Tratamiento de imágenes en Java

El API de Java dispone de varias clases para trabajar con imágenes, algunas ya obsoletas disponibles en paquetes como AWT.

Además, también existen bibliotecas externas más completas, como ***ImageJ***, ***OpenIMAJ*** o ***TwelveMonkeys***.

AWT/SWING

AWT fue la primera biblioteca Java integrada en el API que permitía al usuario realizar operaciones simples relacionadas con la visualización, como crear ventanas, listeners (oyentes) o componentes visuales ya desaprobados. Además, incluye métodos que permiten al usuario editar y trabajar imágenes.

Cargar una imagen con ***BufferedImage***

Lo primero es crear un objeto de tipo *BufferedImage* de una archivo imagen del disco:

```
String rutaImagen = "ruta/imagen.jpg";
BufferedImage miImagen = ImageIO.read(new File(rutaImagen));
```

BufferedImage es hereda de ***Image***:

Package java.awt.image

Class BufferedImage

java.lang.Object
    java.awt.Image
        java.awt.image.BufferedImage

All Implemented Interfaces:

RenderedImage, WritableRenderedImage, Transparency

También podría crearse por medio de alguno de sus constructores:

```
public BufferedImage(int width, int height, int imageType)
```

Editando la imagen. Dibujando figuras.

Para dibujar una figura en una imagen, tendremos que usar el **objeto *Graphics*** relacionado con la imagen cargada:

```
miImagen.getGraphics();
```

El objeto *Graphics* encapsula las propiedades necesarias para realizar operaciones básicas de representación.

Graphics2D es una clase que hereda de *Graphics* y proporciona más control y posibilidades sobre las formas bidimensionales.

Para imágenes 2D necesitamos **convertir el objeto gráfico en un *Graphic2D***. En el siguiente ejemplo cambiamos el acho del pincel (Stroke), cambiamos el color para dibujar a azul y dibujamos un rectángulo en la posición 10,10 con ancho de la imagen menos 20 píxeles:

```
Graphics2D g = (Graphics2D) miImagen.getGraphics();
g.setStroke(new BasicStroke(3));
g.setColor(Color.BLUE);
g.drawRect(10, 10, miImagen.getWidth() - 20, miImagen.getHeight() - 20);
```

Mostrando la imagen

Ahora que hemos dibujado un rectángulo en nuestra imagen, nos gustaría mostrarlo. Podemos hacerlo usando objetos de la biblioteca Swing.

Primero, **creamos un objeto *JLabel*** que representa un área de visualización para texto y/o imagen y que tiene un constructor que recoge un objeto de tipo Icon (ImageIcon):

```
JLabel lblImaxe = new JLabel(new ImageIcon(miImagen));
```

Podemos añadir la imagen a un JPanel:

```
JPanel panel = new JPanel();
panel.add(lblImaxe);
```

A modo de ejemplol, añadimos todo a *JFrame*, que es una ventana que se muestra en una pantalla. Tenemos que establecer el tamaño para que no tengamos que expandir esta ventana cada vez que ejecutamos nuestro programa:

```
JFrame f = new JFrame();
f.setSize(new Dimension(
      miImagen.getWidth(), miImagen.getHeight()));
f.add(panel);
f.setVisible(true);
```

IMAGEJ

ImageJ es un software basado en Java creado para trabajar con imágenes. Tiene bastantes plugins, pero sólo emplearemos la biblioteca para realizar el procesamiento por nosotros mismos.

Dependencia Maven

```
<dependency>
      <groupId>net.imagej</groupId>
      <artifactId>ij</artifactId>
      <version>1.51h</version>
</dependency>
```

Cargar la imagen

Para cargar la imagen se debe usar el **método estático *openImage()***, *de la clase IJ*:

```
ImagePlus imp = IJ.openImage("ruta/a/imagen/imagen.jpg");
```

Edición de la imagen

Para editar una imagen, tendremos que usar métodos del objeto de tipo ***ImageProcessor*** adjunto a nuestro objeto *ImagePlus*, que es similar al objeto *Graphics* en AWT:

```
ImageProcessor ip = imp.getProcessor();
```

```
ip.setColor(Color.BLUE);
ip.setLineWidth(4);
ip.drawRect(10, 10, imp.getWidth() - 20, imp.getHeight() - 20);
```

Mostrar la imagen

Sólo se precisa llamar al método ***show()***:

```
imp.show();
```

OPENIMAJ

OpenIMAJ es un **conjunto de bibliotecas de Java** enfocadas no solo en la **visión por computadora o el procesamiento de video**, sino también en el aprendizaje automático, el procesamiento de audio, el trabajo con Hadoop y mucho más.

Todas las partes del proyecto OpenIMAJ se pueden encontrar, en el apartado "Módulos". En este caso sólo necesitamos la parte de procesamiento de imágenes.

Dependencia Maven

https://search.maven.org/search?q=g:org.openimaj%20AND%20a:core-image

```
<dependency>
	<groupId>org.openimaj</groupId>
	<artifactId>core-image</artifactId>
	<version>1.3.10</version>
</dependency>
```

Cargar la imagen

Para cargar una imagen emplearemos el método estático *ImageUtilities.readMBF()*:

```
MBFImage imaxe
	= ImageUtilities.readMBF(new File("ruta/a/imagen/img.jpg"));
```

MBF significa ***imagen de punto flotante multibanda*** (RGB en este ejemplo, pero no es la única forma de representar colores).

Editando una imagen

Para dibujar un rectángulo necesitamos definir su forma, que es un polígono que consta de 4 puntos (arriba a la izquierda, abajo a la izquierda, abajo a la derecha, arriba a la derecha):

```
Point2d arribaIzq = new Point2dImpl(10, 10);
Point2d abajoIzq = new Point2dImpl(10, image.getHeight()-10);
Point2d abajoDer
      = new Point2dImpl(image.getWidth()-10,
            image.getHeight()-10);
Point2d arribaDer = new Point2dImpl(image.getWidth()-10, 10);
Polygon poligono =
      new Polygon(Arrays.asList(arribaIzq,
            abajoIzq, abajoDer, arribaDer));
```

En el procesamiento de imágenes, el eje Y está invertido. Después de definir la forma, necesitamos dibujarla:

```
imaxe.drawPolygon(poligono, 4, new Float[] { 0f, 0f, 255.0f });
```

El método de dibujo toma 3 argumentos: la figura, grosor de línea y valores de canal RGB representados por la matriz *flotante.*

Mostrar imagen

Necesitamos emplear el método estático ***display*** de *DisplayUtilities*:

```
DisplayUtilities.display(imaxe);
```

TWELVEMONKEYS IMAGEIO

La biblioteca *TwelveMonkeys ImageIO está* pensada como una **extensión de la API de Java *ImageIO***, con soporte para una mayor cantidad de formatos.

La mayoría de las veces, el código tendrá el mismo aspecto que el código Java incorporado, **pero funcionará con formatos de imagen adicionales**, después de agregar las dependencias necesarias.

De forma predeterminada, **Java admite cinco formatos para imágenes: *JPEG* , *PNG* , *BMP* , *WEBMP* , *GIF*** .

Si intentamos trabajar con un archivo de imagen en un formato diferente, nuestra aplicación no podrá leerlo y lanzará una *NullPointerException* al acceder a la variable *BufferedImage.*

TwelveMonkeys agrega soporte para los siguientes formatos:

PNM, *PSD*, *TIFF*, *HDR*, *IFF*, *PCX*, *PICT*, *SGI*, *TGA*, *ICNS*, *ICO*, *CUR*, *Thumbs.db*, *SV*, *WMF* .

Para trabajar con imágenes en un formato específico, necesitamos agregar la dependencia correspondiente , como imageio-jpeg o imageio-tiff.

La lista completa de dependencias se muestra en la documentación de TwelveMonkeys.

Vamos a crear un ejemplo que lea una imagen *.ico*. El código tendrá el mismo aspecto que la sección *AWT*/Swing, excepto que abriremos un tipo de imagen diferente:

```
String ruta = "ruta/a/imagen/imaxe.ico";
BufferedImage miImagen = ImageIO.read(new File(ruta))
```

Dependencia Maven

Para que este ejemplo funcione, necesitamos agregar la dependencia *TwelveMonkeys* que contiene soporte para imágenes *.ico*, que es la dependencia imageio-bmp, junto con la dependencia imageio-core:

```
<dependency>
	<groupId>com.twelvemonkeys.imageio</groupId>
	<artifactId>imageio-bmp</artifactId>
	<version>3.3.2</version>
</dependency>
<dependency>
	<groupId>com.twelvemonkeys.imageio</groupId>
	<artifactId>imageio-core</artifactId>
	<version>3.3.2</version>
</dependency>
```

Ahora la API *ImageIO* Java integrada carga los complementos automáticamente en tiempo de ejecución. Ahora nuestro proyecto también funcionará con imágenes *.ico*.