Programación orientada a objetos. Obligatorio

Segunda parte.

Octubre de 2012.

Las fotos e imágenes que vemos todos los días en nuestra computadora pueden representarse fácilmente como una matriz de colores. Cada uno de los elementos de esa matriz dice que color debe tomar cada uno de los puntos de nuestra pantalla.

Para representar los 65536 colores, que normalmente pueden mostrar los diversos dispositivos, se utilizan combinaciones de tres componentes distintos: rojo, verde y azul (RGB, por sus siglas en inglés). Cada uno de los componentes puede variar en intensidad de 0 a 255. Por ejemplo, para representar un color puramente rojo, se utiliza un rojo de 255, un verde de 0 y un azul de 0, para representar un blanco, los tres colores al máximo de intensidad, y para representar un negro los tres colores al mínimo.

Las imágenes digitales pueden transformarse mediante filtros, que analizan esta matriz de colores y la modifican aplicando distintas transformaciones. Por ejemplo, un filtro para transformar una imagen a blanco y negro, podría evaluar cada punto y si tiene colores muy intensos transformarlo a negro o en otro caso a blanco.

Parte 1

La siguiente interfaz representa una imagen.

```
using System;
using System.Drawing;
namespace PicfilLib
{
   public interface IPicture
   {
      Int32 Width { get; }
      Int32 Height { get; }
      Color GetColor(Int32 x, Int32 y);
      void SetColor(Int32 x, Int32 y, Color color);
      void Resize(Int32 width, Int32 height);
      IPicture Clone();
   }
}
```

El código que te proveemos se encuentra bajo el espacio de nombres PicfilLib. Por lo tanto, para referenciar y usar nuestras clases e interfaces, debes agregar `using PicfilLib;` al comienzo de tu código.

Consulta la documentación de `IPicture` en el código que te proveemos para conocer que debe hacer cada una de las operaciones.

Para representar una matriz de colores, puedes usar la siguiente declaración

```
Color[,] colors = new Color[width, height];
```

Puedes acceder al color del punto (x, y) de la matriz haciendo colors[x, y] y obtener el ancho de la matriz haciendo colors.GetLength(0) y el alto haciendo colors.GetLength(1). Consulta la documentación de C# por más información.

Asegúrate de agregar el espacio de nombres System.Drawing; para poder usar la clase Color, que es parte de la plataforma .NET. Debes agregar a "Referencias" la biblioteca System.Drawing en caso de que no la tengas.

Parte 2

Una vez que tengas el tipo implementado, puedes usar la implementación de IPictureProvider que te proveemos para cargar una imagen de su disco y la implementación de IPictureRenderer para mostrar la imagen por pantalla.

Para hacerlo, pueden usar una clase Program basada en la siguiente.

```
using System.Windows.Forms;
using PicfilLib;
....
public class Program
{
    [STAThread]
    public static void Main()
    {
      IPictureProvider provider = ...;

      IPicture picture = ...;
        ....
      IPictureRenderer renderer = ...;
      renderer.Render(picture);

      Application.Run();
    }
}
```

Mira el código que te damos para identificar las clases que implementan estas dos interfaces y leer la documentación de las mismas.

Para que el programa funciona debes haber creado tu proyecto como una aplicación de ventanas (*Window Application*) o agregar a las referencias las librerías System.Windows.Forms y System.Drawing.

Parte 3 - Filtros

Implementa dos filtros de imágenes, uno para obtener el negativo de una imagen y el otro para transformar la imagen a una imagen monocromática en blanco y negro.

Negativo

Para obtener el negativo de una imagen, solo debes restar a cada uno de los componentes de cada punto de la imagen 255 y obtener el valor absoluto del mismo. Por ejemplo, asumiendo que un punto de la imagen tiene como componentes

```
Rojo:    34
Verde:   55
Azul:    100
```

el negativo es

```
Rojo:     Abs(34 – 255) = 221
Verde:    Abs(55 – 255) = 200
Azul:     Abs(100 – 255) = 155.
```

Blanco y Negro

Para obtener la imagen en blanco y negro, debe definir un límite del color a partir del cual tomas el punto como blanco. Si el punto no supera este color, entonces se debe mostrar como negro. Para calcular el valor absoluto puedes usar la función de .NET Math.Abs(Int32).

Por ejemplo, si los componentes del color del punto son:

```
Rojo:     34
Verde:    55
Azul:     100
```

Y defines como limite 99, puedes asumir que el punto es blanco ya que el Azul supera 99. Si en cambio defines como limite 10, el punto debe ser negro, ya que todos los puntos son superiores a ese nivel.

Puedes ir probando con distintos niveles para determinar cual es el ideal. Opcionalmente, puedes usar el promedio de todos los componentes en lugar de fijarte si solamente uno de ellos supera el límite.

Para implementar el filtro, debes implementar el siguiente tipo:

```
namespace PicfilLib
{
  public interface IFilter
  {
    IPicture Filter(IPicture image);
  }
}
```

Consulta la documentación de la interfaz en el código entregado.

Puedes probar aplicar los filtros y mostrar las imágenes generadas modificando levemente el código programado en la parte 3.