(EndY - Y, 2) + Math.Pow(EndX - X,
↪   2));

            // Define a tolerance value for how close the point can be to the line
            double tolerance = 5.0; // Adjust as needed

            // Check if the distance is within the tolerance
            return distance <= tolerance;
        }

    }
}
```


Drawing Shape