Exercise NMCGJ:
Image Processing

A digital picture (or image) is internally stored as an array or a matrix of pixels (= picture elements), each of them containing a specific color. This exercise is devoted to perform image processing.

Colors

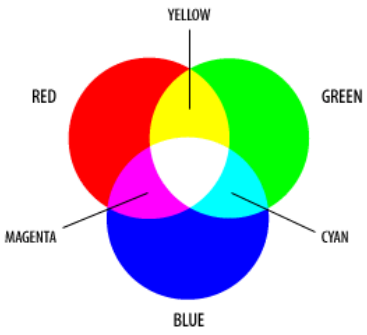
In a (8-bit) grayscale image each pixel has an assigned intensity that ranges from 0 to 255. Such an image is what people usually call a black-and-white image, but the name emphasizes that such an image also includes many shades of grey.

A true color image usual has 24 bit color depth = 8x8x8 bits = 256x256x256 colors ~ 16 million colors. For science communication, the two main color spaces are RGB and CMYK.

RGB

The RGB color relates very closely to the way we perceive color with the red (R), green (G) and blue (B) receptors in our retinas in the eyes. RGB uses additive color mixing. It is the basic color model used in television or any other medium that projects color with light. It is also the basic color model used in computers and graphics, but it is not used for print production.

The secondary colors of RGB – cyan, magenta and yellow – are formed by mixing two of the primary colors (red, green or blue) and excluding the third color. Red and green combine to make yellow. Green and blue form cyan; and blue and red form magenta. The combination of red, green and blue in full intensity makes white.



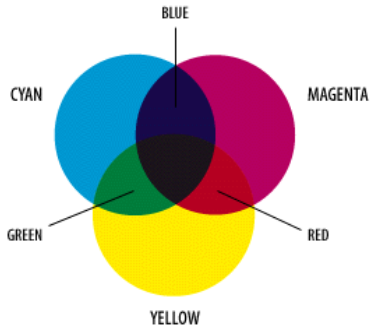
A general color in Java can be created as follows:

```
Color color = new Color(red, green, blue);
```

where `red`, `green` and `blue` have to be integer values between 0 and 255. The class `Color` is part of the package `java.awt`. It has the methods `getRed()`, `getGreen()`, and `getBlue()`.

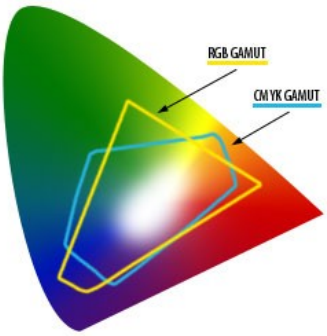
CMYK

The four-color CMYK color model used in printing works by overlapping layers of varying percentages of transparant cyan (C), magenta (M) and yellow (Y) inks on a white background. In addition, a layer of black (K) ink can be added. The CMYK model uses the subtractive color model because inks “subtract” brightness from white.



Gamut

The range, or gamut, of human color perception is quite large. The two color spaces discussed here span only a fraction of the colors we can see. Furthermore, the two spaces do not have the same gamut. This means that converting from one color space to the other may cause problems for colors in the outer regions of the gamuts.



Problem

The classes `Picture` and `PictureFrame` are prepared for visualizing images. The class `PictureFrame` is a subclass of the class `GraphicsFrame`, so it provides all the original graphical functionality of this class (see the previous exercise *Drawing Fractal Patterns*). In addition, it has specific functionality to visualize and manipulate pictures, which is accessible through the menu bar.

drawPicture()	Draws the picture to the graphics panel.
processPicture(int index)	Performs picture processing. The index specifies the processing action.
String[] getPictureActionNames()	Returns a description of all available picture processing actions.

The real work of the picture processing is delegated to the class `Picture`. The last two methods are designed so that easily new algorithms for picture processing can be added to the menu bar.

The class `Picture` contains internally a matrix of colors:

```
protected Color[][] pixels;
```

The following methods are available in this class.

getWidth()	Gets the width of the picture.
getHeight()	Gets the height of the picture.
getPixel(int i, int j): Color	Gets the color of a given pixel.
setPixel(int i, int j, Color color)	Sets the color of a given pixel.

Additional methods should be implemented in the class `Picture` for performing some picture processing on the internal matrix of colors.

Recoloring

A first way to manipulate pictures is by modifying the pixel colors. Several basic operations can be done by taking certain linear combinations of the RGB color components of each pixel.

Conversion to grayscale
Luminance is a measure to describe the perceived brightness of a color. Suppose a pixel is represented by the triple (R, G, B) of intensities for red, green, and blue. The luminance of this pixel is obtained by the formula

$$Y = 0.2126\,R + 0.7152\,G + 0.0722\,B.$$

Conversion of a color image to grayscale is not unique, but in general it replaces each RGB color component by an averaged version. The above formula can be used for this: replace the red, green and blue component by their luminance value.

Negative image
A negative grayscale image is a totally inversed image, meaning that light areas appear dark and vice versa. A negative color image is additionally color-reversed, with red areas appearing cyan, greens appearing magenta, and blues appearing yellow. This means that each new RGB values is replaced by its complement (255 – value).

Brightening and darkening
Making an image darker or brighter can be simply obtained by multiplying each RGB color component with a constant value less or larger than 1, respectively.

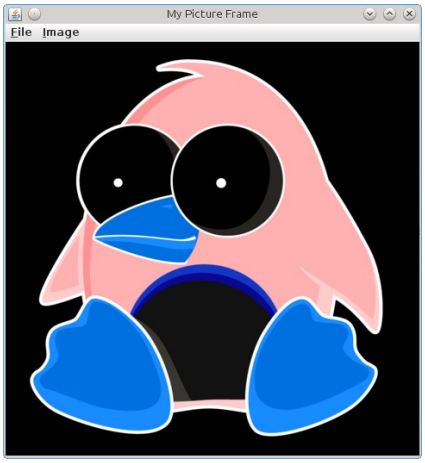
Implementation
Write three methods that take care of these functionalities:

grayscale()	Converts the picture to a grayscale picture.
negative()	Inverts the colors of the picture.
brightness(double factor)	Adjusts the brightness of the picture.

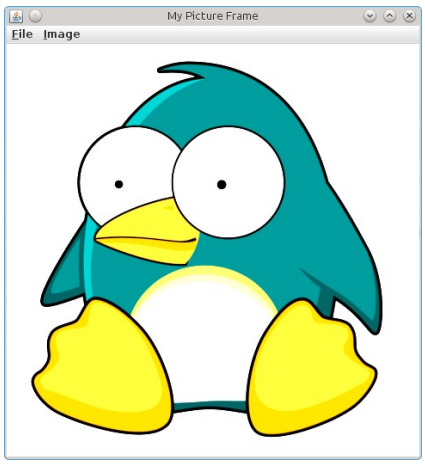
original

grayscale

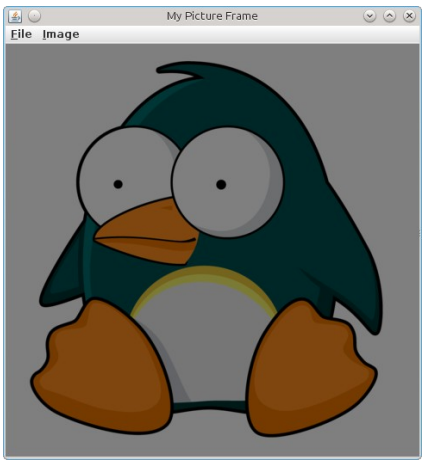
negative



brighter
(f = 2)



darker
(f = 1/2)



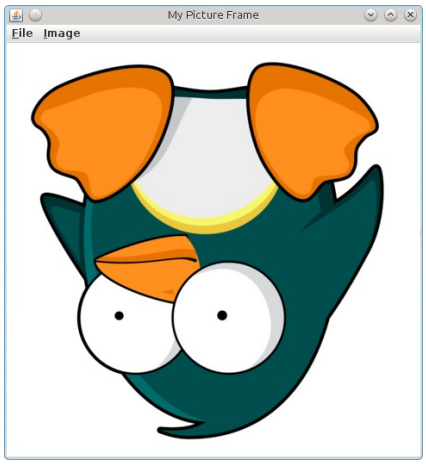
Flipping

A second way to manipulate pictures is by reorganizing the positions of the current pixels. A common manipulation is to flip the picture along the central horizontal or vertical axes.

Write two methods that take care of this functionality:

flipH()	Flips the picture horizontally (along central vertical axis).
flipV()	Flips the picture vertically (along central horizontal axis).

flipH



flipV

Tiling

For the next task, only the result is shown. How to obtain the result, is up to you. This effect is called tiling, and it is sometimes used in television to make a transition to another subject.

Lens effect

A (fish-eye) lens effect is created by mapping an image onto the surface of a (part of a) sphere. The surface, however, does not have to be a perfect sphere as long as the effect looks similar: the image is “zoomed” in the center and “shrinked” at the border. A simple lens effect can be obtained as follows. Suppose the picture is mapped into the unit box [−1, 1] x [−1, 1]. Then, consider any point inside the unit circle, so any point with polar coordinates (r, θ) such that $r \leq 1$. The color of this point is taken as the color of the picture point with polar coordinates (R, θ) where

$$R=(r+d)/2 \qquad d=1-\sqrt{1-r^2}$$

Add a method that takes care of this functionality:

lens()	Creates a lens effect (fish eye).