APENDICE A

Image Processor

```
package imagenes;

import java.awt.*;
import javax.swing.*;

public class ImageProcessor {
  boolean packFrame = false;

  public ImageProcessor() {
    GraphicInterface frame = new GraphicInterface();
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation( (screenSize.width - frameSize.width) / 2,
                       (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        try {
          UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
        catch (Exception exception) {
          exception.printStackTrace();
        }
        new ImageProcessor();
      }
    });
  }
}
```

Image File Filter

```
package imagenes;

import java.io.*;
import java.util.*;

import javax.swing.filechooser.FileFilter;

public class ImageFileFilter
    extends FileFilter {

  private static String TYPE_UNKNOWN = "Type Unknown";
  private static String HIDDEN_FILE = "Hidden File";

  private Hashtable filters = null;
  private String description = null;
  private String fullDescription = null;
  private boolean useExtensionsInDescription = true;

  public ImageFileFilter() {
    this.filters = new Hashtable();
  }

  public ImageFileFilter(String extension) {
    this(extension, null);
  }

  public ImageFileFilter(String extension, String description) {
    this();
    if (extension != null) {
      addExtension(extension);
    }
    if (description != null) {
      setDescription(description);
    }
  }

  public ImageFileFilter(String[] filters) {
    this(filters, null);
  }

  public ImageFileFilter(String[] filters, String description) {
    this();
    for (int i = 0; i < filters.length; i++) {
      addExtension(filters[i]);
```

```
  }
  if (description != null) {
    setDescription(description);
  }
}

public boolean accept(File f) {
  if (f != null) {
    if (f.isDirectory()) {
      return true;
    }
    String extension = getExtension(f);
    if (extension != null && filters.get(getExtension(f)) != null) {
      return true;
    }
    ;
  }
  return false;
}

public String getExtension(File f) {
  if (f != null) {
    String filename = f.getName();
    int i = filename.lastIndexOf('.');
    if (i > 0 && i < filename.length() - 1) {
      return filename.substring(i + 1).toLowerCase();
    }
    ;
  }
  return null;
}

public void addExtension(String extension) {
  if (filters == null) {
    filters = new Hashtable(5);
  }
  filters.put(extension.toLowerCase(), this);
  fullDescription = null;
}

public String getDescription() {
  if (fullDescription == null) {
    if (description == null || isExtensionListInDescription()) {
      fullDescription = description == null ? "(" : description + " (";
      Enumeration extensions = filters.keys();
      if (extensions != null) {
```

```
        fullDescription += "." + (String) extensions.nextElement();
        while (extensions.hasMoreElements()) {
          fullDescription += ", ." + (String) extensions.nextElement();
        }
      }
      fullDescription += ")";
    }
    else {
      fullDescription = description;
    }
  }
  return fullDescription;
}

public void setDescription(String description) {
  this.description = description;
  fullDescription = null;
}

public void setExtensionListInDescription(boolean b) {
  useExtensionsInDescription = b;
  fullDescription = null;
}

public boolean isExtensionListInDescription() {
  return useExtensionsInDescription;
}
}
```

Graphic Interface

```
package imagenes;

import java.io.*;
import javax.imageio.*;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;

import imagenes.editor.*;
import imagenes.editor.operators.*;
import imagenes.editor.operators.dyadic.Subtraction;
import imagenes.editor.operators.monadic.Inverse;
import java.util.Vector;
import imagenes.editor.layout.Thumbnails;
import java.util.Calendar;

public class GraphicInterface
    extends JFrame {
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  JMenuBar jMenuBar1 = new JMenuBar();
  JMenu jMenuFile = new JMenu();
  JMenuItem jMenuFileExit = new JMenuItem();
  JMenuItem jMenuOpenFile = new JMenuItem();
  JMenuItem jMenuSaveAs = new JMenuItem();
  JFileChooser jFileChooser1 = new JFileChooser();
  ImageFileFilter filter1 = new ImageFileFilter();
  ImageFileFilter filter2 = new ImageFileFilter();
  boolean enabled = false;
  JMenu jMenuMonadic = new JMenu();
  JMenu jMenuDyadic = new JMenu();
  JMenuItem jMenuIdentity = new JMenuItem();
  JMenuItem jMenuInverse = new JMenuItem();
  JMenuItem jMenuThreshold = new JMenuItem();
  JMenuItem jMenuIT = new JMenuItem();
  JMenuItem jMenuBT = new JMenuItem();
  JMenuItem jMenuIBT = new JMenuItem();
  JMenuItem jMenuGST = new JMenuItem();
  JMenuItem jMenuIGST = new JMenuItem();
  JMenuItem jMenuStretch = new JMenuItem();
  JMenuItem jMenuGLR = new JMenuItem();
  JSplitPane jSplitPane1 = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
```

```
JSplitPane jSplitPane2 = new JSplitPane();
JPanel panelExtra = new JPanel();
String file;
MatrixProcessor mp = new MatrixProcessor();
ImageEditor imageEditor = new ImageEditor(this, mp);
JMenu jMenu1 = new JMenu();
public JMenuItem jMenuUndo = new JMenuItem();
JMenuItem jMenuAdd = new JMenuItem();
JMenu jMenuFile2 = new JMenu();
JMenuItem jMenuOpen2 = new JMenuItem();
JProgressBar jProgressBar1 = new JProgressBar();
JScrollPane jsp = new JScrollPane();
JMenuItem jMenuGrayScale = new JMenuItem();
JMenuItem jMenuSub = new JMenuItem();
JMenuItem jMenuMult = new JMenuItem();
JMenuItem jMenuConvolution = new JMenuItem();
JMenu jMenuGeometrics = new JMenu();
JMenuItem BlurItem = new JMenuItem();
JMenuItem jMenuRotate = new JMenuItem();
JMenuItem jMenuScale = new JMenuItem();
JMenuItem jMenuShare = new JMenuItem();
JMenu jMenuEffects = new JMenu();
JMenuItem jMenuEmboss = new JMenuItem();
JMenuItem jMenuSharpen = new JMenuItem();
JMenuItem jMenuEdge = new JMenuItem();
public JScrollPane scroll1 = new JScrollPane();
public JScrollPane scroll2 = new JScrollPane();
public JLabel label1 = new JLabel();
public JLabel label2 = new JLabel();
JMenu jMenuMotion = new JMenu();
JMenuItem jMenuDetection = new JMenuItem();

public GraphicInterface() {
  try {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    jbInit();
  }
  catch (Exception exception) {
    exception.printStackTrace();
  }
}

private void jbInit() throws Exception {
  Image image = java.awt.Toolkit.getDefaultToolkit().getImage(
      "Iconos/PICS_8.GIF");
  this.setIconImage(image);
```

```
contentPane = (JPanel) getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setResizable(false);
    setSize(new Dimension(1024, 768));
    setTitle("Images Processor");
    jMenuFile.setText("File 1");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new
                              GraphicInterface_jMenuFileExit_ActionAdapter(this));
    Icon c = new ImageIcon("Iconos/SHUTDOWN4.gif");
    jMenuFileExit.setIcon(c);
    jMenuSaveAs.setEnabled(false);
    jMenuSaveAs.setText("Save As");
    jMenuSaveAs.addActionListener(new
                             GraphicInterface_jMenuSaveAs_actionAdapter(this));
    Icon a = new ImageIcon("Iconos/FLOPPY.gif");
    jMenuSaveAs.setIcon(a);
    jMenuOpenFile.setText("Open File");
    jMenuOpenFile.addActionListener(new
                              GraphicInterface_jMenuOpenFile_actionAdapter(this));
    Icon b = new ImageIcon("Iconos/MY_PICTURES3.gif");
    jMenuOpenFile.setIcon(b);
    Icon d = new ImageIcon("Iconos/PEN_BLUE.gif");

    jMenuMonadic.setEnabled(false);
    jMenuMonadic.setText("Monadic");
    jMenuDyadic.setEnabled(false);
    jMenuDyadic.setText("Dyadic");
    jMenuIdentity.setText("Identity");
    jMenuIdentity.setIcon(d);
    jMenuInverse.setText("Inverse");
    jMenuInverse.setIcon(d);
    jMenuInverse.addActionListener(new
                             GraphicInterface_jMenuInverse_actionAdapter(this));
    jMenuThreshold.setText("Threshold");
    jMenuThreshold.setIcon(d);
    jMenuThreshold.addActionListener(new
                               GraphicInterface_jMenuThreshold_actionAdapter(this));
    jMenuIT.setText("Inverted Threshold");
    jMenuIT.setIcon(d);
    jMenuIT.addActionListener(new GraphicInterface_jMenuIT_actionAdapter(this));
    jMenuBT.setText("Binary Threshold");
    jMenuBT.setIcon(d);
    jMenuBT.addActionListener(new GraphicInterface_jMenuBT_actionAdapter(this));
    jMenuIBT.setText("Inverted Binary Threshold");
    jMenuIBT.setIcon(d);
```

```
    jMenuIBT.addActionListener(new GraphicInterface_jMenuIBT_actionAdapter(this));
    jMenuGST.setText("Gray Scale Threshold");
    jMenuGST.setIcon(d);
    jMenuGST.addActionListener(new
  GraphicInterface_jMenuGST_actionAdapter(this));
    jMenuIGST.setText("Inverted Gray Scale Threshold");
    jMenuIGST.addActionListener(new
  GraphicInterface_jMenuIGST_actionAdapter(this));
    jMenuIGST.setIcon(d);
    jMenuStretch.setText("Stretch");
    jMenuStretch.setIcon(d);
    jMenuStretch.addActionListener(new
                          GraphicInterface_jMenuStretch_actionAdapter(this));
    jMenuGLR.setText("Gray Level Reduction");
    jMenuGLR.addActionListener(new
  GraphicInterface_jMenuGLR_actionAdapter(this));
    jMenuGLR.setIcon(d);
    jMenu1.setText("Edit");
    Icon e = new ImageIcon("Iconos/UP_F1.gif");
    jMenuUndo.setIcon(e);
    jMenuUndo.setEnabled(false);
    jMenuUndo.setText("Undo");
    jMenuUndo.addActionListener(new
  GraphicInterface_jMenuUndo_actionAdapter(this));
    Icon f = new ImageIcon("Iconos/PEN_RED.gif");
    jMenuAdd.setText("Addition");
    jMenuAdd.setIcon(f);
    jMenuAdd.addActionListener(new GraphicInterface_jMenuAdd_actionAdapter(this));
    jMenuFile2.setText("File 2");
    jMenuOpen2.setText("Open File");
    jMenuOpen2.addActionListener(new
  GraphicInterface_jMenuOpen2_actionAdapter(this));
    jMenuOpen2.setIcon(b);
    jMenuGrayScale.setText("Gray Scale");
    jMenuGrayScale.setIcon(d);
    jMenuGrayScale.addActionListener(new
                            GraphicInterface_jMenuGrayScale_actionAdapter(this));
    jMenuSub.setText("Subtraction");
    jMenuSub.addActionListener(new GraphicInterface_jMenuSub_actionAdapter(this));
    jMenuSub.setIcon(f);
    jMenuMult.setText("Multiplication");
    jMenuMult.addActionListener(new
  GraphicInterface_jMenuMult_actionAdapter(this));
    jMenuMult.setIcon(f);
    jMenuConvolution.setText("Convolution");
    jMenuConvolution.setIcon(d);
```

```
    jMenuConvolution.addActionListener(new
                            GraphicInterface_jMenuConvolution_actionAdapter(this));
    Icon effects = new ImageIcon("Iconos/PEN4.gif");
    jMenuGeometrics.setEnabled(false);
    jMenuGeometrics.setText("Geometrics");
    BlurItem.setIcon(effects);
    BlurItem.setText("Blur");
    BlurItem.addActionListener(new
                        GraphicInterface_ConvolutionItem_actionAdapter(this));
    Icon rotate = new ImageIcon("Iconos/RECICLE.gif");
    jMenuRotate.setIcon(rotate);
    jMenuRotate.setText("Rotate");
    jMenuRotate.addActionListener(new
                            GraphicInterface_jMenuRotate_actionAdapter(this));
    jMenuScale.setText("Scale");
    Icon scale = new ImageIcon("Iconos/DESKT2P.gif");
    jMenuScale.setIcon(scale);

    jMenuScale.addActionListener(new
GraphicInterface_jMenuScale_actionAdapter(this));
    Icon share = new ImageIcon("Iconos/PAINTBRUSH_1.gif");
    jMenuShare.setIcon(share);
    jMenuShare.setText("Shear");
    jMenuShare.addActionListener(new
GraphicInterface_jMenuShare_actionAdapter(this));
    jMenuEffects.setEnabled(false);
    jMenuEffects.setText("Effects");

    jMenuEmboss.setIcon(effects);
    jMenuEmboss.setText("Emboss");
    jMenuEmboss.addActionListener(new
                            GraphicInterface_jMenuEmboss_actionAdapter(this));
    jMenuSharpen.setIcon(effects);
    jMenuSharpen.setText("Sharpen");
    jMenuSharpen.addActionListener(new
                            GraphicInterface_jMenuSharpen_actionAdapter(this));
    jMenuEdge.setIcon(effects);
    jMenuEdge.setText("Edge Detection");
    jMenuEdge.addActionListener(new
GraphicInterface_jMenuEdge_actionAdapter(this));
    jSplitPane1.setOneTouchExpandable(true);
    jSplitPane2.setOneTouchExpandable(true);
    jMenuMotion.setText("Motion");
    jMenuDetection.setText("Detection");
    jMenuDetection.addActionListener(new
                            GraphicInterface_jMenuDetection_actionAdapter(this));
```

```
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenu1);
jMenuBar1.add(jMenuMonadic);
jMenuBar1.add(jMenuGeometrics);
jMenuBar1.add(jMenuEffects);
jMenuBar1.add(jMenuFile2);
jMenuBar1.add(jMenuDyadic);
jMenuBar1.add(jMenuMotion);
jMenuFile.add(jMenuOpenFile);
jMenuFile.add(jMenuSaveAs);
jMenuFile.add(jMenuFileExit);
setJMenuBar(jMenuBar1);
jSplitPane2.setLeftComponent(scroll1);
jSplitPane2.setRightComponent(scroll2);
jSplitPane2.setDividerLocation(512);
jSplitPane1.setBottomComponent(panelExtra);
jSplitPane1.setTopComponent(jSplitPane2);
jSplitPane1.setDividerLocation(550);
contentPane.add(jProgressBar1, java.awt.BorderLayout.SOUTH);
jMenuMonadic.add(jMenuIdentity);
jMenuMonadic.add(jMenuGrayScale);
jMenuMonadic.add(jMenuInverse);
jMenuMonadic.add(jMenuThreshold);
jMenuMonadic.add(jMenuIT);
jMenuMonadic.add(jMenuBT);
jMenuMonadic.add(jMenuIBT);
jMenuMonadic.add(jMenuGST);
jMenuMonadic.add(jMenuIGST);
jMenuMonadic.add(jMenuStretch);
jMenuMonadic.add(jMenuGLR);
jMenuMonadic.add(jMenuConvolution);
jMenu1.add(jMenuUndo);
jMenuDyadic.add(jMenuAdd);
jMenuDyadic.add(jMenuSub);
jMenuDyadic.add(jMenuMult);
jMenuFile2.add(jMenuOpen2);
jMenuGeometrics.add(jMenuRotate);
jMenuMotion.add(jMenuDetection);
contentPane.add(jSplitPane1, java.awt.BorderLayout.CENTER);
jMenuEffects.add(jMenuEmboss);
jMenuEffects.add(BlurItem);
jMenuEffects.add(jMenuSharpen);
jMenuEffects.add(jMenuEdge);
jMenuGeometrics.add(jMenuScale);
jMenuGeometrics.add(jMenuShare);
jMenuGeometrics.add(jMenuShare);
```

```
  jMenuGeometrics.add(jMenuScale);
  jMenuGeometrics.add(jMenuRotate);
  initFilters();
}

void jMenuFileExit_actionPerformed(ActionEvent actionEvent) {
  System.exit(0);
}

public void jMenuOpenFile_actionPerformed(ActionEvent e) {
  setProgress(true);
  file = openFile();
  if (file != null) {
    imageEditor.openImage(file, 1, true);
    enableMenu(1);
  }
  setProgress(false);
}

public String openFile() {
  jFileChooser1.resetChoosableFileFilters();
  jFileChooser1.setFileFilter(filter1);
  int returnVal = jFileChooser1.showOpenDialog(this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    return jFileChooser1.getSelectedFile().getPath();
  }
  else {
    return null;
  }
}

public void initFilters() {
  filter1.addExtension("jpg");
  filter1.addExtension("gif");
  filter1.setDescription("JPG and GIF Images");
  filter2.addExtension("jpg");
  filter2.setDescription("JPG Images");
  jFileChooser1.setAcceptAllFileFilterUsed(false);

}

public void jMenuSaveAs_actionPerformed(ActionEvent e) {
  setProgress(true);
  jFileChooser1.resetChoosableFileFilters();
  jFileChooser1.setFileFilter(filter2);
  jFileChooser1.setDialogTitle("Save As");
```

```
  int returnVal = jFileChooser1.showSaveDialog(this);
  if (returnVal == JFileChooser.APPROVE_OPTION) {
    saveFile(imageEditor.getImage(1),
            jFileChooser1.getSelectedFile().getPath());
  }
  setProgress(false);
}

public void saveFile(Image image, String path) {
  if (!path.endsWith(".jpg")) {
    path = path + ".jpg";
  }
  BI_Operations s = new BI_Operations();
  BufferedImage bi = s.toBufferedImage(image);
  try {
    ImageIO.write(bi, "jpg", new File(path));
  }
  catch (IOException e) {
    e.printStackTrace();
  }
}

public void enableMenu(int i) {
  if (i == 1) {
    if (!enabled) {
      jMenuSaveAs.setEnabled(true);
      jMenuMonadic.setEnabled(true);
      jMenuGeometrics.setEnabled(true);
      jMenuEffects.setEnabled(true);
      enabled = true;
    }
  }
  else if (i == 2) { //**************************manejar activación de menús
    jMenuDyadic.setEnabled(true);
  }
}

public void setProgress(boolean state) {
  jProgressBar1.setStringPainted(true);
  if (state) {
    jProgressBar1.setString("Processing...");
  }
  else {
    jProgressBar1.setString("");
  }
  jProgressBar1.setIndeterminate(state);
```

```
  }

  public void jMenuOpen2_actionPerformed(ActionEvent e) {
    setProgress(true);
    file = openFile();
    if (file != null) {
      imageEditor.openImage(file, 2, false);
      enableMenu(2);
    }
    setProgress(false);
  }


//Otros*****************************************************************
  public void getPanel() {
    jSplitPane1.setBottomComponent(imageEditor.getImageLayout());
    jSplitPane1.setDividerLocation(550);
    setProgress(false);
  }

  public void jMenuUndo_actionPerformed(ActionEvent e) {
    imageEditor.undo();
  }


//Monadic***************************************************************
*****
  public void jMenuGrayScale_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("GrayScale");
    getPanel();
  }

  public void jMenuThreshold_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Threshold");
    getPanel();
  }

  public void jMenuInverse_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Inverse");
    getPanel();
  }

  public void jMenuIT_actionPerformed(ActionEvent e) {
```

```
    setProgress(true);
    imageEditor.setImageLayout("IT");
    getPanel();
  }

  public void jMenuBT_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("BTI");
    getPanel();
  }

  public void jMenuIBT_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("IBTI");
    getPanel();
  }

  public void jMenuGST_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("GSThreshold");
    getPanel();
  }

  public void jMenuIGST_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("IGSThreshold");
    getPanel();
  }

  public void jMenuStretch_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Stretch");
    getPanel();
  }

  public void jMenuGLR_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("GLR");
    getPanel();
  }


//Dyadic*************************************************************
****
  public void jMenuAdd_actionPerformed(ActionEvent e) {
    setProgress(true);
```

```
    imageEditor.setImageLayout("Addition");
    getPanel();
  }

  public void jMenuSub_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Subtraction");
    getPanel();
  }

  public void jMenuMult_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Multiplication");
    setProgress(false);
  }

  public void jMenuConvolution_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Convolution");
    getPanel();
  }


//Geometric**********************************************************
****
  public void jMenuRotate_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Rotate");
    getPanel();
  }

  public void jMenuScale_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Scale");
    getPanel();
  }

  public void jMenuShare_actionPerformed(ActionEvent e) {
    setProgress(true);
    imageEditor.setImageLayout("Shear");
    getPanel();
  }


//Efects************************************************************
********
```

```
public void jMenuEmboss_actionPerformed(ActionEvent e) {
  setProgress(true);
  imageEditor.setImageLayout("Emboss");
  getPanel();
}

public void BlurItem_actionPerformed(ActionEvent e) {
  setProgress(true);
  imageEditor.setImageLayout("Blur");
  getPanel();
}

public void jMenuSharpen_actionPerformed(ActionEvent e) {
  setProgress(true);
  imageEditor.setImageLayout("Sharpen");
  getPanel();
}

public void jMenuEdge_actionPerformed(ActionEvent e) {
  setProgress(true);
  imageEditor.setImageLayout("Edge");
  getPanel();
}

public void jMenuDetection_actionPerformed(ActionEvent e) {
  String folder = "\\Motion", detected = folder + "\\detected",
      inverted = folder + "\\inverted";
  Calendar calendar = Calendar.getInstance();
  int tmp1, tmp2, tmp3;
  String temp1 = "", temp2 = "", temp3 = "";
  tmp1 = calendar.get(Calendar.DAY_OF_MONTH);
  tmp2 = calendar.get(Calendar.MONTH);
  tmp3 = calendar.get(Calendar.YEAR);
  if (tmp1 < 10) {
    temp1 = "0";
  }
  if (tmp2 < 10) {
    temp2 = "0";
  }
  temp1 += new Integer(tmp1).toString();
  temp2 += new Integer(tmp2).toString();
  temp3 = new Integer(tmp3).toString();
  String name = temp1 + temp2 + temp3.substring(2);
  MatrixProcessor mp3 = new MatrixProcessor();
  setProgress(true);
  jFileChooser1.resetChoosableFileFilters();
```

```
jFileChooser1.setFileFilter(filter1);
jFileChooser1.setMultiSelectionEnabled(true);
int returnVal = jFileChooser1.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
  File[] archivos = jFileChooser1.getSelectedFiles();
  Image original = Toolkit.getDefaultToolkit().getImage(archivos[0].getPath());
  Vector filtrados = new Vector();
  for (int i = 1; i < archivos.length; i++) {
    Image image = Toolkit.getDefaultToolkit().getImage(archivos[i].getPath());
    mp3.setOneDPix(Subtraction.getSubtraction(original, 240, 320, image,
                                    240, 320));
    mp3.setImgRows(Subtraction.getRows());
    mp3.setImgCols(Subtraction.getCols());
    mp3.convertToImage();
    File f = new File(folder);
    f.mkdir();
    f = new File(detected);
    f.mkdir();
    f = new File(inverted);
    f.mkdir();
    calendar = Calendar.getInstance();
    temp1 = "";
    temp2 = "";
    temp3 = "";
    tmp1 = calendar.get(Calendar.HOUR)-6;
    tmp2 = calendar.get(Calendar.MINUTE);
    tmp3 = calendar.get(Calendar.SECOND);
    if (tmp1 < 10) {
      temp1 = "0";
    }
    if (tmp2 < 10) {
      temp2 = "0";
    }
    if (tmp3 < 10) {
      temp3 = "0";
    }
    temp1 += new Integer(tmp1).toString();
    temp2 += new Integer(tmp2).toString();
    temp3 += new Integer(tmp3).toString();
    String name2 = temp1 + temp2 + temp3;
    saveFile(mp3.getModImg(), detected + "\\" + name + name2 + i);
    if (DMotion.getMotion(mp3.getOneDPix())) {
      mp3.setOneDPix(Inverse.getInverse(mp3.getOneDPix()));
      mp3.convertToImage();
      saveFile(mp3.getModImg(), inverted + "\\" + name + name2 + i);
      filtrados.addElement(new File(inverted + "\\" + name + name2 + i +
```

```
                                ".jpg"));

          }
        }
        JOptionPane.showMessageDialog(new JFrame(),
                                      "Motion Detection Finilized",
                                      "Finish", JOptionPane.INFORMATION_MESSAGE);

        JPanel x = new Thumbnails(imageEditor, filtrados, archivos.length);
        jSplitPane1.setBottomComponent(x);
        jSplitPane1.setDividerLocation(550);
        setProgress(false);
      }
    }
  }

class GraphicInterface_jMenuDetection_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuDetection_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuDetection_actionPerformed(e);
  }
}

class GraphicInterface_jMenuMult_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuMult_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuMult_actionPerformed(e);
  }
}

class GraphicInterface_jMenuConvolution_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuConvolution_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }
```

```
  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuConvolution_actionPerformed(e);
  }
}

class GraphicInterface_jMenuEdge_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuEdge_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuEdge_actionPerformed(e);
  }
}

class GraphicInterface_jMenuSharpen_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuSharpen_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuSharpen_actionPerformed(e);
  }
}

class GraphicInterface_jMenuEmboss_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuEmboss_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuEmboss_actionPerformed(e);
  }
}

class GraphicInterface_jMenuShare_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuShare_actionAdapter(GraphicInterface adaptee) {
```

```
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuShare_actionPerformed(e);
  }
}

class GraphicInterface_jMenuScale_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuScale_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuScale_actionPerformed(e);
  }
}

class GraphicInterface_jMenuRotate_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuRotate_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuRotate_actionPerformed(e);
  }
}

class GraphicInterface_ConvolutionItem_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_ConvolutionItem_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {

    adaptee.BlurItem_actionPerformed(e);
  }
}

class GraphicInterface_jMenuUndo_actionAdapter
```

```
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuUndo_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuUndo_actionPerformed(e);
  }
}

class GraphicInterface_jMenuSub_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuSub_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuSub_actionPerformed(e);
  }
}

class GraphicInterface_jMenuGrayScale_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuGrayScale_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuGrayScale_actionPerformed(e);
  }
}

class GraphicInterface_jMenuGLR_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuGLR_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuGLR_actionPerformed(e);
  }
}
```

```
class GraphicInterface_jMenuStretch_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuStretch_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuStretch_actionPerformed(e);
  }
}

class GraphicInterface_jMenuIGST_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuIGST_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuIGST_actionPerformed(e);
  }
}

class GraphicInterface_jMenuGST_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuGST_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuGST_actionPerformed(e);
  }
}

class GraphicInterface_jMenuIBT_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuIBT_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuIBT_actionPerformed(e);
```

```
  }
}

class GraphicInterface_jMenuIT_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuIT_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuIT_actionPerformed(e);
  }
}

class GraphicInterface_jMenuBT_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuBT_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuBT_actionPerformed(e);
  }
}

class GraphicInterface_jMenuAdd_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuAdd_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuAdd_actionPerformed(e);
  }
}

class GraphicInterface_jMenuOpen2_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuOpen2_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }
```

```
  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuOpen2_actionPerformed(e);
  }
}

class GraphicInterface_jMenuInverse_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuInverse_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuInverse_actionPerformed(e);
  }
}

class GraphicInterface_jMenuThreshold_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuThreshold_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuThreshold_actionPerformed(e);
  }
}

class GraphicInterface_jMenuSaveAs_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuSaveAs_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuSaveAs_actionPerformed(e);
  }
}

class GraphicInterface_jMenuOpenFile_actionAdapter
    implements ActionListener {
  private GraphicInterface adaptee;
  GraphicInterface_jMenuOpenFile_actionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
```

```
  }

  public void actionPerformed(ActionEvent e) {
    adaptee.jMenuOpenFile_actionPerformed(e);
  }
}

class GraphicInterface_jMenuFileExit_ActionAdapter
    implements ActionListener {
  GraphicInterface adaptee;

  GraphicInterface_jMenuFileExit_ActionAdapter(GraphicInterface adaptee) {
    this.adaptee = adaptee;
  }

  public void actionPerformed(ActionEvent actionEvent) {
    adaptee.jMenuFileExit_actionPerformed(actionEvent);
  }
}
```

Matriz Processor

```
package imagenes.editor;

import java.awt.*;
import java.awt.image.*;

public class MatrixProcessor
    extends Component {
  int[] oneDPix;
  int imgCols, imgRows;
  Image rawImg, modImg;

  public MatrixProcessor() {
  }

  public void getPixelGrabber() {
    oneDPix = new int[imgCols * imgRows];
    try {
      PixelGrabber pgObj = new PixelGrabber(rawImg, 0, 0, imgCols, imgRows,
                                            oneDPix, 0, imgCols);
      pgObj.grabPixels();
    }
    catch (InterruptedException e) {
      e.printStackTrace();
    }
  }

  public void convertToMatrix(Image image, int cols, int rows) {
    rawImg = image;
    imgCols = cols;
    imgRows = rows;
    getPixelGrabber();
  }

  public void convertToImage() {
    modImg = createImage(new MemoryImageSource(imgCols, imgRows, oneDPix, 0,
                                               imgCols));
  }

  public int[][][] convertToThreeDim(int[] oneDPix, int imgRows,
                                     int imgCols) {
    int[][][] data = new int[imgRows][imgCols][4];
    for (int row = 0; row < imgRows; row++) {
      int[] aRow = new int[imgCols];
      for (int col = 0; col < imgCols; col++) {
```

```
      int element = row * imgCols + col;
      aRow[col] = oneDPix[element];
    }
    for (int col = 0; col < imgCols; col++) {
      data[row][col][0] = (aRow[col] >> 24) & 0xFF;
      data[row][col][1] = (aRow[col] >> 16) & 0xFF;
      data[row][col][2] = (aRow[col] >> 8) & 0xFF;
      data[row][col][3] = (aRow[col]) & 0xFF;
    }
  }
  return data;
}

public int[] convertToOneDim(int[][][] data, int imgRows, int imgCols) {
  int[] oneDPix = new int[imgCols * imgRows * 4];
  for (int row = 0, cnt = 0; row < imgRows; row++) {
    for (int col = 0; col < imgCols; col++) {
      oneDPix[cnt] = ( (data[row][col][0] << 24) & 0xFF000000)
          | ( (data[row][col][1] << 16) & 0x00FF0000)
          | ( (data[row][col][2] << 8) & 0x0000FF00)
          | ( (data[row][col][3]) & 0x000000FF);
      cnt++;
    }
  }
  return oneDPix;
}

public int[] getOneDPix() {
  return oneDPix;
}

public int getImgCols() {
  return imgCols;
}

public int getImgRows() {
  return imgRows;
}

public Image getRawImg() {
  return rawImg;
}

public Image getModImg() {
  return modImg;
}
```

```
  public void setOneDPix(int[] array) {
    oneDPix = array;
  }

  public void setImgCols(int cols) {
    imgCols = cols;
  }

  public void setImgRows(int rows) {
    imgRows = rows;
  }

  public void setRawImg(Image raw) {
    rawImg = raw;
  }

  public void setModImg(Image mod) {
    modImg = mod;
  }
}
```

Image Editor

```
package imagenes.editor;

import java.util.*;

import java.awt.*;
import java.awt.image.*;
import javax.swing.*;

import imagenes.*;
import imagenes.editor.layout.*;
import imagenes.editor.operators.*;
import imagenes.editor.operators.dyadic.*;
import imagenes.editor.operators.monadic.*;

public class ImageEditor
    extends JComponent {
  public JLabel display1;
  public JLabel display2;
  JPanel panel;
  public final int queueSize = 3;
  public Vector queue = new Vector(queueSize);
  int queueCounter = 0;
  public GraphicInterface g;
  Vector image1, image2;
  public MatrixProcessor mp, mp2, mp3;
  ExtraImage exImg;

  public ImageEditor(GraphicInterface g, MatrixProcessor mp) {
    this.g = g;
    this.mp = mp;
    mp2 = new MatrixProcessor();
    mp3 = new MatrixProcessor();

    display1 = g.label1;
    display2 = g.label2;
    exImg = new ExtraImage(this);
    for (int i = 0; i < queueSize; i++) {
      queue.addElement(null);
    }
  }

  public void push(Image image) {
    for (int i = 0; i < queueSize - 1; i++) {
      queue.setElementAt(queue.elementAt(i + 1), i);
```

```
  }
  queue.setElementAt(image, queueSize - 1);
  if (queueCounter < queueSize) {
    queueCounter++;
  }
  if (queueCounter > 1) {
    g.jMenuUndo.setEnabled(true);
  }
}

public Image pop() {
  Image tmp = (Image) queue.elementAt(queueSize - 2);
  for (int i = queueSize - 1; i > 0; i--) {
    queue.setElementAt(queue.elementAt(i - 1), i);
  }
  queue.setElementAt(null, 0);
  if (queueCounter > 1) {
    queueCounter--;
  }
  if (queueCounter < 2) {
    g.jMenuUndo.setEnabled(false);
  }
  return tmp;
}

public void openImage(String file, int display, boolean push) {
  Image image = Toolkit.getDefaultToolkit().getImage(file);
  setImage(image, display);
  if(push)
    push(image);
}

public void setImage(Image image, int display) {
  if (display == 1) {
    display1 = new JLabel(new ImageIcon(image));
    g.scroll1.setViewportView(display1);

  }
  else if (display == 2) {
    display2 = new JLabel(new ImageIcon(image));
    g.scroll2.setViewportView(display2);
  }
}

public Image getImage(int display) {
  ImageIcon ii = null;
```

```
    if (display == 1) {
      ii = (ImageIcon) display1.getIcon();
    }
    else if (display == 2) {
      ii = (ImageIcon) display2.getIcon();
    }
    return ii.getImage();
  }

  public void setImageLayout(String layout) {

//Monadic*************************************************************
****
    if (layout.equals("GrayScale")) {
      Image image = (Image) queue.elementAt(queueSize - 1);
      mp.convertToMatrix(image, image.getWidth(this), image.getHeight(this));
      mp.setOneDPix(GrayScale.getGrayScale(mp.getOneDPix()));
      mp.convertToImage();
      setImage(mp.getModImg(), 1);
      push(mp.getModImg());
      panel = new JPanel();
    }

    else if (layout.equals("Threshold")) {
      panel = new LayoutThreshold(this, mp);
    }

    else if (layout.equals("Inverse")) {
      Image image = (Image) queue.elementAt(queueSize - 1);
      mp.convertToMatrix(image, image.getWidth(this), image.getHeight(this));
      mp.setOneDPix(Inverse.getInverse(mp.getOneDPix()));
      mp.convertToImage();
      setImage(mp.getModImg(), 1);
      push(mp.getModImg());
      panel = new JPanel();
    }

    else if (layout.equals("IT")) {
      panel = new LayoutInvertedThreshold(this, mp);
    }

    else if (layout.equals("BTI")) {
      panel = new Layout2opc(this, mp, 1, "Binary Threshold");
    }

    else if (layout.equals("IBTI")) {
```

```
      panel = new Layout2opc(this, mp, 2, "Inverted Binary Threshold");
    }

    else if (layout.equals("GSThreshold")) {
      panel = new Layout2opc(this, mp, 3, "Gray Scale Threshold");
    }

    else if (layout.equals("IGSThreshold")) {
      panel = new Layout2opc(this, mp, 4, "Inverse Gray Scale Threshold");
    }

    else if (layout.equals("Stretch")) {
      panel = new Layout2opc(this, mp, 5, "Stretch");
    }

    else if (layout.equals("GLR")) {
      panel = new LayoutGrayScaleReduction(this, mp);
    }


//Effects*************************************************************
    else if (layout.equals("Emboss")) {
      Image image = (Image) queue.elementAt(queueSize - 1);
      BI_Operations bi = new BI_Operations();
      BufferedImage source = bi.toBufferedImage(image);
      Kernel kernel = new Kernel(3, 3,
                          new float[] {
                          -2, 0, 0,
                          0, 1, 0,
                          0, 0, 2});
      BufferedImageOp op = new ConvolveOp(kernel);
      BufferedImage destination = op.filter(source, null);
      Image imagemod = bi.toImage(destination);
      setImage(imagemod, 1);
      push(imagemod);
      panel = new JPanel();
    }

    else if (layout.equals("Sharpen")) {
      Image image = (Image) queue.elementAt(queueSize - 1);
      BI_Operations bi = new BI_Operations();
      BufferedImage source = bi.toBufferedImage(image);
      Kernel kernel = new Kernel(3, 3,
                          new float[] {0.0f, -1.0f, 0.0f,
                          -1.0f, 5.f, -1.0f,
                          0.0f, -1.0f, 0.0f});
```

```
  BufferedImageOp op = new ConvolveOp(kernel, ConvolveOp.EDGE_NO_OP, null);
  BufferedImage destination = op.filter(source, null);
  Image imagemod = bi.toImage(destination);
  setImage(imagemod, 1);
  push(imagemod);
  panel = new JPanel();
 }

 else if (layout.equals("Edge")) {
  Image image = (Image) queue.elementAt(queueSize - 1);
  BI_Operations bi = new BI_Operations();
  BufferedImage source = bi.toBufferedImage(image);

  float[] edgeKernel = {
     0.0f, -1.0f, 0.0f,
     -1.0f, 4.0f, -1.0f,
     0.0f, -1.0f, 0.0f};

  BufferedImageOp blur = new ConvolveOp( (new Kernel(3, 3, edgeKernel)),
                                ConvolveOp.EDGE_NO_OP, null);
  BufferedImage destination = blur.filter(source, null);
  Image imagemod = bi.toImage(destination);
  setImage(imagemod, 1);
  push(imagemod);
  panel = new JPanel();
 }

 else if (layout.equals("Blur")) {
  Image image = (Image) queue.elementAt(queueSize - 1);
  BI_Operations bi = new BI_Operations();
  BufferedImage source = bi.toBufferedImage(image);
  float ninth = 1.0f / 9.0f;
  float[] blurKernel = {
     ninth, ninth, ninth,
     ninth, ninth, ninth,
     ninth, ninth, ninth};

  BufferedImageOp blur = new ConvolveOp(new Kernel(3, 3, blurKernel));
  BufferedImage destination = blur.filter(source, null);
  Image imagemod = bi.toImage(destination);
  setImage(imagemod, 1);
  push(imagemod);
  panel = new JPanel();
 }
```

```
//Geometric*************************************************************
***

   else if (layout.equals("Rotate")) {
     panel = new LayoutRotate(this, mp);
   }

   else if (layout.equals("Scale")) {
     panel = new LayoutScale(this, mp, "Scale");
   }

   else if (layout.equals("Shear")) {
     panel = new LayoutScale(this, mp, "Shear");
   }

//Dyadic****************************************************************
*******
   else if (layout.equals("Addition")) {
     Image image1 = (Image) queue.elementAt(queueSize - 1);
     Image image2 = getImage(2);
     mp3.setOneDPix(Addition.getAddition(image1, image1.getHeight(this),
                                image1.getWidth(this), image2,
                                image2.getHeight(this),
                                image2.getWidth(this)));
     mp3.setImgRows(Addition.getRows());
     mp3.setImgCols(Addition.getCols());
     mp3.convertToImage();
     exImg.printImage(mp3.getModImg());
     panel = new JPanel();
   }

   else if (layout.equals("Subtraction")) {
     Image image1 = (Image) queue.elementAt(queueSize - 1);
     Image image2 = getImage(2);
     mp3.setOneDPix(Subtraction.getSubtraction(image1, image1.getHeight(this),
                                   image1.getWidth(this), image2,
                                   image2.getHeight(this),
                                   image2.getWidth(this)));
     mp3.setImgRows(Subtraction.getRows());
     mp3.setImgCols(Subtraction.getCols());
     mp3.convertToImage();
     exImg.printImage(mp3.getModImg());
     panel = new JPanel();
   }
```

```
    else if (layout.equals("Multiplication")) {
      Image image1 = (Image) queue.elementAt(queueSize - 1);
      Image image2 = getImage(2);
      mp3.setOneDPix(Multiplication.getMultiplication(image1, image1.getHeight(this),
          image1.getWidth(this), image2,
          image2.getHeight(this),
          image2.getWidth(this)));
      mp3.setImgRows(Multiplication.getRows());
      mp3.setImgCols(Multiplication.getCols());
      mp3.convertToImage();
      exImg.printImage(mp3.getModImg());
      panel = new JPanel();
    }

    else if (layout.equals("Convolution")) {
      panel = new LayoutConvolution(this, mp);
    }
  }

  public JPanel getImageLayout() {
    return panel;
  }

  public void undo() {
    Image image = pop();
    if (image != null) {
      mp.convertToMatrix(image, image.getWidth(this), image.getHeight(this));
      mp.convertToImage();
      setImage(mp.getModImg(), 1);
    }
  }
}
```

Nota: Resto del Código está en el CD que va adjunto.