

APPENDIX

Mainwindow.java

```
1. package com.appdev.example;
2.
3. import java.awt.BorderLayout;
4. import java.awt.EventQueue;
5.
6. import java.awt.*;
7. import java.awt.event.*;
8. import javax.swing.*;
9. import javax.swing.JFrame;
10. import javax.swing.JPanel;
11. import javax.swing.border.EmptyBorder;
12. import javax.swing.JLayeredPane;
13. import javax.swing.GroupLayout;
14. import javax.swing.GroupLayout.Alignment;
15. import javax.swing.JLabel;
16. import java.awt.Toolkit;
17. import javax.swing.JButton;
18. import javax.swing.LayoutStyle.ComponentPlacement;
19. import com.jgoodies.forms.layout.FormLayout;
20. import com.jgoodies.forms.layout.ColumnSpec;
21. import com.jgoodies.forms.layout.FormSpecs;
22. import com.jgoodies.forms.layout.RowSpec;
23. import java.awt.GridLayout;
24. import javax.swing.SpringLayout;
25. import javax.swing.JFormattedTextField;
26. import javax.swing.JTree;
27. import javax.swing.JList;
28. import java.awt.FlowLayout;
29. import javax.swing.JToolBar;
30. import javax.swing.JScrollPane;
31. import javax.swing.SwingConstants;
32. import java.awt.event.ActionListener;
33. import java.awt.event.ActionEvent;
34. import javax.swing.JFileChooser;
35. import java.io.File;
36. import javax.swing.JSeparator;
37. import javax.swing.border.TitledBorder;
38. import javax.swing.JTextArea;
39. import java.awt.CardLayout;
40. import javax.swing.DropMode;
41. import javax.swing.JTextField;
42. import javax.swing.JDialog;
43. import java.io.FileWriter; // Import the FileWriter class
44. import java.io.IOException; // Import the IOException
        class to handle errors
45. import java.nio.ByteBuffer;
46. import java.util.Base64;
47.
48. public class Mainwindow extends JFrame {
49.
```

```

50.     private JPanel contentPane;
51.
52.     /**
53.      * Launch the application.
54.     */
55.     static JFileChooser fileChooser;
56.     static Mainwindow mainFrame;
57.     private JTextField tf_message_in;
58.
59.     // Global Variables
60.     private String audioInPathEnc;
61.     private String audioInPathDec;
62.     private String audioOutPath;
63.
64.     // Method to convert long to byte array
65.     public byte[] longToBytes(long x) {
66.         ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
67.         buffer.putLong(x);
68.         return buffer.array();
69.     }
70.
71.     // Method to convert byte array to long
72.     public long bytesToLong(byte[] bytes) {
73.         ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES);
74.         buffer.put(bytes);
75.         buffer.flip();
76.         return buffer.getLong();
77.     }
78.
79.     // Auto-generated Method
80.     public static void main(String[] args) {
81.         EventQueue.invokeLater(new Runnable() {
82.             public void run() {
83.                 try {
84.                     Mainwindow frame = new Mainwindow();
85.                     mainFrame = frame;
86.                     frame.setVisible(true);
87.                     fileChooser = new JFileChooser();
88.                     fileChooser.setCurrentDirectory(new File(System.getProperty("user.home")));
89.                 } catch (Exception e) {
90.                     e.printStackTrace();
91.                 }
92.             }
93.         });
94.     }
95. }
96.
97. /**
98.  * Create the frame.
99. */
100.    public Mainwindow() {
101.        setTitle("Audio Enc-Dec Tools");
102.        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
103.        setBounds(200, 200, 544, 531);
104.        contentPane = new JPanel();
105.        contentPane.setToolTipText("");

```

```

106.           contentPane.setBorder(new Empty-
    Border(5, 5, 5, 5));
107.           setContentPane(contentPane);
108.
109.           JPanel panel_2 = new JPanel();
110.           panel_2.setBorder(new TitledBorder(null, "Encryp-
    tion", TitledBorder.LEADING, TitledBorder.TOP, null, null));
111.
112.           JPanel panel_4 = new JPanel();
113.           panel_4.setBorder(new TitledBorder(null, "Decryp-
    tion", TitledBorder.LEADING, TitledBorder.TOP, null, null));
114.           GroupLayout gl_contentPane = new GroupLayout(con-
    tentPane);
115.           gl_contentPane.setHorizontalGroup(
116.               gl_contentPane.createParallelGroup(Align-
    ment.LEADING)
117.                   .addComponent(panel_2, GroupLayout.DE-
    FAULT_SIZE, 518, Short.MAX_VALUE)
118.                   .addComponent(panel_4, GroupLayout.DE-
    FAULT_SIZE, 518, Short.MAX_VALUE)
119.               );
120.           gl_contentPane.setVerticalGroup(
121.               gl_contentPane.createParallelGroup(Align-
    ment.LEADING)
122.                   .addGroup(gl_contentPane.createSequen-
    tialGroup())
123.                   .addComponent(panel_2, GroupLayout.PRE-
    FERRED_SIZE, 229, GroupLayout.PREFERRED_SIZE)
124.                   .addPreferredGap(ComponentPlacement.RE-
    LATED)
125.                   .addComponent(panel_4, GroupLayout.PRE-
    FERRED_SIZE, 246, GroupLayout.PREFERRED_SIZE)
126.                   .addContainer-
    Gap(116, Short.MAX_VALUE)
127.               );
128.           panel_4.setLayout(null);
129.
130.           JPanel panel_5 = new JPanel();
131.           panel_5.setBounds(10, 21, 498, 54);
132.           panel_5.setToolTipText("");
133.           panel_5.setBorder(new TitledBorder(UIManager.getBorder(
    "TitledBorder.border"), "Audio File Input", TitledBorder.LEAD-
    ING, TitledBorder.TOP, null, new Color(0, 0, 0)));
134.           panel_4.add(panel_5);
135.
136.           JLabel lbl_dec_audio_in = new JLabel("No Files Se-
    lected");
137.
138.           JButton button = new JButton("Select File");
139.           button.addActionListener(new ActionListener() {
140.               public void actionPerformed(ActionEvent-
    Event arg0) {
141.                   // Prepares file chooser dialog
142.                   int result = fileChooser.showOpenDialog(con-
    tentPane);
143.                   if (result == JFileChooser.APPROVE_OP-
    TION) {
144.                       // user selects a file

```

```

145.             File selectedFile = fileChooser.get-
146.                 SelectedFile();
147.             audioInPathDec = selectedFile.getAbsolutePath();
148.             //Makes sure selected audio file for-
149.             mat is WAV
150.             if(audioInPathDec.contains(".wav")) {
151.                 lbl_dec_audio_in.setText(audioIn-
152.                     PathDec);
153.             } else {
154.                 // Throws a not supported for-
155.                 mat warning to user
156.                 JOptionPane.showMessageDialog(main-
157.                     Frame,
158.                     "Please pick a '.wav' file!",
159.                     JOptionPane.WARNING_MESSAGE);
160.             }
161.             button.setHorizontalAlignment(SwingConstants.LEADING);
162.             GroupLayout gl_panel_5 = new GroupLayout(panel_5);
163.             gl_panel_5.setHorizontalGroup(
164.                 gl_panel_5.createParallelGroup(Alignment.TRAILING)
165.                     .addGap(0, 506, Short.MAX_VALUE)
166.                     .addGroup(gl_panel_5.createSequentialGroup()
167.                         .addContainerGap()
168.                         .addComponent(lbl_dec_a-
169.                             dio_in, GroupLayout.DEFAULT_SIZE, 386, Short.MAX_VALUE)
170.                         .addGap(18)
171.                         .addComponent(button, GroupLayout.PREFERRED_SIZE, 80, GroupLayout.PREFERRED_SIZE)
172.                     );
173.             gl_panel_5.setVerticalGroup(
174.                 gl_panel_5.createParallelGroup(Alignment.LEADING)
175.                     .addGap(0, 49, Short.MAX_VALUE)
176.                     .addGroup(gl_panel_5.createSequentialGroup()
177.                         .addComponent(lbl_dec_audio_in)
178.                         .addComponent(button))
179.                         .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
180.                     );
181.             panel_5.setLayout(gl_panel_5);
182.
183.             JPanel panel_6 = new JPanel();

```

```

184.         panel_6.setBorder(new TitledBorder(null, "Mes-
185.             sage", TitledBorder.LEADING, TitledBorder.TOP, null, null));
186.         panel_6.setBounds(10, 78, 498, 133);
187.         panel_6.setLayout(null);
188.
189.         JLabel lbl_message_out = new JLabel();
190.         lbl_message_out.setVerticalAlignment(SwingConstants-
191.             TOP);
192.         lbl_message_out.setBounds(10, 21, 478, 101);
193.         panel_6.add(lbl_message_out);
194.
195.         JButton btn_decrypt = new JButton("Decrypt");
196.         btn_decrypt.addActionListener(new Action-
197.             Listener() {
198.                 public void actionPerformed(ActionEvent-
199.                     arg0) {
200.                         if(audioInPathDec == null || audioIn-
201.                             PathDec.isEmpty()) {
202.                             // Throws an error to user
203.                             JOptionPane.showMessageDialog(main-
204.                                 Frame,
205.                                 "Error",
206.                                 "Audio Input not Specified",
207.                                 JOptionPane.ERROR_MESSAGE);
208.                         return;
209.                     }
210.                     long t_start = System.nanoTime();
211.                     // Decrypting messages
212.                     try {
213.                         // buffer to handle message length
214.                         long[] bufferIn = new long[64];
215.                         // Open audio file
216.                         WavFile wavIn = WavFile.open-
217.                             WavFile(new File(audioInPathDec));
218.                         // Get number of channels
219.                         int numChannels = wavIn.getNumChan-
220.                             nels();
221.                         // Read 64 bit LSB from the most begin-
222.                         //ning frames (as message length)
223.                         wavIn.readFrames(bufferIn, 64 / numChan-
224.                             nels());
225.                         // ===== Convert LSB bit of mes-
226.                         // sage length to byte array =====
227.                         // Variable to store mes-
228.                         sage length as byte
229.                         byte[] bytes = new byte[8];
230.
231.                         for(int i = 0, j = -1; i < 64; i++) {
232.                             if( i % 8 == 0) j++;
233.                             bytes[j] += (byte)(buff-
234.                                 erIn[i] & 1) * Math.pow(2, i % 8);
235.                         }

```

```

229.          // === End of the part ===
230.
231.          // Convert byte array to long
232.          long msgLength = bytesToLong(bytes);
233.
234.          // buffer to handle encrypted message
235.          long[] buff-
236.          erIn2 = new long[(int)msgLength * 8];
237.          // Read whole message
238.          wavIn.readFrames(buff-
239.          erIn2, (int)msgLength * 8 / numChannels);
240.          // === Convert LSB bit of mes-
241.          sage to byte array ===
242.          // Variable to store message as byte ar-
243.          // ray
244.          byte[] msgBytes = new byte[(int)msgLengt-
245.          h];
246.          for(int i = 0, j = -1; i < (int)msgLength * 8; i++) {
247.              if( i % 8 == 0) j++;
248.              msgBytes[j] += (byte)(buff-
249.              erIn2[i] & 1) * Math.pow(2, i % 8);
250.          }
251.          // === End of the part ===
252.          // === Retrieve base64 encoded mes-
253.          sage as real message ===
254.          // Instantiate string from byte array
255.          String decMsg = new String(msgBytes);
256.          // Decode encoded message
257.          String realMsg = new String(Base64.getDe-
258.          coder().decode(decMsg));
259.          // Display message to user
260.          lbl_mes-
261.          sage_out.setText("<html>" + realMsg.re-
262.          place(" ", "<wbr>") + "</html>");
263.          // === End of the part ===
264.
265.          wavIn.close();
266.          long t_end = System.nanoTime();
267.          String time_str = "Elapsed Time:\n";
268.          time_str += (t_end - t_start)/1000000 + " ms\n";
269.
270.          }catch(Exception e) {

```

```

271.           JOptionPane.showMessageDialog(main-
272.               Frame,
273.                   "An Error Occured: " + e.getMes-
274.                   sage(),
275.                   "Error",
276.                   JOptionPane.ERROR_MESSAGE);
277.           }
278.           btn_decrypt.setBounds(425, 215, 80, 23);
279.           panel_4.add(btn_decrypt);
280.           panel_2.setLayout(null);
281.
282.           JPanel panel_1 = new JPanel();
283.           panel_1.setBounds(6, 76, 506, 55);
284.           panel_2.add(panel_1);
285.           panel_1.setBorder(new TitledBorder(null, "Mes-
286.               sage", TitledBorder.LEADING, TitledBorder.TOP, null, null));
287.           panel_1.setLayout(null);
288.           tf_message_in = new JTextField();
289.           tf_message_in.setBounds(10, 22, 486, 20);
290.           panel_1.add(tf_message_in);
291.           tf_message_in.setColumns(10);
292.
293.           JPanel panel = new JPanel();
294.           panel.setBounds(6, 16, 506, 49);
295.           panel_2.add(panel);
296.           panel.setBorder(new TitledBorder(UIManager.getBorder(
297.               "TitledBorder.border"), "Audio File Input", TitledBorder.LEADING,
298.               TitledBorder.TOP, null, new Color(0, 0, 0)));
299.           panel.setToolTipText("");
300.
301.           JLabel lbl_enc_audio_in = new JLabel("No Files Se-
302.               lected");
303.           JButton btn_select_audio = new JButton("Se-
304.               lect File");
305.           btn_select_audio.addActionListener(new Ac-
306.               tionListener());
307.           public void actionPerformed(Action-
308.               Event arg0) {
309.               int result = fileChooser.showOpen-
310.                   Dialog(contentPane);
311.               if (result == JFileChooser.AP-
312.                   PROVE_OPTION) {
313.                   Chooser.getSelectedFile();
314.                   File selectedFile = file-
315.                       Chooser.getSelectedFile();
316.                   audioInPathEnc = selected-
317.                       File.getAbsolutePath();
318.                   if (audioInPathEnc.con-
319.                       tains(".wav")) {
320.                       audioInPathEnc =
321.                           audioInPathEnc;
322.                       lbl_enc_audio_in.setText(au-
323.                           dioInPathEnc);
324.                   } else {
325.                   }
326.               }
327.           }
328.       }
329.   }
330. }

```

```

314.                                     JOptionPane.showMessageDia-
315.         log(mainFrame,
316.                           "Please pick a '.wav"
317.                           "File For-
318.                           mat Not Supported",
319.                           JOptionPane.WARNING_MESSAGE);
320.         }
321.     });
322.     btn_select_audio.setHorizontalAlign-
323.         ment(SwingConstants.LEADING);
324.     GroupLayout gl_panel = new GroupLay-
325.         out(panel);
326.         gl_panel.setHorizontalGroup(
327.             gl_panel.createParallelGroup(Align-
328.               ment.TRAILING)
329.             .addGroup(gl_panel.createSequen-
330.               tialGroup()
331.               .addContainerGap()
332.               .addComponent(lbl_enc_a-
333.                 dio_in, GroupLayout.DEFAULT_SIZE, 385, Short.MAX_VALUE)
334.               .addGap(18)
335.               .addComponent(btn_select_a-
336.                 dio, GroupLayout.PREFERRED_SIZE, 80, GroupLayout.PRE-
337.                   FERRED_SIZE))
338.             );
339.             gl_panel.setVerticalGroup(
340.                 gl_panel.createParallelGroup(Align-
341.                   ment.LEADING)
342.                   .addGroup(gl_panel.createSequen-
343.                     tialGroup()
344.                     .addGroup(gl_panel.createParal-
345.                       lelGroup(Alignment.BASELINE)
346.                         .addComponent(lbl_enc_a-
347.                           dio_in)
348.                           .addComponent(btn_select_a-
349.                             dio))
350.                           .addContainerGap(GroupLayout.DE-
351.                               FAULT_SIZE, Short.MAX_VALUE)
352.                           );
353.                           panel.setLayout(gl_panel);
354.                           JPanel panel_3 = new JPanel();
355.                           panel_3.setToolTipText("");
356.                           panel_3.setBorder(new TitledBorder(UIMan-
357.                               ger.getBorder("TitledBorder.border"), "Audio File Output", Ti-
358.                               tledBorder.LEADING, TitledBorder-
359.                                 .TOP, null, new Color(0, 0, 0)));
360.                           panel_3.setBounds(6, 142, 506, 49);
361.                           panel_2.add(panel_3);
362.                           JLabel lbl_save_loc = new JLa-
363.                             bel("No Save Location Specified");
364.                           JButton btnBrowse = new JButton("Browse");

```

```
351.         btnBrowse.addActionListener(new Action-
352.                               Listener() {
353.                                     public void actionPerformed(Action-
354.                                         Event arg0) {
355.                                           JFrame parentFrame = new JFrame();
356.                                           JFileChooser file-
357.                                             Chooser = new JFileChooser();
358.                                               fileChooser.setDialogTitle("Spec-
359.                                                 ify a file to save");
360.                                               int userSelection = file-
361.                                                 Chooser.showSaveDialog(parentFrame);
362.                                               if (userSelection == JFile-
363.                                                 Chooser.APPROVE_OPTION) {
364.                                                   File fileToSave = file-
365.                                                     fileChooser.getSelectedFile();
366.                                                   absolutePath();
367.                                                   lbl_save_loc.setText("Save as fi-
368.                                                     le: " + fileToSave.getAbsolutePath());
369.                                                 }
370.                                               }
371.                                               GroupLayout gl_panel_3 = new GroupLayout(
372.                                                 gl_panel_3.setHorizontalGroup(
373.                                                   gl_panel_3.createParallelGroup(Align-
374.                                                     ment.TRAILING)
375.                                                   .addGroup(gl_panel_3.createSequen-
376.                                                       tialGroup())
377.                                                       .addCompo-
378.                                                       nent(lbl_save_loc, GroupLayout.DE-
379.                                                         FAULT_SIZE, 407, Short.MAX_VALUE)
380.                                                       .addPreferredGap(ComponentPlace-
381.                                                         ment.UNRELATED)
382.                                                       .addCompo-
383.                                                       nent(btnBrowse, GroupLayout.PREFERRED_SIZE, 80, GroupLayout.PRE-
384.                                                         FERRED_SIZE))
385.                                                       );
386.                                               panel_3.setLayout(gl_panel_3);
```

```

387.
388.                     JButton btnNewButton = new JButton("En-
   cypt");
389.                     btnNewButton.addActionListener(new Action-
   Listener());
390.                     public void actionPerformed(ActionEvent
   arg0) {
391.                         if(audioInPathEnc == null || audio-
   InPathEnc.isEmpty()) {
392.                             JOptionPane.showMessageDia-
   log(mainFrame,
393.                                 "Audio Input not Speci-
   fied",
394.                                 "Error",
395.                                 JOptionPane.ERROR_MES-
   SAGE);
396.                         return;
397.                     }
398.                     if(tf_message_in.getText().is-
   Empty()) {
399.                         JOptionPane.showMessageDia-
   log(mainFrame,
400.                                 "Message Could not Be Empty",
401.                                 "Error",
402.                                 JOptionPane.ERROR_MES-
   SAGE);
403.                         return;
404.                     }
405.                     if(audioOutPath == null || audioIn-
   PathEnc.isEmpty()) {
406.                         JOptionPane.showMessageDia-
   log(mainFrame,
407.                                 "Audio Output not Speci-
   fied",
408.                                 "Error",
409.                                 JOptionPane.ERROR_MES-
   SAGE);
410.                         return;
411.                     }
412.                     long t_start = System.nanoTime();
413.                     try {
414.                         // Original Message
415.                         String message = tf_mes-
   sage_in.getText();
416.                         // Base64 Encoded Message
417.                         String encodedMes-
   sage = Base64.getEncoder().encodeToString(message.getBytes());
418.                         // Debug Purpose
419.                         System.out.println("Mes-
   sage: " + message);
420.                         System.out.println("En-
   coded: " + encodedMessage);
421.                         System.out.println("En-
   coded: " + encodedMessage);
422.                         System.out.println("En-
   coded: " + encodedMessage);
423.                         System.out.println("En-
   coded: " + encodedMessage);
424.                         System.out.println("En-
   coded: " + encodedMessage);
425.

```

```

426.                                     // Get encoded mes-
427.         sage as byte array
428.         byte[] encodedMsgBytes = encod-
429.             edMessage.getBytes();
430.                                     // Inserted encoded message
431.         byte[] formattedMs-
432.             gBytes = new byte[8 + encodedMsgBytes.length];
433.                                     // Get encoded message length
434.         byte[] encodedMs-
435.             gLen = longToBytes(encodedMsgBytes.length);
436.                                     for(int i = 0; i < 8; i++) {
437.             formattedMsgBytes[i] = en-
438.                 codedMsgLen[i];
439.         }
440.     }
441.     for(int i = 0; i < encodedMs-
442.         gBytes.length; i++) {
443.         gBytes[8 + i] = encodedMsgBytes[i];
444.     }
445.     // Extract encoded mes-
446.     sage byte array into bit
447.     byte[] mes-
448.         sageBits = new byte[(formattedMsgBytes.length * 8)];
449.         for(int i = 0, bit_count = 0; i <
450.             formattedMsgBytes.length; i++) {
451.             for(int j = 0; j < 8; j++, b
452.                 it_count++) {
453.                     sageBits[bit_count] = (byte) ((formattedMs-
454.                         gBytes[i] & (byte) Math.pow(2, j)) >> j);
455.                 }
456.             }
457.         // Open WAV input
458.         WavFile wavIn = WavFile.open-
459.             WavFile(new File(audioInPathEnc));
460.             wavIn.display();
461.             // Get the number of audio chan-
462.             nels in the WAV file
463.             int numChannels = wavIn.getNum-
464.                 Channels();
465.             long numFrames = wavIn.getNum-
466.                 Frames();
467.             // Preparing output
468.             WavFile wavOut = WavFile.new-
469.                 WavFile(new File(
470.                     audioOutPath),
471.                     numChannels,
472.                     numFrames);

```

```

466.                                         numFrames,
467.                                         wavIn.getValidBits(),
468.                                         wavIn.get-
469.                                         SampleRate());
470.                                         // Cre-
471.                                         ate a buffer of 100 frames
472.                                         long[] buff-
473.                                         erIn = new long[100 * numChannels];
474.                                         long[][] bufferOut = new long[nu-
475.                                         mChannels][100];
476.                                         int messageBitsRemaining = mes-
477.                                         sageBits.length;
478.                                         int framesRead;
479.                                         do {
480.                                         // Read frames into buffer
481.                                         framesRead = wavIn.read-
482.                                         long remaining = wavOut.get-
483.                                         int toWrite = (remain-
484.                                         Write ; i++)
485.                                         for (int i = 0, k = 0; i < to
486.                                         { for(int j = 0; j < num-
487.                                         Channels; j++, k++, messageBitsRemaining--) {
488.                                         if(messageBitsRemain-
489.                                         ing > 0) { bufferIn[k] >>= 1; bufferIn[k] <<= 1; bufferOut[j][i] =
490.                                         bufferIn[k] + (long) messageBits[messageBits.length - mes-
491.                                         sageBitsRemaining];
492.                                         } else {
493.                                         bufferOut[j][i] =
494.                                         bufferIn[k];
495.                                         }
496.                                         wavOut.writeFrames(bufferOut,
497.                                         toWrite);
498.                                         } while (framesRead != 0);
499.                                         wavIn.close();
500.                                         wavOut.close();
501.                                         long t_end = System.nano-
502.                                         Time();
503.                                         String time_str = "Elapsed Time:
504.                                         \n";
505.                                         time_str += (t_end-
t_start)/1000000 + " ms\n";

```

```

506.                                     JOptionPane.showMessageDialog(
507.             log(mainFrame,
508.                 "Successfully En-
509.                 crpyt the Message\n" + time_str,
510.             "Done",
511.             JOptionPane.INFOR-
512.                 MATION_MESSAGE);
513.             }catch (Exception e) {
514.                 JOptionPane.showMessageDialog(
515.                     log(mainFrame,
516.                         "An Error Oc-
517.                         cured: " + e.getMessage(),
518.                         "Error",
519.                         JOptionPane.ERROR_MESSAGE));
520.             panel_2.setLayout(gl_contentPane);
521.         }
522.     }
}
});
```

WavFile.java

```

1. package com.appdev.example;
2.
3. import java.io.*;
4.
5. public class WavFile
6. {
7.     private enum IOState {READING, WRITING, CLOSED};
8.     private final static int BUFFER_SIZE = 4096;
9.
10.    private final static int FMT_CHUNK_ID = 0x20746D66;
11.    private final static int DATA_CHUNK_ID = 0x61746164;
12.    private final static int RIFF_CHUNK_ID = 0x46464952;
13.    private final static int RIFF_TYPE_ID = 0x45564157;
14.
15.    pri-
16.        vate File file;                                // File that will be read fro
17.        m or written to
18.        private IOState ioState;                      // Speci-
19.        fies the IO State of the Wav File (used for sanity checking)
20.        private int bytesPerSample;                  // Number of bytes re-
21.        quired to store a single sample
22.        private long numFrames;                      // Num-
23.        ber of frames within the data section
24.        private FileOutputStream oStream;          // Out-
25.        put stream used for writing data
26.        private FileInputStream iStream;           // In-
27.        put stream used for reading data
}
}
});
```

```

21.     private double floatScale;           // Scaling fac-
   tor used for int <-> float conversion
22.     private double floatOffset;          // Offset fac-
   tor used for int <-> float conversion
23.     private boolean wordAlignAdjust;      // Specify if an ex-
   tra byte at the end of the data chunk is required for word align-
   ment
24.
25.     // Wav Header
26.     private int numChannels;             // 2 bytes un-
   signed, 0x0001 (1) to 0xFFFF (65,535)
27.     private long sampleRate;            // 4 bytes un-
   signed, 0x00000001 (1) to 0xFFFFFFFF (4,294,967,295)
28.                                         // Alt-
   though a java int is 4 bytes, it is signed, so need to use a long
29.     private int blockAlign;              // 2 bytes un-
   signed, 0x0001 (1) to 0xFFFF (65,535)
30.     private int validBits;              // 2 bytes un-
   signed, 0x0002 (2) to 0xFFFF (65,535)
31.
32.     // Buffering
33.     private byte[] buffer;              // Lo-
   cal buffer used for IO
34.     private int bufferPointer;          // Points to the cur-
   rent position in local buffer
35.     private int bytesRead;              // Bytes read af-
   ter last read into local buffer
36.     private long frameCounter;         // Current num-
   ber of frames read or written
37.
38.     // Cannot instantiate WavFile directly, must either use new-
   WavFile() or openWavFile()
39.     private WavFile()
40.     {
41.         buffer = new byte[BUFFER_SIZE];
42.     }
43.
44.     public int getNumChannels()
45.     {
46.         return numChannels;
47.     }
48.
49.     public long getNumFrames()
50.     {
51.         return numFrames;
52.     }
53.
54.     public long getFramesRemaining()
55.     {
56.         return numFrames - frameCounter;
57.     }
58.
59.     public long getSampleRate()
60.     {
61.         return sampleRate;
62.     }
63.
64.     public int getValidBits()

```

```

65.      {
66.          return validBits;
67.      }
68.
69.      public static WavFile newWavFile(File file, int numChan-
    nels, long numFrames, int validBits, long sampleRate) throws IOException,
        WavFileException
70.      {
71.          // Instantiate new Wavfile and initialise
72.          WavFile wavFile = new WavFile();
73.          wavFile.file = file;
74.          wavFile.numChannels = numChannels;
75.          wavFile.numFrames = numFrames;
76.          wavFile.sampleRate = sampleRate;
77.          wavFile.bytesPerSample = (validBits + 7) / 8;
78.          wavFile.blockAlign = wavFile.bytesPerSample * numChan-
    nels;
79.          wavFile.validBits = validBits;
80.
81.          // Sanity check arguments
82.          if (numChannels < 1 || numChan-
    nels > 65535) throw new WavFileException("Illegal number of chan-
    nels, valid range 1 to 65536");
83.          if (numFrames < 0) throw new WavFileException("Num-
    ber of frames must be positive");
84.          if (validBits < 2 || valid-
    Bits > 65535) throw new WavFileException("Illegal num-
    ber of valid bits, valid range 2 to 65536");
85.          if (sampleRate < 0) throw new WavFileException("Sam-
    ple rate must be positive");
86.
87.          // Create output stream for writing data
88.          wavFile.oStream = new FileOutputStream(file);
89.
90.          // Calculate the chunk sizes
91.          long dataChunkSize = wavFile.blockAlign * numFrames;
92.          long mainChunkSize = 4 + // Riff Type
93.                               8 + // Format ID and size
94.                               16 + // Format data
95.                               8 + // Data ID and size
96.                               dataChunkSize;
97.
98.          // Chunks must be word aligned, so if odd number of au-
    dio data bytes
99.          // adjust the main chunk size
100.         if (dataChunkSize % 2 == 1) {
101.             mainChunkSize += 1;
102.             wavFile.wordAlignAdjust = true;
103.         }
104.         else {
105.             wavFile.wordAlignAdjust = false;
106.         }
107.
108.         // Set the main chunk size
109.         putLE(RIFF_CHUNK_ID, wavFile.buffer, 0, 4);
110.         putLE(mainChunkSize, wavFile.buffer, 4, 4);
111.         putLE(RIFF_TYPE_ID, wavFile.buffer, 8, 4);
112.

```

```

113.          // Write out the header
114.          wavFile.oStream.write(wavFile.buffer, 0, 12);
115.
116.          // Put format data in buffer
117.          long averageBytesPerSecond = sampleRate * wavFile.blockAlign;
118.
119.          putLE(FMT_CHUNK_ID, wavFile.buffer, 0, 4);
120.          putLE(16, wavFile.buffer, 4, 4); // Chunk Data Size
121.          putLE(1, wavFile.buffer, 8, 2); // Compression Code (Uncompressed)
122.          putLE(numChannels, wavFile.buffer, 10, 2); // Number of channels
123.          putLE(sampleRate, wavFile.buffer, 12, 4); // Sample Rate
124.          putLE(averageBytesPerSecond, wavFile.buffer, 16, 4); // Average Bytes Per Second
125.          putLE(wavFile.blockAlign, wavFile.buffer, 20, 2); // Block Align
126.          putLE(validBits, wavFile.buffer, 22, 2); // Valid Bits
127.
128.          // Write Format Chunk
129.          wavFile.oStream.write(wavFile.buffer, 0, 24);
130.
131.          // Start Data Chunk
132.          putLE(DATA_CHUNK_ID, wavFile.buffer, 0, 4); // Chunk ID
133.          putLE(dataChunkSize, wavFile.buffer, 4, 4); // Chunk Data Size
134.
135.          // Write Format Chunk
136.          wavFile.oStream.write(wavFile.buffer, 0, 8);
137.
138.          // Calculate the scaling factor for converting to a normalised double
139.          if (wavFile.validBits > 8)
140.          {
141.              // If more than 8 validBits, data is signed
142.              // Conversion required multiplying by magnitude of max positive value
143.              wavFile.floatOffset = 0;
144.              wavFile.floatScale = Long.MAX_VALUE >> (64 - wavFile.validBits);
145.          }
146.          else
147.          {
148.              // Else if 8 or less validBits, data is unsigned
149.              // Conversion required dividing by max positive value
150.              wavFile.floatOffset = 1;

```

```

151.           wavFile.floatScale = 0.5 * ((1 << wavFile.valid-
   Bits) - 1);
152.       }
153.
154.           // Finally, set the IO State
155.           wavFile.bufferPointer = 0;
156.           wavFile.bytesRead = 0;
157.           wavFile.frameCounter = 0;
158.           wavFile.ioState = IOState.WRITING;
159.
160.       return wavFile;
161.   }
162.
163.   public static WavFile openWavFile(File file) throws IO-
   Exception, WavFileException
164.   {
165.       // Instantiate new Wavfile and store the file refer-
   ence
166.       WavFile wavFile = new WavFile();
167.       wavFile.file = file;
168.
169.       // Create a new file input stream for read-
   ing file data
170.       wavFile.iStream = new FileInputStream(file);
171.
172.       // Read the first 12 bytes of the file
173.       int bytesRead = wavFile.iStream.read(wavFile.buffer,
   0, 12);
174.       if (bytesRead != 12) throw new WavFileExcep-
   tion("Not enough wav file bytes for header");
175.
176.       // Extract parts from the header
177.       long riffChunkID = getLE(wavFile.buffer, 0, 4);
178.       long chunkSize = getLE(wavFile.buffer, 4, 4);
179.       long riffTypeID = getLE(wavFile.buffer, 8, 4);
180.
181.       // Check the header bytes contains the correct sig-
   nature
182.       if (riff-
   ChunkID != RIFF_CHUNK_ID) throw new WavFileException("Inva-
   lid Wav Header data, incorrect riff chunk ID");
183.       if (riffTypeID != RIFF_TYPE_ID) throw new WavFileEx-
   ception("Invalid Wav Header data, incorrect riff type ID");
184.
185.       // Check that the file size matches the num-
   ber of bytes listed in header
186.       if (file.length() != chunkSize+8) {
187.           throw new WavFileExcep-
   tion("Header chunk size (" + chunkSize + ") does not match file si-
   ze (" + file.length() + ")");
188.       }
189.
190.       boolean foundFormat = false;
191.       boolean foundData = false;
192.
193.       // Search for the Format and Data Chunks
194.       while (true)
195.   {

```

```

196.          // Read the first 8 bytes of the chunk (ID and c
    hunk size)
197.          bytesRead = wavFile.iStream.read(wavFile.buffer,
        0, 8);
198.          if (bytesRead == -1) throw new WavFileException(
    "Reached end of file without finding format chunk");
199.          if (bytesRead != 8) throw new WavFileException(
    "Could not read chunk header");
200.
201.          // Extract the chunk ID and Size
202.          long chunkID = getLE(wavFile.buffer, 0, 4);
203.          chunkSize = getLE(wavFile.buffer, 4, 4);
204.
205.          // Word align the chunk size
206.          // chunkSize specifies the number of bytes hold-
    ing data. However,
207.          // the data should be word aligned (2 bytes) so
    we need to calculate
208.          // the actual number of bytes in the chunk
209.          long numChun-
    Bytes = (chunkSize%2 == 1) ? chunkSize+1: chunkSize;
210.
211.          if (chunkID == FMT_CHUNK_ID)
212.          {
213.              // Flag that the for-
    mat chunk has been found
214.              foundFormat = true;
215.
216.              // Read in the header info
217.              bytesRead = wavFile.iStream.read(wavFile.buf
    fer, 0, 16);
218.
219.              // Check this is uncompressed data
220.              int compres-
    sionCode = (int) getLE(wavFile.buffer, 0, 2);
221.              if (compres-
    sionCode != 1) throw new WavFileException("Compre-
    ssion Code " + compressionCode + " not supported");
222.
223.              // Extract the format information
224.              wavFile.numChan-
    nels = (int) getLE(wavFile.buffer, 2, 2);
225.              wavFile.sam-
    pleRate = getLE(wavFile.buffer, 4, 4);
226.              wavFile.block-
    Align = (int) getLE(wavFile.buffer, 12, 2);
227.              wavFile.valid-
    Bits = (int) getLE(wavFile.buffer, 14, 2);
228.
229.              if (wavFile.numChan-
    nels == 0) throw new WavFileException("Number of channels speci-
    fied in header is equal to zero");
230.              if (wavFile.block-
    Align == 0) throw new WavFileException("Block Align speci-
    fied in header is equal to zero");
231.              if (wavFile.valid-
    Bits < 2) throw new WavFileException("Valid Bits speci-
    fied in header is less than 2");

```

```

232.                     if (wavFile.valid-
   Bits > 64) throw new WavFileException("Valid Bits speci-
   fied in header is greater than 64, this is greater than a long can
   hold");
233.
234.                     // Calculate the number of bytes re-
   quired to hold 1 sample
235.                     wavFile.bytesPerSample = (wavFile.valid-
   Bits + 7) / 8;
236.                     if (wavFile.bytesPerSample * wavFile.num-
   Channels != wavFile.blockAlign)
237.                     throw new WavFileExcep-
   tion("Block Align does not agree with bytes required for valid-
   Bits and number of channels");
238.
239.                     // Account for number of for-
   mat bytes and then skip over
240.                     // any extra format bytes
241.                     numChunkBytes -= 16;
242.
   Bytes > 0) wavFile.iStream.skip(numChunkBytes);
243.     }
244.     else if (chunkID == DATA_CHUNK_ID)
245.     {
246.         // Check if we've found the format chunk,
247.         // If not, throw an excep-
   tion as we need the format information
248.         // before we can read the data chunk
249.         if (foundFor-
   mat == false) throw new WavFileException("Data chunk found be-
   fore Format chunk");
250.
251.         // Check that the chunkSize (wav data length
   ) is a multiple of the
252.         // block align (bytes per frame)
253.         if (chunkSize % wavFile.block-
   Align != 0) throw new WavFileExcep-
   tion("Data Chunk size is not multiple of Block Align");
254.
255.         // Calculate the number of frames
256.         wavFile.num-
   Frames = chunkSize / wavFile.blockAlign;
257.
258.         // Flag that we've found the wave data chunk
259.         foundData = true;
260.
261.         break;
262.     }
263.     else
264.     {
265.         // If an un-
   known chunk ID is found, just skip over the chunk data
266.         wavFile.iStream.skip(numChunkBytes);
267.     }
268. }
269.

```

```

270.          // Throw an exception if no data chunk has been found
271.          if (foundData == false) throw new WavFileException("Did not find a data chunk");
272.
273.          // Calculate the scaling factor for converting to a normalised double
274.          if (wavFile.validBits > 8)
275.          {
276.              // If more than 8 validBits, data is signed
277.              // Conversion required dividing by magnitude of max negative value
278.              wavFile.floatOffset = 0;
279.              wavFile.floatScale = 1 << (wavFile.validBits - 1);
280.          }
281.          else
282.          {
283.              // Else if 8 or less validBits, data is unsigned
284.              // Conversion required dividing by max positive value
285.              wavFile.floatOffset = -1;
286.              wavFile.floatScale = 0.5 * ((1 << wavFile.validBits) - 1);
287.          }
288.
289.          wavFile.bufferPointer = 0;
290.          wavFile.bytesRead = 0;
291.          wavFile.frameCounter = 0;
292.          wavFile.ioState = IOState.READING;
293.
294.          return wavFile;
295.      }
296.
297.      // Get and Put little endian data from local buffer
298.      // -----
299.      private static long getLE(byte[] buffer, int pos, int numBytes)
300.      {
301.          numBytes--;
302.          pos += numBytes;
303.
304.          long val = buffer[pos] & 0xFF;
305.          for (int b=0 ; b<numBytes ; b++) val = (val << 8) +
306.              (buffer[--pos] & 0xFF);
307.
308.          return val;
309.      }
310.
311.      private static void putLE(long val, byte[] buffer, int pos, int numBytes)
312.      {
313.          for (int b=0 ; b<numBytes ; b++)
314.          {
315.              buffer[pos] = (byte) (val & 0xFF);
            val >>= 8;
        }
    }
}

```

```

316.             pos++;
317.         }
318.     }
319.
320.     // Sample Writing and Reading
321.     // -----
322.     private void writeSample(long val) throws IOException
323.     {
324.         for (int b=0 ; b<bytesPerSample ; b++)
325.         {
326.             if (bufferPointer == BUFFER_SIZE)
327.             {
328.                 ostream.write(buffer, 0, BUFFER_SIZE);
329.                 bufferPointer = 0;
330.             }
331.
332.             buffer[bufferPointer] = (byte) (val & 0xFF);
333.             val >>= 8;
334.             bufferPointer++;
335.         }
336.     }
337.
338.     private long readSample() throws IOException, WavFileException
339.     {
340.         long val = 0;
341.
342.         for (int b=0 ; b<bytesPerSample ; b++)
343.         {
344.             if (bufferPointer == bytesRead)
345.             {
346.                 int read = iStream.read(buffer, 0, BUFFER_SIZE);
347.
348.                 if (read == -1) throw new WavFileException("Not enough data available");
349.                 bytesRead = read;
350.                 bufferPointer = 0;
351.
352.                 int v = buffer[bufferPointer];
353.                 if (b < bytesPerSample-1 || bytesPer-
354.                     Sample == 1) v &= 0xFF;
355.                 val += v << (b * 8);
356.
357.             }
358.
359.             return val;
360.         }
361.
362.         // Integer
363.         // -----
364.         public int readFrames(int[] sampleBuffer, int numFrame-
365.             sToRead) throws IOException, WavFileException
366.         {
367.             return readFrames(sampleBuffer, 0, numFrame-
sToRead);
367.         }

```

```
368.
369.         public int readFrames(int[] sampleBuffer, int off-
   set, int numFramesToRead) throws IOException, WavFileException
370.         {
371.             if (ioState != IOState.READING) throw new IOException(
   "Cannot read from WavFile instance");
372.
373.             for (int f=0 ; f<numFramesToRead ; f++)
374.             {
375.                 if (frameCounter == numFrames) return f;
376.
377.                 for (int c=0 ; c<numChannels ; c++)
378.                 {
379.                     sampleBuffer[offset] = (int) readSample();
380.                     offset++;
381.                 }
382.
383.                 frameCounter++;
384.             }
385.
386.             return numFramesToRead;
387.         }
388.
389.         public int readFrames(int[][] sampleBuffer, int num-
   FramesToRead) throws IOException, WavFileException
390.         {
391.             return readFrames(sampleBuffer, 0, numFrame-
   sToRead);
392.         }
393.
394.         public int readFrames(int[][][] sampleBuffer, int off-
   set, int numFramesToRead) throws IOException, WavFileException
395.         {
396.             if (ioState != IOState.READING) throw new IOException(
   "Cannot read from WavFile instance");
397.
398.             for (int f=0 ; f<numFramesToRead ; f++)
399.             {
400.                 if (frameCounter == numFrames) return f;
401.
402.                 for (int c=0 ; c<numChannels ; c++) sam-
   pleBuffer[c][offset] = (int) readSample();
403.
404.                 offset++;
405.                 frameCounter++;
406.             }
407.
408.             return numFramesToRead;
409.         }
410.
411.         public int writeFrames(int[] sampleBuffer, int numFrame-
   sToWrite) throws IOException, WavFileException
412.         {
413.             return writeFrames(sampleBuffer, 0, numFrame-
   sToWrite);
414.         }
415.
```

```
416.         public int writeFrames(int[] sampleBuffer, int off-
417.             set, int numFramesToWrite) throws IOException, WavFileException
418.             {
419.                 if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
420.                 for (int f=0 ; f<numFramesToWrite ; f++)
421.                 {
422.                     if (frameCounter == numFrames) return f;
423.                     for (int c=0 ; c<numChannels ; c++)
424.                     {
425.                         writeSample(sampleBuffer[offset]);
426.                         offset++;
427.                     }
428.                 }
429.                 frameCounter++;
430.             }
431.             return numFramesToWrite;
432.         }
433.     }
434. }
435.
436.     public int writeFrames(int[][] sampleBuffer, int num-
437.     FramesToWrite) throws IOException, WavFileException
438.     {
439.         return writeFrames(sampleBuffer, 0, numFrame-
440.         sToWrite);
441.     }
442.     public int writeFrames(int[][] sampleBuffer, int off-
443.         set, int numFramesToWrite) throws IOException, WavFileException
444.         {
445.             if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
446.             for (int f=0 ; f<numFramesToWrite ; f++)
447.             {
448.                 if (frameCounter == numFrames) return f;
449.                 for (int c=0 ; c<numChan-
450.                     nels ; c++) writeSample(sampleBuffer[c][offset]);
451.                     offset++;
452.                     frameCounter++;
453.             }
454.             return numFramesToWrite;
455.         }
456.     }
457.
458. // Long
459. // ----
460.     public int readFrames(long[] sampleBuffer, int numFrame-
461.     sToRead) throws IOException, WavFileException
462.     {
463.         return readFrames(sampleBuffer, 0, numFrame-
464.         sToRead);
465.     }
```

```
465.         public int readFrames(long[] sampleBuffer, int off-
    set, int numFramesToRead) throws IOException, WavFileException
466.         {
467.             if (ioState != IOState.READING) throw new IOException(
    "Cannot read from WavFile instance");
468.
469.             for (int f=0 ; f<numFramesToRead ; f++)
470.             {
471.                 if (frameCounter == numFrames) return f;
472.
473.                 for (int c=0 ; c<numChannels ; c++)
474.                 {
475.                     sampleBuffer[offset] = readSample();
476.                     offset++;
477.                 }
478.
479.                 frameCounter++;
480.             }
481.
482.             return numFramesToRead;
483.         }
484.
485.         public int readFrames(long[][] sampleBuffer, int num-
    FramesToRead) throws IOException, WavFileException
486.         {
487.             return readFrames(sampleBuffer, 0, numFrame-
    sToRead);
488.         }
489.
490.         public int readFrames(long[][][] sampleBuffer, int off-
    set, int numFramesToRead) throws IOException, WavFileException
491.         {
492.             if (ioState != IOState.READING) throw new IOException(
    "Cannot read from WavFile instance");
493.
494.             for (int f=0 ; f<numFramesToRead ; f++)
495.             {
496.                 if (frameCounter == numFrames) return f;
497.
498.                 for (int c=0 ; c<numChannels ; c++) sam-
    pleBuffer[c][offset] = readSample();
499.
500.                 offset++;
501.                 frameCounter++;
502.             }
503.
504.             return numFramesToRead;
505.         }
506.
507.         public int writeFrames(long[] sampleBuffer, int num-
    FramesToWrite) throws IOException, WavFileException
508.         {
509.             return writeFrames(sampleBuffer, 0, numFrame-
    sToWrite);
510.         }
511.
512.         public int writeFrames(long[] sampleBuffer, int off-
    set, int numFramesToWrite) throws IOException, WavFileException
```

```
513.        {
514.            if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
515.
516.            for (int f=0 ; f<numFramesToWrite ; f++)
517.            {
518.                if (frameCounter == numFrames) return f;
519.
520.                for (int c=0 ; c<numChannels ; c++)
521.                {
522.                    writeSample(sampleBuffer[offset]);
523.                    offset++;
524.                }
525.
526.                frameCounter++;
527.            }
528.
529.            return numFramesToWrite;
530.        }
531.
532.    public int writeFrames(long[][] sampleBuffer, int num-
      FramesToWrite) throws IOException, WavFileException
533.    {
534.        return writeFrames(sampleBuffer, 0, numFrame-
      sToWrite);
535.    }
536.
537.    public int writeFrames(long[][] sampleBuffer, int off-
      set, int numFramesToWrite) throws IOException, WavFileException
538.    {
539.        if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
540.
541.        for (int f=0 ; f<numFramesToWrite ; f++)
542.        {
543.            if (frameCounter == numFrames) return f;
544.
545.            for (int c=0 ; c<numChan-
      nels ; c++) writeSample(sampleBuffer[c][offset]);
546.
547.            offset++;
548.            frameCounter++;
549.        }
550.
551.        return numFramesToWrite;
552.    }
553.
554.    // Double
555.    // -----
556.    public int readFrames(double[] sampleBuffer, int num-
      FramesToRead) throws IOException, WavFileException
557.    {
558.        return readFrames(sampleBuffer, 0, numFrame-
      sToRead);
559.    }
560.
561.    public int readFrames(double[] sampleBuffer, int off-
      set, int numFramesToRead) throws IOException, WavFileException
```

```
562.        {
563.            if (ioState != IOState.READING) throw new IOException("Cannot read from WavFile instance");
564.
565.            for (int f=0 ; f<numFramesToRead ; f++)
566.            {
567.                if (frameCounter == numFrames) return f;
568.
569.                for (int c=0 ; c<numChannels ; c++)
570.                {
571.                    sampleBuffer[offset] = floatOffset + (double) readSample() / floatScale;
572.                    offset++;
573.                }
574.
575.                frameCounter++;
576.            }
577.
578.            return numFramesToRead;
579.        }
580.
581.        public int readFrames(double[][] sampleBuffer, int num-
      FramesToRead) throws IOException, WavFileException
582.        {
583.            return readFrames(sampleBuffer, 0, numFrame-
      sToRead);
584.        }
585.
586.        public int readFrames(double[][] sampleBuffer, int off-
      set, int numFramesToRead) throws IOException, WavFileException
587.        {
588.            if (ioState != IOState.READING) throw new IOException("Cannot read from WavFile instance");
589.
590.            for (int f=0 ; f<numFramesToRead ; f++)
591.            {
592.                if (frameCounter == numFrames) return f;
593.
594.                for (int c=0 ; c<numChannels ; c++) sam-
      pleBuffer[c][offset] = floatOffset + (double) readSam-
      ple() / floatScale;
595.
596.                offset++;
597.                frameCounter++;
598.            }
599.
600.            return numFramesToRead;
601.        }
602.
603.        public int writeFrames(double[] sampleBuffer, int num-
      FramesToWrite) throws IOException, WavFileException
604.        {
605.            return writeFrames(sampleBuffer, 0, numFrame-
      sToWrite);
606.        }
607.
608.        public int writeFrames(double[] sampleBuffer, int off-
      set, int numFramesToWrite) throws IOException, WavFileException
```

```
609.          {
610.              if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
611.
612.              for (int f=0 ; f<numFramesToWrite ; f++)
613.              {
614.                  if (frameCounter == numFrames) return f;
615.
616.                  for (int c=0 ; c<numChannels ; c++)
617.                  {
618.                      writeSample((long) (floatScale * (floatOffset + sampleBuffer[offset])));
619.                      offset++;
620.                  }
621.
622.                  frameCounter++;
623.              }
624.
625.              return numFramesToWrite;
626.          }
627.
628.          public int writeFrames(double[][] sampleBuffer, int num-
   FramesToWrite) throws IOException, WavFileException
629.          {
630.              return writeFrames(sampleBuffer, 0, numFrame-
   sToWrite);
631.          }
632.
633.          public int writeFrames(double[][] sampleBuffer, int off-
   set, int numFramesToWrite) throws IOException, WavFileException
634.          {
635.              if (ioState != IOState.WRITING) throw new IOException("Cannot write to WavFile instance");
636.
637.              for (int f=0 ; f<numFramesToWrite ; f++)
638.              {
639.                  if (frameCounter == numFrames) return f;
640.
641.                  for (int c=0 ; c<numChan-
   nels ; c++) writeSample((long) (floatScale * (floatOffset + sam-
   pleBuffer[c][offset])));
642.
643.                  offset++;
644.                  frameCounter++;
645.              }
646.
647.              return numFramesToWrite;
648.          }
649.
650.
651.          public void close() throws IOException
652.          {
653.              // Close the input stream and set to null
654.              if (iStream != null)
655.              {
656.                  iStream.close();
657.                  iStream = null;
658.              }
}
```

```
659.             if (oStream != null)
660.             {
661.                 // Write out anything still in the lo-
662.                 cal buffer
663.                 if (bufferPointer > 0) oS-
664.                     tream.write(buffer, 0, bufferPointer);
665.                 // If an extra byte is required for word align-
666.                 // ment, add it to the end
667.                 if (wordAlignAdjust) oStream.write(0);
668.                 // Close the stream and set to null
669.                 oStream.close();
670.                 oStream = null;
671.             }
672.             // Flag that the stream is closed
673.             ioState = IOState.CLOSED;
674.         }
675.     }
676.     public void display()
677.     {
678.         display(System.out);
679.     }
680.     public void display(PrintStream out)
681.     {
682.         out.printf("File: %s\n", file);
683.         out.printf("Channels: %d, Frames: %d\n", numChan-
684.             nels, numFrames);
685.         out.printf("IO State: %s\n", ioState);
686.         out.printf("Sam-
687.             ple Rate: %d, Block Align: %d\n", sampleRate, blockAlign);
688.         out.printf("Valid Bits: %d, Bytes per sam-
689.             ple: %d\n", validBits, bytesPerSample);
690.     }
691.     public static void main(String[] args)
692.     {
693.         if (args.length < 1)
694.         {
695.             System.err.println("Must supply filename");
696.             System.exit(1);
697.         }
698.         try
699.         {
700.             for (String filename : args)
701.             {
702.                 WavFile readWavFile = open-
703.                     WavFile(new File(filename));
704.                     readWavFile.display();
705.                     long numFrames = readWavFile.getNum-
706.                         Frames();
707.                     int numChannels = readWavFile.getNumChan-
708.                         nels();
```

```

708.                     int validBits = readWavFile.getValid-
    Bits();
709.                     long sampleRate = readWavFile.get-
    SampleRate();
710.
711.                     WavFile writeWavFile = new-
    WavFile(new File("out.wav"), numChannels, numFrames, valid-
    Bits, sampleRate);
712.
713.                     final int BUF_SIZE = 5001;
714.
715.                     //                     int[] buffer = new int[BUF_SIZE * numChan-
    nels];
716.                     //                     long[] buffer = new long[BUF_SIZE * numChan-
    nels];
717.                     double[] buffer = new double[BUF_SIZE * num-
    Channels];
718.
719.                     int framesRead = 0;
720.                     int framesWritten = 0;
721.
722.                     do
723.                     {
724.                         framesRead = readWavFile.read-
    Frames(buffer, BUF_SIZE);
725.                         framesWritten =
    writeWavFile.writeFrames(buffer, BUF_SIZE);
726.                         Sys-
    tem.out.printf("%d %d\n", framesRead, framesWritten);
727.                         }
728.                     while (framesRead != 0);
729.
730.                     readWavFile.close();
731.                     writeWavFile.close();
732.                 }
733.
734.                 WavFile writeWavFile = new-
    WavFile(new File("out2.wav"), 1, 10, 23, 44100);
735.                 double[] buffer = new double[10];
736.                 writeWavFile.writeFrames(buffer, 10);
737.                 writeWavFile.close();
738.             }
739.             catch (Exception e)
740.             {
741.                 System.err.println(e);
742.                 e.printStackTrace();
743.             }
744.         }
745.     }

```

WavFileException.java

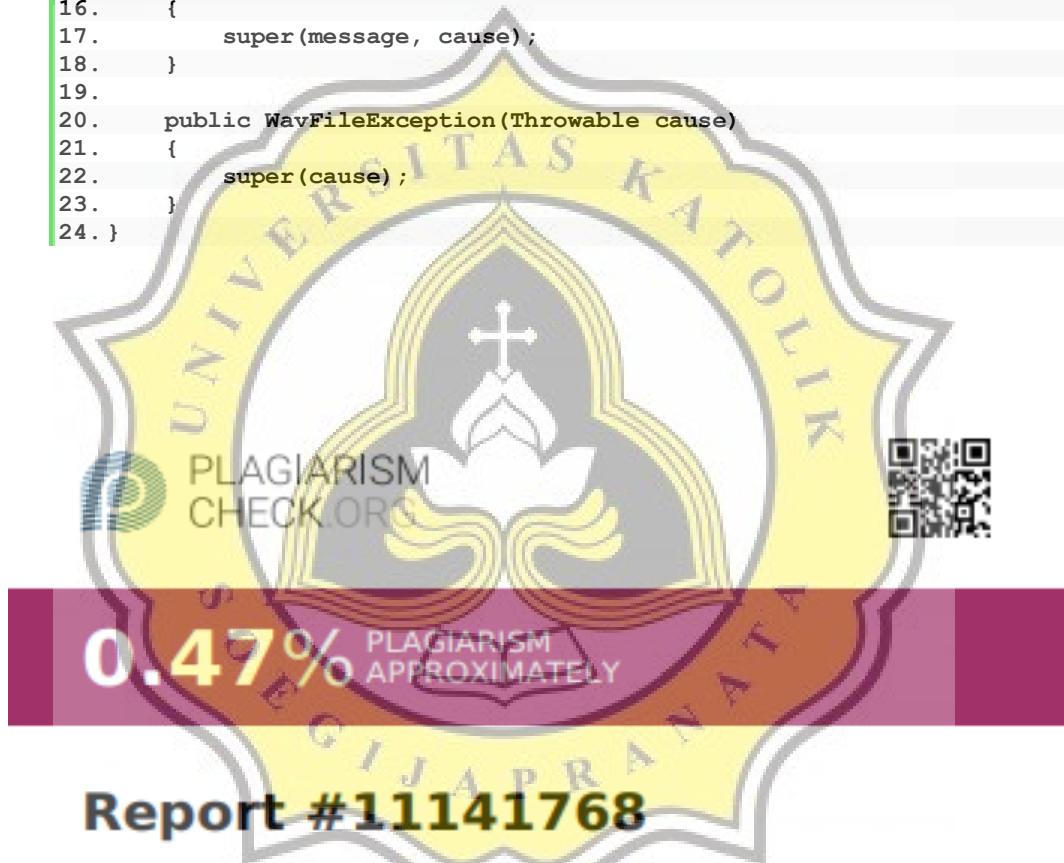
```

1. package com.appdev.example;
2.

```

```

3. public class WavFileException extends Exception
4. {
5.     public WavFileException()
6.     {
7.         super();
8.     }
9.
10.    public WavFileException(String message)
11.    {
12.        super(message);
13.    }
14.
15.    public WavFileException(String message, Throwable cause)
16.    {
17.        super(message, cause);
18.    }
19.
20.    public WavFileException(Throwable cause)
21.    {
22.        super(cause);
23.    }
24.}
```



IntroductionBackgroundWAV Audio File (WAVEform Audio) is a standard audio file format used by windows as an uncompressed audio file format. WAV audio files use the RIFF structure. There are 3 parts in the WAV audio file format, namely RIFF, FMT and data. Each part has its own function. RIFF saves the identity, size and format of the chunk. Whereas FMT stores audio information and sub-chunk data. Data stores audio size information and audio data. Because WAV audio files are not compressed, encryption data can be added to the file without damaging the file structure. The encryption process utilizes a combination of Base64 and LSB (Least Significant Bit) algorithms. The workings of this algorithm are that data in the form of messages will be encrypted using the Base64 algorithm. The output of Base64 encryption will be used as input for the LSB algorithm. In the process of inserting a message into a WAV audio file, the RIFF and FMT sections were not modified. Modifications are made in the part of the data where the encrypted message will be inserted. After the data has been modified, the RIFF and fmt sections will be merged back with the data where the encrypted message has been inserted. The decryption process only needs to do the opposite of the combination of the 2 algorithms. Based on the discussions above, this project discusses research on implementing a combination