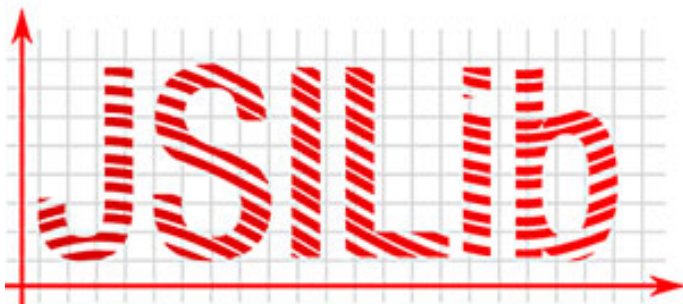


# Processamento de Imagens



**JSILib**  
**Java Simple Image Library**

**Acadêmico: Marcel Campos Inocencio**  
**Professor: Altamiro Susin**

# Proposta

Desenvolver uma biblioteca de processamento de imagens em Java, de fácil utilização e que possa ser instalada por qualquer usuário.



# Classe Image

Classe principal das bibliotecas;

Encapsula uma Imagem, fornecendo operações básicas e persistência;

## Principais Métodos:

getWidth  
getHeight  
saveToFile  
getImgHistogram  
getRaster

# Raster

Objeto que percorre a imagem, retornando (getSample) e alterando (setSample) os valores dos pixels.

```
RGBImage imagem = new RGBImage("quad.jpg");  
int banda = 0;  
for (int x = 0; x < imagem.getWidth(); x++) {  
    for (int y = 0; y < imagem.getHeight(); y++) {  
        int valor = imagem.getRaster().getSample(x, y, banda);  
        System.out.println("Valor do Pixel(" + x + "," + y + "): " + valor);  
    }  
}
```



# Classe RGBImage

Derivada de Image

## Principais Métodos:

convertToBW  
convertToBinary  
getRImage  
getGImage  
getBImage  
getImgHistogramR  
getImgHistogramG  
getImgHistogramB



# Classe BWImage (Black and White)

Derivada de Image

## Principais Métodos:

- convertToBinary
- calculaDFT
- saveTransformToFile
- aplicaFiltro
- getMagnitude
- getPhase
- imageSum
- bright



# Classe BImage (Binary)

Derivada de Image.

Principais métodos herdados de Image.



# Transformadas: DFT

Implementação de DFT em imagens Preto e Branco (PB).  
Resultado é armazenado no atributo `spectro`.

```
RGBImage imagem = new RGBImage("64x64.jpg");  
BWImage pb = imagem.convertToBW();  
try {  
    pb.calculaDFT();  
} catch (ImgLibError ex) {  
    System.out.println(ex.getMessage());  
}
```





# Transformada DFT Inversa

Retorna uma imagem restaurada a partir do espectro.

```
RGBImage imagem = new RGBImage("64x64.jpg");  
BWImage pb = imagem.convertToBW();  
BWImage aux = new BWImage(pb.getWidth(), pb.getHeight());  
try {  
    pb.calculaDFT();  
    aux.setSpectro(pb.getSpectro());  
    aux = aux.calculaDFTInversa();  
    Visual.show(aux);  
} catch (ImgLibError ex) {  
    System.out.println(ex.getMessage());  
}
```



# Transformadas: FFT

Implementação da Transformada Rápida de Fourier  
Resultado é armazenado no atributo `spectro`, e retorna o `spectro`.

```
RGBImage imagem = new RGBImage("teste256.jpg");  
BWImage pb = imagem.convertToBW();  
try {  
    FFT.fft(pb);  
} catch (ImgLibError ex) {  
    System.out.println(ex.getMessage());  
}
```



# Transformada Inversa FFT

Implementação da Transformada Inversa Rápida de Fourier. A imagem restaurada é retornada.

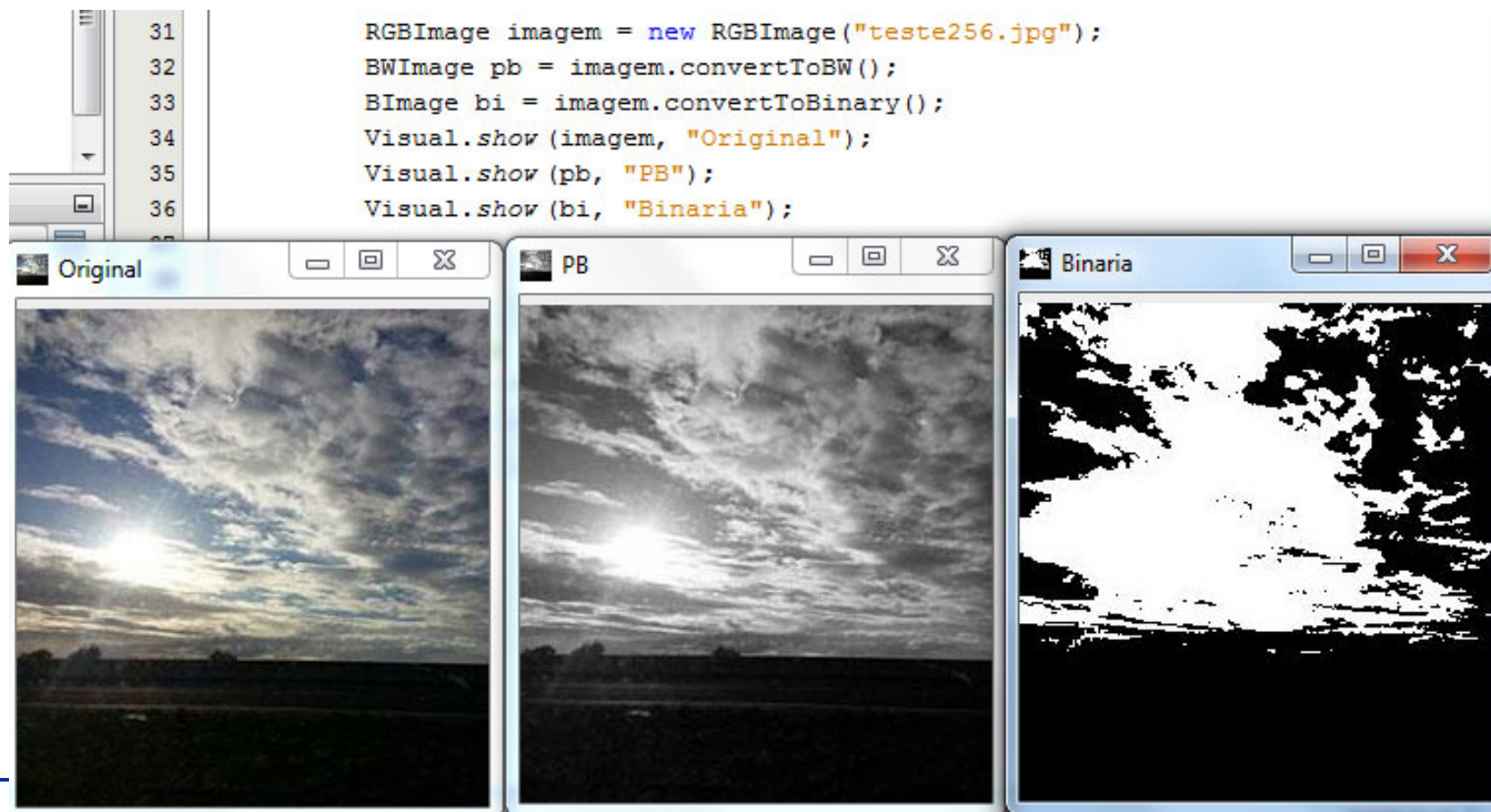
```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    // TODO code application logic here

    RGBImage imagem = new RGBImage("teste256.jpg");
    BWImage pb = imagem.convertToBW();
    BWImage aux = new BWImage(pb.getWidth(), pb.getHeight());
    try {
        aux.setSpectro(FFT.fft(pb));
        FFT.ifft(aux);
        Visual.show(aux, "Imagem Restaurada");
    } catch (ImgLibError ex) {
        System.out.println(ex.getMessage());
    }
}
```



# Camada Visual

Com apenas **um** método é possível exibir qualquer tipo de Imagem. (legenda opcional)



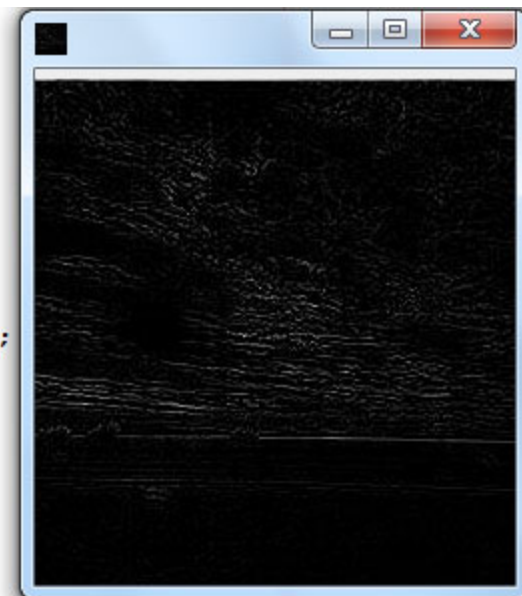
# Classe Filters

Classe com diversos métodos getXYZ, que retornam arrays de 9 posições que serão utilizados para filtragem espacial de imagens preto e branco.

```
@param args the command line arguments

lic static void main(String[] args) {
    // TODO code application logic here

    RGBImage imagem = new RGBImage("teste256.jpg");
    BWImage pb = imagem.convertToBW();
    BWImage aux = pb.aplicaFiltro(Filters.getLaplaciano());
    Visual.show(aux);
}
```



Aplicação Filtro Laplaciano

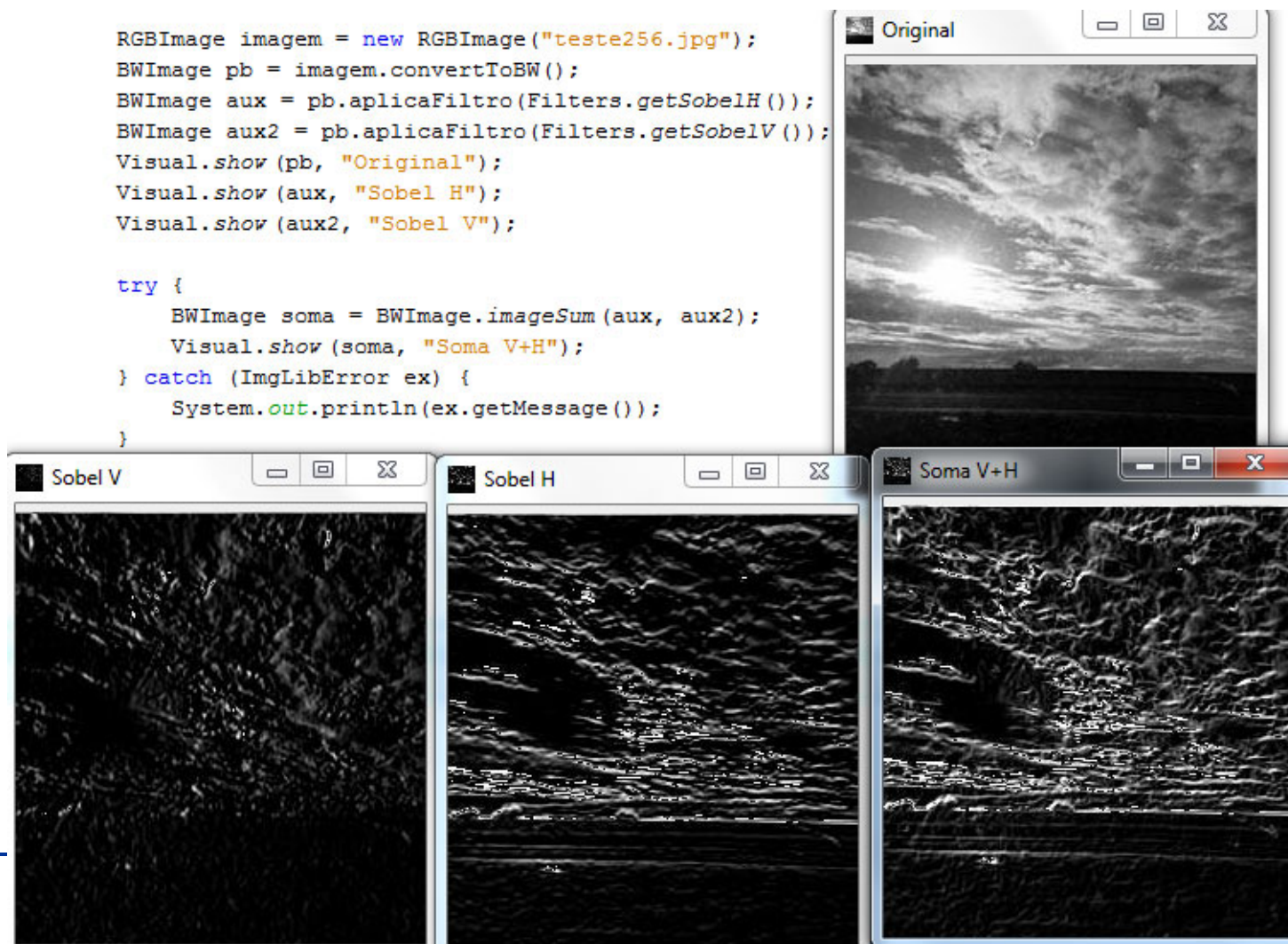




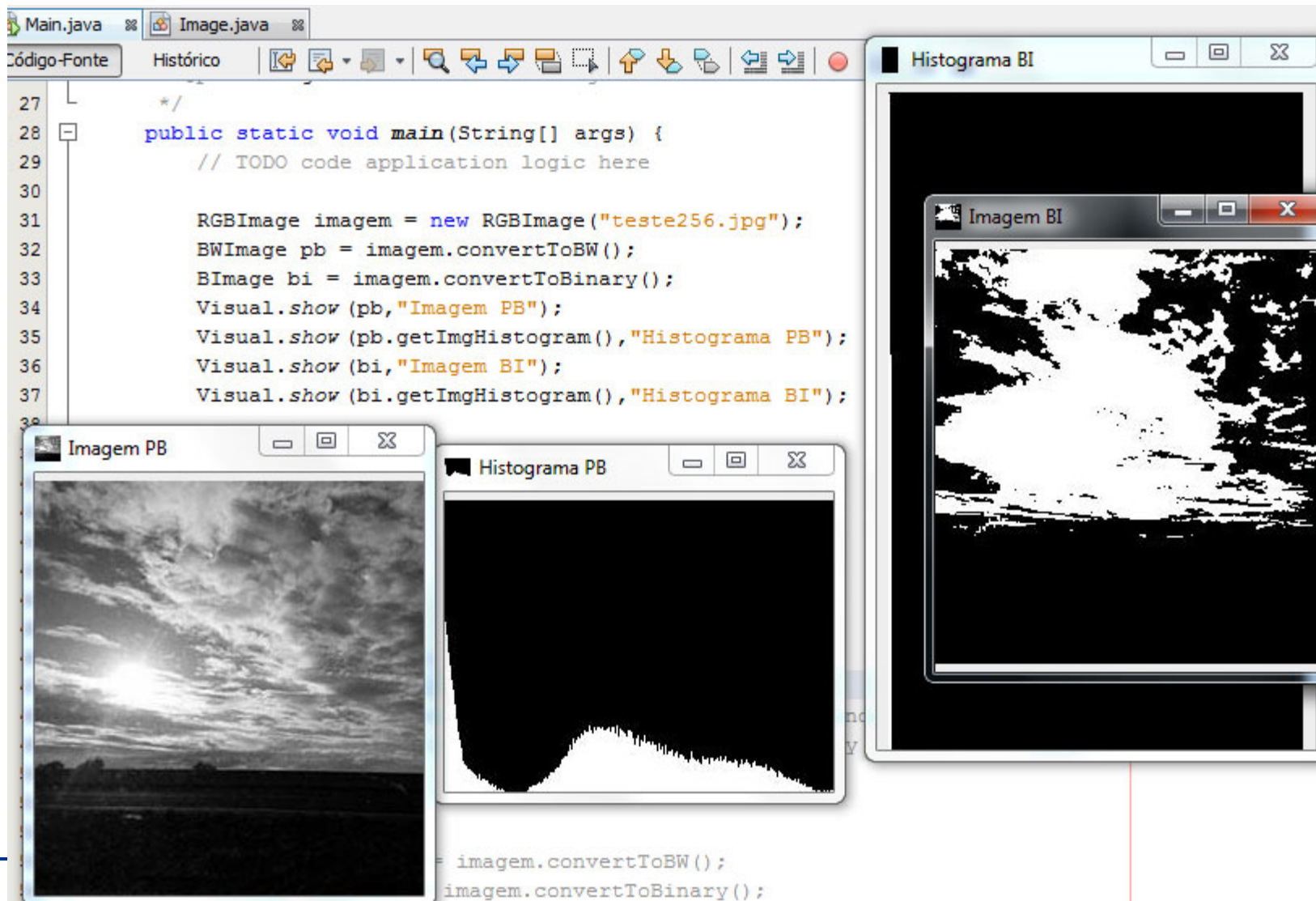
# Aplicação de Múltiplos Filtros

É possível aplicar 2 filtros em uma imagem, bastando para isto somar os resultados obtidos a partir das imagens originais.

```
RGBImage imagem = new RGBImage("teste256.jpg");  
BWImage pb = imagem.convertToBW();  
BWImage aux = pb.aplicaFiltro(Filters.getSobelH());  
BWImage aux2 = pb.aplicaFiltro(Filters.getSobelV());  
Visual.show(pb, "Original");  
Visual.show(aux, "Sobel H");  
Visual.show(aux2, "Sobel V");  
  
try {  
    BWImage soma = BWImage.imageSum(aux, aux2);  
    Visual.show(soma, "Soma V+H");  
} catch (ImgLibError ex) {  
    System.out.println(ex.getMessage());  
}
```



# Histograma PB e BI



The screenshot displays a Java IDE with a code editor and three application windows. The code editor shows the following Java code:

```
27  */
28  public static void main(String[] args) {
29      // TODO code application logic here
30
31      RGBImage imagem = new RGBImage("teste256.jpg");
32      BWImage pb = imagem.convertToBW();
33      BImage bi = imagem.convertToBinary();
34      Visual.show(pb, "Imagem PB");
35      Visual.show(pb.getImgHistogram(), "Histograma PB");
36      Visual.show(bi, "Imagem BI");
37      Visual.show(bi.getImgHistogram(), "Histograma BI");
38  }
```

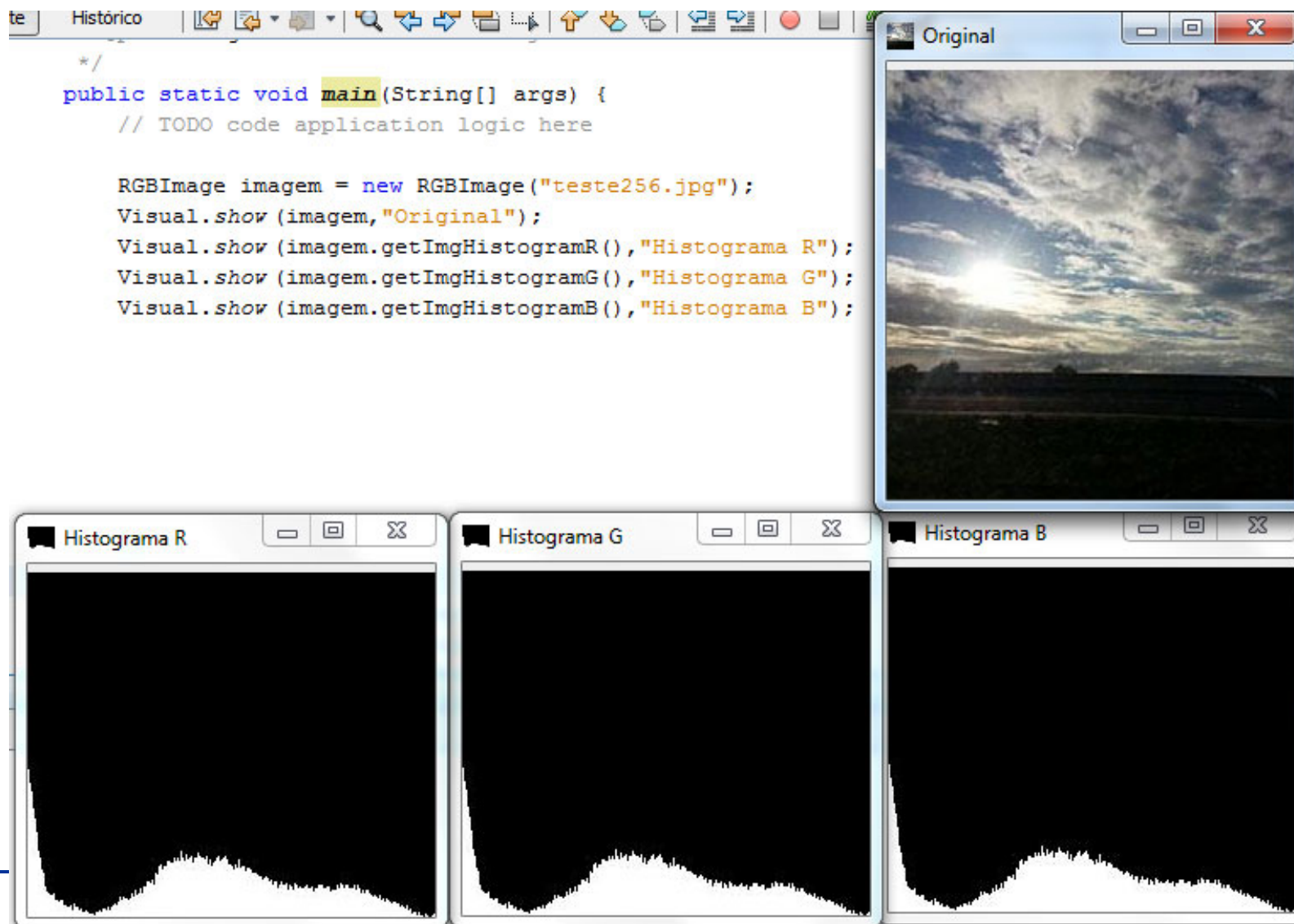
The three application windows are:

- Imagem PB**: Displays a grayscale image of a landscape with a bright sun and clouds.
- Histograma PB**: Displays a histogram of the grayscale image, showing a distribution with a prominent peak on the left side, indicating many dark pixels.
- Histograma BI**: Displays a histogram of the binary image, showing a bimodal distribution with peaks at both the minimum (black) and maximum (white) intensity values.

Below the code editor, the following code snippets are visible:

```
= imagem.convertToBW();
imagem.convertToBinary();
```

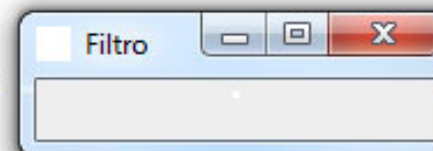
# Histograma R, G, B



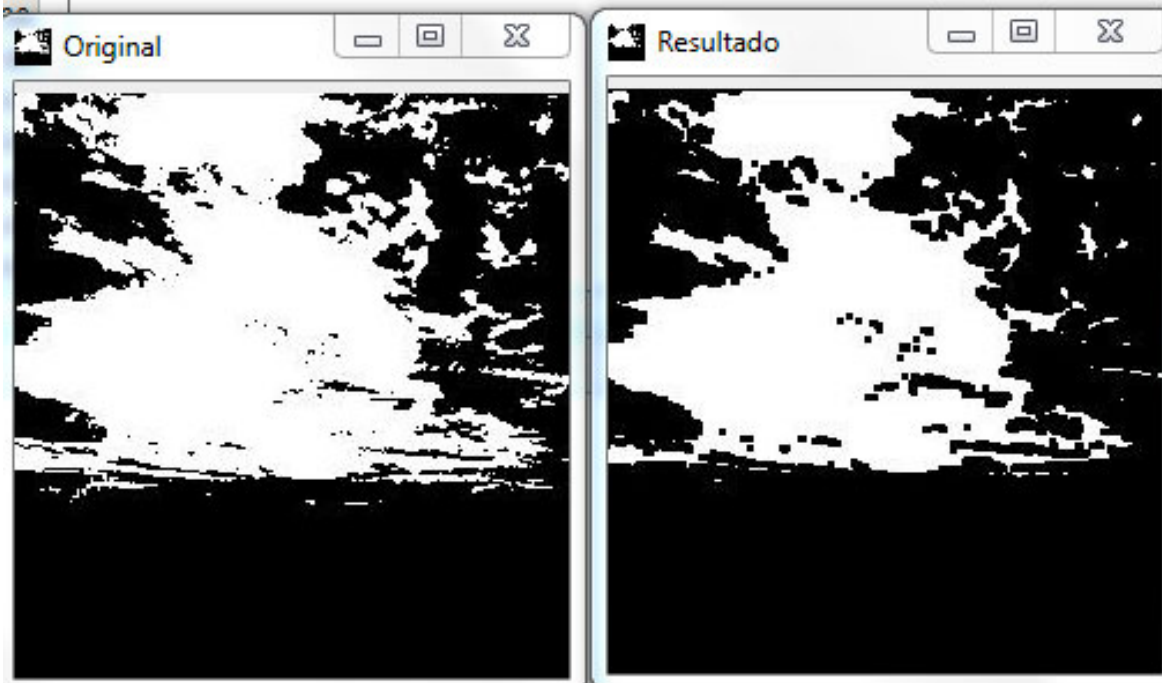


# Erosão

```
31     RGBImage imagem = new RGBImage("teste256.jpg");
32     BImage bi = imagem.convertToBinary();
33     Visual.show(bi, "Original");
34     RGBImage fi = new RGBImage("filter.bmp");
35     BImage filtro = fi.convertToBinary();
36     Visual.show(filtro, "Filtro");
37     BImage resultado = Morfologia.erosion(bi, filtro);
38     Visual.show(resultado, "Resultado");
```

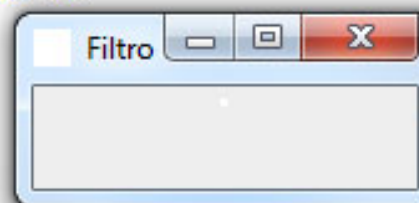


Filtro 3x3 branco

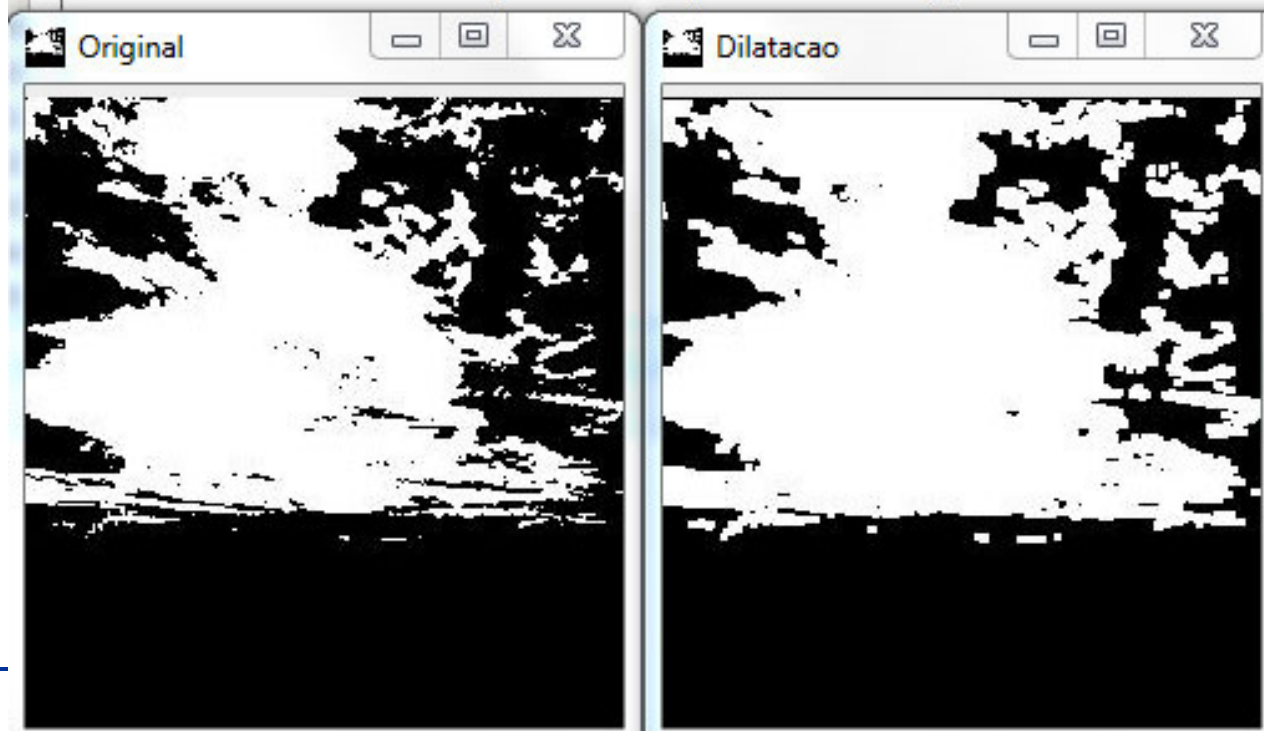


# Dilatação

```
0  
1  RGBImage imagem = new RGBImage("teste256.jpg");  
2  BImage bi = imagem.convertToBinary();  
3  Visual.show (bi, "Original");  
4  RGBImage fi = new RGBImage("filter.bmp");  
5  BImage filtro = fi.convertToBinary();  
6  Visual.show (filtro, "Filtro");  
7  BImage resultado = Morfologia.dilation(bi, filtro);  
8  Visual.show (resultado, "Dilatacao");
```

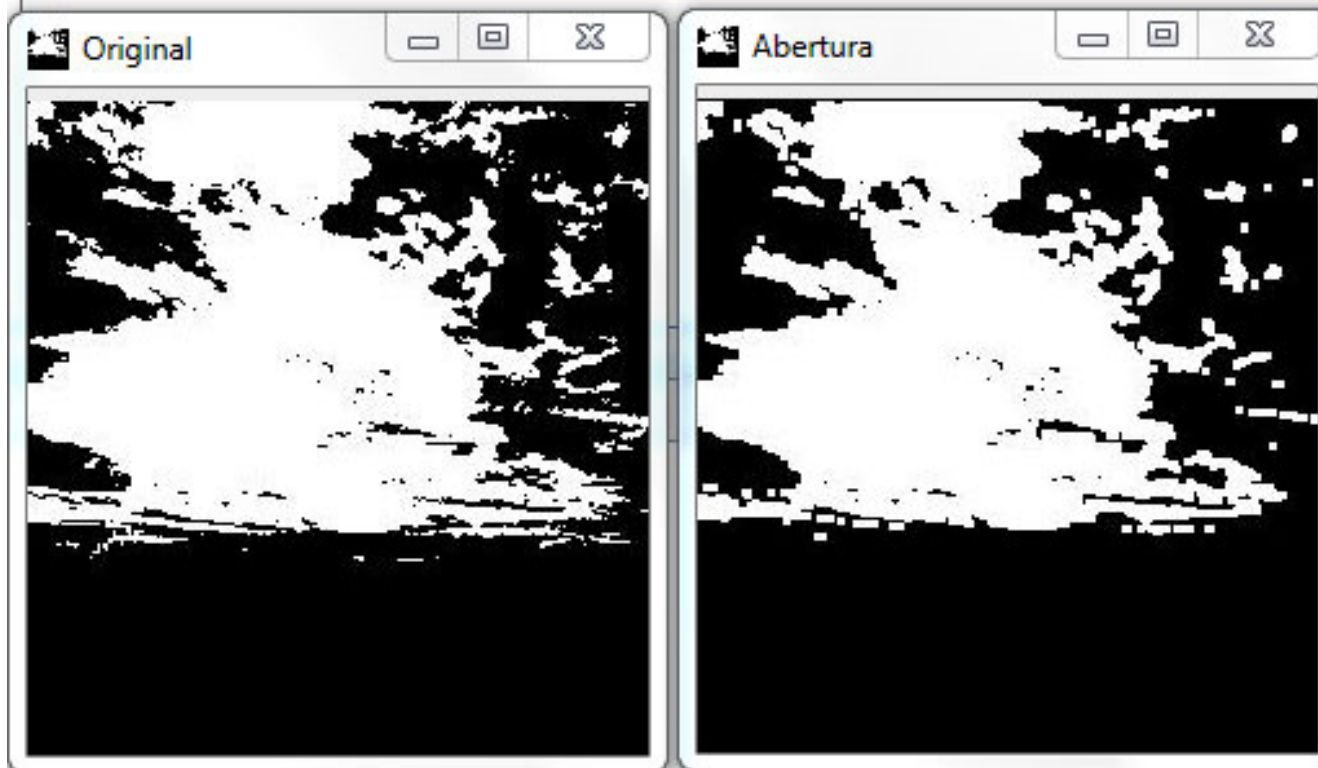
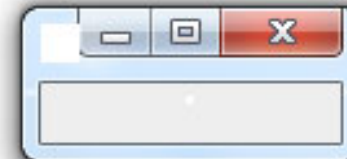


Filtro 3x3 branco



# Abertura

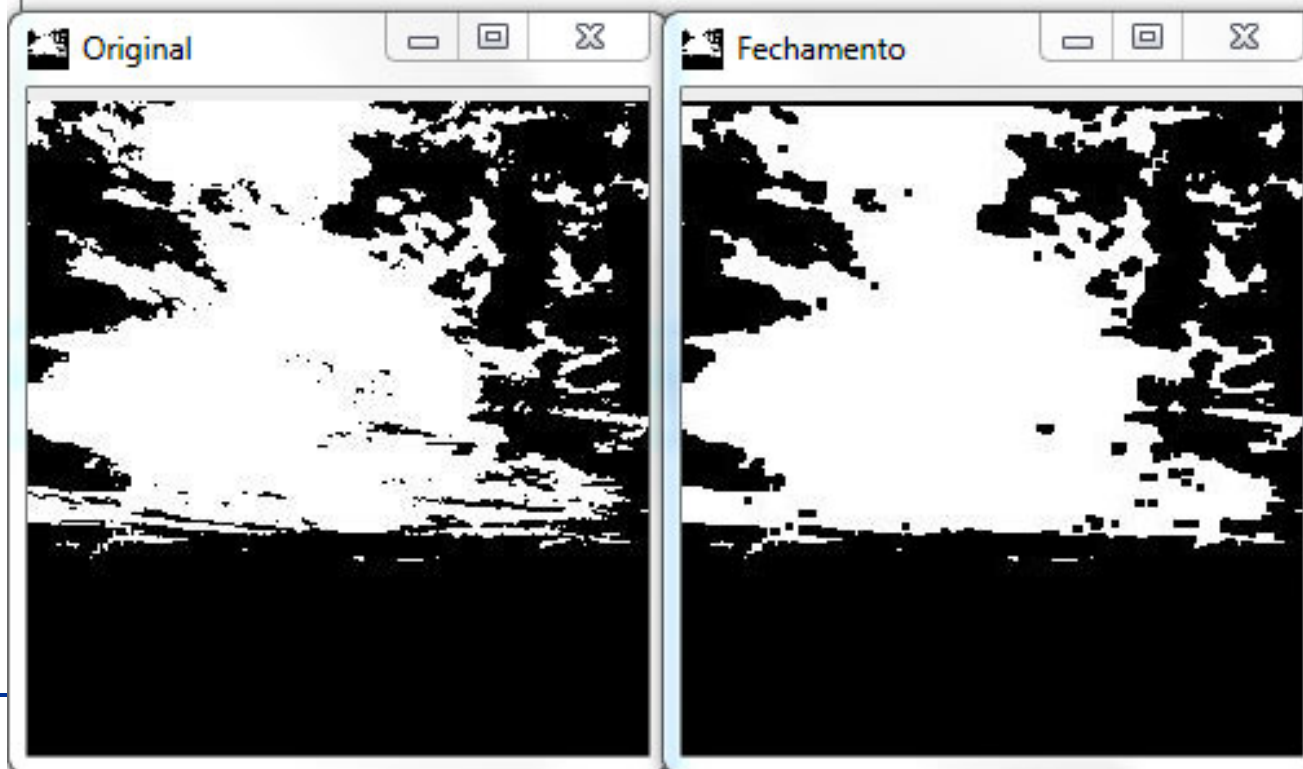
```
RGBImage imagem = new RGBImage("teste256.jpg");  
BImage bi = imagem.convertToBinary();  
Visual.show (bi, "Original");  
RGBImage fi = new RGBImage("filter.bmp");  
BImage filtro = fi.convertToBinary();  
Visual.show (filtro, "Filtro");  
BImage resultado = Morfologia.opening (bi, filtro);  
Visual.show (resultado, "Abertura");
```



Filtro 3x3 branco

# Fechamento

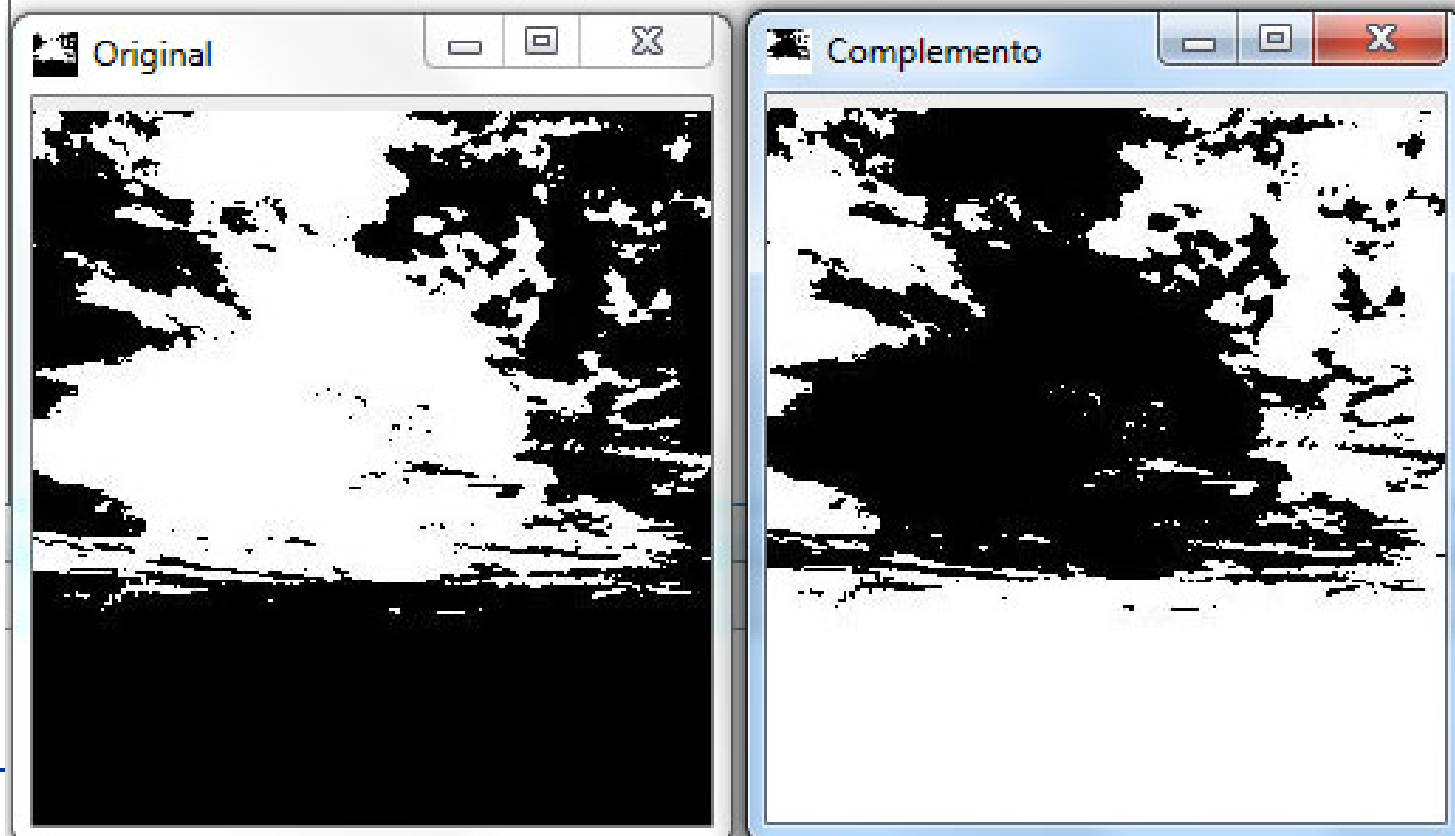
```
RGBImage imagem = new RGBImage("teste256.jpg");  
BImage bi = imagem.convertToBinary();  
Visual.show (bi, "Original");  
RGBImage fi = new RGBImage("filter.bmp");  
BImage filtro = fi.convertToBinary();  
Visual.show (filtro, "Filtro");  
BImage resultado = Morfologia.closing (bi, filtro);  
Visual.show (resultado, "Fechamento");
```



Filtro 3x3 branco

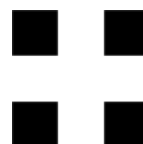
# Complemento

```
RGBImage imagem = new RGBImage("teste256.jpg");  
BImage bi = imagem.convertToBinary();  
Visual.show (bi, "Original");  
BImage comp = Morfologia.complement (bi);  
Visual.show (comp, "Complemento");
```



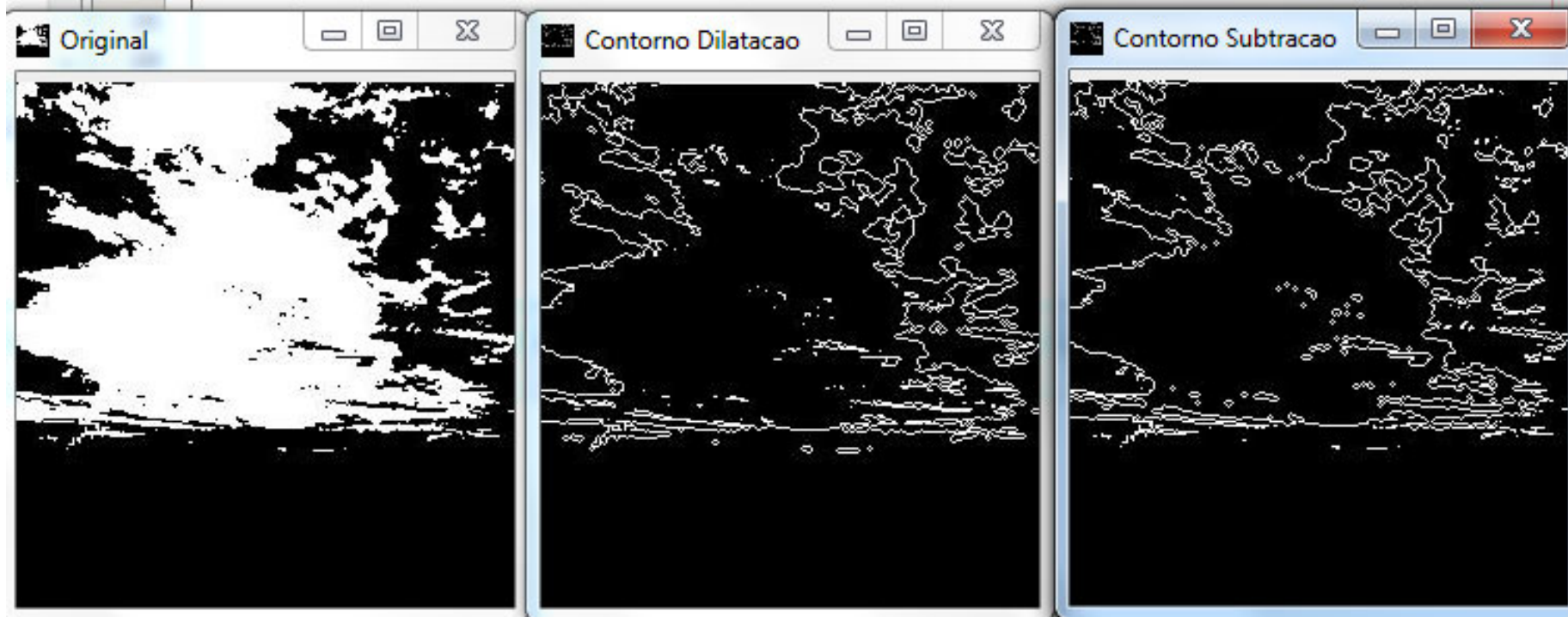


# Detecção de Bordas

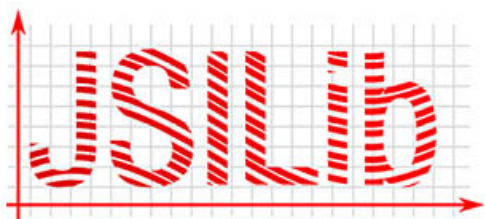


Filtro utilizado automaticamente  
pelo algoritmo 3x3 cruz.

```
31     RGBImage imagem = new RGBImage("teste256.jpg");  
32     BImage bi = imagem.convertToBinary();  
33     Visual.show (bi,"Original");  
34     BImage di = Morfologia.contornoPorDilatacao (bi);  
35     Visual.show (di,"Contorno Dilatacao");  
36     BImage su = Morfologia.contornoPorSubtracao (bi);  
37     Visual.show (su,"Contorno Subtracao");
```



Website ( <http://jsilib.org> )



*Home Sobre Download Contato*

## Java Simple Image Library

Biblioteca código aberto, desenvolvida em Java para o estudo de Processamento de Imagens. Apresenta funcionalidades básicas para a manipulação de imagens RGB, Preto e Branco e Binárias.

Latest Version



0.1

**12/12/2013**

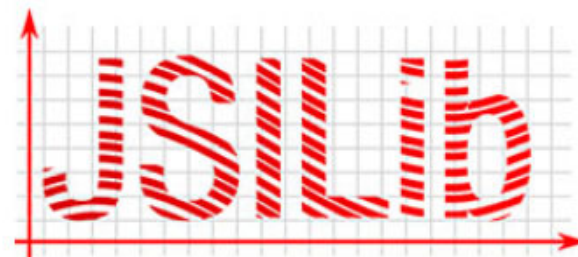
Lançamento da primeira versão da biblioteca (v 0.1).



# Manual de Instalação - PDF

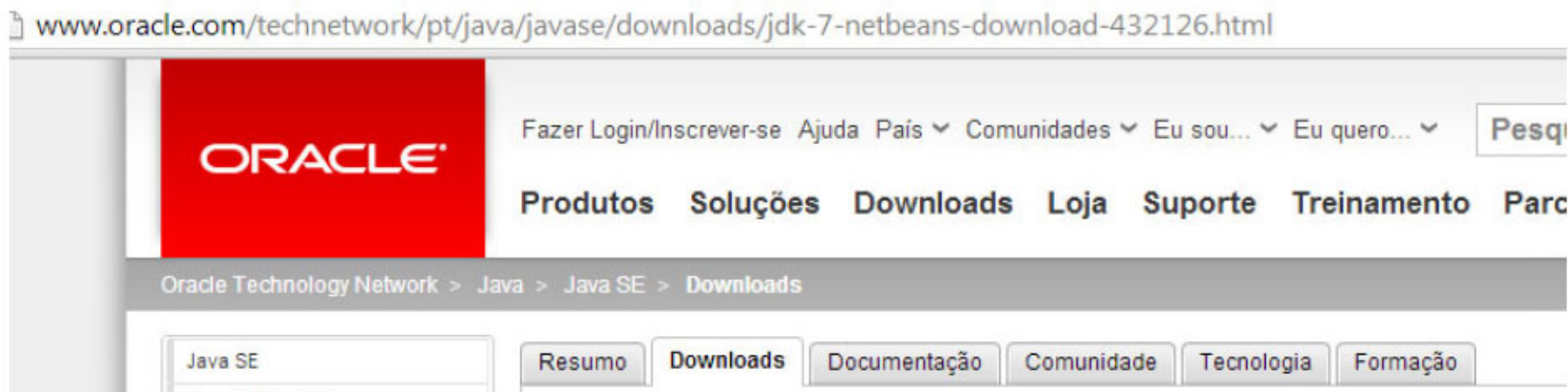
(disponível no site)

Manual de Instalação




- 1) Instale o ambiente de desenvolvimento (IDE) e o pacote de desenvolvimento Java (JDK), preferencialmente, utilize a versão disponibilizada pela Oracle contendo o netbeans com a JDK;

Link: <http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk-7-netbeans-download-432126.html>






# Fontes (GitHub)




 marcelinocencio/jsilib · GitHub


GitHub, Inc. (US) | <https://github.com/marcelinocencio/jsilib>

**GitHub** This repository ▾ Search or type a command ⓘ Explore Features E


PUBLIC  marcelinocencio / jsilib




Java Simple Image Library

 2 commits  1 branch  0 releases

 branch: master ▾ jsilib / +

versao 0.1

 marcelinocencio authored 4 hours ago latest

 fontes	versao 0.1
 jar	versao 0.1
 README.md	versao 0.1

# Obrigado !

## Perguntas ?

