Swinburne University of Technology

COS20007 Object Oriented Programming

Drawing Program - Multiple Shape Kinds

PDF generated at 17:59 on Friday 29th September, 2023

```csharp
using System;
using System.Drawing;
using System.Runtime.CompilerServices;
using SplashKitSDK;

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

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

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

                if (SplashKit.MouseClicked(MouseButton.LeftButton))
                {
                    Shape newShape = null;
                    if (kindToAdd == ShapeKind.Circle)
                    {
                        Mycircle newCircle = new Mycircle();
                        newCircle.X = SplashKit.MouseX();
                        newCircle.Y = SplashKit.MouseY();
                        newShape = newCircle;
                    }
                    else if (kindToAdd == ShapeKind.Rectangle)
                    {
                        Myrectangle newRect = new Myrectangle();
                        newRect.X = SplashKit.MouseX();
                        newRect.Y = SplashKit.MouseY();
                        newShape = newRect;
                    }
                    else if (kindToAdd == ShapeKind.Line)
                    {
                        MyLine newLine = new MyLine();
                        newLine.startpoint = new Point2D()
                        {
                            X = SplashKit.MouseX(),
                            Y = SplashKit.MouseY()
                        };
                        newShape = newLine;
                    }

                    if (newShape != null)
                    {
                        drawing.AddShape(newShape);
                    }
                }

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

                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))
                {
                    float x = SplashKit.MouseX();
                    float y = SplashKit.MouseY();
                    Point2D mouseposition = new Point2D()
                    {
                        X = x,
                        Y = y
                    };
                    drawing.SelectShapeAt(mouseposition);
                }

                drawing.Draw();
                SplashKit.RefreshScreen();

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

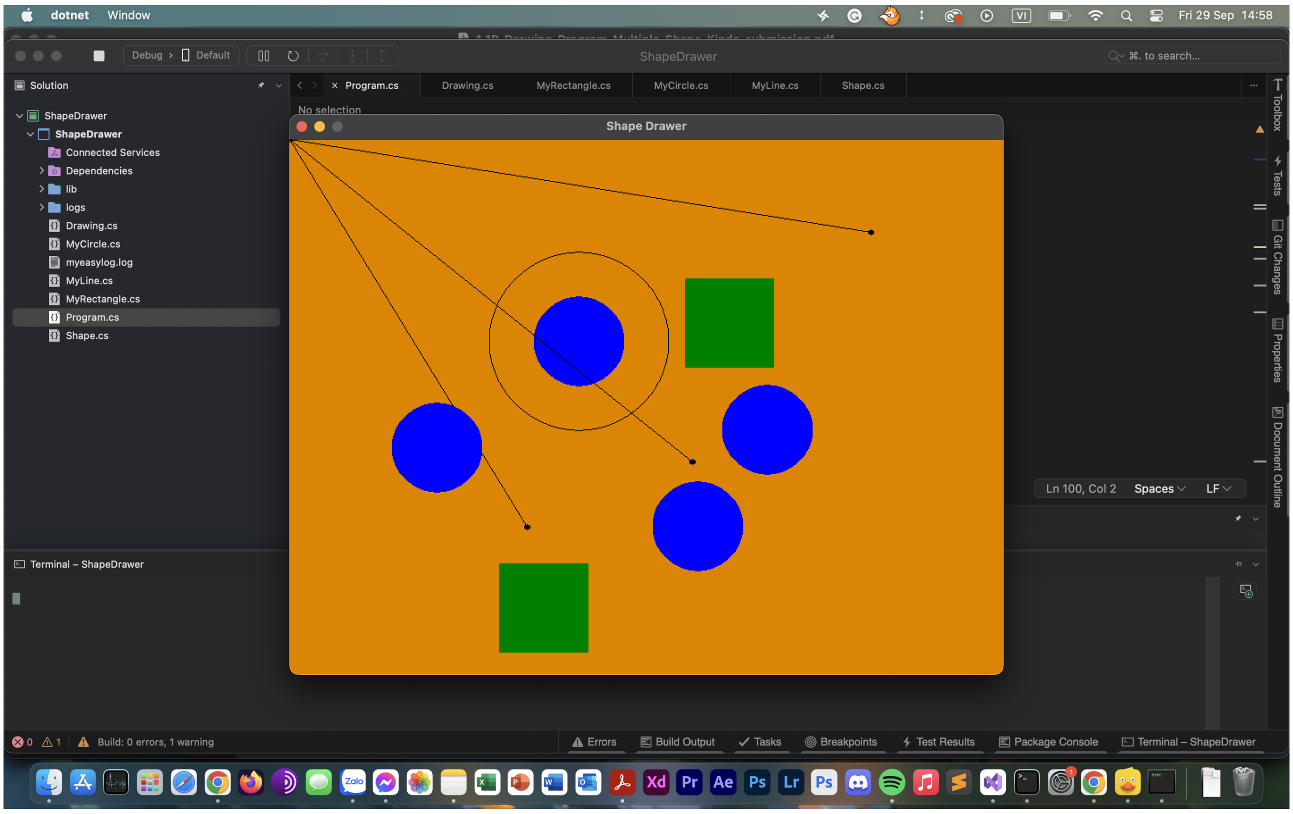
```
using System.Collections.Generic;
using SplashKitSDK;
using Color = SplashKitSDK.Color;

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

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

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

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

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

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

```
using System;
using SplashKitSDK;

namespace ShapeDrawer
{
    public abstract class Shape
    {
        public bool _selected;
        public Color _color { get; set; }

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

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

        public abstract bool IsAt(Point2D pt);

        public void ChangeColor()
        {
            _color = SplashKit.RandomRGBColor(255);
        }
        public bool Selected
        {
            get { return _selected; }
            set { _selected = value; }
        }

        public abstract void Draw();
        public abstract void Drawoutline();
    }
}
```

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

namespace ShapeDrawer
{
    public class Myrectangle : Shape
    {
        private float _width;
        private float _height;
        public float Width
        {
            get { return _width; }
            set { _width = value; }
        }

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

        public float X { get; set; }
        public float Y { get; set; }

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

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

        public override void Draw()
        {
            if (_selected)
            {
                Drawoutline();
            }
            SplashKit.FillRectangle(_color, X, Y, Width, Height);
        }

        public override void Drawoutline()
        {
            float outlineX = X - 2;
            float outlineY = Y - 2;
            float outlineWidth = Width + 4;
            float outlineHeight = Height + 4;
            SplashKit.DrawRectangle(Color.Black, outlineX, outlineY, outlineWidth,
↪   outlineHeight);
        }

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

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

namespace ShapeDrawer
{
    public class Mycircle : Shape
    {
        public float X { get; set; }
        public float Y { get; set; }
        public float _radius { get; set; }

        public Mycircle(Color _color, float x, float y, float radius) : base(_color)
        {
            X = x;
            Y = y;
            _radius = radius;
        }

        public Mycircle() : this(Color.Blue, 0, 0, 50)
        {
        }

        public override void Draw()
        {
            if (_selected)
            {
                Drawoutline();
            }
            SplashKit.FillCircle(_color, X, Y, _radius);
        }

        public override void Drawoutline()
        {
            float outlineX = X - _radius;
            float outlineY = Y - _radius;
            float outlineDiameter = _radius * 2;
            SplashKit.DrawCircle(Color.Black, X, Y, outlineDiameter);
        }

        public override bool IsAt(Point2D pt)
        {
            double distance = System.Math.Sqrt(System.Math.Pow(pt.X - X, 2) +
 ↪  System.Math.Pow(pt.Y - Y, 2));
            return distance <= _radius;
        }
    }
}
```

```
using SplashKitSDK;
using System;

namespace ShapeDrawer
{
    public class MyLine : Shape
    {
        public Point2D startpoint { get; set; }
        public Point2D endpoint { get; set; }

        public MyLine(Point2D startPoint, Point2D endPoint, Color _color) : base(_color)
        {
            startpoint = startPoint;
            endpoint = endPoint;
        }

        public MyLine() : this(new Point2D(), new Point2D(), Color.Black)
        {
        }

        public override void Draw()
        {
            if (_selected)
            {
                Drawoutline();
            }
            SplashKit.DrawLine(_color, startpoint.X, startpoint.Y, endpoint.X, endpoint.Y);
            Drawoutline();
        }

        public override void Drawoutline()
        {
            const int outlineRadius = 3;
            SplashKit.FillCircle(Color.Black, startpoint.X, startpoint.Y, outlineRadius);
            SplashKit.FillCircle(Color.Black, endpoint.X, endpoint.Y, outlineRadius);
        }

        public override bool IsAt(Point2D pt)
        {
            const int tolerance = 2;
            Line line = SplashKit.LineFrom(startpoint, endpoint);
            return SplashKit.PointOnLine(pt, line, tolerance);
        }
    }
}
```


dotnet
Window
Fri 29 Sep 14:58
Debug
Default
ShapeDrawer
Solution
ShapeDrawer
ShapeDrawer
Connected Services
Dependencies
lib
logs
Drawing.cs
MyCircle.cs
myeasylog.log
MyLine.cs
MyRectangle.cs
Program.cs
Shape.cs
Program.cs
Drawing.cs
MyRectangle.cs
MyCircle.cs
MyLine.cs
Shape.cs
No selection
Shape Drawer
Ln 100, Col 2
Spaces
LF
Terminal – ShapeDrawer
Build: 0 errors, 1 warning
Errors
Build Output
Tasks
Breakpoints
Test Results
Package Console
Terminal – ShapeDrawer
Toolbox
Tests
Git Changes
Properties
Document Outline