```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.image.*;
import javax.swing.border.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.swing.filechooser.FileFilter;
import javax.imageio.ImageIO;
import java.net.URL;

/*************************************************************
MenuActionListener provides the link between the dropdown menu
items and the methods they actually call.

@author James Houghton
@version 06/05/2014
*************************************************************/
class MenuActionListener implements ActionListener
{
    public void actionPerformed(ActionEvent evt)
    {
        if (evt.getActionCommand()=="Invert")
            Tasks.task1();
        if (evt.getActionCommand()=="Fade")
            Tasks.task2();
        if (evt.getActionCommand()=="Tint")
            Tasks.task3();
        if (evt.getActionCommand()=="Black/White")
            Tasks.task4();
        if (evt.getActionCommand()=="Remove color")
            Tasks.task5();
        if (evt.getActionCommand()=="Grayscale")
            Tasks.task6();
        if (evt.getActionCommand()=="Colorize")
            Tasks.task7();
        if (evt.getActionCommand()=="Clear effects")
            Display.clearEffects();
        if (evt.getActionCommand()=="Encode")
            Tasks.encode();
        if(evt.getActionCommand()=="Decode")
            Tasks.decode();
        if (evt.getActionCommand()=="Open...")
            Tasks.open();
        if (evt.getActionCommand()=="Save")
            Tasks.save();
        if (evt.getActionCommand()=="Save as...")
            Tasks.saveAs();
        if (evt.getActionCommand()=="Exit")
            System.exit(0);
        if (evt.getActionCommand()=="Help")
            Tasks.help();
        if (evt.getActionCommand()=="Info")
            Tasks.info();
```

```
        if (evt.getActionCommand()=="Copy to clipboard")
            Tasks.copy();
        if (evt.getActionCommand()=="Clear")
            Display.clear();
    }
}

/*************************************************************
Display contains methods for initializing and opening a GUI to
run CRYPTICON.
MenuActionListener contains the bindings between every button in
the GUI and their respective actions.

@author James Houghton
@version 06/05/2014
*************************************************************/
public class Display
{
    /*************************************************************
    Main frame of CRYPTICON.
    *************************************************************/
    public static JFrame pictureFrame;
    /*************************************************************
    Side panel placed on the left of the main frame.
    *************************************************************/
    public static JPanel sidePanel;

    /*************************************************************
    The menu bar of crypticon. Contains the three dropdown menus.
    *************************************************************/
    public static JMenuBar menuBar;
    /*************************************************************
    Effects dropdown menu. Contains all effects.
    *************************************************************/
    public static JMenu effects;
    public static JMenuItem effect1;
    public static JMenuItem effect2;
    public static JMenuItem effect3;
    public static JMenuItem effect4;
    public static JMenuItem effect5;
    public static JMenuItem effect6;
    public static JMenuItem effect7;
    public static JMenuItem cleareffects;

    /*************************************************************
    Main, image-containing panel.
    *************************************************************/
    public static JPanel picturePanel;
    /*************************************************************
    The label in which the loaded image is placed.
    *************************************************************/
    public static JLabel imageLabel;
    /*************************************************************
    The last instance of imageLabel.
```

```
    **************************************************************/
    public static JLabel prevImageLabel;

    /**************************************************************
    File dropdown menu. Contains File I/O options.
    **************************************************************/
    public static JMenu fileMenu;
    /**************************************************************
    Menu item used to open an image.
    **************************************************************/
    public static JMenuItem open;
    /**************************************************************
    Menu item used to save an image.
    **************************************************************/
    public static JMenuItem save;
    /**************************************************************
    Menu item used to save an image with a specific filename and location.
    **************************************************************/
    public static JMenuItem saveAs;
    /**************************************************************
    Menu item used to exit the program.
    **************************************************************/
    public static JMenuItem quit;

    /**************************************************************
    Dropdown menu containing help, information about the program, and its
    developers.
    **************************************************************/
    public static JMenu resources;
    /**************************************************************
    Menu item used to assist the user in operating the program.
    **************************************************************/
    public static JMenuItem help;
    /**************************************************************
    Menu item used to show information about the program and its developers.
    **************************************************************/
    public static JMenuItem info;

    /**************************************************************
    Input text field. Where text to be encoded is placed.
    **************************************************************/
    public static JTextArea inputTextField;
    /**************************************************************
    Output text field. Where the decoded message is displayed.
    **************************************************************/
    public static JTextArea outputTextField;

    /**************************************************************
    Button that encodes the image with text.
    **************************************************************/
    public static JButton encode;
    /**************************************************************
    Button that displays the encoded message in the output text field.
    **************************************************************/
```

```
    public static JButton decode;
    /*************************************************************
    Button that allows the user to copy the output text to the system
    clipboard.
    *************************************************************/
    public static JButton copy;
    /*************************************************************
    Button that clears and resets the output text field.
    *************************************************************/
    public static JButton clear;

    /*************************************************************
    Height of the loaded image.
    *************************************************************/
    public static int height;
    /*************************************************************
    Width of the loaded image.
    *************************************************************/
    public static int width;

    /*************************************************************
    The loaded image. Typically encoded with text.
    *************************************************************/
    public static BufferedImage loadedImage;
    /*************************************************************
    The loaded image that is used when effects are applied. Typically
    not encoded with text.
    *************************************************************/
    public static BufferedImage imageToBeSent;
    /*************************************************************
    The image that is loaded when the user wishes to revert the
    effects applied to it.
    *************************************************************/
    public static BufferedImage imageNoEffects;
    /*************************************************************
    When opening an image, filters visible files.
    *************************************************************/
    public static FileFilter fileFilter;

    /*************************************************************
    Previously opened file path.
    *************************************************************/
    public static String prevOpen;
    /*************************************************************
    Previously opened file.
    *************************************************************/
    public static String prevOpenFile;
    /*************************************************************
    Verified previously opened file path.
    *************************************************************/
    public static String prevOpenChecked;
    /*************************************************************
    Verified previously opened file.
    *************************************************************/
```

```
    public static String prevOpenFileChecked;
    /**************************************************************
    Text encoded in the loaded image.
    **************************************************************/
    public static String encodedText;

    /**************************************************************
    If true, no message was found in the loaded image.
    **************************************************************/
    public static boolean blankErrorOccurred;

    /**************************************************************
    Takes the defined JMenu entries in the beginning of the class
    and creates the action listeners and binds them together.
    **************************************************************/
    public static void initMenu()
    {
        menuBar = new JMenuBar();
        fileMenu = new JMenu("File");
        effects = new JMenu("Effects");
        resources = new JMenu("Resources");
        effect1 = new JMenuItem("Invert");
        effect2 = new JMenuItem("Fade");
        effect3 = new JMenuItem("Tint");
        effect4 = new JMenuItem("Black/White");
        effect5 = new JMenuItem("Remove color");
        effect6 = new JMenuItem("Grayscale");
        effect7 = new JMenuItem("Colorize");
        cleareffects = new JMenuItem("Clear effects");
        open = new JMenuItem("Open...");
        save = new JMenuItem("Save");
        saveAs = new JMenuItem("Save as...");
        quit = new JMenuItem("Exit");
        help = new JMenuItem("Help");
        info = new JMenuItem("Info");

        effect1.addActionListener(new MenuActionListener());
        effect2.addActionListener(new MenuActionListener());
        effect3.addActionListener(new MenuActionListener());
        effect4.addActionListener(new MenuActionListener());
        effect5.addActionListener(new MenuActionListener());
        effect6.addActionListener(new MenuActionListener());
        effect7.addActionListener(new MenuActionListener());
        cleareffects.addActionListener(new MenuActionListener());
        open.addActionListener(new MenuActionListener());
        save.addActionListener(new MenuActionListener());
        saveAs.addActionListener(new MenuActionListener());
        quit.addActionListener(new MenuActionListener());
        help.addActionListener(new MenuActionListener());
        info.addActionListener(new MenuActionListener());

        effects.add(effect1);
        effects.add(effect2);
        effects.add(effect3);
```

```
        effects.add(effect4);
        effects.add(effect5);
        effects.add(effect6);
        effects.add(effect7);
        effects.add(cleareffects);
        fileMenu.add(open);
        fileMenu.add(save);
        fileMenu.add(saveAs);
        fileMenu.add(quit);
        resources.add(info);
        resources.add(help);

        menuBar.add(fileMenu);
        menuBar.add(effects);
        menuBar.add(resources);

        pictureFrame.setJMenuBar(menuBar);
    }
    /*************************************************************
    Performs basic tasks to initialize the GUI.
    *************************************************************/
    public static void initFrame()
    {
        pictureFrame = new JFrame();
        pictureFrame.setResizable(false);
        pictureFrame.getContentPane().setLayout(new BorderLayout());
        pictureFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        pictureFrame.setTitle("CRYPTICON");
        Dimension dimension = new Dimension(40, 400);
        pictureFrame.setPreferredSize(dimension);
    }
    /*************************************************************
    Creates buttons to apply inputText to the image.
    *************************************************************/
    public static void initButtonsAndMenu()
    {
        encode = new JButton("Encode");
        decode = new JButton("Decode");
        clear = new JButton("Clear");
        copy = new JButton("Copy to clipboard");
        encode.addActionListener(new MenuActionListener());
        decode.addActionListener(new MenuActionListener());
        copy.addActionListener(new MenuActionListener());
        clear.addActionListener(new MenuActionListener());

        setDisabled();
        open.setEnabled(true);
        quit.setEnabled(true);

        fileFilter = new FileFilter()
        {
            public boolean accept(File file)
            {
                if (file.isDirectory())
```

```
            {
                return true;
            }
            String name = file.getName();
            return name.toLowerCase().endsWith(".png")||name.toLowerCase().endsWith(".jpg");
        }
        public String getDescription()
        {
            return "Image files (*.png, *.jpg)";
        }
    };
}
/*************************************************************
Disables all menu items that can be used when an image is loaded.
*************************************************************/
public static void setDisabled()
{
    effect1.setEnabled(false);
    effect2.setEnabled(false);
    effect3.setEnabled(false);
    effect4.setEnabled(false);
    effect5.setEnabled(false);
    effect6.setEnabled(false);
    effect7.setEnabled(false);
    cleareffects.setEnabled(false);
    save.setEnabled(false);
    saveAs.setEnabled(false);
    encode.setEnabled(false);
    decode.setEnabled(false);
    copy.setEnabled(false);
    clear.setEnabled(false);
}
/*************************************************************
Enables all menu items that can be used when an image is loaded.
*************************************************************/
public static void setEnabled()
{
    effect1.setEnabled(true);
    effect2.setEnabled(true);
    effect3.setEnabled(true);
    effect4.setEnabled(true);
    effect5.setEnabled(true);
    effect6.setEnabled(true);
    effect7.setEnabled(true);
    cleareffects.setEnabled(true);
    save.setEnabled(true);
    saveAs.setEnabled(true);
    encode.setEnabled(true);
    decode.setEnabled(true);
}
/*************************************************************
Creates and formats input and output text fields to be displayed.
*************************************************************/
public static void initSidePanelandTextField()
```

```
    {
        sidePanel = new JPanel();
        sidePanel.setLayout(new FlowLayout());
        sidePanel.setBackground(new Color(160,160,160));
        sidePanel.setBorder(BorderFactory.createLineBorder(Color.BLACK));
        inputTextField = new JTextArea(10,11);
        inputTextField.setBounds(5,5,100,100);
        inputTextField.setPreferredSize(new Dimension(150,200));
        inputTextField.setLineWrap(true);
        inputTextField.setWrapStyleWord(true);
        inputTextField.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
        createLineBorder(Color.BLACK,2),BorderFactory.createEmptyBorder(5,5,5,5)));
        sidePanel.add(inputTextField);
        outputTextField = new JTextArea(10,11);
        outputTextField.setBounds(5,5,100,100);
        outputTextField.setPreferredSize(new Dimension(150,200));
        outputTextField.setLineWrap(true);
        outputTextField.setWrapStyleWord(true);
        outputTextField.setEditable(false);
        outputTextField.setBorder(BorderFactory.createCompoundBorder(BorderFactory.
        createLineBorder(Color.BLACK,2),BorderFactory.createEmptyBorder(5,5,5,5)));
        outputTextField.setBackground(new Color(216,216,216));
    }
    /*************************************************************
    Adds the output text field to the side panel.
    *************************************************************/
    public static void applyOutputTextField()
    {
        sidePanel.add(outputTextField);
    }
    /*************************************************************
    Adds encode and decode buttons to the side panel.
    *************************************************************/
    public static void applyButtons()
    {
        sidePanel.add(encode);
        sidePanel.add(decode);
    }
    /*************************************************************
    Adds the side panel to the main frame of the program.
    *************************************************************/
    public static void applySidePanel()
    {
        pictureFrame.add(sidePanel);
    }
    /*************************************************************
    Creates the frame which a scaled version of the loaded image
    will be held inside of.
    *************************************************************/
    public static void initImagePanel()
    {
         picturePanel = new JPanel(new BorderLayout());
    }
    /*************************************************************
```

```
    Converts an Image to a BufferedImage.
    ***************************************************************/
    public static BufferedImage toBufferedInstance(Image image,int width,int height)
    {
        BufferedImage bi = new BufferedImage(width,height,BufferedImage.TYPE_INT_ARGB);
        Graphics2D g2d = bi.createGraphics();
        g2d.drawImage(image, 0, 0, null);
        g2d.dispose();
        return bi;
    }
    /***************************************************************
    Returns a filled image label, with a blank or custom image.
    ***************************************************************/
    public static JLabel initImageLabel(String filename,int mode)
    {
        JLabel imageLabel;
        BufferedImage image=null;
        if(mode==0)
        {
            boolean success=false;
            while(success==false && filename!=null){
                try
                {
                    image = ImageIO.read(new File(filename));
                    prevOpenChecked=prevOpen;
                    prevOpenFileChecked=prevOpenFile;
                    success=true;
                }
                catch(IOException e)
                {
                boolean again=tryAgain();
                if (again==true)
                {
                    filename=fnPrompt();
                    if(filename!=null)
                    {
                        if(filename.length()>4)
                        {
                            if(!filename.substring(filename.length()-4).toLowerCase().equals(
                            ".png")||!filename.substring(filename.length()-4).toLowerCase().
                            equals(".jpg"))
                            {
                                filename+=".png";
                                try
                                {
                                    ImageIO.read(new File(filename));
                                    setEnabled();
                                }
                                catch(IOException e1)
                                {
                                    filename = filename.substring(0,filename.length()-4);
                                    filename+=".jpg";
                                    try
                                    {
```

```
                                    ImageIO.read(new File(filename));
                                    setEnabled();
                                }
                                catch(IOException e2)
                                {
                                    filename = filename.substring(0,filename.length()-4);
                                }
                            }
                        }
                        else
                            setEnabled();
                    }
                }
                else
                    return prevImageLabel;
            }
            else
                return prevImageLabel;
            }
        }
        else{
            try
            {
                InputStream stream = Display.class.getResourceAsStream(
                "resources/images/blank.png");
                image = ImageIO.read(stream);
            }
            catch(IOException e){}
        }
        imageNoEffects=image;
        imageLabel = initImageLabel(image);
        setImageToBeSent(image);
        prevImageLabel = imageLabel;
        return imageLabel;
    }
    /*************************************************************
    Opens a dialog prompting the user to attempt to find a file
    again.  Is called until a valid filename is retrieved.
    *************************************************************/
    public static boolean tryAgain()
    {
        JOptionPane op = new JOptionPane();
        op.setMessageType(JOptionPane.ERROR_MESSAGE);
        int i=op.showConfirmDialog(null,"File does not exist or cannot be read!\nEnsure the
        selected file is a JPG or PNG image.\nTry again?","File cannot be read!",JOptionPane.
        YES_NO_OPTION);
        if(i==JOptionPane.YES_OPTION)
            return true;
        if(i==JOptionPane.NO_OPTION)
            return false;
        else
            return false;
    }
```

```
    /**************************************************************
    Creates and returns a filled label with a custom filename.
    **************************************************************/
    public static JLabel initImageLabel(String filename)
    {
        JLabel label = initImageLabel(filename,0);
        return label;
    }
    /**************************************************************
    Returns a filled label with a scaled BufferedImage, rather than
    a file that must be read from a disk.
    **************************************************************/
    public static JLabel initImageLabel(BufferedImage image)
    {
        blankErrorOccurred=false;
        JLabel imageLabel;
        height = image.getHeight();
        width = image.getWidth();
        loadedImage = image;
        double scaler = 600.0/width;
        double dsWidth = width*scaler;
        double dsHeight = height*scaler;
        int sWidth = (int) dsWidth;
        int sHeight = (int) dsHeight;
        Image sImage=image.getScaledInstance(sWidth,sHeight,Image.SCALE_SMOOTH);
        BufferedImage bsImage=toBufferedInstance(sImage,sWidth,sHeight);
        imageLabel = new JLabel(new ImageIcon(bsImage));
        return imageLabel;
    }
    /**************************************************************
    Gets input from the input text field and converts it to a string.
    This method can also retrieve the loaded image's text.
    **************************************************************/
    public static String getText(int mode)
    {
        if(mode==0)
        {
            return encodedText;
        }
        else
            return inputTextField.getText();
    }
    /**************************************************************
    Returns the encoded text in the image when no parameter is
    passed.
    **************************************************************/
    public static String getText()
    {
        return getText(0);
    }
    /**************************************************************
    Returns the text from the output text field.
    **************************************************************/
    public static String getOutput()
```

```
    {
        return outputTextField.getText();
    }
    /**************************************************************
    Sets the text of the output text field to the decrypted text if a
    message was successfully decrypted from the image.
    **************************************************************/
    public static void setText(String string)
    {
        if(blankErrorOccurred==false)
        {
            outputTextField.setBackground(Color.WHITE);
            outputTextField.setText(string);
            clear.setEnabled(true);
            copy.setEnabled(true);
        }
    }
    /**************************************************************
    Sets BufferedImage references to the passed in BufferedImage.
    Afterwards, the initialized label containing the scaled image is
    displayed.
    **************************************************************/
    public static void loadBI(BufferedImage image, int mode)
    {
        if (mode==1)
        {
            imageToBeSent = image;
            return;
        }
        else if (mode==2)
        {

        }
        else if (mode==3)
        {
            imageToBeSent = image;
        }
        else
            imageNoEffects = image;
        imageLabel = initImageLabel(image);
        Border paddingBorder = BorderFactory.createEmptyBorder(10,10,10,10);
        Border border = BorderFactory.createLineBorder(Color.BLACK,5);
        imageLabel.setBorder(BorderFactory.createCompoundBorder(border,paddingBorder));
        picturePanel.add(BorderLayout.EAST,imageLabel);
        pictureFrame.getContentPane().add(BorderLayout.EAST,picturePanel);
        pictureFrame.validate();
        pictureFrame.repaint();
    }
    /**************************************************************
    Runs loadImage again.
    **************************************************************/
    public static void loadImage(String filename)
    {
        loadImage(filename,0);
```

```
    }
    /*************************************************************
    Resets variables that would have changed upon image opening and
    changing. Takes given String and checks if the extension is
    present, and which extensions are valid. Then, it initializes and
    displays the image specified by the given String filename.
    *************************************************************/
    public static void loadImage(String filename,int mode)
    {
        setEncodedText(null);
        if(mode==0)
        {
            filename=fnPrompt();
            if(filename!=null)
            {
                if(filename.length()>4)
                {
                    if(!filename.substring(filename.length()-4).toLowerCase().equals(".png")||!
                    filename.substring(filename.length()-4).toLowerCase().equals(".jpg"))
                    {
                        filename+=".png";
                        try
                        {
                            ImageIO.read(new File(filename));
                            setEnabled();
                        }
                        catch(IOException e1)
                        {
                            filename = filename.substring(0,filename.length()-4);
                            filename+=".jpg";
                            try
                            {
                                ImageIO.read(new File(filename));
                                setEnabled();
                            }
                            catch(IOException e2)
                            {
                                filename = filename.substring(0,filename.length()-4);
                            }
                        }
                    }
                    else
                        setEnabled();
                }
                else
                    filename+=".png";
                    try
                    {
                        ImageIO.read(new File(filename));
                        setEnabled();
                    }
                    catch(IOException e1)
                    {
                        filename = filename.substring(0,filename.length()-4);
```

```
                    filename+=".jpg";
                    try
                    {
                        ImageIO.read(new File(filename));
                        setEnabled();
                    }
                    catch(IOException e2)
                    {
                        filename = filename.substring(0,filename.length()-4);
                    }
                }
            unloadImage();
            clear();
            initImageLabelS2(filename);
            Tasks.setLoadedImage(filename);
        }
    }
    else
    {
        filename = "resources/images/blank.png";
        initImageLabelS2(filename,1);
    }
 }
 /*************************************************************
 Performs the post-creation steps to accurately display the
 new image label created by initImageLabel.
 *************************************************************/
 public static void initImageLabelS2(String filename,int mode)
{
    imageLabel=initImageLabel(filename,mode);
    Border paddingBorder = BorderFactory.createEmptyBorder(10,10,10,10);
    Border border = BorderFactory.createLineBorder(Color.BLACK,5);
    imageLabel.setBorder(BorderFactory.createCompoundBorder(border,paddingBorder));
    picturePanel.add(BorderLayout.EAST,imageLabel);
    pictureFrame.getContentPane().add(BorderLayout.EAST,picturePanel);
    pictureFrame.validate();
    pictureFrame.repaint();
}
 /*************************************************************
 When not passed a mode, performs the default initImageLabel
 initialization and finishes the post-creation phase to display
 the image label.
 *************************************************************/
public static void initImageLabelS2(String filename)
{
    imageLabel=initImageLabel(filename);
    Border paddingBorder = BorderFactory.createEmptyBorder(10,10,10,10);
    Border border = BorderFactory.createLineBorder(Color.BLACK,5);
    imageLabel.setBorder(BorderFactory.createCompoundBorder(border,paddingBorder));
    picturePanel.add(BorderLayout.EAST,imageLabel);
    pictureFrame.getContentPane().add(BorderLayout.EAST,picturePanel);
    pictureFrame.validate();
    pictureFrame.repaint();
 }
```

```
    /*************************************************************
    Prompts the user, by opening a JFileChooser, for a valid
    filename. This filename will be given back to the method it was
    called in.
    *************************************************************/
   public static String fnPrompt()
   {
        JFileChooser fileChooser = new JFileChooser(prevOpen);
        fileChooser.setFileFilter(fileFilter);
        Action details = fileChooser.getActionMap().get("viewTypeDetails");
        details.actionPerformed(null);
        int returnval = fileChooser.showOpenDialog(pictureFrame);
            if(returnval == JFileChooser.APPROVE_OPTION)
            {
                String prevOpenFull = fileChooser.getSelectedFile().getAbsolutePath();
                prevOpen = prevOpenFull;
                int length=prevOpen.length();
                int i=length;
                while(i>0 && !prevOpen.substring(i-1,i).equals("\\"))
                {
                    i--;
                }
                prevOpen = prevOpen.substring(0,i);
                prevOpenFile = prevOpenFull.substring(i,length);
                if(prevOpenFile.length()>4)
                {
                    if(prevOpenFile.substring(prevOpenFile.length()-4).toLowerCase().equals(
                    ".jpg"))
                    {
                        prevOpenFile=prevOpenFile.substring(0,prevOpenFile.length()-4);
                        prevOpenFile+=".png";
                    }
                    if(!prevOpenFile.substring(prevOpenFile.length()-4).toLowerCase().equals(
                    ".png"))
                    {
                        prevOpenFile+=".png";
                    }
                }
                else prevOpenFile+=".png";
                return fileChooser.getSelectedFile().getAbsolutePath();
            }
          else
            return null;

   }
    /*************************************************************
    Prompts the user for a location to save the loaded BufferedImage.
    *************************************************************/
   public static String fnSave()
   {
        JFileChooser fileChooser = new JFileChooser(prevOpenChecked);
        fileChooser.setSelectedFile(new File(prevOpenFileChecked));
        Action details = fileChooser.getActionMap().get("viewTypeDetails");
        details.actionPerformed(null);
```

```
        int returnval = fileChooser.showSaveDialog(pictureFrame);
        if(returnval == JFileChooser.APPROVE_OPTION)
            return fileChooser.getSelectedFile().getAbsolutePath();
        else
            return null;
    }
    /*************************************************************
    Returns the height of the loaded BufferedImage.
    *************************************************************/
    public static int getHeight()
    {
        return height;
    }
    /*************************************************************
    Returns the width of the loaded BufferedImage.
    *************************************************************/
    public static int getWidth()
    {
        return width;
    }
    /*************************************************************
    Unloads and nullifies the imageLabel containing the loaded
    BufferedImage.
    *************************************************************/
    public static void unloadImage()
    {
        if(imageLabel!=null)
        {
            picturePanel.remove(imageLabel);
            imageLabel = null;
        }
    }
    /*************************************************************
    Returns the unencrypted BufferedImage.
    *************************************************************/
    public static BufferedImage getImage()
    {
        return getImage(0);
    }
    /*************************************************************
    Depending on the passed in mode, returns the unencrypted loaded
    BufferedImage, or just the encrypted or not BufferedImage.
    *************************************************************/
    public static BufferedImage getImage(int mode)
    {
        if(mode==1)
            return loadedImage;
        else
            return imageToBeSent;
    }
    /*************************************************************
    Add the 'Copy to clipboard' button to the side panel.
    *************************************************************/
    public static void applyCopy()
```

```
    {
        sidePanel.add(copy);
    }
    /*************************************************************
    Adds each element to the side panel in order.
    *************************************************************/
    public static void apply()
    {
        applyButtons();
        applyOutputTextField();
        applyCopy();
        applyClear();
        applySidePanel();
    }
    /*************************************************************
    Centers CRYPTICON on the desktop.
    *************************************************************/
    public static void centerFrame()
    {
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Point middle = new Point(screenSize.width/2,screenSize.height/2);
        Point middleCorrected = new Point(middle.x-(pictureFrame.getWidth()/2),middle.y-(
        pictureFrame.getHeight()/2));
        pictureFrame.setLocation(middleCorrected);
    }
    /*************************************************************
    Set the passed in BufferedImage to the unencrypted BufferedImage
    reference in the Display class.
    *************************************************************/
    public static void setImageToBeSent(BufferedImage image)
    {
        imageToBeSent=image;
    }
    /*************************************************************
    Add the 'clear' button to the side panel.
    *************************************************************/
    public static void applyClear()
    {
        sidePanel.add(clear);
    }
    /*************************************************************
    Clear and reset the formatting of the output text field.
    Disable buttons that would have no function when output text field
    contains no text.
    *************************************************************/
    public static void clear()
    {
        outputTextField.setText("");
        outputTextField.setBackground(new Color(216,216,216));
        copy.setEnabled(false);
        clear.setEnabled(false);
    }
    /*************************************************************
    Loads the BufferedImage that was not used when effects were applied.
```

```
    All effects applied to a loaded, unsaved image are reset.
    **************************************************************/
    public static void clearEffects()
    {
        Display.unloadImage();
        loadBI(imageNoEffects,1);
        BufferedImage tmpimage = imageNoEffects;
        try
        {
            imageNoEffects=Steg.encrypt(imageNoEffects);
            loadBI(imageNoEffects,2);
        }
        catch(Exception e){loadBI(tmpimage,1);}
    }
    /*************************************************************
    Store text encoded in the image.
    **************************************************************/
    public static void setEncodedText(String message)
    {
        encodedText = message;
    }
    /*************************************************************
    If a desired string to be encoded is too long, produce an error
    and tell the user how many characters need to be removed.
    **************************************************************/
    public static void lengthError(int over)
    {
        String s="s";
        if(over==1)
            s="";
        JOptionPane.showMessageDialog(null,"Your message is "+over+" character"+s+" too
        long.\nYou must shorten your message or choose a larger image.","Error!",JOptionPane.
        ERROR_MESSAGE);
    }
    /*************************************************************
    If no message is found in the image, display an error.
    **************************************************************/
    public static void blankError()
    {
        JOptionPane.showMessageDialog(null,"No message was found in this image.\nEnsure that it
        was encoded using CRYPTICON.","Error!",JOptionPane.ERROR_MESSAGE);
        blankErrorOccurred=true;
        clear();
    }
    /*************************************************************
    Let the user know, if the encryption may take a long time, when
    the process has started.
    **************************************************************/
   public static void eTimeErrorStart()
    {
       JOptionPane.showMessageDialog(null,"Encryption of this message may take a substantial
       amount of time,\ndepending on the processing power of your machine.\nAnother popup will
       inform you when this process is finished.\nContinue?","Encrypting...",JOptionPane.
       ERROR_MESSAGE);
```

```
    }
    /**************************************************************
    If the encryption was thought to take a long time, when the process
    has finished, notify the user.
    **************************************************************/
   public static void eTimeErrorEnd()
    {
        JOptionPane.showMessageDialog(null,"The encryption process has been completed.\nYou are
        now free to save this image.","Process completed.",JOptionPane.INFORMATION_MESSAGE);
    }
    /**************************************************************
    If the decryption of a certain image is thought to take a long time,
    notifies the user of this, and asks them
    if they want to proceed.
    **************************************************************/
    public static int timeError()
    {
        int choice = JOptionPane.showConfirmDialog(null,"Decryption of this message may take a
        substantial amount of time due to its length,\ndepending on the processing power of
        your machine.\nHowever, the message should be able to be retrieved.\nContinue?",
        "Decrypting...",JOptionPane.YES_NO_OPTION);
        if(choice==JOptionPane.YES_OPTION)
            return 1;
        else
            return 0;
    }
    /**************************************************************
    Sets up the frames to hold the images and menus and loads them.
    **************************************************************/
    public static void init()
    {
        initFrame();
        initMenu();
        initSidePanelandTextField();
        initButtonsAndMenu();
        apply();
        pictureFrame.pack();
        initImagePanel();
        loadImage("",1);
        Dimension dim = new Dimension(800,600);
        pictureFrame.setPreferredSize(dim);
        pictureFrame.setSize(dim);
        centerFrame();
        blankErrorOccurred=false;
        pictureFrame.setIconImage(new ImageIcon(Display.class.getResource(
        "resources/images/icon.ico")).getImage());
        pictureFrame.setVisible(true);

    }
}
```