Listing 1: AbstractShape.cs

```cs
using System;
using System.Text;

namespace SmellyShapes.Source
{
  public abstract class AbstractShape : IShape
  {
    public abstract bool Contains(int x, int y);

    public string ToXml()
    {
      var builder = new StringBuilder();

      if (this is Circle circle)
      {
        builder.Append("<circle");
        builder.Append(" x=\"" + circle.X + "\"");
        builder.Append(" y=\"" + circle.Y + "\"");
        builder.Append(" radius=\"" + circle.Radius + "\"");
        builder.Append(" />\n");
      }
      else if (this is Square square)
      {
        builder.Append("<square");
        builder.Append(" x=\"" + square.X + "\"");
        builder.Append(" y=\"" + square.Y + "\"");
        builder.Append(" edgeLength=\"" + square.Width + "\"");
        builder.Append(" />\n");
      }
      else if (this is Rectangle rectangle)
      {
        builder.Append("<rectangle");
        builder.Append(" x=\"" + rectangle.X + "\"");
        builder.Append(" y=\"" + rectangle.Y + "\"");
        builder.Append(" width=\"" + rectangle.Width + "\"");
        builder.Append(" height=\"" + rectangle.Height + "\"");
        builder.Append(" />\n");
      }
      else if (this is ShapeGroup group)
      {
        builder.Append("<shapegroup>\n");
        for (var i = 0; i < group.Size; i++)
          builder.Append(group.Shapes[i].ToXml());
        builder.Append("</shapegroup>\n");
      }
      else
      {
        throw new ArgumentException("Unknown shape type: " + GetType());
      }

      return builder.ToString();
    }
  }
}
```

Listing 2: Circle.cs

```cs
namespace SmellyShapes.Source;

public class Circle : SimpleShape
{
  private int numberOfContainingPoints;
  private readonly int x;
  private readonly int y;

  public Circle(int x, int y, int radius)
  {
    this.x = x;
    this.y = y;
    this.Radius = radius;
  }

  /// <summary>
  ///   Gets or sets the shape color.
  /// </summary>
  public Color Color { get; set; } = new Color("Green");

  public int X
  {
    get { return this.x; }
  }

  public int Y => this.y;

  public int Radius { get; }

  public override bool Contains(int x, int y)
  {
    var result = (x - this.X) * (x - this.X) + (y - Y) * (y - Y) <= Radius * Radius;
    // Increase number of Points?
    if (result) numberOfContainingPoints++;
    return result;
  }

  public int CountContainingPoints(int[] xCords, int[] yCords)
  {
    numberOfContainingPoints = 0;
    for (var i = 0; i < xCords.Length; ++i) Contains(xCords[i], yCords[i]);
    return numberOfContainingPoints;
  }

  public override string ToString()
  {
    return "Circle: (" + X + "," + Y + ") radius= " + Radius
        + " RGB=" + Color.ColorAsRgbRed + ","
        + Color.ColorAsRgbGreen + ","
        + Color.ColorAsRgbBlue;
  }
}
```

Listing 3: Color.cs

```cs
namespace SmellyShapes.Source
{
  public class Color
  {
    private readonly string colorAsText;

    public string ColorAsHex { get; private set; }
    public string ColorAsRgbBlue { get; private set; }
    public string ColorAsRgbGreen { get; private set; }
    public string ColorAsRgbRed { get; private set; }
    public string ErrorMessage { get; private set; }

    public Color(string colorAsText)
    {
      this.colorAsText = colorAsText;
      ConvertTextValueToRgbAndHex();
    }

    public string GetColorFormatted(bool includeHexAndRgb)
    {
      if (includeHexAndRgb)
        return colorAsText + " " + ColorAsHex + " " + ColorAsRgbRed + ":" + ColorAsRgbGreen + ":"
          +
               ColorAsRgbBlue;
      return colorAsText;
    }

    private void ConvertTextValueToRgbAndHex()
    {
      ErrorMessage = string.Empty;

      // set to Red
      if (colorAsText == "Red")
      {
        ColorAsRgbRed = "255";
        ColorAsRgbBlue = "0";
        ColorAsRgbGreen = "0";
        ColorAsHex = "#FF0000";
      }
      else if (colorAsText == "Blue")
      {
        // set to Blue
        ColorAsRgbRed = "0";
        ColorAsRgbBlue = "255";
        ColorAsRgbGreen = "0";
        ColorAsHex = "#00FF00";
      }
      else if (colorAsText == "Green")
      {
        // set to Green
        ColorAsRgbRed = "0";
        ColorAsRgbBlue = "0";
        ColorAsRgbGreen = "255";
        ColorAsHex = "#0000FF";
      }
      else
      {
        ErrorMessage = "Color not recognized";
      }
    }
  }
}
```

Listing 4: ComplexShape.cs

```cs
namespace SmellyShapes.Source
{
  public abstract class ComplexShape : AbstractShape
  {
    protected bool ReadOnly { get; set; }

    public void SetReadOnly(bool readOnly)
    {
      ReadOnly = readOnly;
    }
  }
}
```

Listing 5: DrawingBoard.cs

```cs
namespace SmellyShapes.Source
{
  public class DrawingBoard : ShapeGroup
  {
    public Color BackgroundColor { get; set; }

    public static void Main()
    {
      var drawingBoard = new DrawingBoard
      {
        BackgroundColor = new Color("Green")
      };

      drawingBoard.Add(new Square(-10, -10, 20));
      drawingBoard.Load("testFile");
      drawingBoard.DrawOnScreen();
    }

    public void DrawOnScreen()
    {
      // ... removed for exercise
    }

    public void Load(string file)
    {
      // ... removed for exercise
    }
  }
}
```

Listing 6: Rectangle.cs

```cs
namespace SmellyShapes.Source
{
  public class Rectangle : SimpleShape
  {
    private readonly int height;

    public Rectangle(int x, int y, int width, int height)
    {
      X = x;
      Y = y;
      Width = width;
      this.height = height;
    }

    public int Width { get; }

    public virtual int Height => height;

    public int X { get; }

    public int Y { get; }

    protected Color C { get; set; } = new Color("Blue");

    public override bool Contains(int x, int y)
    {
      return X <= x && x <= X + Width && Y <= y && y <= Y + height;
    }

    public int Calculate()
    {
      return Width * height;
    }

    public override string ToString()
    {
      return
        $"Rectangle: ({X},{Y}) width={Width} height={height} color={C.ColorAsHex}";
    }
  }
}
```

Listing 7: IShape.cs

```cs
namespace SmellyShapes.Source;

public interface IShape
{
  bool Contains(int x, int y);

  string ToXml();
}
```

Listing 8: ShapeGroup.cs

```cs
namespace SmellyShapes.Source
{
  public class ShapeGroup : ComplexShape
  {
    public IShape[] Shapes = new IShape[10];

    public int Size;

    public ShapeGroup()
    {
    }

    public ShapeGroup(IShape[] shapes, bool readOnly)
    {
      this.Shapes = shapes;
      Size = shapes.Length;
      ReadOnly = readOnly;
    }

    public void Add(IShape shape)
    {
      if (!ReadOnly)
      {
        var newSize = Size + 1;
        if (newSize > Shapes.Length)
        {
          var newShapes = new IShape[Shapes.Length + 10];
          for (var i = 0; i < Size; i++) newShapes[i] = Shapes[i];
          Shapes = newShapes;
        }

        if (Contains(shape)) return;
        Shapes[Size++] = shape;
      }
    }

    public bool Contains(IShape shape)
    {
      for (var i = 0; i < Size; i++)
        if (Shapes[i].Equals(shape))
          return true;
      return false;
    }

    public override bool Contains(int x, int y)
    {
      foreach (var shape in Shapes)
        if (shape != null)
          if (shape.Contains(x, y))
            return true;
      return false;
    }
  }
}
```

Listing 9: SimpleShape.cs

```cs
namespace SmellyShapes.Source
{
  public abstract class SimpleShape : AbstractShape
  {
  }
}
```

Listing 10: Square.cs

```cs
using System;

namespace SmellyShapes.Source
{
  public class Square : Rectangle
  {
    public Square(int x, int y, int edgeLength)
      : base(x, y, edgeLength, edgeLength)
    {
    }

    public Square(int x, int y, int edgeLength, Color color)
      : base(x, y, edgeLength, edgeLength)
    {
      C = color;
    }

    public override int Height =>
      throw new InvalidOperationException("Square does not have a height, only edgeLength");

    public bool ContainsPoint(int x, int y)
    {
      return X <= x && x <= X + Width && Y <= y && y <= Y + Width;
    }

    public override string ToString()
    {
      return $"Square: ({X}:{Y}) edgeLength={Width} color={C.ColorAsHex}";
    }

    public bool Contains(int x1, int y1, int x2, int y2)
    {
      return Contains(x1, y1) && Contains(x2, y2);
    }
  }
}
```