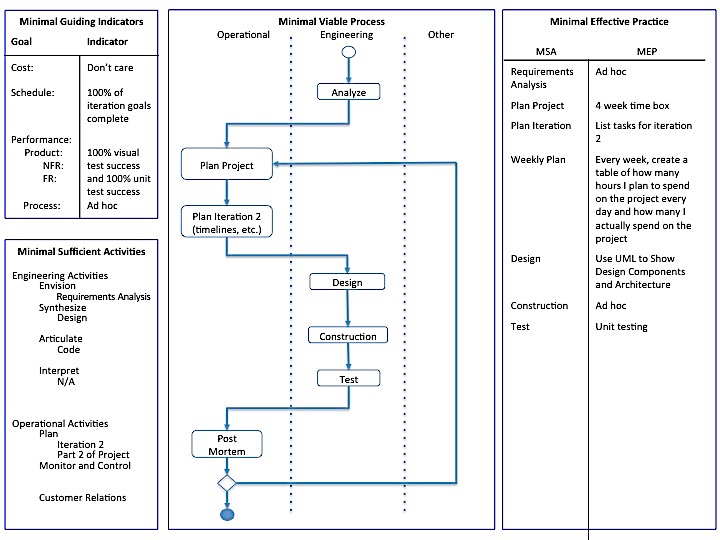
Iteration 5

# Process for Iteration 5



The process has not changed since Iteration 4.

# Requirements Analysis

The requirements have not changed since Iteration 1. They are as follows:

Given a secret image file and two innocent image files, the tool should

* Be able to read in image files and store the pixel information
* Use the extended visual cryptography scheme to encode the secret image pixels into the two innocent images
* Store the encoded images in new image files
  + The filenames and location can be specified by the user. If not, the files are named share1 and share2 and gets stored on the Desktop.

Given two encoded image files, the tool should

* Be able to read in the files and store the pixel information
* Use the extended visual cryptography scheme to decode the secret image from the encoded images (similar to super imposing them)
* The image revealing the secret gets stored in a new image file
  + The filename and location can be specified by the user. If not, the file is named secretMsg and gets stored on the Desktop.

The visual cryptography tool will only work with PNG and JPEG images. The images involved with the encoding process must have the same dimensions. The tool can handle images of any coloring.

# Plans for Project

Iteration 1 (Sept. 11 – Oct. 9):

* Create a graphical user interface
* Get the tool working for strictly black and white images
* Test the tool to check the quality of the encoded shares and the decoded message

Iteration 2 (Oct. 10 – Nov. 6):

* Research visual cryptography schemes with gray scale images

Iteration 3 (Nov. 7 – Dec. 4):

* Implement the visual cryptography scheme with grayscale images
* Begin researching how to modify the current algorithm to handle color images

Iteration 4 (Dec. 5 – Jan. 1):

* Research how to add the ability to encode and decode multicolor images

Iteration 5 (Jan. 2 – Jan. 29):

* Implement the encoding and decoding of color images
* Add unit tests for the components of the visual cryptography tool

Iteration 6 (Jan. 30 – Feb. 26):

* Analyze the tool and look for ways to improve efficiency (performance and memory storage)
* Add features to project to help boost robustness (i.e. add in checks to keep the user from breaking the tool easily)

# Plans for Iteration 5

* Implement the encoding and decoding of color images
* Add unit tests for the components of the visual cryptography tool

# Weekly Plans

Week 1:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, January 2nd | 0 | 0 |
| Sunday, January 3rd | 0 | 0 |
| Monday, January 4th | 2 | 0 |
| Tuesday, January 5th | 2 | 3 |
| Wednesday, January 6th | 3 | 1.5 |
| Thursday, January 7th | 0 | 0 |
| Friday, January 8th | 0 | 0 |

Week 2:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, January 9th | 0 | 0 |
| Sunday, January 10th | 0 | 0 |
| Monday, January 11th | 0 | 0 |
| Tuesday, January 12th | 0 | 0 |
| Wednesday, January 13th | 0 | 0 |
| Thursday, January 14th | 0 | 0 |
| Friday, January 15th | 3 | 2 |

Week 3:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, January 16th | 0 | 0 |
| Sunday, January 17th | 0 | 0 |
| Monday, January 18th | 4 | 4 |
| Tuesday, January 19th | 2 | 1 |
| Wednesday, January 20th | 4 | 3 |
| Thursday, January 21st | 0 | 0 |
| Friday, January 22nd | 0 | 0 |

Week 4:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, January 23rd | 0 | 0 |
| Sunday, January 24th | 0 | 0 |
| Monday, January 25th | 4 | 2 |
| Tuesday, January 26th | 2 | 3 |
| Wednesday, January 27th | 3 | 3 |
| Thursday, January 28th | 4 | 5 |
| Friday, January 29th | 3 | 4 |

# Design

Figure 1: UML Diagram from Iteration 1

Note the design has not changed since Iteration 2.

Before changing the Java files to handle the gray scale images, I exported the PlantUML diagram of the visual cryptography tool. Figure 1 shows the class relations.

In iteration 4, I decided to focus on researching techniques for encrypting color images. The most promising technique I found came from Varalakshmi, R, and Parameswari, and it utilized Visual Information Pixel (VIP) synchronization. VIP synchronization helps hide the secret image pixel information inside the innocent pixels. The process for encrypting a secret image is as follows:

1. Gather and process the two innocent images and one secret image.
2. Half-tone the innocent images using error diffusion.
3. Split the secret image into three images. One image represents only the red concentration of the picture, the second represents the green concentration, and the third represents blue.
4. Perform VIP synchronization on the innocent images and the three secret images.
5. Use error diffusion on the encrypted shares to smooth any pixels that cause the encoded image to look as if they are hiding something.

Decryption for this technique does not require the user to have a computer. The images can be printed on transparencies and stacked to reveal the secret image.

# Construction

I began the construction phase by fixing my decryption technique. Using the XOR operation is equivalent to an individual stacking the two encoded shares printed on transparencies and viewing the secret message.

The second portion of the construction phase was focused on implementing the VIP synchronization technique described in the design section. Three new methods were added to assist in the encryption technique. The first method constructed was used to split the secret image into three images. One image holds the red portion of the secret image, one holds the green portion, and the third holds the blue portion. The next step was to implement error diffusion. Error diffusion is used to blend the pixels such that the colors blend together more naturally. In terms of the encoded images, the error diffusion assists in blending the hidden information with the original image.

The final and most difficult method performed the VIP synchronization and encoding the secret data into the cover images. Once the method construction was completed, there were still issues with printing the encoded images to a file. Previously, I had been able to use TYPE\_INT\_ARGB as the BufferedImage type. Since I am creating the RGB values from scratch, I no longer had values for the alpha. Alpha handles the opacity of the image, so I thought my images were never being printed to the file. The images were being printed; they were just 100% transparent. The issue was fixed when I made the BufferedImage type TYPE\_INT\_RGB. Tests of encryption and decryption are shown in the Test section for grayscale and colored images.

The decryption technique appeared to be working well. Tests are shown in the section below.

The final change I made to the cryptography this iteration was fixing the frames to appear in the center of the screen instead of in the upper left corner.

# Test

I was testing different images throughout the construction process. The second step of implementation was programming the error diffusion.

|  |
| --- |
|  |
| Figure 2: Original Images (left), Images after Error Diffusion (right) |

Figure 2 shows the results of performing error diffusion on two grayscale images. The puppy pictures appear to be the same, while the rose looks lighter after being processed. The technique worked well enough that, if you only saw the final images, you would not know anything was unusual about the photos.

After being satisfied with the error diffusion, I completed the implementation of the VIP synchronization by adding the code that combined the pixel information of the cover images and the secret image. The first test of the encryption and decryption process was with my grayscale images from Iteration 3. Figures 3 shows the original images. The encoded images are displayed in Figure 4, while Figure 5 shows the secret revealed from stacking the two images in Figure 4.

|  |
| --- |
|  |
| Figure 3: Secret Image to be Encoded (left), Innocent Image 1 (middle), Innocent Image 2 (right) |

|  |
| --- |
|  |
| Figure 4: Encoded Share A (left), Encoded Share B (right) |



Figure 5: Result of decoding shares A and B from Figure 4.

Encryption and Decryption Color Images

The grayscale image testing did not go well, so I wanted to see how the tool handled colorful images. The goal was to determine if the grayscaling was the problem or if the problem was with the implementation. Figure 6 shows the original color images to be used for this visual test of the encryption and decryption process. The images created from the encoding process are shown in Figure 7. Finally, Figure 8 displays the result of decrypting the two encoded shares from Figure 7.

|  |
| --- |
|  |
| Figure 6: Secret Image to be Encoded (left), Innocent Image 1 (middle), Innocent Image 2 (right) |

|  |
| --- |
|  |
| Figure 7: Encoded Share A (left), Encoded Share B (right) |

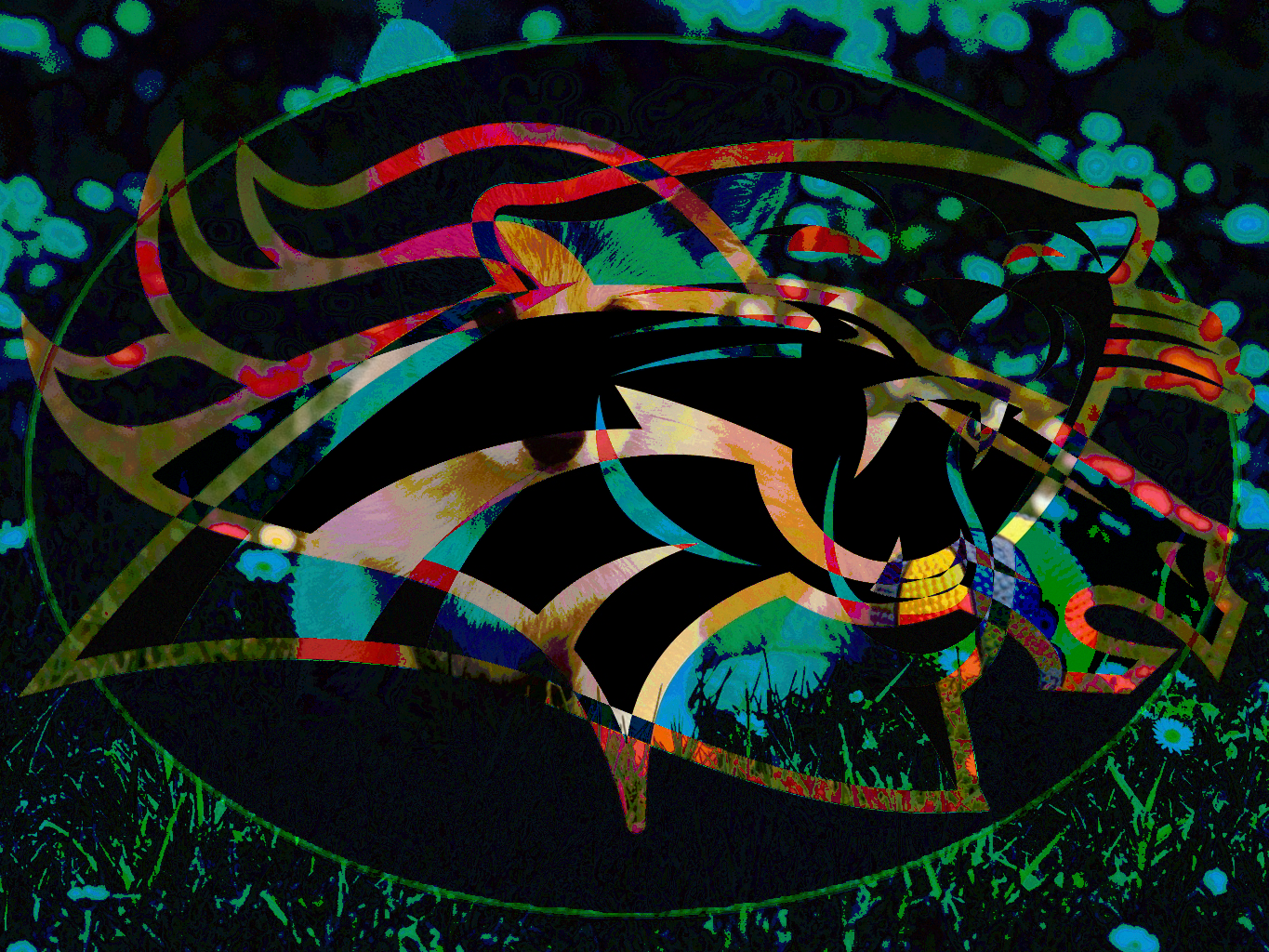


Figure 8: Result of decoding shares A and B from Figure 7.

# Post Mortem

Based on the results from testing, the encryption process needs to be re-evaluated. I had modified one portion of the algorithm to keep the encoded images the same size as the original images, which is a property of a size invariant scheme. During Iteration 6, I want to see what happens when I perform the expansion such that 1 pixel becomes four pixels.

# Source Code

MainFrame.java

1 package Masters\_Proj;  
 2   
 3 /\*  
 4 \* To change this license header, choose License Headers in Project Properties.  
 5 \* To change this template file, choose Tools | Templates  
 6 \* and open the template in the editor.  
 7 \*/  
 8   
 9 /\*\*  
 10 \*  
 11 \* @author allisonholt  
 12 \*/  
 13 public class MainFrame extends javax.swing.JFrame {  
 14   
 15 /\*\*  
 16 \* Creates new form StartFrame  
 17 \*/  
 18 public MainFrame() {  
 19 initComponents();  
 20 this.setLocationRelativeTo(null);  
 21 }  
 22   
 23 /\*\*  
 24 \* This method is called from within the constructor to initialize the form.  
 25 \* WARNING: Do NOT modify this code. The content of this method is always  
 26 \* regenerated by the Form Editor.  
 27 \*/  
 28 @SuppressWarnings("unchecked")  
 29 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 30 private void initComponents() {  
 31   
 32 welcomeBanner = new javax.swing.JLabel();  
 33 jScrollPane1 = new javax.swing.JScrollPane();  
 34 descriptionArea = new javax.swing.JTextArea();  
 35 jScrollPane2 = new javax.swing.JScrollPane();  
 36 directionsArea = new javax.swing.JTextArea();  
 37 encodeButton = new javax.swing.JButton();  
 38 decodeButton = new javax.swing.JButton();  
 39   
 40 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 41 setTitle("Holt Visual Cryptography");  
 42   
 43 welcomeBanner.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
 44 welcomeBanner.setText("Welcome to the Holt Visual Cryptography Tool!");  
 45   
 46 descriptionArea.setEditable(false);  
 47 descriptionArea.setColumns(20);  
 48 descriptionArea.setLineWrap(true);  
 49 descriptionArea.setRows(5);  
 50 descriptionArea.setText("The Holt Cryptography Tool allows you to encrypt or decrypt a secret image using extended visual cryptography. The secret image gets embedded into two innocent images that must be superimposed in order to reveal the secret information.");  
 51 descriptionArea.setWrapStyleWord(true);  
 52 jScrollPane1.setViewportView(descriptionArea);  
 53   
 54 directionsArea.setColumns(20);  
 55 directionsArea.setLineWrap(true);  
 56 directionsArea.setRows(5);  
 57 directionsArea.setText("If you wish to encrypt a secret image, then select the encode button. If you wish to decrypt a secret message, then select the decode button.");  
 58 directionsArea.setWrapStyleWord(true);  
 59 jScrollPane2.setViewportView(directionsArea);  
 60   
 61 encodeButton.setText("Encode");  
 62 encodeButton.addActionListener(  
 63 new java.awt.event.ActionListener() {  
 64 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 65 encodePressed(evt);  
 66 }  
 67 });  
 68   
 69 decodeButton.setText("Decode");  
 70 decodeButton.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);  
 71 decodeButton.addActionListener(  
 72 new java.awt.event.ActionListener() {  
 73 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 74 decodePressed(evt);  
 75 }  
 76 });  
 77   
 78 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
 79 getContentPane().setLayout(layout);  
 80 layout.setHorizontalGroup(  
 81 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 82 .addComponent(welcomeBanner, javax.swing.GroupLayout.DEFAULT\_SIZE, 600, Short.MAX\_VALUE)  
 83 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
 84 .addContainerGap()  
 85 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
 86 .addComponent(jScrollPane2)  
 87 .addComponent(jScrollPane1))  
 88 .addContainerGap())  
 89 .addGroup(layout.createSequentialGroup()  
 90 .addGap(66, 66, 66)  
 91 .addComponent(encodeButton)  
 92 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
 93 .addComponent(decodeButton)  
 94 .addGap(66, 66, 66))  
 95 );  
 96 layout.setVerticalGroup(  
 97 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 98 .addGroup(layout.createSequentialGroup()  
 99 .addGap(24, 24, 24)  
100 .addComponent(welcomeBanner, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
101 .addGap(18, 18, 18)  
102 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 64, javax.swing.GroupLayout.PREFERRED\_SIZE)  
103 .addGap(18, 18, 18)  
104 .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 47, javax.swing.GroupLayout.PREFERRED\_SIZE)  
105 .addGap(18, 18, 18)  
106 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
107 .addComponent(encodeButton)  
108 .addComponent(decodeButton))  
109 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
110 );  
111   
112 pack();  
113 }// </editor-fold>//GEN-END:initComponents  
114   
115 private void encodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_encodePressed  
116 // TODO add your handling code here:  
117 new EncodeFrame().setVisible(true);  
118 this.setVisible(false);  
119   
120 }//GEN-LAST:event\_encodePressed  
121   
122 private void decodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_decodePressed  
123 // TODO add your handling code here:  
124 new DecodeFrame().setVisible(true);  
125 this.setVisible(false);  
126 }//GEN-LAST:event\_decodePressed  
127   
128 /\*\*  
129 \* @param args the command line arguments  
130 \*/  
131 public static void main(String args[]) {  
132 /\* Set the Nimbus look and feel \*/  
133 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
134 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
135 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
136 \*/  
137 try {  
138 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
139 if ("Nimbus".equals(info.getName())) {  
140 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
141 break;  
142 }  
143 }  
144 }   
145 catch (ClassNotFoundException ex) {  
146 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
147 }   
148 catch (InstantiationException ex) {  
149 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
150 }   
151 catch (IllegalAccessException ex) {  
152 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
153 }   
154 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
155 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
156 }  
157 //</editor-fold>  
158 //</editor-fold>  
159   
160 /\* Create and display the form \*/  
161 java.awt.EventQueue.invokeLater(  
162 new Runnable() {  
163 public void run() {  
164 new MainFrame().setVisible(true);  
165 }  
166 });  
167 }  
168   
169 // Variables declaration - do not modify//GEN-BEGIN:variables  
170 private javax.swing.JButton decodeButton;  
171 private javax.swing.JTextArea descriptionArea;  
172 private javax.swing.JTextArea directionsArea;  
173 private javax.swing.JButton encodeButton;  
174 private javax.swing.JScrollPane jScrollPane1;  
175 private javax.swing.JScrollPane jScrollPane2;  
176 private javax.swing.JLabel welcomeBanner;  
177 // End of variables declaration//GEN-END:variables  
178 }  
179

EncodeFrame.java

1 package Masters\_Proj;  
 2   
 3 import java.awt.image.BufferedImage;  
 4 import java.io.File;  
 5 import java.io.IOException;  
 6 import javax.imageio.ImageIO;  
 7 import javax.swing.JFileChooser;  
 8 import javax.swing.JOptionPane;  
 9   
 10 /\*  
 11 \* To change this license header, choose License Headers in Project Properties.  
 12 \* To change this template file, choose Tools | Templates  
 13 \* and open the template in the editor.  
 14 \*/  
 15   
 16 /\*\*  
 17 \*  
 18 \* @author allisonholt  
 19 \*/  
 20 public class EncodeFrame extends javax.swing.JFrame {  
 21   
 22 /\*\*  
 23 \* Creates new form EncodeFrame  
 24 \*/  
 25 public EncodeFrame() {  
 26 initComponents();  
 27 this.setLocationRelativeTo(null);  
 28 }  
 29   
 30 public EncodeFrame(EncodeFrame prevState)  
 31 {  
 32 this.secretTextField.setText(prevState.secretTextField.getText());  
 33 }  
 34   
 35 /\*\*  
 36 \* This method is called from within the constructor to initialize the form.  
 37 \* WARNING: Do NOT modify this code. The content of this method is always  
 38 \* regenerated by the Form Editor.  
 39 \*/  
 40 @SuppressWarnings("unchecked")  
 41 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 42 private void initComponents() {  
 43   
 44 imageChooser = new javax.swing.JFileChooser();  
 45 directoryChooser = new javax.swing.JFileChooser();  
 46 cancelButton = new javax.swing.JButton();  
 47 encodeButton = new javax.swing.JButton();  
 48 panel1 = new javax.swing.JPanel();  
 49 secretTextField = new javax.swing.JTextField();  
 50 jLabel1 = new javax.swing.JLabel();  
 51 browseButton1 = new javax.swing.JButton();  
 52 jPanel1 = new javax.swing.JPanel();  
 53 jLabel2 = new javax.swing.JLabel();  
 54 innocentTextField1 = new javax.swing.JTextField();  
 55 browseButton2 = new javax.swing.JButton();  
 56 innocentTextField2 = new javax.swing.JTextField();  
 57 browseButton3 = new javax.swing.JButton();  
 58 optionalPanel = new javax.swing.JPanel();  
 59 jLabel3 = new javax.swing.JLabel();  
 60 jLabel4 = new javax.swing.JLabel();  
 61 jLabel5 = new javax.swing.JLabel();  
 62 filename1 = new javax.swing.JTextField();  
 63 filename2 = new javax.swing.JTextField();  
 64 jLabel6 = new javax.swing.JLabel();  
 65 storageDirectoryTextField = new javax.swing.JTextField();  
 66 browseButton4 = new javax.swing.JButton();  
 67   
 68 imageChooser.setDialogTitle("Choose an Image");  
 69 imageChooser.setFileFilter(new ImageCustomFilter());  
 70   
 71 directoryChooser.setDialogTitle("Choose a Directory");  
 72 directoryChooser.setFileFilter(new DirectoryCustomFilter());  
 73 directoryChooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES\_ONLY);  
 74   
 75 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 76 setResizable(false);  
 77   
 78 cancelButton.setText("Cancel");  
 79 cancelButton.addActionListener(  
 80 new java.awt.event.ActionListener() {  
 81 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 82 cancelPressed(evt);  
 83 }  
 84 });  
 85   
 86 encodeButton.setText("Encode");  
 87 encodeButton.addActionListener(  
 88 new java.awt.event.ActionListener() {  
 89 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 90 encodePressed(evt);  
 91 }  
 92 });  
 93   
 94 panel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Secret Image"));  
 95 panel1.setToolTipText("Secret Image");  
 96   
 97 jLabel1.setText("Please select your secret image file:\*");  
 98   
 99 browseButton1.setText("Browse");  
100 browseButton1.addActionListener(  
101 new java.awt.event.ActionListener() {  
102 public void actionPerformed(java.awt.event.ActionEvent evt) {  
103 imageBrowsePressed(evt);  
104 }  
105 });  
106   
107 javax.swing.GroupLayout panel1Layout = new javax.swing.GroupLayout(panel1);  
108 panel1.setLayout(panel1Layout);  
109 panel1Layout.setHorizontalGroup(  
110 panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
111 .addGroup(panel1Layout.createSequentialGroup()  
112 .addComponent(jLabel1)  
113 .addGap(0, 0, Short.MAX\_VALUE))  
114 .addGroup(panel1Layout.createSequentialGroup()  
115 .addComponent(secretTextField)  
116 .addGap(18, 18, 18)  
117 .addComponent(browseButton1))  
118 );  
119 panel1Layout.setVerticalGroup(  
120 panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
121 .addGroup(panel1Layout.createSequentialGroup()  
122 .addContainerGap()  
123 .addComponent(jLabel1)  
124 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
125 .addGroup(panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
126 .addComponent(secretTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
127 .addComponent(browseButton1))  
128 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
129 );  
130   
131 jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Innocent Images"));  
132   
133 jLabel2.setText("Please select your two innocent image files:\*");  
134   
135 browseButton2.setText("Browse");  
136 browseButton2.addActionListener(  
137 new java.awt.event.ActionListener() {  
138 public void actionPerformed(java.awt.event.ActionEvent evt) {  
139 imageBrowsePressed(evt);  
140 }  
141 });  
142   
143 browseButton3.setText("Browse");  
144 browseButton3.addActionListener(  
145 new java.awt.event.ActionListener() {  
146 public void actionPerformed(java.awt.event.ActionEvent evt) {  
147 imageBrowsePressed(evt);  
148 }  
149 });  
150   
151 javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  
152 jPanel1.setLayout(jPanel1Layout);  
153 jPanel1Layout.setHorizontalGroup(  
154 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
155 .addGroup(jPanel1Layout.createSequentialGroup()  
156 .addContainerGap()  
157 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
158 .addGroup(jPanel1Layout.createSequentialGroup()  
159 .addComponent(jLabel2)  
160 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
161 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()  
162 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
163 .addComponent(innocentTextField2, javax.swing.GroupLayout.Alignment.LEADING)  
164 .addComponent(innocentTextField1))  
165 .addGap(18, 18, 18)  
166 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
167 .addComponent(browseButton2)  
168 .addComponent(browseButton3)))))  
169 );  
170 jPanel1Layout.setVerticalGroup(  
171 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
172 .addGroup(jPanel1Layout.createSequentialGroup()  
173 .addContainerGap()  
174 .addComponent(jLabel2)  
175 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
176 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
177 .addComponent(innocentTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
178 .addComponent(browseButton2))  
179 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
180 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
181 .addComponent(innocentTextField2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
182 .addComponent(browseButton3))  
183 .addContainerGap(10, Short.MAX\_VALUE))  
184 );  
185   
186 optionalPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Optional"));  
187   
188 jLabel3.setText("Names for your encoded shares (without file extension):");  
189   
190 jLabel4.setText("File 1:");  
191   
192 jLabel5.setText("File 2:");  
193   
194 jLabel6.setText("Directory for Image Shares:");  
195   
196 browseButton4.setText("Browse");  
197 browseButton4.addActionListener(  
198 new java.awt.event.ActionListener() {  
199 public void actionPerformed(java.awt.event.ActionEvent evt) {  
200 dirBrowsePressed(evt);  
201 }  
202 });  
203   
204 javax.swing.GroupLayout optionalPanelLayout = new javax.swing.GroupLayout(optionalPanel);  
205 optionalPanel.setLayout(optionalPanelLayout);  
206 optionalPanelLayout.setHorizontalGroup(  
207 optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
208 .addGroup(optionalPanelLayout.createSequentialGroup()  
209 .addContainerGap()  
210 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
211 .addGroup(optionalPanelLayout.createSequentialGroup()  
212 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
213 .addComponent(jLabel3)  
214 .addComponent(jLabel6))  
215 .addContainerGap())  
216 .addGroup(optionalPanelLayout.createSequentialGroup()  
217 .addGap(6, 6, 6)  
218 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
219 .addGroup(optionalPanelLayout.createSequentialGroup()  
220 .addComponent(jLabel5)  
221 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
222 .addComponent(filename2))  
223 .addGroup(optionalPanelLayout.createSequentialGroup()  
224 .addComponent(jLabel4)  
225 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
226 .addComponent(filename1))  
227 .addGroup(optionalPanelLayout.createSequentialGroup()  
228 .addGap(0, 3, Short.MAX\_VALUE)  
229 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, 480, javax.swing.GroupLayout.PREFERRED\_SIZE)  
230 .addGap(18, 18, 18)  
231 .addComponent(browseButton4))))))  
232 );  
233 optionalPanelLayout.setVerticalGroup(  
234 optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
235 .addGroup(optionalPanelLayout.createSequentialGroup()  
236 .addContainerGap()  
237 .addComponent(jLabel3)  
238 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
239 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
240 .addComponent(jLabel4)  
241 .addComponent(filename1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
242 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
243 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
244 .addComponent(jLabel5)  
245 .addComponent(filename2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
246 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
247 .addComponent(jLabel6)  
248 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
249 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
250 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
251 .addComponent(browseButton4))  
252 .addGap(0, 6, Short.MAX\_VALUE))  
253 );  
254   
255 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
256 getContentPane().setLayout(layout);  
257 layout.setHorizontalGroup(  
258 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
259 .addGroup(layout.createSequentialGroup()  
260 .addContainerGap()  
261 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
262 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
263 .addGap(0, 0, Short.MAX\_VALUE)  
264 .addComponent(encodeButton)  
265 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
266 .addComponent(cancelButton))  
267 .addComponent(panel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
268 .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
269 .addComponent(optionalPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
270 .addContainerGap())  
271 );  
272 layout.setVerticalGroup(  
273 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
274 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
275 .addContainerGap()  
276 .addComponent(panel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
277 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
278 .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
279 .addGap(12, 12, 12)  
280 .addComponent(optionalPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
281 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
282 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
283 .addComponent(cancelButton)  
284 .addComponent(encodeButton))  
285 .addContainerGap())  
286 );  
287   
288 pack();  
289 }// </editor-fold>//GEN-END:initComponents  
290   
291 private void cancelPressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_cancelPressed  
292 // TODO add your handling code here:  
293 this.setVisible(false);  
294 new MainFrame().setVisible(true);  
295 }//GEN-LAST:event\_cancelPressed  
296   
297 private void dirBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_dirBrowsePressed  
298   
299 int returnVal = directoryChooser.showOpenDialog(this);  
300 if(returnVal == JFileChooser.APPROVE\_OPTION)  
301 {  
302 File dir = directoryChooser.getSelectedFile();  
303 if(evt.getSource() == browseButton4)  
304 {  
305 storageDirectoryTextField.setText(dir.getAbsolutePath());  
306 directoryForStorage = dir.getAbsolutePath();  
307 }  
308 }  
309   
310 }//GEN-LAST:event\_dirBrowsePressed  
311   
312 private void imageBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_imageBrowsePressed  
313   
314 int returnVal = imageChooser.showOpenDialog(this);  
315 if(returnVal == JFileChooser.APPROVE\_OPTION)  
316 {  
317 File imageFile = imageChooser.getSelectedFile();  
318 if(evt.getSource() == browseButton1)  
319 {  
320 secretTextField.setText(imageFile.getAbsolutePath());  
321 secretFile = imageFile.getAbsolutePath();  
322 }  
323 else if(evt.getSource() == browseButton2)  
324 {  
325 innocentTextField1.setText(imageFile.getAbsolutePath());  
326 innocentFiles[0] = imageFile.getAbsolutePath();  
327 }  
328 else if(evt.getSource() == browseButton3)  
329 {  
330 innocentTextField2.setText(imageFile.getAbsolutePath());  
331 innocentFiles[1] = imageFile.getAbsolutePath();  
332 }  
333 }  
334 }//GEN-LAST:event\_imageBrowsePressed  
335   
336 private void encodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_encodePressed  
337 //Code to encode secret message  
338 BufferedImage secretImage = null;  
339 boolean fileFound;  
340 try  
341 {  
342 secretImage = ImageIO.read(new File(secretFile));  
343 fileFound = true;  
344 }  
345 catch (IOException e)  
346 {  
347 JOptionPane.showMessageDialog(null, "Error reading your secret file",  
348 "ERROR", JOptionPane.ERROR\_MESSAGE);  
349 fileFound = false;  
350 }  
351   
352 BufferedImage[] innocentShares = new BufferedImage[0];  
353 if(fileFound)  
354 {  
355 innocentShares = new BufferedImage[2];  
356 for(int i = 0; i < 2; i++)  
357 {  
358 try  
359 {  
360 innocentShares[i] = ImageIO.read(new File(innocentFiles[i]));  
361 fileFound = true;  
362 }  
363 catch(IOException e)  
364 {  
365 JOptionPane.showMessageDialog(null,   
366 ("Error reading innocent file " + (i + 1)),  
367 "ERROR", JOptionPane.ERROR\_MESSAGE);  
368 fileFound = false;  
369 }  
370 }  
371 }  
372   
373 if(fileFound)  
374 {  
375 ExtendedVCS myEVCS = new ExtendedVCS(secretImage, innocentShares);  
376 myEVCS.encryptImage();  
377   
378 int[][] encodedRGB = myEVCS.getRGBPixelsForShares();  
379   
380 if(storageDirectoryTextField.getText().equals(""))  
381 {  
382 //Get path to users desktop  
383 //BUG!!! Not working.  
384 directoryForStorage = "C:/Users/allisonholt/Desktop";  
385 //makeDir = false;  
386 }  
387   
388 String[] shareFiles = new String[2];  
389   
390 if(filename1.getText().equals(""))  
391 {  
392 shareFiles[0] = directoryForStorage + "/share1.png";  
393 }  
394 else  
395 {  
396 shareFiles[0] = directoryForStorage + "/" + filename1.getText() +".png";  
397 }  
398   
399 if(filename2.getText().equals(""))  
400 {  
401 shareFiles[1] = directoryForStorage + "/share2.png";  
402 }  
403 else  
404 {  
405 shareFiles[1] = directoryForStorage + "/" + filename2.getText() +".png";  
406 }  
407   
408   
409 try  
410 {  
411 BufferedImage tempShare1 = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_RGB);  
412 tempShare1.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), encodedRGB[0], 0, myEVCS.getImgWidth());  
413 File tempOutput1 = new File(shareFiles[0]);  
414 ImageIO.write(tempShare1, "png", tempOutput1);  
415   
416 BufferedImage tempShare2 = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_RGB);  
417 tempShare2.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), encodedRGB[1], 0, myEVCS.getImgWidth());  
418 File tempOutput2 = new File(shareFiles[1]);  
419 ImageIO.write(tempShare2, "png", tempOutput2);  
420   
421 new MainFrame().setVisible(true);  
422 this.setVisible(false);  
423 JOptionPane.showMessageDialog(null, "Your encrypted shares have been created.",  
424 "SUCCESS", JOptionPane.PLAIN\_MESSAGE);  
425 }  
426 catch (IOException e)  
427 {  
428 JOptionPane.showMessageDialog(null, "Error encrypting your secret message",  
429 "ERROR", JOptionPane.ERROR\_MESSAGE);  
430 }  
431   
432 }  
433 }//GEN-LAST:event\_encodePressed  
434   
435 /\*\*  
436 \* @param args the command line arguments  
437 \*/  
438 public static void main(String args[]) {  
439 /\* Set the Nimbus look and feel \*/  
440 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
441 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
442 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
443 \*/  
444 try {  
445 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
446 if ("Nimbus".equals(info.getName())) {  
447 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
448 break;  
449 }  
450 }  
451 }   
452 catch (ClassNotFoundException ex) {  
453 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
454 }   
455 catch (InstantiationException ex) {  
456 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
457 }   
458 catch (IllegalAccessException ex) {  
459 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
460 }   
461 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
462 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
463 }  
464 //</editor-fold>  
465   
466 /\* Create and display the form \*/  
467 java.awt.EventQueue.invokeLater(  
468 new Runnable() {  
469 public void run() {  
470 new EncodeFrame().setVisible(true);  
471 }  
472 });  
473 }  
474   
475 //Variables for encoding  
476 private String secretFile = "";  
477 private String[] innocentFiles = new String[2];  
478 private String directoryForStorage = "";  
479   
480 // Variables declaration - do not modify//GEN-BEGIN:variables  
481 private javax.swing.JButton browseButton1;  
482 private javax.swing.JButton browseButton2;  
483 private javax.swing.JButton browseButton3;  
484 private javax.swing.JButton browseButton4;  
485 private javax.swing.JButton cancelButton;  
486 private javax.swing.JFileChooser directoryChooser;  
487 private javax.swing.JButton encodeButton;  
488 private javax.swing.JTextField filename1;  
489 private javax.swing.JTextField filename2;  
490 private javax.swing.JFileChooser imageChooser;  
491 private javax.swing.JTextField innocentTextField1;  
492 private javax.swing.JTextField innocentTextField2;  
493 private javax.swing.JLabel jLabel1;  
494 private javax.swing.JLabel jLabel2;  
495 private javax.swing.JLabel jLabel3;  
496 private javax.swing.JLabel jLabel4;  
497 private javax.swing.JLabel jLabel5;  
498 private javax.swing.JLabel jLabel6;  
499 private javax.swing.JPanel jPanel1;  
500 private javax.swing.JPanel optionalPanel;  
501 private javax.swing.JPanel panel1;  
502 private javax.swing.JTextField secretTextField;  
503 private javax.swing.JTextField storageDirectoryTextField;  
504 // End of variables declaration//GEN-END:variables  
505 }  
506

DecodeFrame.java

1 /\*  
 2 \* To change this license header, choose License Headers in Project Properties.  
 3 \* To change this template file, choose Tools | Templates  
 4 \* and open the template in the editor.  
 5 \*/  
 6 package Masters\_Proj;  
 7   
 8 import java.awt.image.BufferedImage;  
 9 import java.io.File;  
 10 import java.io.IOException;  
 11 import javax.imageio.ImageIO;  
 12 import javax.swing.JFileChooser;  
 13 import javax.swing.JOptionPane;  
 14   
 15 /\*\*  
 16 \*  
 17 \* @author allisonholt  
 18 \*/  
 19 public class DecodeFrame extends javax.swing.JFrame {  
 20   
 21 /\*\*  
 22 \* Creates new form DecodeFrame  
 23 \*/  
 24 public DecodeFrame() {  
 25 initComponents();  
 26 this.setLocationRelativeTo(null);  
 27 }  
 28   
 29 /\*\*  
 30 \* This method is called from within the constructor to initialize the form.  
 31 \* WARNING: Do NOT modify this code. The content of this method is always  
 32 \* regenerated by the Form Editor.  
 33 \*/  
 34 @SuppressWarnings("unchecked")  
 35 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 36 private void initComponents() {  
 37   
 38 imageChooser = new javax.swing.JFileChooser();  
 39 directoryChooser = new javax.swing.JFileChooser();  
 40 jPanel1 = new javax.swing.JPanel();  
 41 jLabel1 = new javax.swing.JLabel();  
 42 encodedTextField1 = new javax.swing.JTextField();  
 43 browseButton1 = new javax.swing.JButton();  
 44 encodedTextField2 = new javax.swing.JTextField();  
 45 browseButton2 = new javax.swing.JButton();  
 46 jPanel2 = new javax.swing.JPanel();  
 47 jLabel2 = new javax.swing.JLabel();  
 48 jLabel3 = new javax.swing.JLabel();  
 49 stackedTextField = new javax.swing.JTextField();  
 50 jLabel4 = new javax.swing.JLabel();  
 51 storageDirectoryTextField = new javax.swing.JTextField();  
 52 browseButton3 = new javax.swing.JButton();  
 53 jButton2 = new javax.swing.JButton();  
 54 jButton3 = new javax.swing.JButton();  
 55   
 56 imageChooser.setDialogTitle("Choose an Image");  
 57 imageChooser.setFileFilter(new ImageCustomFilter());  
 58   
 59 directoryChooser.setDialogTitle("Choose a Directory");  
 60 directoryChooser.setFileFilter(new DirectoryCustomFilter());  
 61 directoryChooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES\_ONLY);  
 62   
 63 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 64   
 65 jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Encoded Images"));  
 66   
 67 jLabel1.setText("Please select your two encoded image files:\*");  
 68   
 69 browseButton1.setText("Browse");  
 70 browseButton1.addActionListener(  
 71 new java.awt.event.ActionListener() {  
 72 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 73 imageBrowsePressed(evt);  
 74 }  
 75 });  
 76   
 77 browseButton2.setText("Browse");  
 78 browseButton2.addActionListener(  
 79 new java.awt.event.ActionListener() {  
 80 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 81 imageBrowsePressed(evt);  
 82 }  
 83 });  
 84   
 85 javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  
 86 jPanel1.setLayout(jPanel1Layout);  
 87 jPanel1Layout.setHorizontalGroup(  
 88 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 89 .addGroup(jPanel1Layout.createSequentialGroup()  
 90 .addContainerGap()  
 91 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 92 .addGroup(jPanel1Layout.createSequentialGroup()  
 93 .addComponent(jLabel1)  
 94 .addGap(0, 310, Short.MAX\_VALUE))  
 95 .addGroup(jPanel1Layout.createSequentialGroup()  
 96 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
 97 .addComponent(encodedTextField2)  
 98 .addComponent(encodedTextField1))  
 99 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
100 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
101 .addComponent(browseButton1)  
102 .addComponent(browseButton2))))  
103 .addContainerGap())  
104 );  
105 jPanel1Layout.setVerticalGroup(  
106 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
107 .addGroup(jPanel1Layout.createSequentialGroup()  
108 .addContainerGap()  
109 .addComponent(jLabel1)  
110 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
111 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
112 .addComponent(encodedTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
113 .addComponent(browseButton1))  
114 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
115 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
116 .addComponent(encodedTextField2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
117 .addComponent(browseButton2))  
118 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
119 );  
120   
121 jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("Optional"));  
122   
123 jLabel2.setText("File Name for Decrypted Secret:");  
124   
125 jLabel3.setText("Name (without extension):");  
126   
127 jLabel4.setText("Directory for Decrypted Image:");  
128   
129 browseButton3.setText("Browse");  
130 browseButton3.addActionListener(  
131 new java.awt.event.ActionListener() {  
132 public void actionPerformed(java.awt.event.ActionEvent evt) {  
133 directoryBrowsePressed(evt);  
134 }  
135 });  
136   
137 javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);  
138 jPanel2.setLayout(jPanel2Layout);  
139 jPanel2Layout.setHorizontalGroup(  
140 jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
141 .addGroup(jPanel2Layout.createSequentialGroup()  
142 .addContainerGap()  
143 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
144 .addGroup(jPanel2Layout.createSequentialGroup()  
145 .addComponent(jLabel2)  
146 .addGap(0, 0, Short.MAX\_VALUE))  
147 .addGroup(jPanel2Layout.createSequentialGroup()  
148 .addGap(6, 6, 6)  
149 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
150 .addGroup(jPanel2Layout.createSequentialGroup()  
151 .addComponent(jLabel3)  
152 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
153 .addComponent(stackedTextField))  
154 .addGroup(jPanel2Layout.createSequentialGroup()  
155 .addComponent(jLabel4)  
156 .addGap(0, 0, Short.MAX\_VALUE))  
157 .addGroup(jPanel2Layout.createSequentialGroup()  
158 .addComponent(storageDirectoryTextField)  
159 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
160 .addComponent(browseButton3)))))  
161 .addContainerGap())  
162 );  
163 jPanel2Layout.setVerticalGroup(  
164 jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
165 .addGroup(jPanel2Layout.createSequentialGroup()  
166 .addContainerGap()  
167 .addComponent(jLabel2)  
168 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
169 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
170 .addComponent(jLabel3)  
171 .addComponent(stackedTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
172 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
173 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
174 .addGroup(jPanel2Layout.createSequentialGroup()  
175 .addComponent(jLabel4)  
176 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
177 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
178 .addComponent(browseButton3))  
179 .addContainerGap(19, Short.MAX\_VALUE))  
180 );  
181   
182 jButton2.setText("Cancel");  
183 jButton2.addActionListener(  
184 new java.awt.event.ActionListener() {  
185 public void actionPerformed(java.awt.event.ActionEvent evt) {  
186 cancelPressed(evt);  
187 }  
188 });  
189   
190 jButton3.setText("Decode");  
191 jButton3.addActionListener(  
192 new java.awt.event.ActionListener() {  
193 public void actionPerformed(java.awt.event.ActionEvent evt) {  
194 decodePressed(evt);  
195 }  
196 });  
197   
198 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
199 getContentPane().setLayout(layout);  
200 layout.setHorizontalGroup(  
201 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
202 .addGroup(layout.createSequentialGroup()  
203 .addContainerGap()  
204 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
205 .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
206 .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
207 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
208 .addGap(0, 0, Short.MAX\_VALUE)  
209 .addComponent(jButton3)  
210 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
211 .addComponent(jButton2)))  
212 .addContainerGap())  
213 );  
214 layout.setVerticalGroup(  
215 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
216 .addGroup(layout.createSequentialGroup()  
217 .addContainerGap()  
218 .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
219 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
220 .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
221 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
222 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
223 .addComponent(jButton2)  
224 .addComponent(jButton3))  
225 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
226 );  
227   
228 pack();  
229 }// </editor-fold>//GEN-END:initComponents  
230   
231 private void cancelPressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_cancelPressed  
232 // TODO add your handling code here:  
233 this.setVisible(false);  
234 new MainFrame().setVisible(true);  
235 }//GEN-LAST:event\_cancelPressed  
236   
237 private void imageBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_imageBrowsePressed  
238 // TODO add your handling code here:  
239 int returnVal = imageChooser.showOpenDialog(this);  
240 if(returnVal == JFileChooser.APPROVE\_OPTION)  
241 {  
242 File imageFile = imageChooser.getSelectedFile();  
243 if(evt.getSource() == browseButton1)  
244 {  
245 encodedTextField1.setText(imageFile.getAbsolutePath());  
246 shareFiles[0] = imageFile.getAbsolutePath();  
247 }  
248 else if(evt.getSource() == browseButton2)  
249 {  
250 encodedTextField2.setText(imageFile.getAbsolutePath());  
251 shareFiles[1] = imageFile.getAbsolutePath();  
252 }  
253 }  
254 }//GEN-LAST:event\_imageBrowsePressed  
255   
256 private void directoryBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_directoryBrowsePressed  
257 // TODO add your handling code here:  
258 int returnVal = directoryChooser.showOpenDialog(this);  
259 if(returnVal == JFileChooser.APPROVE\_OPTION)  
260 {  
261 File dir = directoryChooser.getSelectedFile();  
262 if(evt.getSource() == browseButton3)  
263 {  
264 storageDirectoryTextField.setText(dir.getAbsolutePath());  
265 directoryForStorage = dir.getAbsolutePath();  
266 }  
267 }  
268 }//GEN-LAST:event\_directoryBrowsePressed  
269   
270 private void decodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_decodePressed  
271 // TODO add your handling code here:  
272 BufferedImage[] sharesEVCS = new BufferedImage[2];  
273 boolean fileFound = false;  
274   
275 for(int i = 0; i < 2; i++)  
276 {  
277 try  
278 {  
279 sharesEVCS[i] = ImageIO.read(new File(shareFiles[i]));  
280 fileFound = true;  
281 }  
282 catch(IOException e)  
283 {  
284 JOptionPane.showMessageDialog(null,   
285 ("Error reading file share" + (i + 1)),  
286 "ERROR", JOptionPane.ERROR\_MESSAGE);  
287 fileFound = false;  
288 }  
289 }  
290   
291 if(fileFound)  
292 {  
293 ExtendedVCS myEVCS = new ExtendedVCS(sharesEVCS);  
294 myEVCS.decryptImage();  
295   
296 if(storageDirectoryTextField.getText().equals(""))  
297 {  
298 //Get path to users desktop  
299 //BUG!!! Not working.  
300 directoryForStorage = "C:/Users/allisonholt/Desktop";  
301 //makeDir = false;  
302 }  
303   
304 String decodedFileName;  
305 if(stackedTextField.getText().equals(""))  
306 {  
307 //Get path to users desktop  
308 //BUG!!! Not working.  
309 decodedFileName = directoryForStorage + "/secretMsg.png";  
310 //makeDir = false;  
311 }  
312 else  
313 {  
314 decodedFileName = directoryForStorage + "/" + stackedTextField.getText() + ".png";  
315 }  
316   
317 try  
318 {  
319 BufferedImage decryptImage = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_ARGB);  
320 decryptImage.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), myEVCS.getDecryptImgPixels(), 0, myEVCS.getImgWidth());  
321   
322 File tempOutput = new File(decodedFileName);  
323 ImageIO.write(decryptImage, "png", tempOutput);  
324   
325 new MainFrame().setVisible(true);  
326 this.setVisible(false);  
327 JOptionPane.showMessageDialog(null, "Your decrypted image has been created.",  
328 "SUCCESS", JOptionPane.PLAIN\_MESSAGE);  
329 }  
330 catch(IOException e)  
331 {  
332 JOptionPane.showMessageDialog(null, "Error decrypting your secret message",  
333 "ERROR", JOptionPane.ERROR\_MESSAGE);  
334 }  
335   
336 }  
337 }//GEN-LAST:event\_decodePressed  
338   
339 /\*\*  
340 \* @param args the command line arguments  
341 \*/  
342 public static void main(String args[]) {  
343 /\* Set the Nimbus look and feel \*/  
344 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
345 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
346 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
347 \*/  
348 try {  
349 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
350 if ("Nimbus".equals(info.getName())) {  
351 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
352 break;  
353 }  
354 }  
355 }   
356 catch (ClassNotFoundException ex) {  
357 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
358 }   
359 catch (InstantiationException ex) {  
360 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
361 }   
362 catch (IllegalAccessException ex) {  
363 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
364 }   
365 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
366 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
367 }  
368 //</editor-fold>  
369   
370 /\* Create and display the form \*/  
371 java.awt.EventQueue.invokeLater(  
372 new Runnable() {  
373 public void run() {  
374 new DecodeFrame().setVisible(true);  
375 }  
376 });  
377 }  
378 //Variables for decoding  
379 private String[] shareFiles = new String[2];  
380 private String directoryForStorage = "";  
381   
382 // Variables declaration - do not modify//GEN-BEGIN:variables  
383 private javax.swing.JButton browseButton1;  
384 private javax.swing.JButton browseButton2;  
385 private javax.swing.JButton browseButton3;  
386 private javax.swing.JFileChooser directoryChooser;  
387 private javax.swing.JTextField encodedTextField1;  
388 private javax.swing.JTextField encodedTextField2;  
389 private javax.swing.JFileChooser imageChooser;  
390 private javax.swing.JButton jButton2;  
391 private javax.swing.JButton jButton3;  
392 private javax.swing.JLabel jLabel1;  
393 private javax.swing.JLabel jLabel2;  
394 private javax.swing.JLabel jLabel3;  
395 private javax.swing.JLabel jLabel4;  
396 private javax.swing.JPanel jPanel1;  
397 private javax.swing.JPanel jPanel2;  
398 private javax.swing.JTextField stackedTextField;  
399 private javax.swing.JTextField storageDirectoryTextField;  
400 // End of variables declaration//GEN-END:variables  
401 }  
402

ImageCustomFilter.java

1 /\*  
 2 \* To change this license header, choose License Headers in Project Properties.  
 3 \* To change this template file, choose Tools | Templates  
 4 \* and open the template in the editor.  
 5 \*/  
 6 package Masters\_Proj;  
 7   
 8 import java.io.File;  
 9   
10 /\*\*  
11 \*  
12 \* @author allisonholt  
13 \*/  
14 public class ImageCustomFilter extends javax.swing.filechooser.FileFilter {  
15   
16 @Override  
17 public boolean accept(File file)  
18 {  
19 //allow only image file  
20 return file.isDirectory() || file.getAbsolutePath().endsWith(".png")  
21 || file.getAbsolutePath().endsWith(".jpeg")  
22 || file.getAbsolutePath().endsWith(".jpg");  
23 }  
24   
25 @Override  
26 public String getDescription()  
27 {  
28 return "Image files (\*.png, \*.jpeg, \*.jpg)";  
29 }  
30   
31 }  
32

DirectoryCustomFilter.java

1 /\*  
 2 \* To change this license header, choose License Headers in Project Properties.  
 3 \* To change this template file, choose Tools | Templates  
 4 \* and open the template in the editor.  
 5 \*/  
 6 package Masters\_Proj;  
 7   
 8 import java.io.File;  
 9   
10 /\*\*  
11 \*  
12 \* @author allisonholt  
13 \*/  
14 public class DirectoryCustomFilter extends javax.swing.filechooser.FileFilter{  
15   
16 @Override  
17 public boolean accept(File file)  
18 {  
19 //allow only image file  
20 return file.isDirectory();  
21 }  
22   
23 @Override  
24 public String getDescription()  
25 {  
26 return "File Directory";  
27 }  
28 }  
29

ExtendedVCS.java

1 /\*  
 2 \* To change this license header, choose License Headers in Project Properties.  
 3 \* To change this template file, choose Tools | Templates  
 4 \* and open the template in the editor.  
 5 \*/  
 6 package Masters\_Proj;  
 7 import java.awt.Color;  
 8 import java.awt.image.BufferedImage;  
 9 import java.util.Arrays;  
 10 import java.util.Random;  
 11   
 12 /\*\*  
 13 \*  
 14 \* @author allisonholt  
 15 \*/  
 16 public class ExtendedVCS   
 17 {  
 18 private int k;  
 19 private int n;  
 20 private int imgWidth;  
 21 private int imgHeight;  
 22 private int numColumns;  
 23 private BufferedImage secretMsg;  
 24 private BufferedImage[] innocentShares;  
 25 //private int[2][] shareOrigRGBPixels;  
 26 private int[][] encryptedShareRGB;  
 27 private int[][] secretSharesRGB;  
 28   
 29 private int numSharesToDecrypt;  
 30 private BufferedImage[] sharesToDecrypt;  
 31 private int[] secretMsgPixels;  
 32   
 33   
 34 //For encryption purposes  
 35 ExtendedVCS(BufferedImage secretMsgIn, BufferedImage[] innocentSharesIn)  
 36 {  
 37 k = 2;  
 38 n = 2;  
 39 secretMsg = secretMsgIn;  
 40 imgWidth = secretMsg.getWidth();  
 41 imgHeight = secretMsg.getHeight();  
 42 innocentShares = innocentSharesIn;  
 43 encryptedShareRGB = new int[2][imgWidth \* imgHeight];  
 44 }  
 45   
 46 //For decryption purposes  
 47 ExtendedVCS(BufferedImage[] shareImgs)  
 48 {  
 49 numSharesToDecrypt = 2;  
 50 sharesToDecrypt = shareImgs;  
 51 imgWidth = shareImgs[0].getWidth();  
 52 imgHeight = shareImgs[0].getHeight();  
 53 }  
 54   
 55 int getImgWidth()  
 56 {  
 57 return imgWidth;  
 58 }  
 59   
 60 int getImgHeight()  
 61 {  
 62 return imgHeight;  
 63 }  
 64   
 65 int[][] getRGBPixelsForShares()  
 66 {  
 67 return encryptedShareRGB;  
 68 }  
 69   
 70 int[] getDecryptImgPixels()  
 71 {  
 72 return secretMsgPixels;  
 73 }  
 74   
 75 void encryptImage()  
 76 {  
 77 int[] secretRGB = new int[imgWidth \* imgHeight];  
 78 //A cover image is the same as an innocent image  
 79 int[][] coverRGB = new int[2][imgWidth \* imgHeight];  
 80   
 81 //Process the gathered innocent images and the secret image  
 82 secretMsg.getRGB(0, 0, imgWidth, imgHeight, secretRGB, 0, imgWidth);  
 83 innocentShares[0].getRGB(0, 0, imgWidth, imgHeight, coverRGB[0], 0, imgWidth);  
 84 innocentShares[1].getRGB(0, 0, imgWidth, imgHeight, coverRGB[1], 0, imgWidth);  
 85   
 86 //Half-tone Innocent Images  
 87 errorDiffusion(coverRGB[0]);  
 88 errorDiffusion(coverRGB[1]);  
 89   
 90 //Split secret image into three images  
 91 int[] secretRed = new int[secretRGB.length];  
 92 int[] secretGreen = new int[secretRGB.length];  
 93 int[] secretBlue = new int[secretRGB.length];  
 94 splitSecretRGB(secretRGB, secretRed, secretGreen, secretBlue);  
 95   
 96 //VIP synchronization  
 97 vipSynchronization(secretRed, secretGreen, secretBlue, coverRGB);  
 98   
 99 //Perform error diffusion on cover images with secret encoded  
100 errorDiffusion(encryptedShareRGB[0]);  
101 errorDiffusion(encryptedShareRGB[1]);  
102   
103 }  
104   
105 void errorDiffusion(int[] image)  
106 {  
107 int x[][] = new int[imgHeight][imgWidth];  
108 int u[][] = new int [imgHeight][imgWidth];  
109   
110 int i = 0;  
111 for(int n = 0; n < imgHeight; n++)  
112 {  
113 for(int m = 0; m < imgWidth; m++)  
114 {  
115 x[n][m] = image[i];  
116 i += 1;  
117 }  
118 }  
119   
120 for(int n = 0; n < imgHeight; n++)  
121 {  
122 for(int m = 0; m < imgWidth; m++)  
123 {  
124 u[n][m] += x[n][m];  
125   
126 int xRed = (x[n][m] & 0x00ff0000) >> 16;  
127 int xGreen = (x[n][m] & 0x0000ff00) >> 8;  
128 int xBlue = (x[n][m] & 0x000000ff);  
129   
130 int uRed = (u[n][m] & 0x00ff0000) >> 16;  
131 int uGreen = (u[n][m] & 0x0000ff00) >> 8;  
132 int uBlue = (u[n][m] & 0x000000ff);  
133   
134 int quantErrorRed = uRed - xRed;  
135 int quantErrorGreen = uGreen - xGreen;  
136 int quantErrorBlue = uBlue - xBlue;  
137   
138 if(xRed > 127)  
139 {  
140 if((m + 1) < imgWidth)  
141 {  
142 int temp = quantErrorRed \* 7 / 16;  
143 temp = temp << 16;  
144 u[n][m + 1] += temp;  
145 }  
146 if((m - 1) >= 0 && (n + 1) < imgHeight)  
147 {  
148 int temp = quantErrorRed \* 3 / 16;  
149 temp = temp << 16;  
150 u[n + 1][m - 1] += temp;  
151 }  
152 if((n + 1) < imgHeight)  
153 {  
154 int temp = quantErrorRed \* 5 / 16;  
155 temp = temp << 16;  
156 u[n + 1][m] += temp;  
157 }  
158 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
159 {  
160 int temp = quantErrorRed \* 1 / 16;  
161 temp = temp << 16;  
162 u[n + 1][m + 1] += temp;  
163 }  
164 }  
165 if(xGreen > 127)  
166 {  
167 if((m + 1) < imgWidth)  
168 {  
169 int temp = quantErrorGreen \* 7 / 16;  
170 temp = temp << 8;  
171 u[n][m + 1] += temp;  
172 }  
173 if((m - 1) >= 0 && (n + 1) < imgHeight)  
174 {  
175 int temp = quantErrorGreen \* 3 / 16;  
176 temp = temp << 8;  
177 u[n + 1][m - 1] += temp;  
178 }  
179 if((n + 1) < imgHeight)  
180 {  
181 int temp = quantErrorGreen \* 5 / 16;  
182 temp = temp << 8;  
183 u[n + 1][m] += temp;  
184 }  
185 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
186 {  
187 int temp = quantErrorGreen \* 1 / 16;  
188 temp = temp << 8;  
189 u[n + 1][m + 1] += temp;  
190 }  
191 }  
192 if(xBlue > 127)  
193 {  
194 if((m + 1) < imgWidth)  
195 {  
196 int temp = quantErrorBlue \* 7 / 16;  
197 u[n][m + 1] += temp;  
198 }  
199 if((m - 1) >= 0 && (n + 1) < imgHeight)  
200 {  
201 int temp = quantErrorBlue \* 3 / 16;  
202 u[n + 1][m - 1] += temp;  
203 }  
204 if((n + 1) < imgHeight)  
205 {  
206 int temp = quantErrorBlue \* 5 / 16;  
207 u[n + 1][m] += temp;  
208 }  
209 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
210 {  
211 int temp = quantErrorBlue \* 1 / 16;  
212 u[n + 1][m + 1] += temp;  
213 }  
214 }  
215 }  
216 }  
217   
218 int j = 0;  
219 for(int n = 0; n < imgHeight; n++)  
220 {  
221 for(int m = 0; m < imgWidth; m++)  
222 {  
223 image[j] = u[n][m];  
224 j += 1;  
225 }  
226 }  
227 }  
228   
229 void splitSecretRGB(int[] secret, int[] red, int[] green, int[] blue)  
230 {  
231 for(int i = 0; i < secret.length; i++)  
232 {  
233 int redVal = (secret[i] & 0x00ff0000) >> 16;  
234 int greenVal = (secret[i] & 0x0000ff00) >> 8;  
235 int blueVal = (secret[i] & 0x000000ff);  
236   
237 Pixel redPix = new Pixel(redVal, 0, 0);  
238 Pixel greenPix = new Pixel(0, greenVal, 0);  
239 Pixel bluePix = new Pixel (0, 0, blueVal);  
240   
241 int redCon = redPix.getConcentration('r');  
242 int greenCon = greenPix.getConcentration('g');  
243 int blueCon = bluePix.getConcentration('b');  
244   
245 Color redColor = new Color(redCon, 0, 0);  
246 Color greenColor = new Color(0, greenCon, 0);  
247 Color blueColor = new Color(0, 0, blueCon);  
248   
249 red[i] = redColor.getRGB();  
250 green[i] = greenColor.getRGB();  
251 blue[i] = blueColor.getRGB();  
252 }  
253 }  
254   
255 void vipSynchronization(int[] red, int[] green, int[] blue, int[][] cover)  
256 {  
257 for(int i = 0; i < cover[0].length; i++)  
258 {  
259 int c1Red = (cover[0][i] & 0x00ff0000) >> 16;  
260 int c2Red = (cover[1][i] & 0x00ff0000) >> 16;  
261 int secretRed = (red[i] & 0x00ff0000) >> 16;  
262 String c1RedBinary = String.format("%8s", Integer.toBinaryString(c1Red)).replace(" ", "0");  
263 String c2RedBinary = String.format("%8s", Integer.toBinaryString(c2Red)).replace(" ", "0");  
264 String secretRedBinary = String.format("%8s",Integer.toBinaryString(secretRed)).replace(" ", "0");  
265   
266 for(int j = 0; j < secretRedBinary.length(); j++)  
267 {  
268 if(secretRedBinary.charAt(j) == '1'  
269 && c1RedBinary.charAt(j) == c2RedBinary.charAt(j))  
270 {  
271 Random rand = new Random();  
272 int temp = rand.nextInt(20) % 2;  
273 /\*  
274 if temp == 0 then c1 stays the same and c2 is flipped  
275 if temp == 1 then c2 stays the same and c1 is flipped  
276 \*/  
277 if(temp == 0 && c2RedBinary.charAt(j) == '1')  
278 {  
279 char[] c2Array = c2RedBinary.toCharArray();  
280 c2Array[j] = '0';  
281 c2RedBinary = new String(c2Array);  
282 }  
283 else if(temp == 0 && c2RedBinary.charAt(j) == '0')  
284 {  
285 char[] c2Array = c2RedBinary.toCharArray();  
286 c2Array[j] = '1';  
287 c2RedBinary = new String(c2Array);  
288 }  
289 else if(temp == 1 && c1RedBinary.charAt(j) == '1')  
290 {  
291 char[] c1Array = c1RedBinary.toCharArray();  
292 c1Array[j] = '0';  
293 c1RedBinary = new String(c1Array);  
294 }  
295 else  
296 {  
297 char[] c1Array = c1RedBinary.toCharArray();  
298 c1Array[j] = '1';  
299 c1RedBinary = new String(c1Array);  
300 }  
301 }  
302 else  
303 {  
304 Random rand = new Random();  
305 int temp = rand.nextInt(20) % 2;  
306 /\*  
307 if temp == 0 then c2 bit is set to c1 bit  
308 if temp == 1 then c1 bit is set to c2 bit  
309 \*/  
310 if(temp == 0)  
311 {  
312 char[] c2Array = c2RedBinary.toCharArray();  
313 c2Array[j] = c1RedBinary.charAt(j);  
314 c2RedBinary = new String(c2Array);  
315 }  
316 else  
317 {  
318 char[] c1Array = c1RedBinary.toCharArray();  
319 c1Array[j] = c2RedBinary.charAt(j);  
320 c1RedBinary = new String(c1Array);  
321 }  
322 }  
323 }  
324   
325 encryptedShareRGB[0][i] = (Integer.parseInt(c1RedBinary, 2)) << 16;  
326 encryptedShareRGB[1][i] = (Integer.parseInt(c2RedBinary, 2)) << 16;  
327   
328 int c1Green = (cover[0][i] & 0x0000ff00) >> 8;  
329 int c2Green = (cover[1][i] & 0x0000ff00) >> 8;  
330 int secretGreen = (green[i] & 0x0000ff00) >> 8;  
331 String c1GreenBinary = String.format("%8s", Integer.toBinaryString(c1Green)).replace(" ", "0");  
332 String c2GreenBinary = String.format("%8s", Integer.toBinaryString(c2Green)).replace(" ", "0");  
333 String secretGreenBinary = String.format("%8s", Integer.toBinaryString(secretGreen)).replace(" ", "0");  
334   
335 for(int j = 0; j < secretGreenBinary.length(); j++)  
336 {  
337 if(secretGreenBinary.charAt(j) == '1'  
338 && c1GreenBinary.charAt(j) == c2GreenBinary.charAt(j))  
339 {  
340 Random rand = new Random();  
341 int temp = rand.nextInt(20) % 2;  
342 /\*  
343 if temp == 0 then c1 stays the same and c2 is flipped  
344 if temp == 1 then c2 stays the same and c1 is flipped  
345 \*/  
346 if(temp == 0 && c2GreenBinary.charAt(j) == '1')  
347 {  
348 char[] c2Array = c2GreenBinary.toCharArray();  
349 c2Array[j] = '0';  
350 c2GreenBinary = new String(c2Array);  
351 }  
352 else if(temp == 0 && c2GreenBinary.charAt(j) == '0')  
353 {  
354 char[] c2Array = c2GreenBinary.toCharArray();  
355 c2Array[j] = '1';  
356 c2GreenBinary = new String(c2Array);  
357 }  
358 else if(temp == 1 && c1GreenBinary.charAt(j) == '1')  
359 {  
360 char[] c1Array = c1GreenBinary.toCharArray();  
361 c1Array[j] = '0';  
362 c1GreenBinary = new String(c1Array);  
363 }  
364 else  
365 {  
366 char[] c1Array = c1GreenBinary.toCharArray();  
367 c1Array[j] = '1';  
368 c1GreenBinary = new String(c1Array);  
369 }  
370 }  
371 else  
372 {  
373 Random rand = new Random();  
374 int temp = rand.nextInt(20) % 2;  
375 /\*  
376 if temp == 0 then c2 bit is set to c1 bit  
377 if temp == 1 then c1 bit is set to c2 bit  
378 \*/  
379 if(temp == 0)  
380 {  
381 char[] c2Array = c2GreenBinary.toCharArray();  
382 c2Array[j] = c1GreenBinary.charAt(j);  
383 c2GreenBinary = new String(c2Array);  
384 }  
385 else  
386 {  
387 char[] c1Array = c1GreenBinary.toCharArray();  
388 c1Array[j] = c2GreenBinary.charAt(j);  
389 c1GreenBinary = new String(c1Array);  
390 }  
391 }  
392 }  
393   
394 encryptedShareRGB[0][i] += (Integer.parseInt(c1GreenBinary, 2)) << 8;  
395 encryptedShareRGB[1][i] += (Integer.parseInt(c2GreenBinary, 2)) << 8;  
396   
397 int c1Blue = (cover[0][i] & 0x000000ff);  
398 int c2Blue = (cover[1][i] & 0x000000ff);  
399 int secretBlue = (blue[i] & 0x000000ff);  
400 String c1BlueBinary = String.format("%8s", Integer.toBinaryString(c1Blue)).replace(" ", "0");  
401 String c2BlueBinary = String.format("%8s", Integer.toBinaryString(c2Blue)).replace(" ", "0");  
402 String secretBlueBinary = String.format("%8s", Integer.toBinaryString(secretBlue)).replace(" ", "0");  
403   
404 for(int j = 0; j < secretBlueBinary.length(); j++)  
405 {  
406 if(secretBlueBinary.charAt(j) == '1'  
407 && c1BlueBinary.charAt(j) == c2BlueBinary.charAt(j))  
408 {  
409 Random rand = new Random();  
410 int temp = rand.nextInt(20) % 2;  
411 /\*  
412 if temp == 0 then c1 stays the same and c2 is flipped  
413 if temp == 1 then c2 stays the same and c1 is flipped  
414 \*/  
415 if(temp == 0 && c2BlueBinary.charAt(j) == '1')  
416 {  
417 char[] c2Array = c2BlueBinary.toCharArray();  
418 c2Array[j] = '0';  
419 c2BlueBinary = new String(c2Array);  
420 }  
421 else if(temp == 0 && c2BlueBinary.charAt(j) == '0')  
422 {  
423 char[] c2Array = c2BlueBinary.toCharArray();  
424 c2Array[j] = '1';  
425 c2BlueBinary = new String(c2Array);  
426 }  
427 else if(temp == 1 && c1BlueBinary.charAt(j) == '1')  
428 {  
429 char[] c1Array = c1BlueBinary.toCharArray();  
430 c1Array[j] = '0';  
431 c1BlueBinary = new String(c1Array);  
432 }  
433 else  
434 {  
435 char[] c1Array = c1BlueBinary.toCharArray();  
436 c1Array[j] = '1';  
437 c1BlueBinary = new String(c1Array);  
438 }  
439 }  
440 else  
441 {  
442 Random rand = new Random();  
443 int temp = rand.nextInt(20) % 2;  
444 /\*  
445 if temp == 0 then c2 bit is set to c1 bit  
446 if temp == 1 then c1 bit is set to c2 bit  
447 \*/  
448 if(temp == 0)  
449 {  
450 char[] c2Array = c2BlueBinary.toCharArray();  
451 c2Array[j] = c1BlueBinary.charAt(j);  
452 c2BlueBinary = new String(c2Array);  
453 }  
454 else  
455 {  
456 char[] c1Array = c1BlueBinary.toCharArray();  
457 c1Array[j] = c2BlueBinary.charAt(j);  
458 c1BlueBinary = new String(c1Array);  
459 }  
460 }  
461 }  
462   
463 encryptedShareRGB[0][i] += (Integer.parseInt(c1BlueBinary, 2));  
464 encryptedShareRGB[1][i] += (Integer.parseInt(c2BlueBinary, 2));  
465 }  
466 }  
467   
468 /\*\*  
469 \*   
470 \* @param secretImgRGB The RGB values of the secret image  
471 \* @param shareOriginalRGB The RGB values of the innocent images  
472 \*/  
473 void createPixelsOfShares(int[] secretImgRGB, int[][] shareOriginalRGB)  
474 {  
475 //Used to store the embedded RGB values  
476 encryptedShareRGB = new int[2][imgWidth \* imgHeight];  
477   
478 //Used to bring the secret image up using a size invarint-ish technique  
479 secretSharesRGB = new int[2][imgWidth \* imgHeight];  
480   
481 for(int i = 0; i < secretImgRGB.length; i++)  
482 {  
483 int redVal = (secretImgRGB[i] & 0x00ff0000) >> 16;  
484 int greenVal = (secretImgRGB[i] & 0x0000ff00) >> 8;  
485 int blueVal = (secretImgRGB[i] & 0x000000ff);  
486 Pixel orig = new Pixel(redVal, greenVal, blueVal);  
487   
488 redVal = (shareOriginalRGB[0][i] & 0x00ff0000) >> 16;  
489 greenVal = (shareOriginalRGB[0][i] & 0x0000ff00) >> 8;  
490 blueVal = (shareOriginalRGB[0][i] & 0x000000ff);  
491 Pixel innocent0 = new Pixel(redVal, greenVal, blueVal);  
492   
493 redVal = (shareOriginalRGB[1][i] & 0x00ff0000) >> 16;  
494 greenVal = (shareOriginalRGB[1][i] & 0x0000ff00) >> 8;  
495 blueVal = (shareOriginalRGB[1][i] & 0x000000ff);  
496 Pixel innocent1 = new Pixel(redVal, greenVal, blueVal);  
497   
498 Random randomGen = new Random();  
499 int maxGrayCon = orig.getConcentration('r');  
500 int grayCon1 = randomGen.nextInt(maxGrayCon + 1);  
501 int grayCon2 = maxGrayCon - grayCon1;  
502   
503 Color secretGray1 = new Color(grayCon1, grayCon1, grayCon1);  
504 secretSharesRGB[0][i] = secretGray1.getRGB();  
505   
506 Color secretGray2 = new Color(grayCon2, grayCon2, grayCon2);  
507 secretSharesRGB[1][i] = secretGray2.getRGB();  
508   
509 int innocent1Con = innocent0.getConcentration('r');  
510 int embedded1Con = (innocent1Con + grayCon1) / 2;  
511 if(embedded1Con < 0)  
512 embedded1Con = 0;  
513 Color embedded1 = new Color(embedded1Con, embedded1Con, embedded1Con);  
514 encryptedShareRGB[0][i] = embedded1.getRGB();  
515   
516 int innocent2Con = innocent1.getConcentration('r');  
517 int embedded2Con = (innocent2Con + grayCon2) / 2;  
518 if(embedded2Con < 0)  
519 embedded2Con = 0;  
520 Color embedded2 = new Color(embedded2Con, embedded2Con, embedded2Con);  
521 encryptedShareRGB[1][i] = embedded2.getRGB();  
522 }  
523 }  
524   
525 void decryptImage()  
526 {  
527 //Make a 2d array of pixel arrays  
528 int[][] embeddedPixels = new int[numSharesToDecrypt][imgWidth \* imgHeight];  
529 secretMsgPixels = new int[imgWidth \* imgHeight];  
530   
531 //getRGB pixels of BufferedImages  
532 for(int i = 0; i < numSharesToDecrypt; i++)  
533 {  
534 sharesToDecrypt[i].getRGB(0, 0, imgWidth, imgHeight, embeddedPixels[i], 0, imgWidth);  
535 }  
536   
537 int numOfPixels = embeddedPixels[0].length;  
538 for(int i = 0; i < numOfPixels; i++)  
539 {  
540   
541 int redVal1 = (embeddedPixels[0][i] & 0x00ff0000) >> 16;  
542 int greenVal1 = (embeddedPixels[0][i] & 0x0000ff00) >> 8;  
543 int blueVal1 = (embeddedPixels[0][i] & 0x000000ff);  
544   
545 int redVal2 = (embeddedPixels[1][i] & 0x00ff0000) >> 16;  
546 int greenVal2 = (embeddedPixels[1][i] & 0x0000ff00) >> 8;  
547 int blueVal2 = (embeddedPixels[1][i] & 0x000000ff);  
548   
549 //Need to XOR the color concentrations  
550 //XORing mimics stacking transparencies  
551 int redVal = (int)(redVal1 ^ redVal2);  
552 int greenVal = (int)(greenVal1 ^ greenVal2);  
553 int blueVal = (int)(blueVal1 ^ blueVal2);  
554   
555 Color decryptedColor = new Color(redVal, greenVal, blueVal);  
556 secretMsgPixels[i] = decryptedColor.getRGB();  
557 }  
558 }  
559   
560 }  
561

Pixel.java

1 /\*  
 2 \* To change this license header, choose License Headers in Project Properties.  
 3 \* To change this template file, choose Tools | Templates  
 4 \* and open the template in the editor.  
 5 \*/  
 6 package Masters\_Proj;  
 7 import java.awt.Color;  
 8   
 9 /\*\*  
10 \*  
11 \* @author allisonholt  
12 \*/  
13 public class Pixel   
14 {  
15   
16 private int redVal;  
17 private int greenVal;  
18 private int blueVal;  
19   
20 public Pixel(int redIn, int greenIn, int blueIn)  
21 {  
22 redVal = redIn;  
23 greenVal = greenIn;  
24 blueVal = blueIn;  
25 }  
26   
27 //Used to determine if pixel is closer to white than black  
28 public int getConcentration(char color)  
29 {  
30 if(color == 'r')  
31 return redVal;  
32 else if(color == 'g')  
33 return greenVal;  
34 else  
35 return blueVal;  
36 }  
37   
38 }  
39