Práctica 5

Curso 2020 – 2021

Descripción de la aplicación.

En esta práctica desarrollaremos una aplicación que nos permitirá realizar diferentes operaciones de tratamiento digital de imágenes.

Una imagen está compuesta por píxeles, distribuidos en una cuadrícula de *F* filas x *C* columnas, de forma que cada píxel viene caracterizado por su color. El color está constituido, aplicando el sistema RGB (Red, Green, Blue) con 8 bits de profundidad, por 3 valores enteros en el rango [0,255], uno para cada capa de color. El color rojo puro tiene el valor (255,0,0). De la misma forma el verde puro es (0,255,0) y el azul puro (0,0,255). Variando el valor de estas tres capas dentro del rango indicado podemos obtener cualquier color de la paleta de 24 bits (la que habitualmente se utiliza). Puede jugar con los colores usando la paleta de Word (*Más Colores…* → *Personalizado*)

En la figura 1 se muestra una imagen ejemplo de 5x4 (20) píxeles, con indicación numérica de los valores de color de cada pixel y el color correspondiente de fondo:

Fila/Columna	0	1	2	3
0	(198,217,241)	(198,217,241)	(198,217,241)	(198,217,241)
1	(198,217,241)	(255,255,0)	(198,217,241)	(255,255,0)
2	(198,217,241)	(198,217,241)	(198,217,241)	(198,217,241)
3	(255,0,0)	(198,217,241)	(198,217,241)	(255,0,0)
4	(84,141,212)	(255,0,0)	(255,0,0)	(84,141,212)

Figura 1. Ejemplo de imagen en color de 5x4 píxeles.

Para referirnos a los píxeles usaremos una dupla formada por dos índices que señalan la fila y columna de confluencia. Así por ejemplo los píxeles (1,1) y (1,3) de la imagen de la figura 1 tienen color amarillo (255,255,0).

Para gestionar las imágenes se ofrecen ya codificadas 2 clases: “ImagenRGB” y “ColorRGB”, dentro del paquete “imagenes" que se obtendrá tras la importación del fichero “gestionImagenes.jar”. Dispone de instrucciones precisas para la instalación de un archivo JAR en un anexo al final de este enunciado.

La relación entre las 2 clases anteriores, junto a su funcionalidad, se recoge en el diagrama UML simplificado de la figura 2:

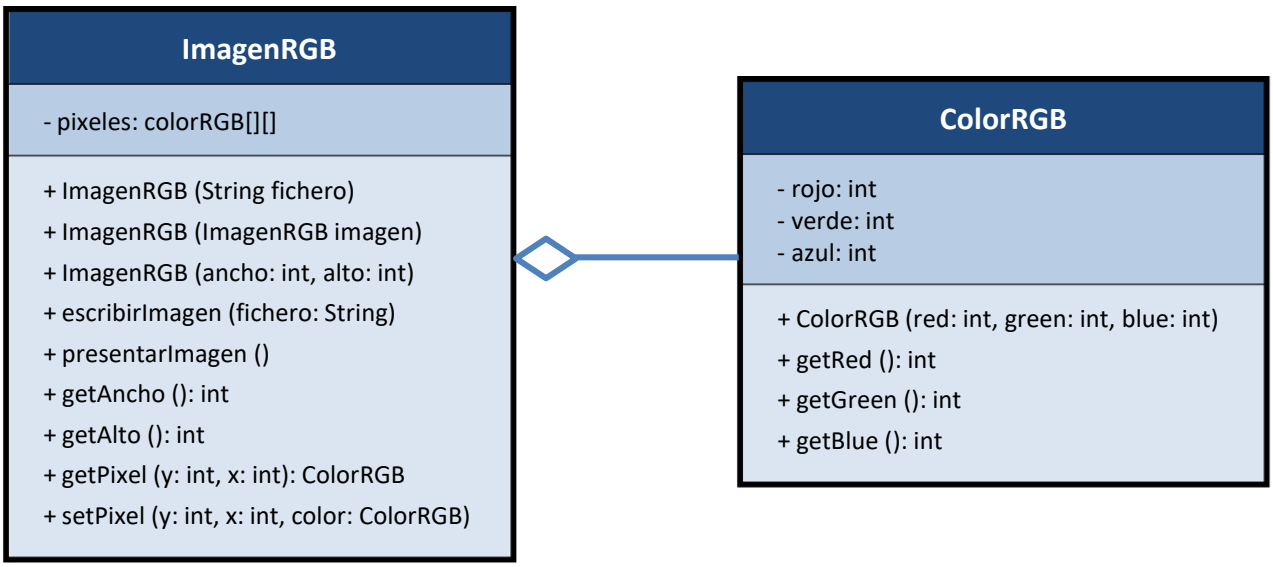

Figura 2. Diagrama UML simplificado para las clases *ImagenRGB* y *ColorRGB*.

La clase “ColorRGB” permite definir el color de los píxeles según el sistema RGB. Los objetos de esta clase son **inmutables**, es decir, una vez generado un color, este no puede modificarse. De esta forma, no es importante que los elementos del *array* que caracteriza la clase “ImagenRGB” contengan todos ellos objetos diferentes (con referencias distintas): si dos píxeles tuviesen el mismo color, las posiciones correspondientes del *array* podrían apuntar al MISMO objeto de color. Si la clase “ColorRGB” fuese **mutable** el hecho que acabamos de describir no sería admisible, ya que si cambiásemos el color de un píxel, variando los atributos de su objeto “ColorRGB” asociado, automáticamente se verían modificados aquellos píxeles que contuviesen la misma referencia que el retocado.

En cuanto a la clase “ImagenRGB”, observe que cuenta con 3 constructores: el primero permite generar un objeto “ImagenRGB” a partir de la imagen contenida en un fichero; el segundo genera una instancia de “ImagenRGB” a partir de otra instancia de la misma clase, es decir **realiza una copia** (evidentemente con distinta referencia) de la segunda instancia; el tercero permite crear una imagen con unas dimensiones determinadas donde el valor de color de los píxeles se generará de forma aleatoria. El resto de métodos de la clase se definen por si solos. Quizás merezca la pena detenerse en “presentarImagen” para aclarar que la presentación se realiza en pantalla usando una ventana emergente de Windows.

El objetivo de la práctica es generar una batería de efectos que puedan ser aplicados sobre una imagen con el fin de procesarla. Dadas las limitaciones de tiempo que tenemos nos centraremos en la implementación de 3 efectos: **Espejo** (vertical u horizontal), que dará como resultado una reflexión de la imagen con respecto a la línea central de la misma (línea vertical u horizontal en consonancia con el efecto); **Marco**, que nos permitirá crear un marco alrededor de la imagen, con un grosor y un color determinados; y **Marco estilo foto**, que resultará en un marco compuesto de 3 zonas: la más cercana al borde, de un color indicado y grosor de 10 píxeles; la central de ancho 6 píxeles y color blanco; y una tercera más próxima al centro de la imagen con grosor 4 píxeles y mismo color que la primera.

Las figuras 3 a 5 muestran ejemplos de estos 3 tipos de efectos.

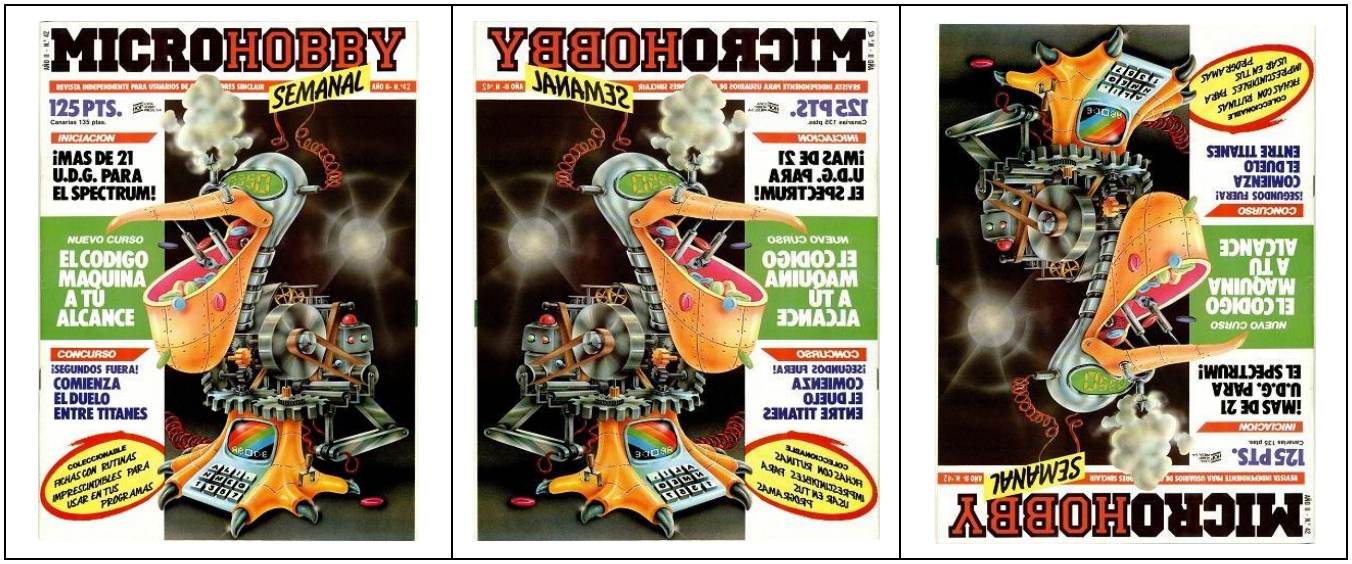

Figura 3. Izda: imagen original. Centro: espejado vertical. Dcha: espejado vertical seguido de espejado horizontal.

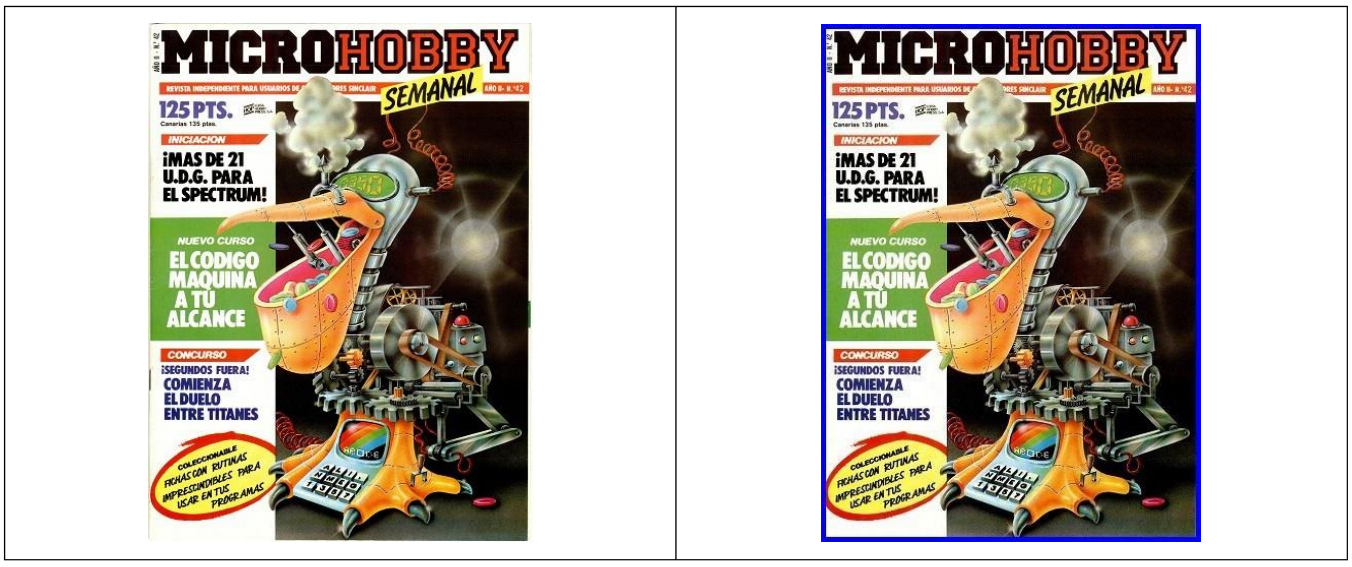

Figura 4. Izda: imagen original. Dcha: imagen enmarcada.

Figura 5. Izda: imagen original. Dcha: imagen enmarcada estilo foto.

La figura 6 recoge por su parte el diagrama UML que describe el bloque de efectos. Todas las clases que componen la estructura deberán integrarse en un paquete denominado “efectos”.

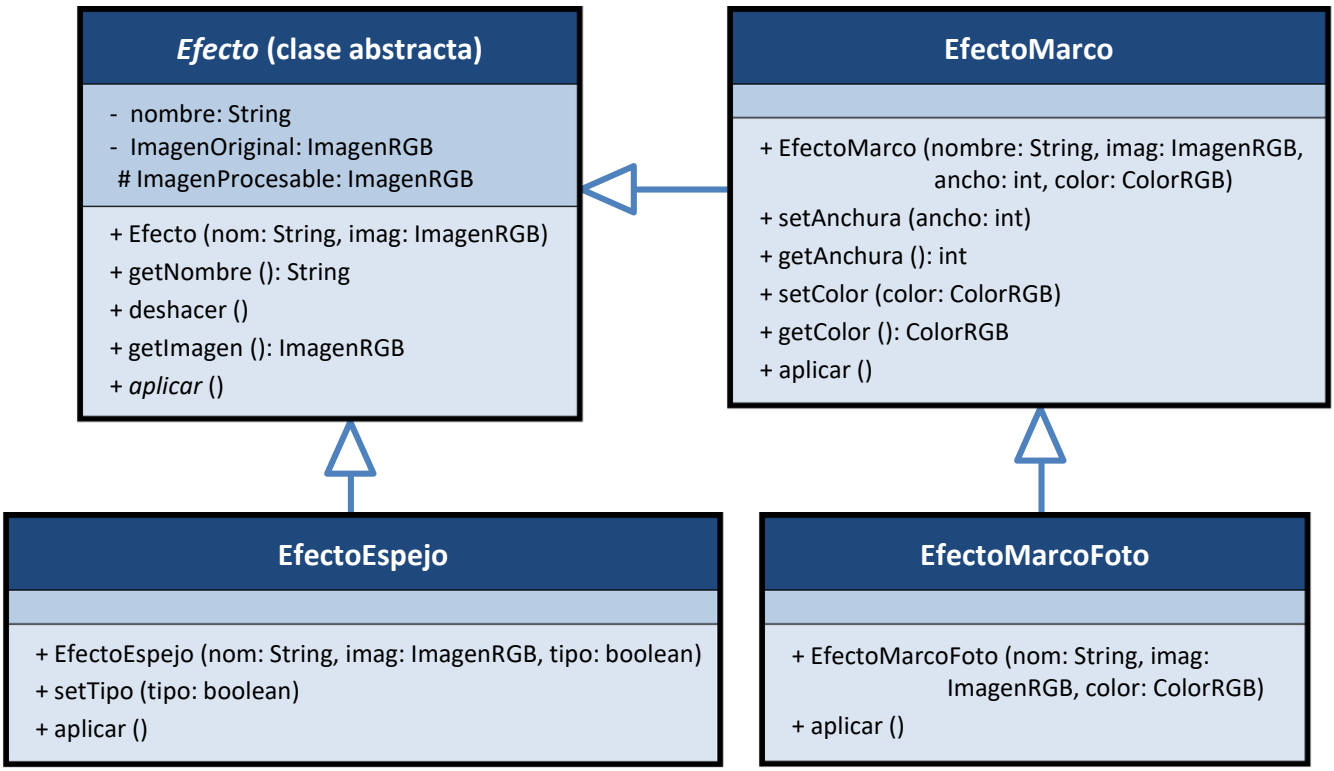

Figura 6. Diagrama UML simplificado del bloque de efectos.

Clase "*Efecto*":

Se trata de una clase abstracta que actúa como superclase, garantizando que todas las subclases implementan el método "aplicar".

Según la estructura diseñada la propia clase debe contener como atributo la imagen a procesar ("ImagenProcesable"). Además, por motivos de eficacia, el tratamiento de "ImagenProcesable" en las subclases se realizará accediendo directamente a esta imagen, por lo que se recomienda que el atributo correspondiente sea definido como *protected*, de forma que las subclases puedan acceder directamente a la imagen, pero se impida el acceso a dicho atributo desde cualquier otra clase que no pertenezca al paquete "efectos".

El método "getImagen" debe entregar **una copia** de la imagen procesada, almacenada siempre en el atributo "ImagenProcesable", ocultando así la referencia directa para evitar que cualquier objeto que llame al método pueda modificar el valor de los píxeles de la imagen contenida en la clase "*Efecto*".

Una particularidad del método "aplicar" es que este podrá ser invocado varias veces, realizando efectos diferentes sobre la imagen en función del valor de los atributos de la subclase. Por tanto, con una única instancia de una de las subclases podremos aplicar distintos procesados sobre la imagen, realizando varias llamadas al método "aplicar".

El método "deshacer" permite **retroceder TODOS los efectos** que se hayan realizado sobre la imagen, debidos a las diversas invocaciones que haya tenido el método "aplicar". En otras palabras, el método "deshacer" permite colocar de nuevo como procesable **una copia** de la imagen original que se usó al crear la instancia de la subclase (la imagen original se encuentra permanentemente almacenada en el atributo "ImagenOriginal")

Por ejemplo, imaginemos que se crea un objeto de la subclase "EfectoEspejo" y se llama 2 veces al método "aplicar", variando el atributo "tipo" en el medio. La imagen resultante habrá sido espejada dos veces: primero en vertical, y en una segunda llamada en horizontal. Si después de esto invocásemos a "deshacer", se anularían los dos efectos y volveríamos a disponer de una copia de la imagen original (la que usamos al crear la instancia) como imagen procesable. A partir de este momento, una nueva llamada a "aplicar" realizaría el tratamiento comenzando desde cero, desde dicha copia de la imagen inicial.

Clase "EfectoEspejo":

Con esta clase podemos aplicar un efecto espejo sobre la imagen, horizontal o vertical dependiendo del parámetro "tipo": con *true* el espejado es vertical, con *false* horizontal.

Como ya hemos visto, cabría la posibilidad de aplicar ambos espejados. El proceso sería el siguiente: creamos la instancia con "tipo = *true*", invocamos "aplicar", modificamos "tipo = *false*" y volvemos a invocar "aplicar".

Clase “EfectoMarco”:

Usando esta clase podemos encuadrar la imagen con un marco del color y ancho indicados en el constructor.

Como en el caso anterior podemos dibujar varios marcos de distintas anchuras y colores llamando repetidas veces al método “aplicar”. En este caso debe tener en cuenta que todos los marcos parten del borde de la imagen, por lo que se sobrepondrán unos a otros.

Tenga en cuenta que para que el marco entre correctamente en la imagen este no puede tener un grosor superior a la mitad del valor más pequeño de entre el ancho y el alto de la imagen. La clase debe controlar este aspecto.

Clase “EfectoMarcoFoto”:

Usando esta clase podemos encuadrar la imagen con un marco triple: la parte más cercana al borde del color indicado en el constructor y grosor de 10 píxeles, la zona central de ancho 6 píxeles y color blanco, y una tercera parte más próxima al centro de la imagen con grosor 4 píxeles y mismo color que la primera. Para realizar el proceso **debemos usar** los componentes heredados de la clase “EfectoMarco”.

Una descripción más detallada de todas las clases (con sus correspondientes métodos) se suministra mediante el *javadoc*, al que se puede acceder desde el Moodle de la asignatura, junto al enunciado de esta práctica.

IMPORTANTE: para aprovechar la sesión de trabajo de laboratorio, es imprescindible haber realizado previamente la lectura detallada del enunciado y del *javadoc* de la práctica.

Desarrollo de la práctica

Se recomienda seguir los siguientes pasos:

1. Crear un proyecto eclipse llamado ***practica5***.
2. Crear los paquetes “efectos” y “usuario”, que albergarán las distintas clases.
3. Importar el archivo “gestionImagenes.jar” que deberá generar el paquete “imagenes” con las clases “ColorRGB” e “ImagenesRGB”.
4. Codificar la clase abstracta “*Efecto*”.
5. Codificar las subclases “EfectoEspejo” y “EfectoMarco”.
6. Implementar la subclase “EfectoMarcoFoto”.
7. Cumplimentar la clase *P5Aplicacion*, que se ofrece como plantilla y que deberá ser incluida en el paquete “usuario”.

Anexo: Importación de un archivo JAR

Los archivos .jar se utilizan en Java para empaquetar y distribuir ficheros binarios (.class). En esta práctica se le proporciona el paquete "gestionImagenes.jar", con clases que le permitirán trabajar con imágenes en formato RGB. Para hacer uso de estas clases deberá incorporar el archivo .jar a su proyecto de Eclipse como "fichero externo". El proceso recomendado es el siguiente:

1. Pinchar con el botón derecho del ratón en el proyecto Java situado en el "explorador de paquetes" de Eclipse.
2. Acceder al submenú "Build Path" y dentro de este a "Configure Build Path".
3. En la ventana emergente (Java Build Path) seleccionar la pestaña "Libraries".
4. Presione el botón "Add external JARs".

5. Navegue hasta el directorio donde tiene guardado el archivo JAR (se recomienda que lo sitúe en el directorio de trabajo del proyecto), selecciónelo y pulse el botón “Abrir”.
6. Pulse el botón “Apply and Close” de la ventana “Java Build Path”.
7. El paquete le aparecerá dentro de un bloque denominado “Referenced Libraries” que cuelga de su proyecto.
8. El explorador de paquetes de Eclipse le permite navegar por las distintas clases que contiene el JAR y explorar sus componentes, pero de ninguna forma podrá acceder al código con el que fueron implementados.

Puede ampliar esta información en el siguiente enlace:

https://es.wikihow.com/a%C3%B1adir-un-jar-a-un-proyecto-en-eclipse-%28java%29

Si sigue este manual, le aconsejamos que utilice el método 3 del apartado “Importación de un .jar externo”, que es precisamente el que le hemos descrito en los pasos anteriores.