

SWINBURNE UNIVERSITY OF TECHNOLOGY

COS20007 OBJECT ORIENTED PROGRAMMING

Drawing Program - Saving and Loading

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

```
1  using SplashKitSDK;
2  using System;
3  //using System.IO;
4
5
6
7  namespace ShapeDrawer
8  {
9      // the main program class
10
11     public class Program
12     {
13         // enumerations for the different shapes
14
15         private enum ShapeKind
16         {
17             Rectangle,
18             Circle,
19             Line
20         }
21
22         // the main entry point of the the program
23
24         public static void Main()
25         {
26             Window window = new Window("Shape Drawer: 800,
↪ 600);
27             Drawing mydrawing = new Drawing();
28             ShapeKind kindToAdd = ShapeKind.Circle;
29
30             do
31             {
32                 //Process user input
33
34                 SplashKit.ProcessEvents();
35
36                 // clear the screen
37
38                 SplashKit.ClearScreen();
39
40
41                 // check if the left mouse button was clicked
42
43                 if (SplashKit.MouseClicked(MouseButton.LeftButton))
44                 {
45                     Shape newShape;
46
47
48                     //Create a new shapes based on the selected kindToAdd
49
50                     if (kindToAdd == ShapeKind.Circle)
51                     {
52                         MyCircle newCirc = new MyCircle();
```

```
53         newCirc.X = SplashKit.MouseX();
54         newCirc.Y = SplashKit.MouseY();
55         newShape = newCirc;
56
57     }
58
59     else if (kindToAdd == ShapeKind.Line)
60     {
61         MyLines newLine = new MyLines();
62         newLine.X = SplashKit.MouseX();
63         newLine.Y = SplashKit.MouseY();
64         newShape = newLine;
65     }
66     else
67     {
68         MyRectangle newRec = new MyRectangle();
69         newRec.X = SplashKit.MouseX();
70         newRec.Y = SplashKit.MouseY();
71         newShape = newRec;
72     }
73
74     //Add the shape to the drawing
75
76     mydrawing.AddShape(newShape);
77 }
78
79
80 // check for key events to change the kind of shape to add
81
82 if (SplashKit.KeyReleased(KeyCode.RKey))
83 {
84     kindToAdd = ShapeKind.Rectangle;
85 }
86 else if (SplashKit.KeyReleased(KeyCode.CKey))
87 {
88     kindToAdd = ShapeKind.Circle;
89 }
90 else if (SplashKit.KeyReleased(KeyCode.LKey))
91 {
92     kindToAdd = ShapeKind.Line;
93 }
94
95
96 // Change the background color if the space key is pressed
97
98 if (SplashKit.KeyDown(KeyCode.SpaceKey))
99 {
100     mydrawing.Background = SplashKit.RandomRGBColor(255);
101 }
102
103
104 // Selected shapes at the current mouse position when the right mouse
↪ button is clicked
```

```
105
106         if (SplashKit.MouseClicked(MouseButton.RightButton))
107         {
108             mydrawing.SelectShapesAt(SplashKit.MousePosition());
109         }
110
111         // Remove selected shapes when the delete key is pressed
112
113         if (SplashKit.KeyDown(KeyCode.DeleteKey) ||
↪ SplashKit.KeyDown(KeyCode.BackspaceKey))
114         {
115             foreach (Shape shape in mydrawing.SelectedShapes)
116             {
117                 mydrawing.RemoveShape(shape);
118             }
119         }
120
121
122
123
124
125         if (SplashKit.KeyDown(KeyCode.SKey))
126         {
127             mydrawing.Save("C:/Users/aarya/Desktop/TestDrawing.txt");
128         }
129
130
131         if (SplashKit.KeyTyped(KeyCode.OKey))
132         {
133             try
134             {
135                 mydrawing.Load("C:/Users/aarya/Desktop/TestDrawing.txt");
136             }
137             catch (Exception e)
138             {
139                 Console.Error.WriteLine("Error loading file: {0}",
↪ e.Message);
140             }
141         }
142
143         // draw the shapes and refresh the screen
144
145         mydrawing.Draw();
146         SplashKit.RefreshScreen();
147
148         // Continue the loop until the user close the window
149
150     } while (!window.CloseRequested);
151 }
152 }
153 }
```

```
1  using System;
2  using SplashScreen;
3  using System.IO;
4  using Shapedrawer;
5
6
7  namespace Shapedrawer
8  {
9      public static class ExtensionMethods
10     {
11         public static int ReadInteger(this StreamReader reader)
12         {
13             return Convert.ToInt32(reader.ReadLine());
14         }
15
16         public static float ReadSingle(this StreamReader reader)
17         {
18             return Convert.ToSingle(reader.ReadLine());
19         }
20
21         public static Color ReadColor(this StreamReader reader)
22         {
23             return Color.RGBColor(reader.ReadSingle(), reader.ReadSingle(),
↵ reader.ReadSingle());
24         }
25
26         public static void WriteColor(this StreamWriter writer, Color color)
27         {
28             writer.WriteLine("{0}\n{1}\n{2}", color.R, color.G, color.B);
29         }
30     }
31 }
```

```
1  using System;
2  using SplashKitSDK;
3  using System.IO;
4  using Shapedrawer;
5
6  namespace ShapeDrawer
7  {
8      // represents a drawing
9      public class Drawing
10     {
11         private readonly List<Shape> _shapes; // list to store shapes
12         private Color _background; // background color of drawing
13
14
15         // constructor to create a drawing with a specified background color
16         public Drawing(Color background)
17         {
18             _shapes = new List<Shape>();
19             _background = background;
20         }
21
22
23         // constructor to create a drawing with a default white background
24         public Drawing() : this(Color.White)
25         {
26
27         }
28
29
30         // gets and sets the background color of the drawing
31         public Color Background
32         {
33             get { return _background; }
34             set { _background = value; }
35         }
36
37
38         // gets the number of shapes in the drawing
39         public int ShapeCount
40         {
41             get { return _shapes.Count; }
42         }
43
44
45         // gets a list of selected shapes in the drawing
46
47         public List<Shape> SelectedShapes
48         {
49             get
50             {
51                 List<Shape> result = new List<Shape>();
52                 foreach (Shape shapez in _shapes)
53                 {
```

```
54         if (shapez.Selected == true)
55         {
56             result.Add(shapez);
57         }
58     }
59     return result;
60 }
61 }
62
63
64 // adds a shape to the drawing
65
66 public void AddShape(Shape shapez)
67 {
68     _shapes.Add(shapez);
69 }
70
71
72 // clears the screen, sets the background color, and drawing all shapes in the
↪ drawing
73
74 public void Draw()
75 {
76     SplashKit.ClearScreen(Background);
77
78     foreach (Shape shapez in _shapes)
79     {
80         shapez.Draw();
81     }
82 }
83
84
85 // selects shapes at a given point
86
87 public void SelectShapesAt(Point2D pt)
88 {
89     foreach (Shape shapez in _shapes)
90     {
91         // check if shape is at point
92         if (shapez.IsAt(pt))
93         {
94             shapez.Selected = true;
95         }
96         else
97         {
98             shapez.Selected = false;
99         }
100     }
101 }
102
103
104 // removes all selected shapes from the drawing
105
```

```
106     public void RemoveShape(Shape shapez)
107     {
108         _shapes.Remove(shapez);
109     }
110
111     // edit below
112
113
114     public void Save(string filename)
115     {
116         StreamWriter writer = new(filename);
117         try
118         {
119             writer.WriteColor(_background);
120             writer.WriteLine(ShapeCount);
121             foreach (Shape shape in _shapes)
122             {
123                 shape.SaveTo(writer);
124             }
125         }
126         finally
127         {
128             writer.Close();
129         }
130     }
131
132
133     public void Load(string filename)
134     {
135         StreamReader reader = new(filename);
136         try
137         {
138             Shape shapez;
139             string kind;
140             Background = reader.ReadColor();
141             int count = reader.ReadInteger();
142
143
144             _shapes.Clear();
145
146             for (int i = 0; i < count; i++)
147             {
148                 kind = reader.ReadLine();
149                 switch (kind)
150                 {
151                     case "Rectangle":
152                         shapez = new MyRectangle();
153                         break;
154                     case "Circle":
155                         shapez = new MyCircle();
156                         break;
157                     case "Line":
158                         shapez = new MyLines();
```



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

```
1  using Shapedrawer;
2  using SplashKitSDK;
3  using System;
4  using System.Collections.Generic;
5
6  namespace ShapeDrawer
7  {
8      // The abstract base class for all shapes
9
10     public abstract class Shape
11     {
12         //color the shape
13
14         private Color _color;
15
16         //X-coordinate of shape
17
18         private float _x;
19
20         //Y-coordinate of shape
21
22         private float _y;
23
24         //whether the shape is selected
25
26         private bool _selected;
27
28
29         // Default constructor for a shape, sets the color to yellow
30
31         public Shape() : this(Color.Yellow) { }
32
33         //Constructor for a shape with a specified color
34
35         public Shape(Color color)
36         {
37             _color = color;
38         }
39
40
41         // Gets and sets the color of the shape
42
43         public Color color
44         {
45             get { return _color; }
46             set { _color = value; }
47         }
48
49
50         //gets and sets the X-coordinate of the shape
51
52         public float X
53         {
```

```
54         get { return _x; }
55         set { _x = value; }
56     }
57
58
59     // gets and sets the Y-coordinate of the shape
60
61     public float Y
62     {
63         get { return _y; }
64         set { _y = value; }
65     }
66
67     // gets and sets whether the shape is selected
68
69     public bool Selected
70     {
71         get { return _selected; }
72         set { _selected = value; }
73     }
74
75
76     // abstract method to draw the shape on the screen
77
78     public abstract void Draw();
79
80     // abstract method to draw the outline of the shape on the screen
81
82     public abstract void DrawOutline();
83
84     // abstract method for checking if a given point is inside the shape
85
86     public abstract bool IsAt(Point2D pt);
87
88
89     public virtual void SaveTo(StreamWriter writer)
90     {
91         writer.WriteColor(color);
92         writer.WriteLine(X);
93         writer.WriteLine(Y);
94     }
95
96
97     public virtual void LoadFrom(StreamReader reader)
98     {
99         color = reader.ReadColor();
100         X = reader.ReadInteger();
101         Y = reader.ReadInteger();
102     }
103
104
105
106 }
```

107 }

```
1  using System;
2  using SplashScreen;
3  using System.Collections.Generic;
4  using Shapedrawer;
5
6
7
8
9  namespace ShapeDrawer
10 {
11     // My rectangle class represents a rectangle
12
13     public class MyRectangle : Shape
14     {
15         // Width and height of rectangle
16
17         private int _width, _height;
18
19         // constructor to create a rectangle with a specified width, height and color
20
21         public MyRectangle(Color color, float x, float y, int width, int height) :
↪     base(color)
22         {
23             X = x;
24             Y = y;
25             Width = width;
26             Height = height;
27         }
28
29         // constructor to create a rectangle with a specified width, height and color
30
31         public MyRectangle() : this(Color.Green, 0, 0, 100, 100) { }
32
33         // gets and sets the width of the rectangle
34
35         public int Height
36         {
37             get
38             {
39                 return _height;
40             }
41             set
42             {
43                 _height = value;
44             }
45         }
46
47
48         // gets and sets the height of the rectangle
49
50         public int Width
51         {
52             get { return _width; }
```

```
53         set { _width = value; }
54     }
55
56     // draws the filled rectangle on the screen and outline if selected
57
58     public override void Draw()
59     {
60         // draw the filled rectangle on the screen
61
62         SplashKit.FillRectangle(color, X, Y, Width, Height);
63
64         // if the rectangle is selected, draw the outline
65
66         if (Selected)
67         {
68             DrawOutline();
69         }
70     }
71
72
73     // draws the outline of the rectangle on the screen
74
75     public override void DrawOutline()
76     {
77         // draw a sligly bigger black rectangle on the screen
78
79         SplashKit.DrawRectangle(Color.Black, X - 2, Y - 2, Width + 4, Height +
↪ 4);
81     }
82
83
84     // checks if the rectangle is inside the rectangle
85
86     public override bool IsAt(Point2D mouseLocation)
87     {
88         if (X < mouseLocation.X && mouseLocation.X < (X + Width) && Y <
↪ mouseLocation.Y && mouseLocation.Y < (Y + Height))
89         {
90             // if the point is within the rectangle, return true
91
92             return true;
93         }
94         else
95         {
96             // if the point is not within the rectangle, return false
97
98             return false;
99         }
100     }
101
102     public override void SaveTo(StreamWriter writer)
103     {
```

```
104         writer.WriteLine("Rectangle");
105         base.SaveTo(writer);
106         writer.WriteLine(Width);
107         writer.WriteLine(Height);
108     }
109
110     public override void LoadFrom(StreamReader reader)
111     {
112         base.LoadFrom(reader);
113         Width = reader.ReadInteger();
114         Height = reader.ReadInteger();
115     }
116
117 }
118 }
```

```
1  using System;
2  using SplashScreen;
3  using Shapedrawer;
4  using System.IO;
5
6  namespace ShapeDrawer
7  {
8      // represents a circle
9
10     public class MyCircle : Shape
11     {
12         private int _radius; // radius of circle
13
14         // constructor to create a circle with a specified radius and color
15
16         public MyCircle() : this(Color.BlueViolet, 50) { }
17
18         // constructor to create a circle with a specified radius and color
19
20         public MyCircle(Color clr, int radius) : base(clr)
21         {
22             _radius = radius;
23         }
24
25         // gets and sets the radius of the circle
26
27         public int Radius
28         {
29             get { return _radius; }
30             set { _radius = value; }
31         }
32
33
34         // checks if the circle is at a specified point
35
36         public override bool IsAt(Point2D pt)
37         {
38             // calculate the distance between the point the circle's center
39
40             double hypotenuse = Math.Sqrt(Math.Pow(X - pt.X, 2) + Math.Pow(Y - pt.Y,
41 ↪ 2));
42
43             // if the distance is less than the radius, the point is inside the
44 ↪ circle
45
46             if (hypotenuse <= Radius)
47             {
48                 return true; // point is inside circle
49             }
50             else
51             {
52                 return false; // point is outside circle
53             }
54         }
55     }
56 }
```

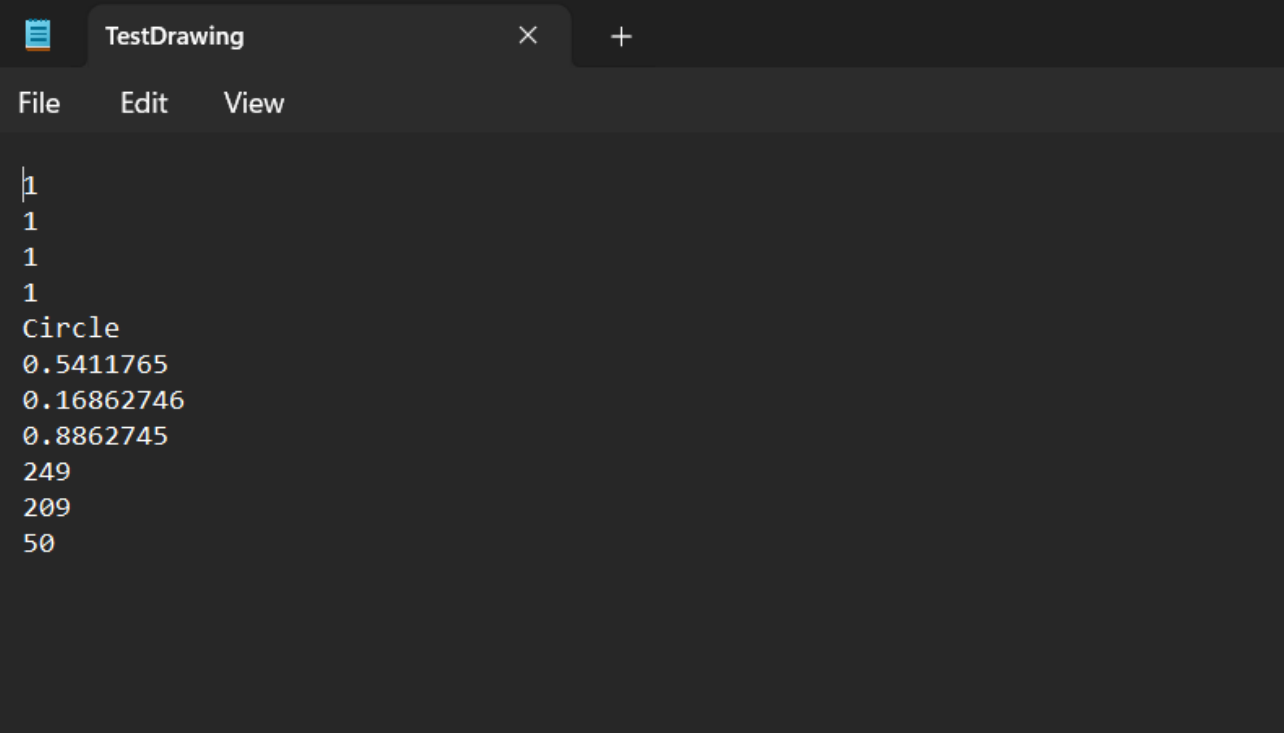


```
52     }
53
54
55     // draws the outline of the circle on the screen
56
57     public override void DrawOutline()
58     {
59         SplashKit.DrawCircle(Color.Black, X, Y, _radius + 2);
60     }
61
62
63     // draws the circle on the screen if selected
64
65     public override void Draw()
66     {
67         SplashKit.FillCircle(color, X, Y, _radius);
68
69         // if selected, draw the outline of the circle
70
71         if (Selected)
72         {
73             DrawOutline();
74         }
75     }
76
77
78     public override void SaveTo(StreamWriter writer)
79     {
80         writer.WriteLine("Circle");
81         base.SaveTo(writer);
82         writer.WriteLine(_radius);
83     }
84
85     public override void LoadFrom(StreamReader reader)
86     {
87         base.LoadFrom(reader);
88         Radius = reader.ReadInteger();
89     }
90
91 }
92 }
```

```
1  using System;
2  using SplashScreen;
3  using System.Collections.Generic;
4  using System.Linq;
5  using Shapedrawer;
6  //using System.IO;
7
8
9  namespace ShapeDrawer
10 {
11     // MyLines class represents a line shapes
12
13     public class MyLines : Shape
14     {
15         // coordinates of lines's end point
16
17         private float _endX, _endY;
18
19         // constructor to create a line with a specified end point and color
20
21         public MyLines() : this(Color.Blue, 100, 100, 400, 300) { }
22
23         // constructor to create a line with a specified end point and color
24
25         public MyLines(Color color, float x, float y, float endx, float endy) :
↪     base(color)
26         {
27             X = x;
28             Y = y;
29             EndX = endx;
30             EndY = endy;
31         }
32
33         // gets and sets the X-coordinate of the line's end point
34
35         public float EndX
36         {
37             get { return _endX; }
38             set { _endX = value; }
39         }
40
41
42         // gets and sets the Y-coordinate of the line's end point
43
44         public float EndY
45         {
46             get { return _endY; }
47             set { _endY = value; }
48         }
49
50
51         // checks if a given point is within of the line
52
```

```
53     public override bool IsAt(Point2D pt)
54     {
55         // calculate the gradient and intercept of the line
56
57         float gradient = (EndY - Y) / (EndX - X);
58         float intercept = EndY - (gradient * EndX);
59
60         // calculate the distance between the point and the line
61
62         double margin = 10;
63
64         // calculate the distance between the point and the line
65
66         double distance = Math.Abs((gradient * pt.X) + intercept - pt.Y);
67
68         // if the distance is less than the margin, the point is on the line
69
70         return distance <= margin;
71     }
72
73     // draws the outline of the line on the screen
74
75     public override void DrawOutline()
76     {
77         SplashKit.DrawCircle(Color.GhostWhite, X, Y, 4);
78         SplashKit.DrawCircle(Color.GhostWhite, EndX, EndY, 4);
79     }
80
81
82     // draws the line on the screen if selected
83
84     public override void Draw()
85     {
86         // draw the line
87
88         SplashKit.DrawLine(color, X, Y, EndX, EndY);
89
90         // if selected, draw the outline of the line
91
92         if (Selected)
93         {
94             DrawOutline();
95         }
96     }
97
98
99     public override void SaveTo(StreamWriter writer)
100    {
101        writer.WriteLine("Line");
102        base.SaveTo(writer);
103        writer.WriteLine(EndX);
104        writer.WriteLine(EndY);
105    }
```

```
106         }
107
108
109
110     public override void LoadFrom(StreamReader reader)
111     {
112         base.LoadFrom(reader);
113         EndX = reader.ReadInteger();
114         EndY = reader.ReadInteger();
115     }
116
117 }
118 }
```



```
1
1
1
1
Circle
0.5411765
0.16862746
0.8862745
249
209
50
```