

```
using System;
using System.Drawing;
using System.Runtime.InteropServices;

namespace sb54_CSAI.GUI
{
    #region Colour CLASS
    /// <summary>
    /// Stores a colour and provides conversion between the Red Green Blue (RGB)
    /// and Hue Saturation Lightness (HSL) colour models
    /// <br></br>
    /// <br></br>
    /// This class was written using an article contained at the following URL
    /// http://www.codeproject.com/cs/miscctrl/CollapsiblePanelBar.asp?df=100&forumid=12638&exp=0&fr=51
    /// <br></br>
    /// GUI components are an area that interest me so I have included the source code in
    this namespace so that
    /// I could understand what was being done by the Author (Derek Lakin) of the original
    classes.
    /// I could have simply used the Authors original C# Dll, but I would not have learned
    much that way
    /// so I chose to examine and use the source code. Some of the names of classes are
    different here, as I
    /// actually wrote the code line by line, using the original article so that I could
    fully comprehend the
    /// original implementation. As such all the comments that are found within the code are
    by and large my own
    /// comments
    /// </summary>
    public class Colour
    {
        #region Instance Fields
        //Instance Fields
        public const int HUEMAX = 360;
        public const float SATMAX = 1.0f;
        public const float BRIGHTMAX = 1.0f;
        public const int RGBMAX = 255;
        private Color clrCurrent = Color.Red;
        #endregion
        #region Public Constructor
        /// <summary>
        /// Simply creates a new Colour object
        /// </summary>
        public Colour()
        {
        }
        #endregion
        #region Public Methods

        /// <summary>
        /// Gets/Sets the current colour (RGB model)
        /// </summary>
        public Color CurrentColour
        {
            get
            {
                return clrCurrent;
            }
            set
            {
                clrCurrent = value;
            }
        }

        /// <summary>
        /// Gets/Sets the Red component of the current colour
        /// </summary>
        public byte Red
        {
            get
            {
                return clrCurrent.R;
            }
        }
    }
}

```

```

    }
    set
    {
        clrCurrent = Color.FromArgb(value, Green, Blue);
    }
}

/// <summary>
/// Gets/Sets the Green component of the current colour
/// </summary>
public byte Green
{
    get
    {
        return clrCurrent.G;
    }
    set
    {
        clrCurrent = Color.FromArgb(Red, value, Blue);
    }
}

/// <summary>
/// Gets/Sets the Blue component of the current colour
/// </summary>
public byte Blue
{
    get
    {
        return clrCurrent.B;
    }
    set
    {
        clrCurrent = Color.FromArgb(Red, Green, value);
    }
}

/// <summary>
/// Gets/Sets the Hue component of the current colour
/// </summary>
public int Hue
{
    get
    {
        //get the Hue for the current colour
        return (int)clrCurrent.GetHue();
    }
    set
    {
        //set the Hue for the current colour
        clrCurrent = HSBToRGB(value,
            clrCurrent.GetSaturation(),
            clrCurrent.GetBrightness());
    }
}

/// <summary>
/// Gets the Hue value
/// </summary>
/// <returns>A float which represents the Hue value</returns>
public float GetHue()
{
    float top = ((float)(2 * Red - Green - Blue)) / (2 * 255);
    float bottom = (float)Math.Sqrt(((Red - Green) * (Red - Green) + (Red - Blue) *
(Green - Blue)) / 255);
    return (float)Math.Acos(top / bottom);
}

/// <summary>
/// Gets the Saturation value
/// </summary>
/// <returns>A float which represents the Saturation value</returns>
public float GetSaturation()

```

```

    {
        return (255 -
            (((float)(Red + Green + Blue)) / 3) * Math.Min(Red, Math.Min(Green, Blue)))
    / 255;
    }

    /// <summary>
    /// Gets the Brightness value
    /// </summary>
    /// <returns>A float which represents the Brightness value</returns>
    public float GetBrightness()
    {
        return ((float)(Red + Green + Blue)) / (255.0f * 3.0f);
    }

    /// <summary>
    /// Gets the Saturation component of the current colour
    /// </summary>
    /// <returns>A float which represents the Saturation component of the current colour
</returns>
    public float Saturation
    {
        get
        {
            if(0.0f == Brightness)
            {
                return 0.0f;
            }
            else
            {
                float fMax = (float)Math.Max(Red, Math.Max(Green, Blue));
                float fMin = (float)Math.Min(Red, Math.Min(Green, Blue));
                return (fMax - fMin) / fMax;
            }
        }
        set
        {
            clrCurrent = HSBToRGB((int)clrCurrent.GetHue(),
                value, clrCurrent.GetBrightness());
        }
    }

    /// <summary>
    /// Gets the Brightness component of the current colour
    /// </summary>
    /// <returns>A float which represents the Brightness component of the current colour
</returns>
    public float Brightness
    {
        get
        {
            //return clrCurrent.GetBrightness();
            return (float)Math.Max(Red, Math.Max(Green, Blue)) / (255.0f);
        }
        set
        {
            clrCurrent = Colour.HSBToRGB((int)clrCurrent.GetHue(),
                clrCurrent.GetSaturation(),
                value);
        }
    }

    /// <summary>
    /// Converts HSB colour components to an RGB System.Drawing.Color
    /// </summary>
    /// <param name="Hue">Hue component</param>
    /// <param name="Saturation">Saturation component</param>
    /// <param name="Brightness">Brightness component</param>
    /// <returns>Returns the RGB value as a System.Drawing.Color</returns>
    public static Color HSBToRGB(int Hue, float Saturation, float Brightness)
    {
        int red = 0; int green = 0; int blue = 0;
        if(Saturation == 0.0f)

```

```

    {
        // Achromatic colour (black and white centre line)
        // Hue should be 0 (undefined), but we'll ignore it.
        // Set shade of grey
        red = green = blue = (int)(Brightness * 255);
    }
else
{
    // Chromatic colour
    // Map hue from [0-255] to [0-360] to hexagonal-space [0-6]
    // (360 / 256) * hue[0-255] / 60
    float fHexHue = (6.0f / 360.0f) * Hue;
    // Determine sector in hexagonal-space (RGB cube projection) {0,1,2,3,4,5}
    float fHexSector = (float)Math.Floor((double)fHexHue);
    // Determine exact position in particular sector [0-1]
    float fHexSectorPos = fHexHue - fHexSector;

    // Convert parameters to in-formula ranges
    float fBrightness = Brightness * 255.0f;
    float fSaturation = Saturation;

    byte bWashOut = (byte)(0.5f + fBrightness * (1.0f - fSaturation));
    byte bHueModifierOddSector = (byte)(0.5f + fBrightness * (1.0f - fSaturation
* fHexSectorPos));
    byte bHueModifierEvenSector = (byte)(0.5f + fBrightness * (1.0f -
fSaturation * (1.0f - fHexSectorPos)));

    // Assign values to RGB components (sector dependent)
    switch((int)fHexSector)
    {
        case 0 :
            // Hue is between red & yellow
            red = (int)(Brightness * 255); green = bHueModifierEvenSector; blue
= bWashOut;
            break;
        case 1 :
            // Hue is between yellow & green
            red = bHueModifierOddSector; green = (int)(Brightness * 255); blue =
bWashOut;
            break;
        case 2 :
            // Hue is between green & cyan
            red = bWashOut; green = (int)(Brightness * 255); blue =
bHueModifierEvenSector;
            break;
        case 3 :
            // Hue is between cyan & blue
            red = bWashOut; green = bHueModifierOddSector; blue = (int)
(Brightness * 255);
            break;
        case 4 :
            // Hue is between blue & magenta
            red = bHueModifierEvenSector; green = bWashOut; blue = (int)
(Brightness * 255);
            break;
        case 5 :
            // Hue is between magenta & red
            red = (int)(Brightness * 255); green = bWashOut; blue =
bHueModifierOddSector;
            break;
        default :
            red = 0; green = 0; blue = 0;
            break;
    }
    return Color.FromArgb(red, green, blue);
}
#endregion
}
#endregion
}

```