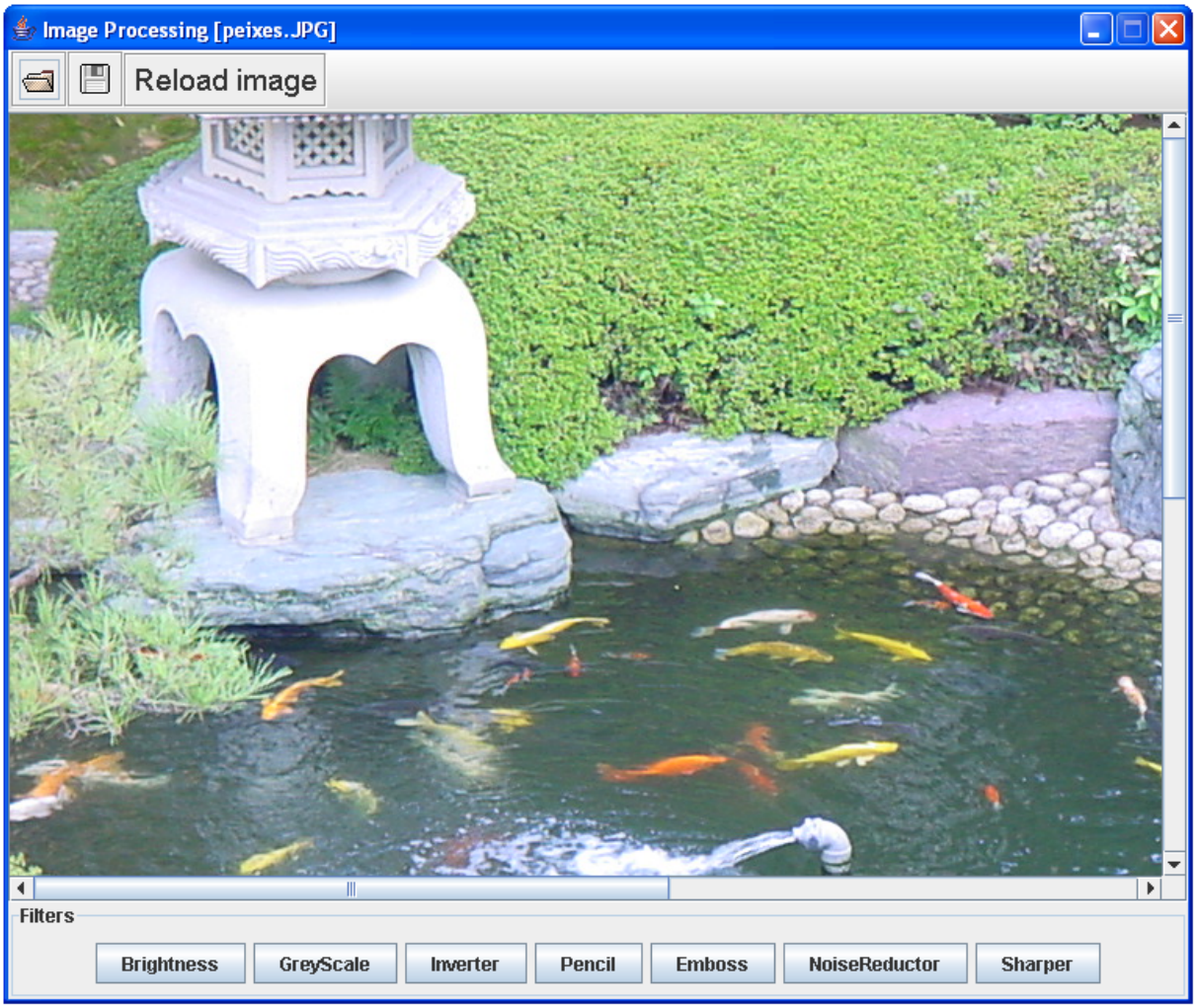
A aplicação **`ImageFilter`**, implementa um conjunto de filtros digitais sobre imagens.

O aspecto gráfico da aplicação é ilustrado na imagem seguinte:



A aplicação tem as seguintes funcionalidades:

- a partir do *toolbar* situado na parte superior, é possível carregar ou salvar imagens localizadas no disco rígido do computador;
- depois de carregada, a imagem é mostrada na parte central da janela;

– na parte inferior, na zona designada por "*Filters*", estão disponíveis botões que permitem manipular a imagem usando os filtros digitais que estão registados, de acordo com o processo explicado a seguir.

Um filtro digital é um filtro que processa sinais digitais – neste caso imagens – implementado em software. O filtro recebe como entrada uma imagem e produz na saída uma nova imagem, que resulta da transformação dos pixéis da imagem fonte. As figuras 1 e 2 ilustram dois exemplos de filtragens típicas.

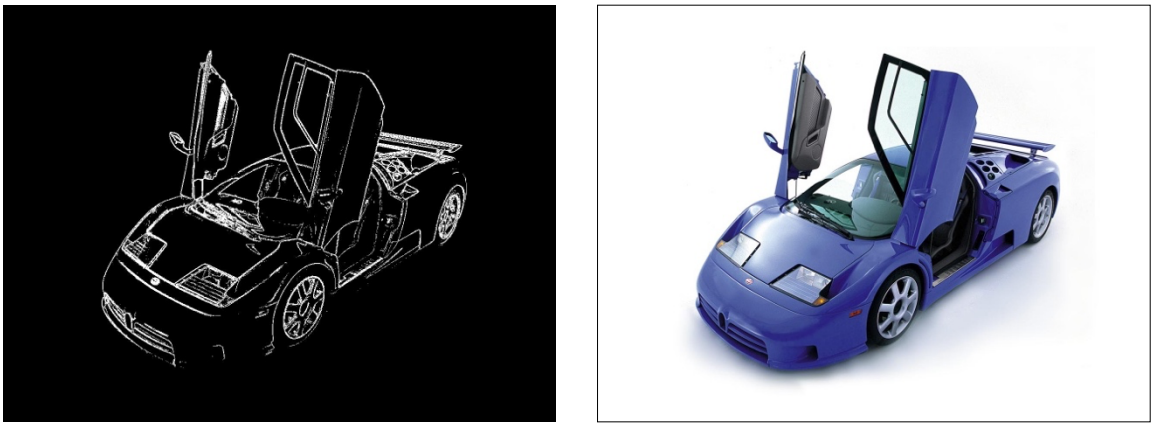


Figura 1 – Resultados obtidos por aplicação do filtro "*Pencil*": imagem original (esquerda); imagem filtrada (direita).

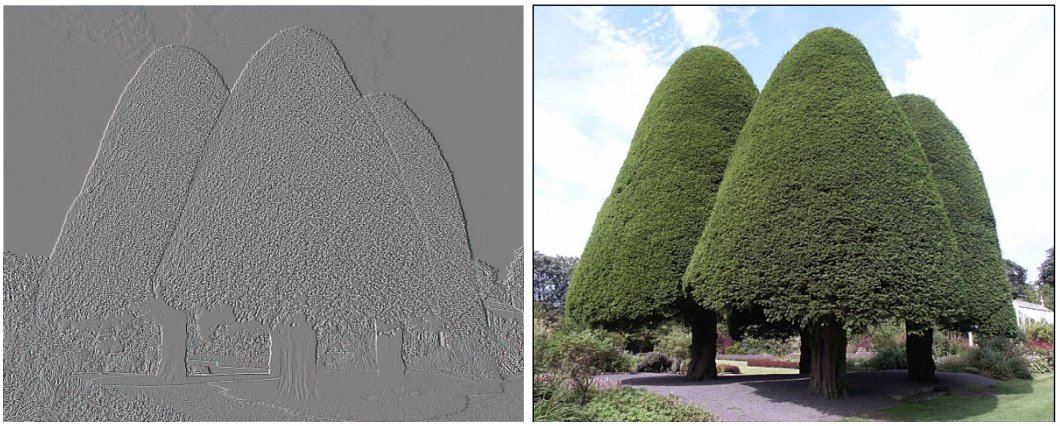


Figura 2 – Resultados obtidos por aplicação do filtro "*Emboss*": imagem original (esquerda); imagem filtrada (direita).

Definição de imagem

Uma imagem consiste numa matriz de pixéis, sendo um pixel um elemento da imagem:

- numa imagem com 256 níveis de cinzento (*greyscale*), cada pixel é representado por um *byte* com valor entre 0 e 255 (0 – preto, 255 – branco, valores intermédios – níveis de cinzento);

- numa imagem policromática *RGB*, cada pixel é representado por três *bytes* correspondendo aos valores das componentes *R*, *G* e *B* (*Red*, *Green* e *Blue*). O modelo *RGB* também permite representar imagens com 256 níveis de cinzento. Basta que todas as componentes *RGB* do pixel assumam o mesmo valor (*R=0*, *G=0*, *B=0* para o preto e *R=255*, *G=255* e *B=255* para o branco, dando os valores intermédios os diferentes níveis de cinzento).

Com o objectivo de simplificar a manipulação de imagens, consideram-se apenas imagens policromáticas, contendo três *bytes* por pixel.

Representação de uma imagem em Java

Uma imagem é representada por um *array* bidimensional de pixéis, onde cada pixel corresponde a um valor inteiro – tipo primitivo **`int`**. As componentes *RGB* correspondem aos três *bytes* menos significativos do valor inteiro que representa o pixel, como ilustrado na figura 3.

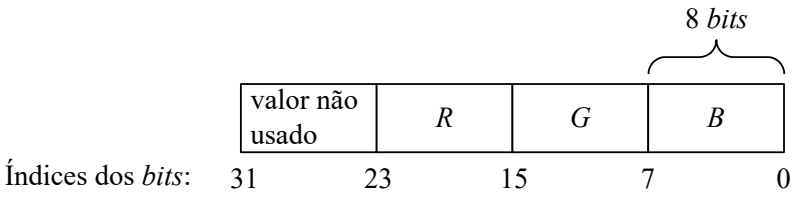


Figura 3 – Componentes *RGB* armazenadas num inteiro representado a 32 *bits*.

Se **`img`** for um *array* bidimensional de pixéis, então **`img[0]`** é o *array* que contém os pixéis da linha superior da imagem e **`img[0][0]`** contém o pixel do canto superior esquerdo

A classe `PGImage`

Para manipular imagens, definiu-se a classe **`PGImage`**. A sua interface simplificada é apresentada na figura 4.

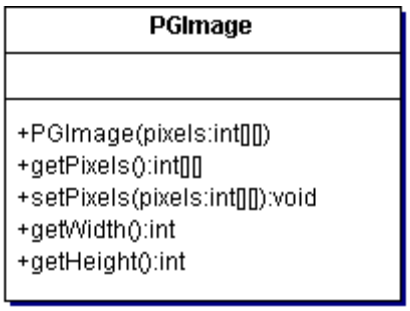


Figura 4 – UML da classe **`PGImage`** disponibilizada.

Esta classe permite a criação de imagens a partir de um *array* bidimensional de pixéis. São também disponibilizados métodos para ler e escrever os pixéis da imagem, obter a altura e largura da imagem.

Implementação de filtros sobre imagens

Existem os seguintes filtros que operam sobre imagens:

- Filtros que operam sobre um pixel isoladamente
  - "*Brightness*" – aumenta a luminância da imagem, tornando-a mais clara;
  - "*GreyScale*" – converte a imagem fonte (policromática) numa imagem com 256 níveis de cinzento;
  - "*Inverter*" – Converte a imagem no seu negativo.
- Filtros que operam na vizinhança de um pixel
  - "*Pencil*" – produz uma imagem contendo os contornos presentes na imagem fonte (fundo – cor preta; contornos – cor branca);
  - "*NoiseReductor*" – reduz imperfeições (ruído) existentes na imagem;
  - "*Emboss*" – produz uma imagem onde se salientam os relevos da imagem fonte;
  - "*Sharper*" – realça as cores e contornos da imagem fonte.

Filtros que operam sobre um pixel isoladamente

Nesta secção descrevem-se as operações aritméticas realizadas por cada um dos filtros sobre um pixel:

- Filtro "*Brightness*": $R' = R+K;\ G' = G+K;\ B' = B+K$, onde K é uma constante arbitrária;

– Filtro "*GreyScale*": *Med = (R+2G+B)/4*; *R' = Med*; *G' = Med*; *B' = Med*; A ponderação superior da componente *G* justifica-se pelo facto do olho humano ser mais sensível à cor verde, comparando com as cores vermelha e azul.

– Filtro "*Inverter*": *R' = 255-R; G' = 255-G; B' = 255-B*.

R, *G* e *B* designam as componentes do pixel na imagem fonte sendo *R'*, *G'* e *B'* as componentes do pixel na imagem destino. No fim de cada operação, os valores das componentes *R'G'B'* são normalizados, de modo a ficarem compreendidos entre 0 e 255.

Filtros que operam na vizinhança de um pixel

Nesta secção descrevem-se as operações realizadas sobre determinado pixel, tendo como informação o valor dos pixéis vizinhos.

Filtro *"Pencil"*

Um pixel pode ser visto como um ponto num mundo tridimensional de eixos *R*, *G* e *B*. Neste mundo, podemos definir a distância entre dois pixéis P_1, P_2 como:

$$distância(P_1,P_2)=\sqrt{(r_2-r_1)^2+(g_2-g_1)^2+(b_2-b_1)^2}$$

onde r_i, g_i e b_i, são as componentes *R*, *G* e *B* do *pixel* P_i.

Neste filtro, cada pixel é comparado com 2 vizinhos: o que está "acima" e o que está "atrás". Se um deles estiver a uma distância superior a uma constante *K* (valores entre 10 e 40), o pixel é colocado a branco. Caso contrário é colocado a preto.

Filtros *"NoiseReductor"*, *"Emboss"* e *"Sharper"*

Nestes filtros, cada pixel da imagem de entrada, *a[i,j]*, é substituído pela **média ponderada** dos valores localizados na sua vizinhança, ou seja, a média ponderada de *a[i+p,j+q]*, com *p = -k até k, q = -k até k* para um valor de *k* que depende da dimensão do *kernel* – *kernel* designa a matriz que contém o valor dos pesos. O *kernel* tem dimensão *2k+1*. Por exemplo, para um *kernel* de dimensão 3×3, *k=1*. A figura 5 ilustra o processo.

(10*1 + 10*2 + 10*1 + 10*2 + 20*4 + 10*2 + 10*1 + 10*2 +

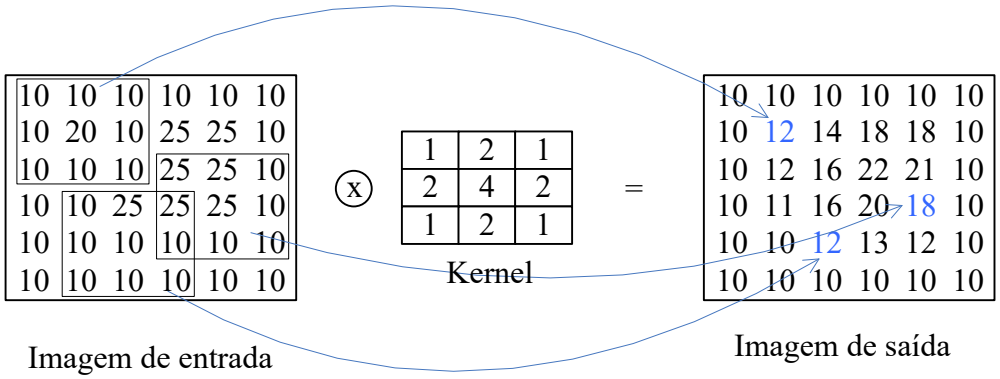


Figura 5 – Neste exemplo, para simplificar a exposição, definiram-se imagens com apenas um *byte* por pixel. Na aplicação é realizado o produto de cada valor do *kernel* pelas componentes *RGB* do pixel. Note também que o filtro não é aplicado às linhas e colunas fronteira da imagem, uma vez que os respectivos pixéis não contêm todos os vizinhos "pesados" pelo filtro.

Os *kernels* usados são apresentados na figura 6.

Emboss Kernel

0	0	-1
1	1	0
0	0	-1

NoiseReductor Kernel

1	2	1
2	4	2
1	2	1

Sharper Kernel

-1	-1	-1
-1	9	-1
-1	-1	-1

Figura 6 – Exemplo de *Kernels* usados na aplicção.

A classe `ImageFilter`

Os filtros que operam sobre imagens são implementados como objectos da classe **`ImageFilter`**. A interface de utilização desta classe é ilustrada na figura 7.

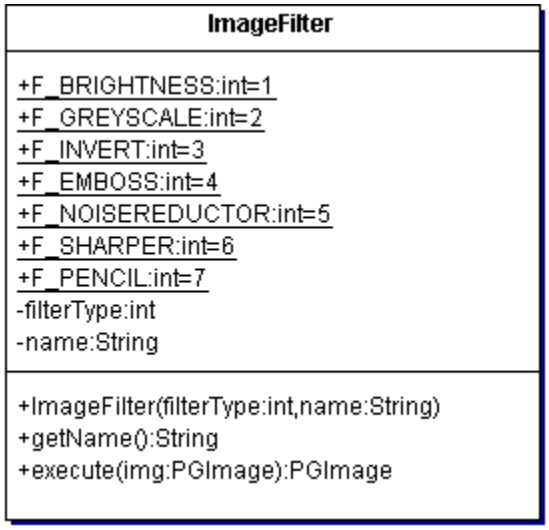


Figura 7 – UML da classe **`ImageFilter`**.

Na classe estão definidas constantes estáticas, **`F_BRIGHTNESS=1`** ... **`F_PENCIL=7`**, que designam os tipos de filtragem suportados. Os atributos de instância **`filterType`** e **`name`** indicam, respectivamente, o tipo de filtragem e o nome do filtro associado a uma instância.

O construtor **`ImageFilter(int filterType, String name)`** permite criar uma nova instância da classe e iniciar os seus atributos de instância, **`filterType`** e **`name`**. O método **`getName`** retorna o nome do filtro. O método **`execute`** retorna uma nova imagem, resultado da aplicação do filtro definido em **`filterType`**, à imagem recebida em parâmetro.

Registo de filtros de imagem

A colecção de filtros a usar pela aplicação de processamento de imagem está especificada na classe **`FilterCollection`**, cuja interface está apresentada na figura 9.

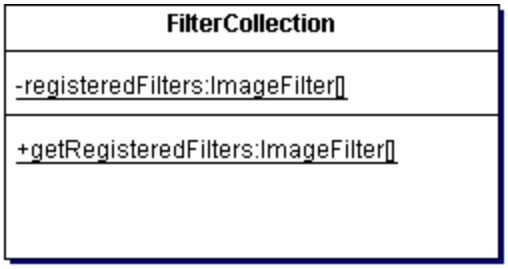


Figura 9 – UML da classe **`FilterCollection`** disponibilizada.

A classe contém apenas membros estáticos:

- atributo **`registeredFilters`** – *array* de objectos do tipo **`ImageFilter`**;
- método **`getRegisteredFilters()`** – retorna o *array* **`registeredFilters`.**

A aplicação gráfica de processamento de imagem constrói os botões situados na zona "*Filters*" (figura 8) a partir do *array* de objectos **`ImageFilter`** devolvido pela invocação: **`FilterCollection.getRegisteredFilters()`**. O nome que consta na face dos botões corresponde à **`String`** retornada pelo método **`getName()`** da classe **`ImageFilter`**.