Programación orientada a objetos. Obligatorio

Tercera parte.

Octubre de 2012.

En esta tercer y última parte, deberás hacer uso de las primeras partes, para crear un único programa capaz que aplicar excelentes filtros sobre imágenes.

• Parte 1

Para poder resolver los próximos ejercicios, debes agregar a tus filtros un nombre que los identifique de forma única. Para ello vamos a modificar un poco la interfaz IFilter y agregarle un atributo Name:

```
public interface IFilter
{
    String Name { get; }
    IPicture Filter(IPicture image);
}
```

Luego deberás implementar una forma de poder crear filtros desde un archive de tags. Para ello, cada tipo de filtro deberá tener un nombre único.

Dentro del archivo de tags, el nombre viene especificado por un atributo de clave nombre, por ejemplo:

```
<filter-negative name="blanco y negro"/>
```

Esto deberá crear un IFilter que convierte la imagen a negativo y tiene como nombre "blanco y negro"

Al leer el archivo de tags, debes validar que no existan dos filtros con el mismo nombre en un mismo archivo.

• Parte 2

Implementa filtros de convolución. Los filtros de convolución son una familia de filtros sencillos que calculan el color de un pixel en base al color de los pixels vecinos.
Por ejemplo, un filtro de convolución es el suavizado, que permite hacer una imagen más borrosa, simplemente tomando como el color de cada pixel el promedio del color de los pixels vecinos. De esta forma, cada uno de los pixels se parece más a sus vecinos, y la imagen se vuelve más borrosa.
Para aplicar un filtro de convolución, se puede utilizar una matriz de 3x3, donde en cada elemento de la matriz es un coeficiente por el cual se debe multiplicar al color del vecino, un divisor y un complemento.
Por ejemplo, para un posible filtro de suavizado, se puede utilizar una matriz (m) como:

1	1	1
1	**1**	1
1	1	1

Un divisor de 9 y un complemento de 0.

El calculo de cada pixel se hace de la siguiente forma:

```
rojo_nuevo[x, y] =
      (
            (
            r[x-1,y-1]*m[0, 0] + r[x,y-1]*m[0,1] + r[x+1, y−1]*m[0, 2] +
            r[x-1,y]*m[1,0] + r[x,y]*m[1,1] + r[x+1,y]*m[1,2] +
            r[x-1,y+1]*m[2,0] + r[x,y+1]*m[2,1] + r[x+1,y+1]*m[2,2]
            )
            / divisor
      ) + complemento
```

Donde r[x,y] corresponde al componente rojo del pixel (x,y) de la imagen original y m[i,j] al valor de la matriz en la posición (i,j), donde 0 <= i, j <= 2 . Es decir, el coeficiente del centro es el peso del pixel que se está filtrando, y el resto el peso de todos los pixels que lo rodean.
En el caso del suavizado, sería:

```
rojo_nuevo[x, y] =
      ((
      r[x-1,y-1]   * 1 + r[x,y-1]    * 1 + r[x+1,y-1]   * 1 +
      r[x-1,y]      * 1 + r[x,y]       * 1 + r[x+1,y]      * 1 +
      r[x-1,y+1]  * 1 + r[x,y+1]  * 1 + r[x+1,y+1] * 1
      ) / 9) + 0
```

Lo que equivale a hacer un promedio del valor del componente rojo para cada uno de los pixels.
Para esta parte debes implementar un filtro de convolución para suavizado y uno para relieve. El primero puedes hacerlo a partir de la matriz, divisor y complemento dado antes. El de relieve utiliza los siguientes parámetros:
matriz =

-1	0	-1
0	4	0
-1	0	-1

Divisor = 1, Complemento = 127.
Puedes ver el resultado de aplicar los filtros de relieve y suavizado en las figuras 1 y 2, respectivamente.

Además, debes agregar soporte para especificar los filtros en el archivo de tags, siguiendo el siguiente formato:

```
<filter-blur name=”suavizado”/>
```

para usar un filtro de suavizado.

```
<filter-emboss name=”relieve”/>
```

para usar un filtro de relieve.

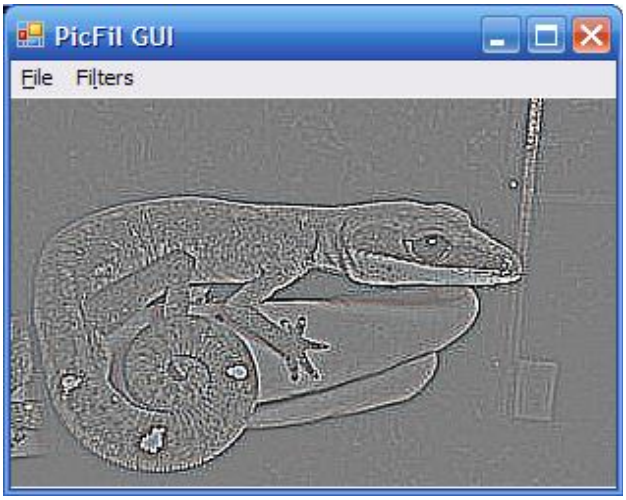

Figura 1: Lagartija con *relieve*

Figura 2: Lagartija *Suavizada*

- **Parte 3**

Hasta el momento tu programa te permite aplicar varios filtros a una imagen específica. Sin embargo, no provee una interfaz de usuario mínimamente razonable que permita aplicar tus filtros sobre otras imágenes de manera interactiva.

Como no queremos que pierdas el tiempo aprendiendo como usar las ventanitas en C#, nosotros ya te resolvemos el problema y ¡te damos la interfaz programada! Claro que, siguiendo buenas practicas de POO, la interfaz no hace demasiado (solo despliega ventanas) y delega la mayor parte del trabajo a un objeto instancia de una clase que tu debes programar y que debes proveerle a la interfaz para que use.

La interfaz de usuario que te proveemos esta programada en una clase que implementa la siguiente interfaz:

```
public interface IPicfilGui: IPictureRenderer
{
        void AddFilter(String filterName);
        Boolean ContainsFilter(String filterName);

        void AddListener(IGuiListener listener);
        Boolean ContainsListener(IGuiListener listener);

        void Run();
}
```

Como puedes ver, la interfaz extiende a IPictureRenderer por lo que además de proveer las operaciones declaradas en la propia interfaz, provee una operación para dibujar una imagen (en este caso, sobre la interfaz, prueba usarla).

La primera operación, void AddFilter(String filterName), te permite registrar un filtro en la interfaz pasando su nombre como argumento. A partir de ese momento, la interfaz mostrará un ítem en un menú cuyo texto es el nombre del filtro, y le permitirá al usuario seleccionarlo.

Además, notarás que hay una operación llamada void AddListener(IGuiListener listener). Esta operación permite registrar un "oyente" (u observador) de la interfaz para que esta le reporte los eventos y acciones que se realicen sobre la misma.

En general, los eventos pueden ser de diverso tipo, por ejemplo, que alguien apretó un botón, ingresó un texto, cerró una ventana, etc., pero en nuestro caso solo usaremos dos: cuando se seleccione una imagen y cuando se selecciona un filtro de la lista de filtros desplegados por la interfaz.

Cada vez que se genera un evento, la interfaz invoca una operación apropiada sobre el observador. La operación es una de las provistas en el tipo del observador:

```
public interface IGuiListener
{
        void ApplyFilter(String filterName);

        void SelectPicture(String pictureName);
}
```

Cuando la interfaz detecta que alguien seleccionó una imagen para trabajar con ella, le avisa al observador invocando la operación void SelectPicture(String pictureName).

Cuando la interfaz detecta que alguien seleccionó un filtro para aplicarlo, le avisa al observador invocando la operación void ApplyFilter(String filterName).

Tu deberás programar una clase que implemente el tipo del observador para procesar el evento y tomar alguna acción.

Debes tomar las siguientes acciones dependiendo del mensaje recibido:

void SelectPicture(String pictureName): Debes cargar la imagen con el nombre recibido en el argumento y desplegarla en la interfaz (recuerda que la interfaz es de tipo IPictureRenderer).

void ApplyFilter(String filterName): Debes aplicar el filtro identificado por el argumento a la última imagen seleccionada y desplegarla en la interfaz. El nombre del filtro será alguno de los que previamente registraste llamando a void AddFilter(String filterName).

- ## Parte 4

Tu programa ya es útil al momento. Sin embargo, aún le falta bastante para ser Photoshop.

Para darle alguna funcionalidad más, agregarás a tu programa la capacidad de combinar múltiples filtros bajo un mismo nombre mediante el uso de Cañerías.

Las cañerías, como su nombre indica, se asemejan a las cañerías de agua que se usan en plomería. Las cañerías se componen de múltiples tramos, de diversas formas, con bifurcaciones, que se ensamblan para guiar el agua a su destino de forma adecuada. Incluso, a lo largo de la cañería pueden agregarse filtros de agua, bombas, medidores, etc.

Debes aplicar la misma idea para tu programa, tomando las imágenes como el agua y las cañerías como objetos de un tipo definido por la siguiente interfaz:

```
public interface IPipe
{
    void Send(IPicture picture);
}
```

Como una cañería de agua, lo único que tu cañería debe hacer es recibir la imagen. Cual es su destino y que pasa con ella, es problema de cada cañería.
Para ensamblar dos cañerías, la primer cañería es responsable de conocer cual es la siguiente cañería, y luego es responsable de enviar la imagen a la siguiente cañería luego de recibirla.
Debes programar las siguientes cañerías:

- Una cañería que reciba una imagen, le aplique un filtro, y envíe el resultado a la siguiente cañería.
- Una cañería que reciba una imagen, construya un duplicado de la misma, y envíe la original por una cañería y el duplicado por otra.
- Una cañería que solo reciba una imagen, pero no la envíe a otra cañería.

Para poder usar estas cañerías en tu programa, deberás programar dos filtro adicionales.
El primero, debe recibir una imagen, enviarla a una cañería, y retornar la misma imagen.
El segundo, debe recibir la imagen, desplegarla en una nueva ventana (usando, por ejemplo, una instancia de PictureFormRenderer) y retornar la misma imagen.

• Parte 5

Agregar soporte para especificar cañerías y los nuevos filtros en el archivo de tags, siguiendo el siguiente formato sugerido:

```
<filter-pipe name=”Proceso 1” pipe=”pipe-1”/>
```

Define un filtro llamado `Proceso 1` que envía la imagen por la cañería llamada `pipe-1`.

```
<filter-render name=”desplegar”/>
```

Define un filtro llamado `desplegar` que despliega una imagen en una nueva ventana.

```
<pipe-serial name=”pipe-1” filter=”suavizado” next=”pipe-2”/>
```

Define una cañería de nombre `pipe-1` que recibe una imagen, le aplica el filtro con nombre `suavizado`, y la envía a la cañería `pipe-2`.

```
<pipe-fork   name=”pipe-2” next1=”pipe-3” next2=”pipe4”/>
```

Define una cañería de nombre `pipe-2` que recibe una imagen, crea un duplicado de la misma, y envía una copia por la cañería `pipe-3` y otra por la cañería `pipe-4`.

```
<pipe-null name=”pipe-null”/>
```

Define una cañería de nombre `pipe-null` que recibe una imagen y no hace nada.

El siguiente es un archivo de ejemplo:

```
<filter-render name=”desplegar”/>
<filter-blur name=”suavizado”/>
<filter-blackwhite name=”blanco y negro” threshold=”99”/>
<filter-negative name=”negativo”/>
<pipe-null name=”pipe-null”/>
<pipe-serial name=”pipe-4” filter=”negativo” next=”pipe-5”/>
<pipe-serial name=”pipe-5” filter=”desplegar” next=”pipe-null”/>
<pipe-serial name=”pipe-3” filter=”blanco y negro” next=”pipe-5”/>
<pipe-fork   name=”pipe-2” next1=”pipe-3” next2=”pipe4”/>
<pipe-serial name=”pipe-1” filter=”suavizado” next=”pipe-2”/>
```

```
<filter-pipe name="Proceso 1" pipe="pipe-1"/>
```

El archivo especifica 5 filtros que deben mostrarse, al igual que en el ejercicio 3, en la interfaz provista.

Cuando se selecciona el filtro llamado `Proceso 1` se toma la imagen actual y se envía por la cañería `pipe-1`. Ésta a su vez aplica un filtro de suavizado sobre la imagen (ver parte 5), y la envía a `pipe-2`, que crea un duplicado de la imagen (ya suavizada) y envía una copia a `pipe-3` y otra a `pipe-4`, que aplican el filtro `blanco y negro` y `negativo` respectivamente, y luego envían la imagen a `pipe-5` que despliega la imagen para luego mandarla a `pipe-null` y terminar el procesamiento.

El archivo nunca hace referencia a un filtro o cañería que aun no haya sido declarado. Ésto permite que proceses el archivo más fácilmente. Si quieres eliminar ésta restricción para que el archivo sea más fácil de leer, hazlo.

Bonus

En la siguiente sección se listan diferentes agregados que puedes hacer a tu entrega para ayudar a complementar otros puntos débiles. De todas formas, ten en cuenta que lo más importante es el diseño de tu aplicación siguiendo los lineamientos de la materia.

Más filtros

Agrega nuevos filtros. Cuanto más filtros originales agregues, mejor!

Definir macros

Un macro es un conjunto de acciones realizadas en la interfaz de usuario que se pueden guardar asignándoles un nombre para luego realizarlas más fácilmente en un único paso.

El uso de macros es el siguiente:

1. Se carga una imagen en la interfaz
2. Se selecciona la opción Grabar Macro (Record) de la interfaz. Esto muestra una pantalla que permite ingresar el nombre del macro.
3. Luego se van seleccionando uno a uno filtros para ejecutar sobre la imagen
4. Cuando se seleccionaron todos los filtros necesarios, se selecciona Detener Grabación de Macro (Stop Macro Recording).
5. Aquí se debe registrar un nuevo filtro en la interfaz con el nombre ingresado en el paso 2, que cuando es seleccionado aplica todos los filtros aplicados en el paso 3, en el orden en que fueron seleccionados.

Si no hay una imagen seleccionada no se puede grabar un macro.

Si se esta grabando un macro y se vuelve a seleccionar la opción de grabar macro, se borra el que se estaba grabando y se empieza a grabar uno nuevo.

Si se ingresa como nombre del macro el nombre de un filtro existente, se debe desplegar un error (por ejemplo, mediante System.Windows.Forms.MessageBox.Show(String)).

La interfaz de usuario notificará a un observador cuando un usuario seleccione grabar macro o detener grabación. El observador debe estar registrado con la interfaz de usuario e implementar la siguiente interfaz:

```
public interface IMacroListener
{
    void RecordMacro(String macroName);
    void StopRecordMacro();
}
```

Persistir Imágenes

La interfaz provee una nueva opción en el menú para grabar la imagen actual. Cuando recibe se selecciona la opción, envía un mensaje a todos los observadores registrados con la interfaz de usuario que implementan la siguiente interfaz:

```
public interface IPicturePersistListener
{
    void PersistPicture(String fileName);
}
```

Para grabar la imagen pueden usar una clase que les proveemos que implementa la siguiente interfaz:

```
public interface IPicturePersister
{
    void Persist(IPicture picture, String fileName);
}
```

Persistir Filtros y Macros

La interfaz provee un botón que permite grabar todos los filtros y macros actualmente usados en el programa a un archivo de configuración de filtros.

El archivo luego debe poder ser cargado como un archivo de configuración de filtros.

La interfaz de usuario notificará a un observador cuando un usuario seleccione grabar filtros. El observador debe estar registrado con la interfaz de usuario e implementar la siguiente interfaz:

```
public interface IFilterPersistListener
{
    void PersistFilters(String fileName);
}
```