Swinburne University of Technology

COS20007 Object Oriented Programming

5.3C - Drawing Program - Saving and Loading

PDF generated at 07:46 on Saturday 25th March, 2023

```
using DrawingProgram.lib;

namespace DrawingProgram
{
    public class Program
    {
        // enumeration for kinds of shapes
        private enum ShapeKind
        {
            Rectangle,
            Circle,
            Line
        }
        public static void Main()
        {
            //initialize a variable with shapekind enum to keep track of which shape is currently equipped
            ShapeKind kindToAdd = ShapeKind.Circle;

            //initialized coordinates for drawing a line, each to hold 2 mouse clicks
            float startX = 0;
            float startY = 0;
            float endX = 0;
            float endY = 0;

            Drawing drawing = new Drawing();

            Window window = new Window("Shape Drawer", 800, 600);

            //event loop starts here
            do
            {
                SplashKit.ProcessEvents();
                SplashKit.ClearScreen();

                //change shape kind depending on key pressed
                if (SplashKit.KeyTyped(KeyCode.RKey))
                {
                    kindToAdd = ShapeKind.Rectangle;
                }
                if (SplashKit.KeyTyped(KeyCode.CKey))
                {
                    kindToAdd = ShapeKind.Circle;
                }
                if (SplashKit.KeyTyped(KeyCode.LKey))
                {
                    kindToAdd = ShapeKind.Line;
                }

                // add new shape
                if (SplashKit.MouseClicked(MouseButton.LeftButton))
                {
```

```
52                      //new rectangle
53                      if (kindToAdd == ShapeKind.Rectangle)
54                      {
55                          // make the new shape, assign coordinates and add it to the
    ↪ drawing list
56                          MyRectangle newRect = new MyRectangle();
57                          newRect.X = SplashKit.MouseX();
58                          newRect.Y = SplashKit.MouseY();
59                          drawing.AddShape(newRect);
60                      }
61                      //new circle
62                      else if (kindToAdd == ShapeKind.Circle)
63                      {
64                          MyCircle newCircle = new MyCircle();
65                          newCircle.X = SplashKit.MouseX();
66                          newCircle.Y = SplashKit.MouseY();
67                          drawing.AddShape(newCircle);
68                      }
69                      //new line
70                      else if (kindToAdd == ShapeKind.Line)
71                      {
72                          // check here if mouse has been clicked once or twice. on
    ↪ first click assign start coordinates,
73                          //on 2nd click assign end values and reset both values so
    ↪ they can be used for the next line
74                          if (startX == 0 && startY == 0)
75                          {
76                              startX = SplashKit.MouseX();
77                              startY = SplashKit.MouseY();
78                          }
79                          else if (endX == 0 && endY == 0)
80                          {
81                              endX = SplashKit.MouseX();
82                              endY = SplashKit.MouseY();
83
84                              MyLine newLine = new MyLine();
85                              newLine.X = startX;
86                              newLine.Y = startY;
87                              newLine.X2 = endX;
88                              newLine.Y2 = endY;
89                              drawing.AddShape(newLine);
90
91                              startX = 0;
92                              startY = 0;
93                              endX = 0;
94                              endY = 0;
95                          }
96                      }
97                  }
98
99                  // delete a shape
100                 if (SplashKit.KeyTyped(KeyCode.BackspaceKey) ||
    ↪ SplashKit.KeyTyped(KeyCode.DeleteKey))
```

```
                {
                    drawing.DeleteShape();
                }
                // select a shape
                if (SplashKit.MouseClicked(MouseButton.RightButton))
                {
                    drawing.SelectShapesAt(SplashKit.MousePosition());
                }

                // change background color
                if (SplashKit.KeyTyped(KeyCode.SpaceKey))
                {
                    drawing.Background = Color.Random();
                }

                // save drawing
                if (SplashKit.KeyTyped(KeyCode.SKey))
                {
                    drawing.Save("/Users/shahn/Desktop/TestDrawing.txt");
                }

                //load drawing
                if (SplashKit.KeyTyped(KeyCode.OKey))
                {
                    try
                    {
                        drawing.Load("C:/Users/shahn/Desktop/TestDrawing.txt");
                    }
                    catch (Exception e)
                    {
                        Console.Error.WriteLine($"Error loading file: {e.Message}");
                    }
                }

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

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

```
using System;
using System.IO;
using DrawingProgram.lib;

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);
        }
    }
}
```

```
using DrawingProgram.lib;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        //properties

        //number of shapes in list, readonly
        public int ShapeCount
        {
            get
            {
                return _shapes.Count;
            }
        }

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

        //list of shapes that are currently selected
        public List<Shape> SelectedShapes
        {
            get
            {
                List<Shape> result = new List<Shape>();

                foreach (Shape s in _shapes)
                {
                    if (s.Selected == true)
                    {
                        result.Add(s);
                    }
                }
```

```
54
55                  return result;
56              }
57          }
58
59          //constructer that accepts color as a parameter for the background
60          public Drawing(Color background)
61          {
62              _shapes = new List<Shape>();
63              _background = background;
64          }
65
66          //default constructor
67          public Drawing() : this(Color.White)
68          {
69
70          }
71
72          //methods
73          public void AddShape(Shape shape)
74          {
75              _shapes.Add(shape);
76          }
77
78          public void Draw()
79          {
80              SplashKit.ClearScreen();
81              foreach (Shape shape in _shapes)
82              {
83                  shape.Draw();
84              }
85          }
86
87          public void SelectShapesAt(Point2D pt)
88          {
89              // checks if mouse position is over a shape, if true then its selected
    ↪  property is set to true
90              foreach (Shape s in _shapes)
91              {
92                  if (s.IsAt(pt))
93                  {
94                      s.Selected = true;
95                  }
96                  else
97                  {
98                      s.Selected = false;
99                  }
100             }
101         }
102
103         public void DeleteShape()
104         {
105             // duplicated shapes list with ToList method to read that list instead
    ↪  of the original one
```

```
106             // because a list cannot be modified while being enumerated
107             foreach (Shape s in _shapes.ToList())
108             {
109                 if (s.Selected)
110                 {
111                     _shapes.Remove(s);
112                 }
113             }
114         }
115 
116         public void Save(string filename)
117         {
118             StreamWriter writer = new StreamWriter(filename);
119             try
120             {
121                 writer.WriteColor(_background);
122                 writer.WriteLine(ShapeCount);
123 
124 
125                 foreach (Shape s in _shapes)
126                 {
127                     s.SaveTo(writer);
128                 }
129             }
130             finally
131             {
132                 writer.Close();
133             }
134         }
135 
136         public void Load(string filename)
137         {
138             StreamReader reader = new StreamReader(filename);
139             try
140             {
141                 Shape s;
142                 string kind;
143 
144                 Background = reader.ReadColor();
145                 int count = reader.ReadInteger();
146 
147                 _shapes.Clear();
148 
149                 for (int i = 0; i < count; i++)
150                 {
151                     kind = reader.ReadLine();
152 
153                     switch (kind)
154                     {
155                         case "Rectangle":
156                             s = new MyRectangle();
157                             break;
158                         case "Circle":
```

```
159                             s = new MyCircle();
160                             break;
161                         case "Line":
162                             s = new MyLine();
163                             break;
164                         default:
165                             throw new InvalidDataException("Unknown shape kind: " +
    ↪  kind);
166                     }
167
168                     s.LoadFrom(reader);
169                     AddShape(s);
170                 }
171             }
172             finally
173             {
174                 reader.Close();
175             }
176
177         }
178     }
179 }
```

```
using System;
using DrawingProgram.lib;

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

        // constructor
        public Shape(Color clr)
        {
            _color = clr;
        }
        //default constructor
        public Shape() : this(Color.Yellow)
        {
        }

        // properties
        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;
            }
        }

        // methods
        public abstract void Draw();
        public abstract bool IsAt(Point2D pt);
        public abstract void DrawOutline();

        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 DrawingProgram.lib;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DrawingProgram
{
    //this class requires common references to have their namespaces explicitly mentioned
    // such as Color, Rectangle(not entirely sure why). not sure why other classes do not give the same error
    public class MyRectangle : Shape
    {
        //local variables
        private int _width;
        private int _height;

        //properties
        public int Width
        {
            get
            {
                return _width;
            }
            set
            {
                _width = value;
            }
        }
        public int Height
        {
            get
            {
                return _height;
            }
            set
            {
                _height = value;
            }
        }

        // constructor (not sure yet why this accepts x y coordinates but other shapes dont, but its not being utilised for now)
        public MyRectangle(lib.Color clr,float x, float y, int width, int height) : base( clr)
        {
            X = x;
            Y = y;
            _width = width;
            _height = height;
```

```
50          }
51          //default constructor
52          public MyRectangle() : this(lib.Color.Green, 0, 0, 100, 100) { }
53
54          //methods
55          public override void Draw()
56          {
57              if (Selected)
58              {
59                  DrawOutline();
60              }
61              SplashKit.FillRectangle(Color, X, Y, _width, _height);
62          }
63
64          public override void DrawOutline()
65          {
66              SplashKit.FillRectangle(lib.Color.Black, X - 2, Y - 2, _width + 4,
    ↪  _height + 4);
67          }
68
69          public override bool IsAt(Point2D pt)
70          {
71              //removed previous logic of checking mouse position
72              //create new rectangle shape because it does not accept raw values on
    ↪  the parameter. will look for better alternatives
73              lib.Rectangle rectangle = new lib.Rectangle()
74              {
75                  X = X,
76                  Y = Y,
77                  Width = _width,
78                  Height = _height
79              };
80              return SplashKit.PointInRectangle(pt, rectangle);
81          }
82
83          public override void SaveTo(StreamWriter writer)
84          {
85              writer.WriteLine("Rectangle");
86              base.SaveTo(writer);
87              writer.WriteLine(Width);
88              writer.WriteLine(Height);
89          }
90
91          public override void LoadFrom(StreamReader reader)
92          {
93              base.LoadFrom(reader);
94              Width = reader.ReadInteger();
95              Height = reader.ReadInteger();
96          }
97      }
98  }
```

```
using DrawingProgram.lib;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DrawingProgram
{
    public class MyCircle : Shape
    {
        //local variables
        private int _radius;

        //properties
        public int Radius
        {
            get
            {
                return _radius;
            }
            set
            {
                _radius = value;
            }
        }
        //constructor
        public MyCircle(Color clr, int radius) : base(clr)
        {
            _radius = radius;
        }
        //default constructor
        public MyCircle() : this(Color.Blue, 50) { }

        //methods
        public override void Draw()
        {
            if (Selected)
            {
                DrawOutline();
            }
            SplashKit.FillCircle(Color, X, Y, _radius);
        }
        public override void DrawOutline()
        {
            SplashKit.FillCircle(Color.Black, X, Y, _radius + 2);
        }

        public override bool IsAt(Point2D pt)
        {
            //checking mouse point with distance formula
            //if (Math.Sqrt(Math.Pow(pt.X - X, 2) +  Math.Pow(pt.Y - Y,2)) <=
↪  _radius )
```

```
            //{
            //    return true;
            //}
            //else
            //{
            //    return false;
            //}

            // -----------------------------

            //new circle to pass it on PointInCircle method
            // because it has no overload methods that take in raw values
            Circle circle = new Circle()
            {
                Center = new Point2D()
                {
                    X = X,
                    Y = Y
                },
                Radius = _radius
            };
            return SplashKit.PointInCircle(pt, circle);

        }

        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();
        }
    }
}
```

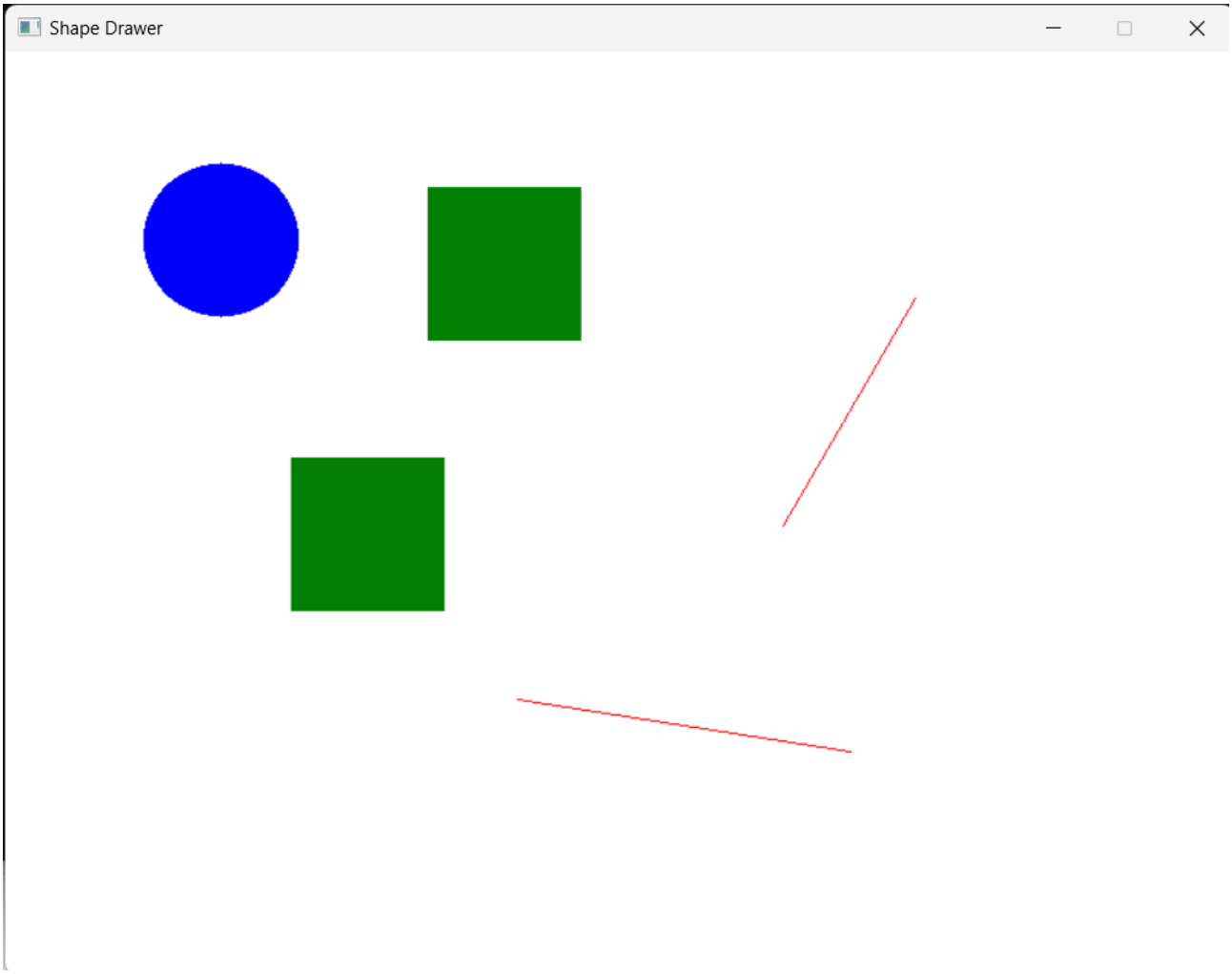
```
using DrawingProgram.lib;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DrawingProgram
{
    public class MyLine : Shape
    {
        //local variables
        // x2 y2 for end coordinates. start coordinate values are already inherited from Shape class
        private float _x2;
        private float _y2;

        //properties
        public float X2
        {
            get
            {
                return _x2;
            }
            set
            {
                _x2 = value;
            }
        }
        public float Y2
        {
            get
            {
                return _y2;
            }
            set
            {
                _y2 = value;
            }
        }
        //constructor
        public MyLine(Color clr, float x2, float y2) : base(clr)
        {
            _x2 = x2;
            _y2 = y2;
        }
        // default constructor
        public MyLine() : this(Color.Red, 0, 0) {}

        // methods
        public override void Draw()
        {
            if (Selected)
```

```
53              {
54                  DrawOutline();
55              }
56              SplashKit.DrawLine(Color, X, Y, X2, Y2);
57          }
58          public override void DrawOutline()
59          {
60              int radius = 2;
61              SplashKit.FillCircle(Color.Black, X, Y, radius);
62              SplashKit.FillCircle(Color.Black, X2, Y2, radius);
63          }
64          public override bool IsAt(Point2D pt)
65          {
66              return SplashKit.PointOnLine(pt, SplashKit.LineFrom(X, Y, X2, Y2), 5);
67          }
68          public override void SaveTo(StreamWriter writer)
69          {
70              writer.WriteLine("Line");
71              base.SaveTo(writer);
72              writer.WriteLine(X2);
73              writer.WriteLine(Y2);
74          }
75          public override void LoadFrom(StreamReader reader)
76          {
77              base.LoadFrom(reader);
78              X2 = reader.ReadInteger();
79              Y2 = reader.ReadInteger();
80          }
81      }
82  }
```


Shape Drawer