

```

1 import java.awt.Desktop;
2 import java.awt.Toolkit;
3 import java.awt.datatransfer.*;
4 import java.util.*;
5 import java.io.*;
6 import javax.imageio.ImageIO;
7 import java.awt.image.BufferedImage;
8 import javax.swing.JOptionPane;
9
10 /*****
11 Tasks are the methods called for by Display
12 through the GUI.
13
14 @author Susanna Bradbury, James Houghton, Pranav Ramanan
15 @version 06/05/2014
16 *****/
17 public class Tasks
18 {
19     /*****
20     Location of where the most recent saved file was saved.
21     *****/
22     public static String saveLocation=null;
23
24     /* Pranav's methods */
25     /*****
26     Inverts the loaded image's colors.
27     *****/
28     public static void task1()
29     {
30         System.out.println(Display.getText());
31         BufferedImage image=Display.getImage();
32
33         if(Display.getText()==null)
34         {
35             String message = Steg.decrypt(image);
36             Display.setEncodedText(message);
37         }
38         image = Effects.Inverse(image,Display.getHeight(),Display.getWidth());
39         Display.unloadImage();
40         Display.loadBI(image, 1);
41         image = Steg.encrypt(image);
42         Display.loadBI(image, 2);
43     }
44     /*****
45     Fades the colors of the image.
46     *****/
47     public static void task2()
48     {
49         BufferedImage image=Display.getImage();
50
51         if(Display.getText()==null)
52         {
53             String message = Steg.decrypt(image);
54             Display.setEncodedText(message);
55         }
56         String color = JOptionPane.showInputDialog("R | G | B | ALL");
57         if ((color != null) && (color.length() > 0))
58         {
59             try

```

```

60         {
61             String tintString = JOptionPane.showInputDialog("Decrease color value by: (
0-225)");
62             if ((tintString!=null) && (tintString.length(>0))
63                 {
64                 int tint = Integer.parseInt(tintString);
65                 image = Effects.Fade(image,Display.getHeight(),Display.getWidth(),tint,c
olor);
66                 Display.unloadImage();
67                 Display.loadBI(image, 1);
68                 image = Steg.encrypt(image);
69                 Display.loadBI(image, 2);
70             }
71         }
72         catch(Exception e){}
73     }
74 }
75 /*****
76 Tints the image's colors.
77 *****/
78 public static void task3()
79 {
80     BufferedImage image=Display.getImage();
81
82     if(Display.getText()==null)
83     {
84         String message = Steg.decrypt(image);
85         Display.setEncodedText(message);
86     }
87
88     String color = JOptionPane.showInputDialog("R | G | B | ALL");
89     if ((color != null) && (color.length() > 0))
90     {
91         try
92         {
93             String tintString = JOptionPane.showInputDialog("Increase color value by: (
0-225)");
94             if ((tintString!=null) && (tintString.length(>0))
95                 {
96                 int tint = Integer.parseInt(tintString);
97                 image = Effects.Tint(image,Display.getHeight(),Display.getWidth(),tint,c
olor);
98                 Display.unloadImage();
99                 Display.loadBI(image, 1);
100                 image = Steg.encrypt(image);
101                 Display.loadBI(image, 2);
102             }
103         }
104         catch(Exception e){}
105     }
106 }
107 /*****
108 Makes the image a mix of only black and white color.
109 *****/
110 public static void task4()
111 {
112     BufferedImage image=Display.getImage();
113
114     if(Display.getText()==null)

```

```

115     {
116         String message = Steg.decrypt(image);
117         Display.setEncodedText(message);
118     }
119
120     image = Effects.BlackWhite(image,Display.getHeight(),Display.getWidth());
121     Display.unloadImage();
122     Display.loadBI(image, 1);
123     image = Steg.encrypt(image);
124     Display.loadBI(image, 2);
125 }
126 /*****
127 Removes a specified color in the loaded image.
128 *****/
129 public static void task5()
130 {
131     BufferedImage image=Display.getImage();
132
133     if(Display.getText()==null)
134     {
135         String message = Steg.decrypt(image);
136         Display.setEncodedText(message);
137     }
138
139     String color = JOptionPane.showInputDialog("R|G|B");
140     if ((color != null) && (color.length() > 0))
141     {
142         image = Effects.Remove(image,Display.getHeight(),Display.getWidth(),color);
143         Display.unloadImage();
144         Display.loadBI(image, 1);
145         image = Steg.encrypt(image);
146         Display.loadBI(image, 2);
147     }
148 }
149 /*****
150 Grayscales the image.
151 *****/
152 public static void task6()
153 {
154     BufferedImage image=Display.getImage();
155
156     if(Display.getText()==null)
157     {
158         String message = Steg.decrypt(image);
159         Display.setEncodedText(message);
160     }
161
162     image = Effects.Grayscale(image,Display.getHeight(),Display.getWidth());
163     Display.unloadImage();
164     Display.loadBI(image, 1);
165     image = Steg.encrypt(image);
166     Display.loadBI(image, 2);
167 }
168 /*****
169 'Colorizes' the image. Dramatically changes the colors of the image.
170 *****/
171 public static void task7()
172 {
173     BufferedImage image=Display.getImage();

```

```

174
175     if(Display.getText()==null)
176     {
177         String message = Steg.decrypt(image);
178         Display.setEncodedText(message);
179     }
180     image = Effects.Colorize(image,Display.getHeight(),Display.getWidth());
181     Display.unloadImage();
182     Display.loadBI(image, 1);
183     image = Steg.encrypt(image);
184     Display.loadBI(image, 2);
185 }
186
187
188 /* Susanna's methods */
189 /*****
190 Encodes the loaded image.
191 *****/
192 public static void encode()
193 {
194     Display.setEncodedText(Display.getText(1));
195     BufferedImage eImage = Steg.encrypt(Display.getImage());
196     Display.unloadImage();
197     Display.loadBI(eImage, 2);
198 }
199 /*****
200 Decodes the loaded image.
201 *****/
202 public static void decode()
203 {
204     String message = Steg.decrypt(Display.getImage());
205     System.out.println(message);
206     if(message.equals(""))
207     {
208         Display.blankError();
209     }
210     else
211         Display.setText(message);
212 }
213
214
215 /* James' methods */
216 /*****
217 Opens a new image.
218 *****/
219 public static void open()
220 {
221     Display.loadImage("");
222 }
223 /*****
224 Saves the loaded image.
225 *****/
226 public static void save()
227 {
228     if(saveLocation!=null)
229     {
230         try {
231             BufferedImage image=Display.getImage(1);

```

```

233         File outputFile = new File(saveLocation);
234         ImageIO.write(image,"png",outputFile);
235     }
236     catch (IOException e){}
237     }
238     else
239         saveAs();
240 }
241 /*****
242 Saves the loaded image as a specific file in a specific place.
243 *****/
244 public static void saveAs()
245 {
246     String filename = Display.fnSave();
247     if(filename!=null)
248     {
249         boolean success=false;
250         while (success==false){
251             try {
252                 BufferedImage image=Display.getImage(1);
253                 if(filename.length(>3)
254                 {
255                     String extension=filename.substring(filename.length()-4).toLowerCase();
256                     if(!extension.equals(".png"))
257                     {
258                         filename+=".png";
259                     }
260                 }
261                 else filename+=".png";
262                 saveLocation=filename;
263                 File outputFile = new File(filename);
264                 ImageIO.write(image,filename.substring(filename.length()-3),outputFile);
265                 success=true;
266             }
267             catch (IOException e){filename = JOptionPane.showInputDialog("Try again.");}}
268     }
269 }
270
271 /*****
272 Displays the user manual to assist the user.
273 *****/
274 public static void help()
275 {
276     Resources.showHelp();
277 }
278
279 /*****
280 Displays a dialog containing information about CRYPTICON
281 and its developers.
282 *****/
283 public static void info()
284 {
285     Resources.displayInfo();
286 }
287 /*****
288 Sets the last save location to a global variable, making it
289 accessible when needed.
290 *****/
291 public static void setLoadedImage(String filename)

```

```
292     {
293         saveLocation=filename;
294     }
295     /*****
296     Copies text in Display's output text field to the system's clipboard.
297     *****/
298     public static void copy()
299     {
300         String output = Display.getOutput();
301         StringSelection stringSelection = new StringSelection(output);
302         Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
303         clipboard.setContents(stringSelection,null);
304     }
305 }
```