AUDIO RECORDER AND PROCESSING FOR ADULTS WITH AUTISM

Software Graphical User Interface for Audio Recording

Design By
Adithi Shankaranarayan Athreya
Winter2015
Under the Guidance of
Dr. Radhika Grover

Table of Contents

Overall Project Objective

To help Autistic adults/students record their tasks and important information in their own voice and set reminders, to be played back by the hand-held device at a later time during the day.

Issue in Hand

Adults with Autism have major issues while communicating in social situations, or expressing their emotions, or following their routine, or displaying their interests and disinterests. It is important for the caretakers to understand them as much as they understand themselves.

Proposed Solution

Through this effort, we propose to introduce a method, a system where these adults have an opportunity to express their feelings, setting reminders for their routine tasks in their own voice. These recordings shall be stored in an SD card, which is then used in a hand-held device (or portable device). At the time of recording, they can schedule these recordings to be played at a particular time of the day. The device then sets an alarm for the scheduled time and plays it back at the set time.
There are two parts to this proposed solution.

- Recording the audio and setting the play back time
- Processing the audio, set the alarm for playback and play it back

Scope of This Project

Main focus of this project was to design and develop software Graphical User Interface (GUI) that is easy and convenient to operate, while functioning to be a basic but necessary audio recorder. Scope of this project was as follows.

- √ Learn and understand Java Sound, its features and functionality
- √ Design a simple, easy to use GUI
- √ Implement the GUI with required functionalities
- √ Integrate the GUI with audio recorder to record, stop recording, play and stop playing audio files
- √ Implement a specific file saving technique to be used by the hardware, hand-help (portable) device, to be able to playback.

Technology and Tools Used

As it was a complete software project, Java swing and Java Sound frameworks were majorly used. Eclipse was used to design and develop the program in Java.

Implementation and Functionality

Following are step by step this program/software was designed and developed.

Step 1. Audio recorder format specifications

The required audio specification were an audio to be recorded at a sampling frequency of 12KHz, with Unsigned 8 bit WAV format, using mono channel, saved in little Endian format.

```
AudioFormat getAudioFormat() {
            float sampleRate = 12000;
            int sampleSizeInBits = 8;
            int channels = 1;
            boolean signed = false;
            boolean bigEndian = false;
            return new AudioFormat(sampleRate, sampleSizeInBits,
                        channels, signed, bigEndian);
}
```

Step 2. Simple audio recorder design

Using the above audio format specifications, design a console-input audio recorder (just to start with), to see and understand Java Sound functionalities.

```
//path of wav file
private File wavFile = new
File("/Users/adithiathreya/Desktop/RecordAudio.wav");
// format of audio file
AudioFileFormat.Type fileType = AudioFileFormat.Type.WAVE;
// the line from which audio data is captured
TargetDataLine line;
```

```
public static void main(String[] args) throws IOException {
        final TestSoundDirectWithThread recorder = new
TestSoundDirectWithThread();
        System.out.println("Enter 1 to start recording");
        @SuppressWarnings("resource")
           Scanner user_input = new Scanner( System.in );
        int ui = user_input.nextInt();
        Runnable run1 = new Runnable() {
             public void run() {
                 try {
                     recorder.start();
                 } catch (IOException e) {
                     e.printStackTrace();
                 }
             }
        };
        Thread recordingThread = new Thread(run1);
        recordingThread.start();
        try {
           Thread.sleep(5000);
        } catch (InterruptedException e) {
           e.printStackTrace();
        }
        recorder.finish();
}
```

```
public void start() throws IOException{
      try {
            AudioFormat format = getAudioFormat();
            DataLine.Info info = new
DataLine.Info(TargetDataLine.class, format);
            // checks if system supports the data line
            if (!AudioSystem.isLineSupported(info)) {
                  System.out.println("Line not supported");
                  System.exit(0);
            }
            line = (TargetDataLine) AudioSystem.getLine(info);
            line.open(format);
            line.start();   // start capturing

            System.out.println("Start capturing...");
            AudioInputStream ais = new AudioInputStream(line);

            System.out.println("Start recording...");
            AudioSystem.write(ais, fileType, wavFile);

      } catch (LineUnavailableException ex) {
            ex.printStackTrace();
      }
}
public void finish() {
      line.stop();
      line.close();
      System.out.println("Finished");
}
```

Step 3. GUI design

Set the Frame and Panels in Java GUI using JAVA Swing framework and initialize them.

```
    private TestSound recorder = new TestSound();
    private RecordTimer timer;

    private JFrame recorderFrame = new JFrame(); //main GUI frame
    private JPanel titlePanel = new JPanel(); //title panel
    private JPanel logoPanel = new JPanel(); //Logo panel
    private JPanel recordingPanel = new JPanel(); //Recording panel
    private JPanel projectTitlePanel = new JPanel(); //Project title
    panel
    private JPanel buttonsPanel = new JPanel(); //Buttons panel
    private JPanel timeDisplayPanel = new JPanel(); //Time display panel
    private JPanel fileLocationPanel = new JPanel(); //File Location
    setting panel
    private JPanel filesListingPanel = new JPanel(); //Files listing
    panel
    private JPanel fileListPanel = new JPanel(); //List of files panel
    private JPanel playbackTimePanel = new JPanel();//Playback time
    choosing panel

    private static final Color scuBackgroundColor =
    Color.decode("#93191B");
    //    private static final Color scuForegroundColor =
    Color.decode("#D2C599");
    private BufferedImage logoPicture = ImageIO.read(new
    File("iconANDLogo/scu-logo-seal.png"));
    private BufferedImage logoTitlePicture = ImageIO.read(new
    File("iconANDLogo/scu-logo-csts.png"));

    private JLabel projectTitle = new JLabel("Audio Recorder");
    private JButton recordStopButton = new JButton();
    private JButton playPauseButton = new JButton();
    private JLabel recordTimeLabel = new JLabel();
    private ImageIcon recordIcon = new
    ImageIcon("iconANDLogo/recorder.png");
    private ImageIcon playIcon = new ImageIcon("iconANDLogo/play.png");
    private ImageIcon stopIcon = new ImageIcon("iconANDLogo/stop.png");
    private ImageIcon pauseIcon = new ImageIcon("iconANDLogo/pause.png");
    private JLabel fileLocationLabel = new JLabel("File Location:");
    private JTextField fileLocationTextField = new JTextField();
    private JButton changeLocationButton = new JButton("Change
    Location");
    private DefaultListModel<String> model;
    private JList<String> fileList;
```

```
private JLabel playbackLabel = new JLabel("Playback Time");
private JSpinner timeSpinner = new JSpinner( new SpinnerDateModel()
);
private JSpinner.DateEditor timeEditor = new
JSpinner.DateEditor(timeSpinner, "hh:mm a");
private JLabel timeErrorLabel = new JLabel();
private String saveTime = "";

private boolean isRecording = false;
private boolean isPlaying = false;
private JFrame optionPaneFrame;
private JTextField playbackTimerTextField;
private String saveFilePath;
private String playFileName;
private Thread playbackThread;
```

Initializing the Main Recorder Frame and setting the skeleton Panels

```
super("Audio Recorder");
recorderFrame.setSize(1000, 1000);
recorderFrame.setLayout(new BorderLayout());
titlePanel.setPreferredSize(new Dimension(1000, 100));
recorderFrame.add(titlePanel, BorderLayout.PAGE_START);
recordingPanel.setPreferredSize(new Dimension(1000, 250));
recordingPanel.setBorder(BorderFactory.createLineBorder(Color.black));
recorderFrame.add(recordingPanel, BorderLayout.CENTER);
filesListingPanel.setPreferredSize(new Dimension(1000, 400));
filesListingPanel.setBorder(BorderFactory.createLineBorder(Color.black
));
recorderFrame.add(filesListingPanel, BorderLayout.PAGE_END);
```

Initializing the Title Panel

This Panel has the SCU logo embedded into it.

```
titlePanel.setBackground(scuBackgroundColor);
titlePanel.setLayout(new FlowLayout(FlowLayout.LEADING));
logoPanel.setPreferredSize(new Dimension(600,100));
JLabel logoPicLabel = new JLabel(new ImageIcon(logoPicture));
logoPanel.add(logoPicLabel);
JLabel logoTitlePicLabel = new JLabel(new
ImageIcon(logoTitlePicture));
logoPanel.add(logoTitlePicLabel);
logoPanel.setOpaque(false);
titlePanel.add(logoPanel);
```

Initializing the Button Panel

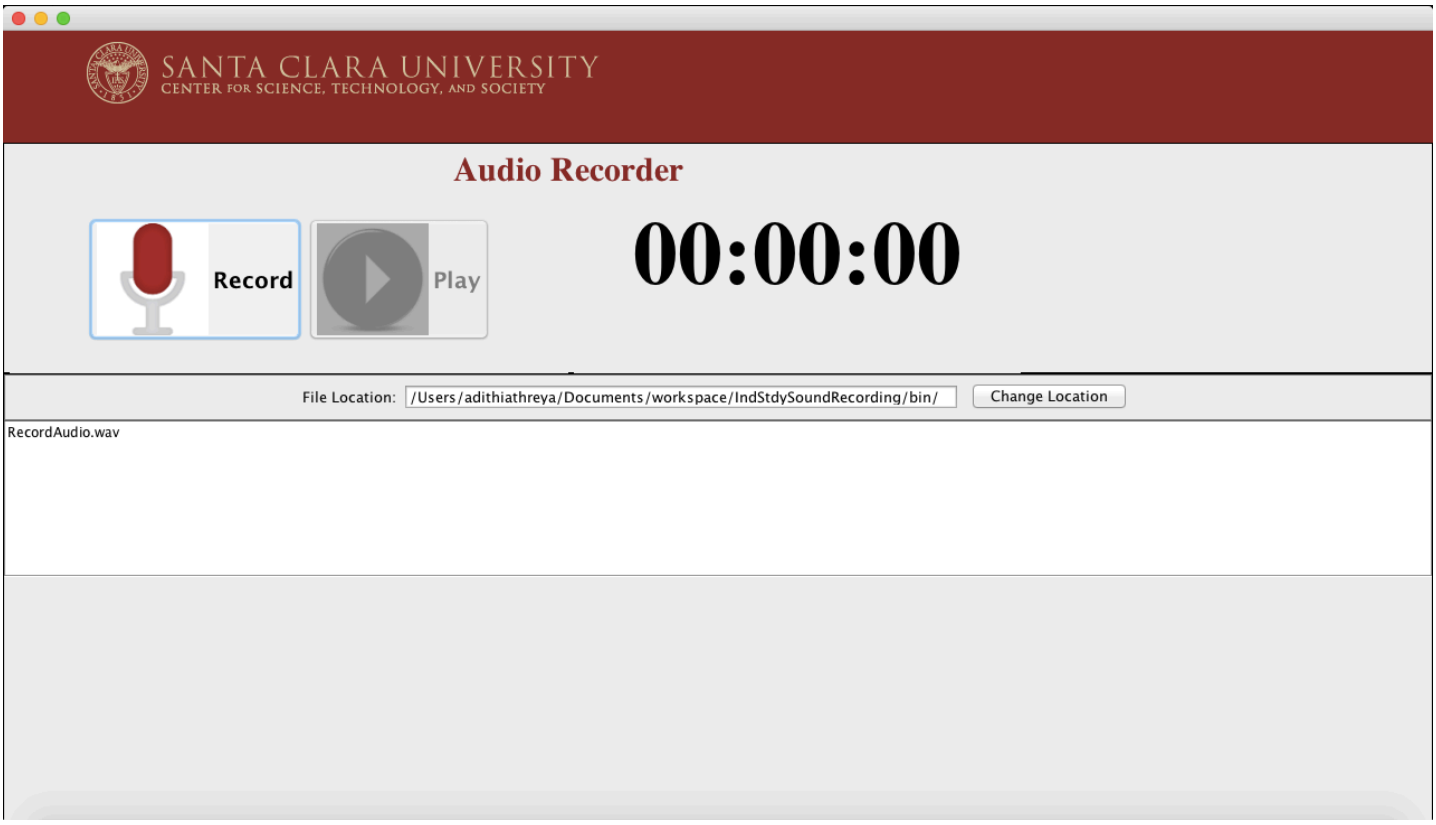
This Panel is the heart of the GUI, where the recording, stopping, playing and pausing buttons are embedded, along with a timer to check the passage of time either while recording or while playing an audio file.

```
recordingPanel.setLayout(new FlowLayout(FlowLayout.LEADING));
projectTitlePanel.setPreferredSize(new Dimension(1000,50));
projectTitle.setFont(new Font("Serif", Font.BOLD, 30));
projectTitle.setForeground(scuBackgroundColor);;
projectTitlePanel.add(projectTitle);
projectTitlePanel.setOpaque(false);
recordingPanel.add(projectTitlePanel);
buttonsPanel.setPreferredSize(new Dimension(500,150));
recordStopButton.setText("Record");
recordStopButton.setFont(new Font("Sans", Font.BOLD, 20));
recordIcon.setImage(recordIcon.getImage().getScaledInstance(100, 100,
java.awt.Image.SCALE_SMOOTH ));
recordStopButton.setIcon(recordIcon);
playPauseButton.setText("Play");
playPauseButton.setFont(new Font("Sans", Font.BOLD, 20));
playIcon.setImage(playIcon.getImage().getScaledInstance(100, 100,
java.awt.Image.SCALE_SMOOTH ));
playPauseButton.setIcon(playIcon);
playPauseButton.setEnabled(false);
buttonsPanel.add(recordStopButton);
buttonsPanel.add(playPauseButton);
recordingPanel.add(buttonsPanel);
timeDisplayPanel.setPreferredSize(new Dimension(400,150));
recordTimeLabel.setText("00:00:00");
recordTimeLabel.setFont(new Font("Serif", Font.BOLD, 80));
timeDisplayPanel.add(recordTimeLabel);
recordingPanel.add(timeDisplayPanel);
recordStopButton.addActionListener(this);
playPauseButton.addActionListener(this);
```

Initializing the File List display Panel

This Panel is used to display the list of files in a particular Directory, so as to be played by the Audio Recorder software. After initializing all the 2 Panels, the Frame is validated to perform the default close operation.

```
filesListingPanel.setLayout(new BorderLayout());
fileLocationPanel.setLayout(new FlowLayout());
fileLocationPanel.setBorder(BorderFactory.createLineBorder(Color.GRAY)
);
fileLocationPanel.add(fileLocationLabel, FlowLayout.LEFT);
fileLocationTextField.setPreferredSize(new Dimension(500, 20));
fileLocationTextField.setText(this.getClass().getClassLoader().getReso
urce("").getPath());
fileLocationPanel.add(fileLocationTextField);
fileLocationPanel.add(changeLocationButton);
filesListingPanel.add(fileLocationPanel, BorderLayout.PAGE_START);
fileListPanel.setPreferredSize(new Dimension(900, 350));
fileListPanel.setLayout(new BorderLayout());
filesListingPanel.add(fileListPanel, BorderLayout.CENTER);
this.displayFileList();
changeLocationButton.addActionListener(this);

recorderFrame.setVisible(true);
recorderFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
recorderFrame.setLocationRelativeTo(null);
```



Step 4. Add actionListeners to the Buttons

Next step is to add functionalities to the buttons Record, Stop, Play, Pause and Change Location.

```
@Override
public void actionPerformed(ActionEvent e) {
      if (e.getSource() == recordStopButton) {
            if (!isRecording) {
                  startRecording();
            } else {
                  if (isPlaying == false) {
                        stopRecording();
                  } else {
                        stopPlaying();
                  }
            }
      } else if (e.getSource() == playPauseButton) {
            if (!isPlaying) {
                  startPlaying();
            } else {
                  if(isRecording == false) {
                        pausePlaying();
                  } else {
                        pauseRecording();
                  }
            }
      } else if (e.getSource() == changeLocationButton) {
            fileListPanel.removeAll();
            JFileChooser chooser = new JFileChooser();
            chooser.setCurrentDirectory(new java.io.File("."));
chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
      chooser.setAcceptAllFileFilterUsed(false);
      if (chooser.showOpenDialog(this) ==
JFileChooser.APPROVE_OPTION) {
            System.out.println("getCurrentDirectory(): "
                        +  chooser.getCurrentDirectory());
fileLocationTextField.setText(chooser.getCurrentDirectory().getPath())
;
            displayFileList();
            fileListPanel.revalidate();
            fileListPanel.updateUI();
            recorderFrame.pack();
      }
      else {
            System.out.println("No Selection ");
            }
      }
}
```

Implement startRecording() method

On the button action of Record Button, a thread spawns to start recording the audio, while the main thread is looking out for stopping the recording. Meantime, the timer also starts, indicating the time elapsed in the recording.

```
private void startRecording() {
	Thread recordThread = new Thread(new Runnable() {
		@Override
		public void run() {
			isRecording = true;
			isPlaying = false;
			recordStopButton.setText("Stop");
		recordStopButton.setFont(new Font("Sans", Font.BOLD, 20));

	stopIcon.setImage(stopIcon.getImage().getScaledInstance(100, 100,
java.awt.Image.SCALE_SMOOTH ));
		recordStopButton.setIcon(stopIcon);
		playPauseButton.setText("Pause");
		playPauseButton.setFont(new Font("Sans", Font.BOLD, 20));

	pauseIcon.setImage(pauseIcon.getImage().getScaledInstance(100,
100, java.awt.Image.SCALE_SMOOTH ));
			playPauseButton.setIcon(pauseIcon);
			playPauseButton.setEnabled(true);
			recorder.start();
		}
	});
	recordThread.start();
	timer = new RecordTimer(recordTimeLabel);
	timer.start();
}
```

Implement stopRecording() method

Once the Stop button is pressed, current recording session stops, thus allowing the user to either save or discard the recording, with the use of a JOptionPane. Timer also stops.

```
private void stopRecording() {
      isRecording = false;
      isPlaying = false;
      recordStopButton.setText("Record");
      recordStopButton.setIcon(recordIcon);
      playPauseButton.setText("Play");
      playPauseButton.setIcon(playIcon);
      playPauseButton.setEnabled(false);
      timer.cancel();
      recorder.finish();
      int n = JOptionPane.showConfirmDialog(optionPaneFrame,
                  "Do you want to save this recording?",
                  "Save File",
                  JOptionPane.YES_NO_OPTION);
      if (n == JOptionPane.YES_OPTION) {
            System.out.println("Saving the recording");
            playbackTimeSelection();
      } else if (n == JOptionPane.NO_OPTION) {
            System.out.println("Cancel saving");
            timer.reset();
      } else {
            System.out.println("Do nothing");
            timer.reset();
      }
}
```

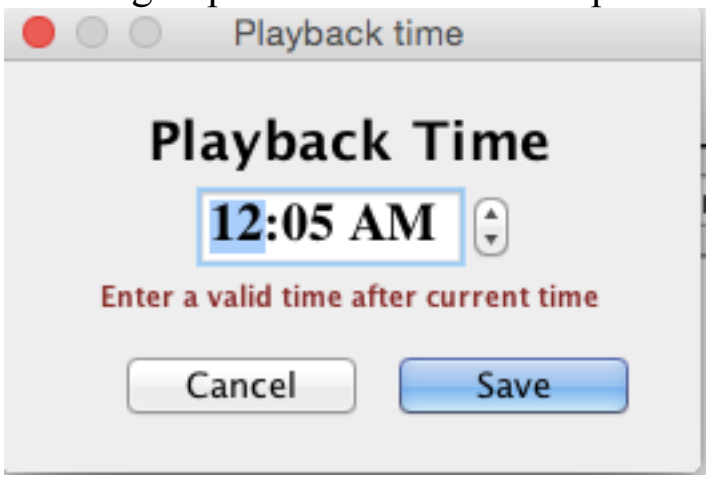
Playback time selection

If the user chooses to proceed on saving the file, another JOptionPane pops-up with a timeSpinner to choose the appropriate time for the audio to be played back. The time chosen for playback has to be a time after the current time.

```
private void playbackTimeSelection() {
      Object[] options = {"Save",
      "Cancel"};
      int n1 = JOptionPane.showOptionDialog(null,
                  getOptionPanel(),
                  "Playback time",
                  JOptionPane.YES_NO_OPTION,
                  JOptionPane.PLAIN_MESSAGE,
                  null, options, options[0]);

      if (n1 == JOptionPane.YES_OPTION && saveTime != "") {
            System.out.println("Playback time choose");
            saveFile();
      } else if (n1 == JOptionPane.YES_OPTION && saveTime == "") {
            System.out.println("Playback time selection error");
            playbackTimeSelection();
      } else if (n1 == JOptionPane.NO_OPTION) {
            System.out.println("Cancel saving");
            timer.reset();
      } else {
            System.out.println("Do nothing");
            timer.reset();
      }
}
```

```
private JPanel getOptionPanel() {
	playbackTimePanel.setPreferredSize(new Dimension(100,80));
	playbackTimePanel.add(playbackLabel);
	playbackLabel.setFont(new Font("Sans", Font.BOLD, 20));
	timeSpinner.setPreferredSize(new Dimension(120,30));
	playbackTimerTextField = timeEditor.getTextField();
	playbackTimerTextField.setFont(new Font("Serif", Font.BOLD,
20));
	timeSpinner.setEditor(timeEditor);
	timeSpinner.setValue(new Date());
	playbackTimePanel.add(timeSpinner);
	playbackTimePanel.add(timeErrorLabel);
	timeErrorLabel.setVisible(false);
	timeSpinner.addChangeListener(new ChangeListener() {
		@Override
		public void stateChanged(ChangeEvent e) {
			Date currentTime = new Date();
			@SuppressWarnings("deprecation")
			int curTime = currentTime.getHours()*60 +
currentTime.getMinutes();
			@SuppressWarnings("deprecation")
			int spnTime =
((Date)timeSpinner.getValue()).getHours()*60 +
((Date)timeSpinner.getValue()).getMinutes();
			if(spnTime <= curTime) {
				System.out.println("Bad Time");
				saveTime = "";
	timeErrorLabel.setText("Enter a valid time after current time");
	timeErrorLabel.setFont(new Font("Sans", Font.BOLD, 10));
	timeErrorLabel.setForeground(scuBackgroundColor);
				timeErrorLabel.setVisible(true);
				playbackTimePanel.repaint();
			} else {
				System.out.println("Good time");
				saveTime = String.format("%04d", spnTime);
				timeErrorLabel.setVisible(false);
				System.out.println(saveTime);
			}
		}
	});
	return playbackTimePanel;
}
```

If an invalid time is entered, or the time is entered in a wrong format or a time entered is before or at current time, an error message is displayed and thus not allowing to proceed to the next step.

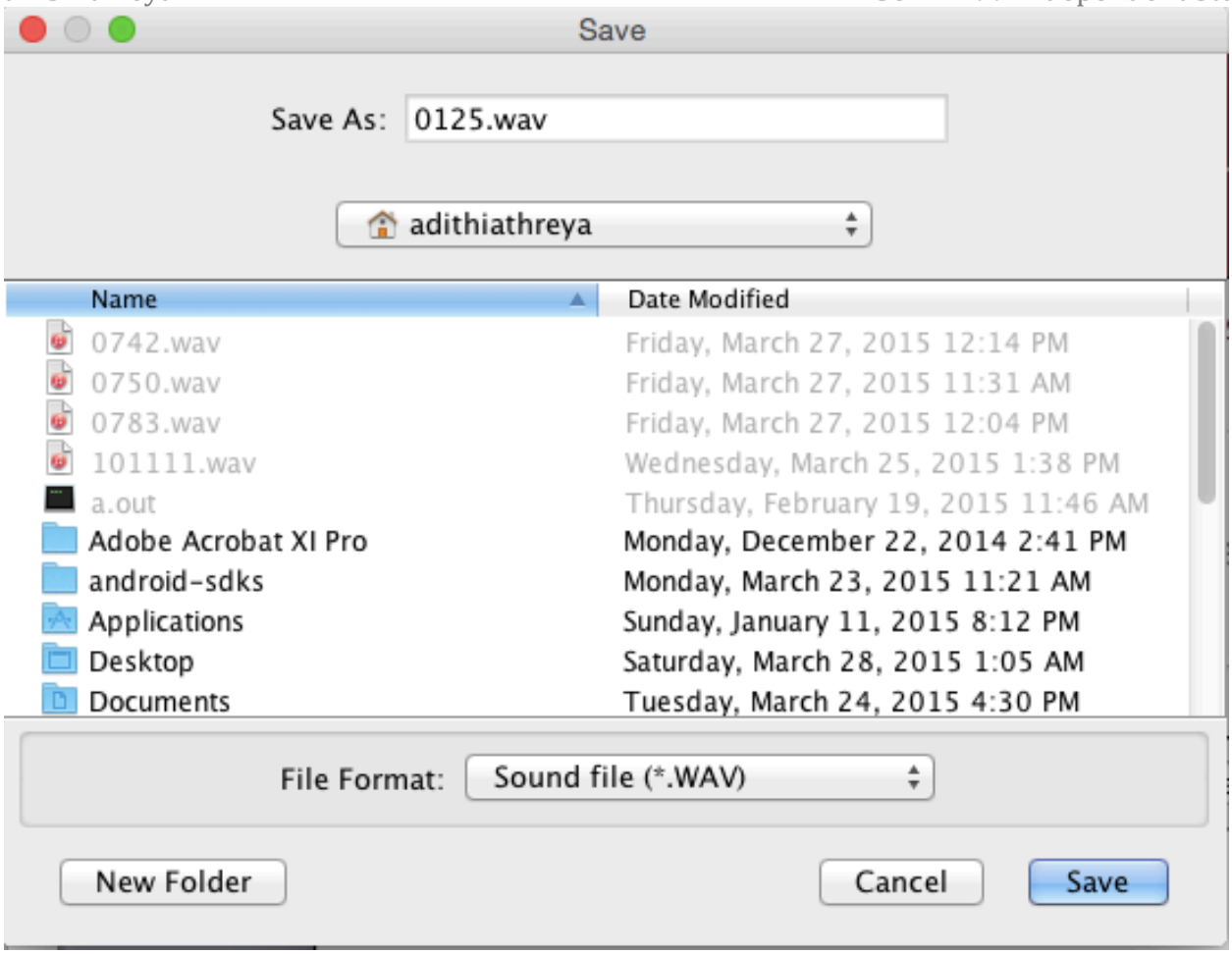


Save file with specific file name in a Directory chosen by the user

Upon entering the valid playback time, another pop-up window appears allowing the user to choose the directory in which the file needs to be saved. Name of file is made un-editable by giving a name to be a calculated integer of playback time.

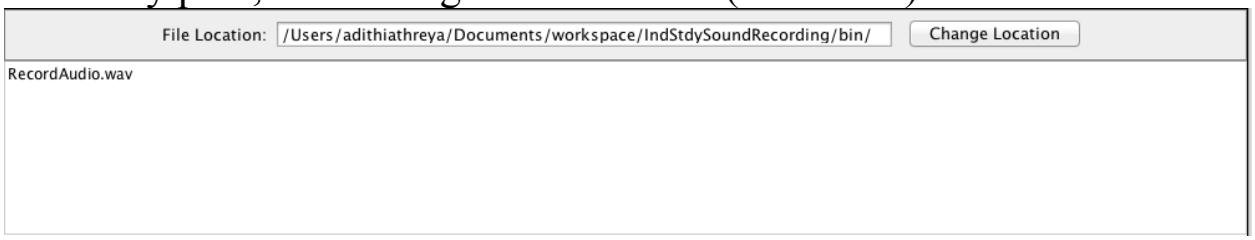
```
int spnTime = ((Date)timeSpinner.getValue()).getHours()*60 +
((Date)timeSpinner.getValue()).getMinutes();
saveTime = String.format("%04d", spnTime);
```

```
  private void saveFile() {
        JFileChooser fileChooser = new JFileChooser();
        FileFilter wavFilter = new FileFilter() {
              @Override
              public String getDescription() {
                    return "Sound file (*.WAV)";
              }

              @Override
              public boolean accept(File file) {
                    if (file.isDirectory()) {
                          return true;
                    } else {
                          return
  file.getName().toLowerCase().endsWith(".wav");
                    }
              }
        };
        fileChooser.setFileFilter(wavFilter);
        fileChooser.setAcceptAllFileFilterUsed(false);
        fileChooser.setSelectedFile(new File(saveTime + ".wav"));
        disableTextField(fileChooser.getComponents());
        int userChoice = fileChooser.showSaveDialog(this);
        if (userChoice == JFileChooser.APPROVE_OPTION) {
              saveFilePath =
  fileChooser.getSelectedFile().getAbsolutePath();
              if (!saveFilePath.toLowerCase().endsWith(".wav")) {
                    saveFilePath += ".wav";
              }
              File wavFile = new File(saveFilePath);
              try {
                    recorder.save(wavFile);

  JOptionPane.showMessageDialog(AudioNewGUI.this,
              "Saved recorded sound to:\n" + saveFilePath);

              } catch (IOException ex) {
  JOptionPane.showMessageDialog(AudioNewGUI.this, "Error",
              "Error saving to sound file!",
                          JOptionPane.ERROR_MESSAGE);
                    ex.printStackTrace();
              }
        }
  }
```

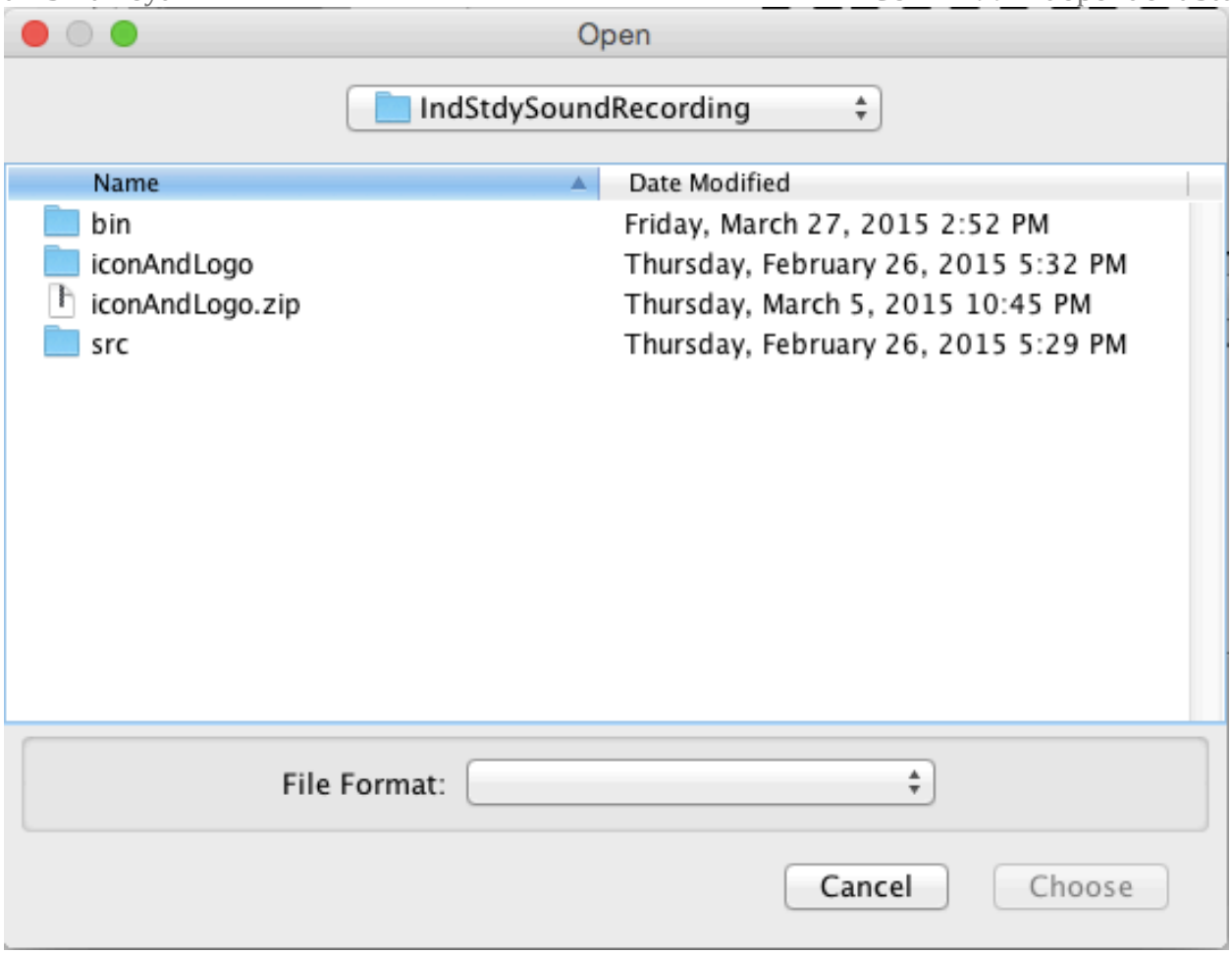


Step 5. File list display

As added functionality, this software program also has option to play the recorded or any .wav audio on the computer. The textfield has default current Directory path, thus listing the audio files (.wav files) in the current Directory.



In case one wishes to change the folder/Directory location to search for desired audio files, they can do so by pressing the Change Location Button. ActionListener for this button pops-up another JFileChooser window, which only allows choosing Directories.


Open
IndStdySoundRecording
Name
Date Modified
bin
Friday, March 27, 2015 2:52 PM
iconAndLogo
Thursday, February 26, 2015 5:32 PM
iconAndLogo.zip
Thursday, March 5, 2015 10:45 PM
src
Thursday, February 26, 2015 5:29 PM
File Format:
Cancel
Choose

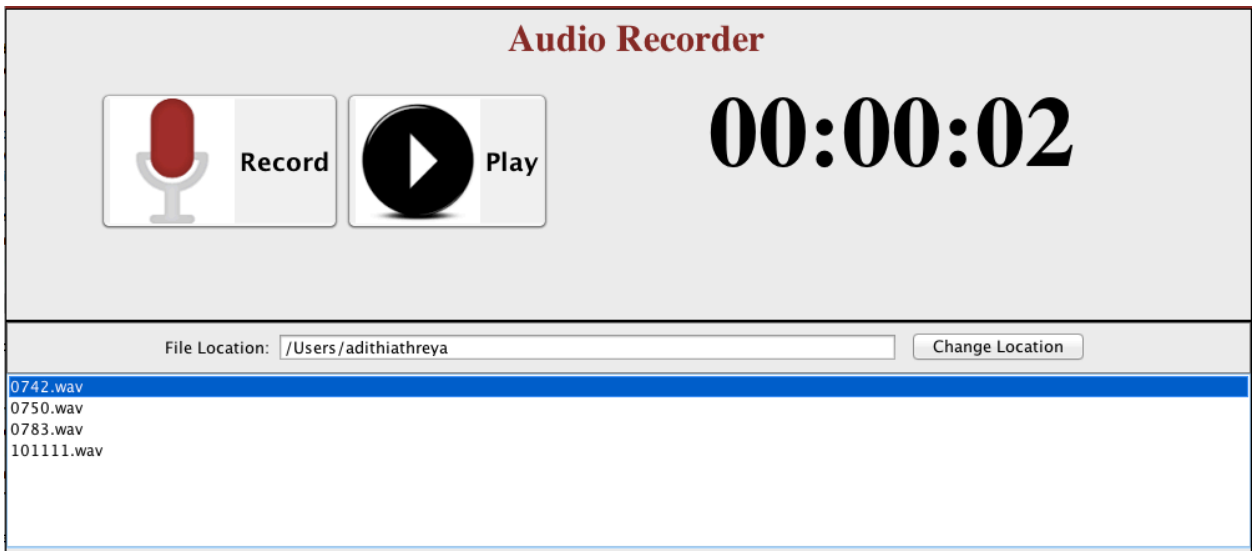
Selection of a new Directory location, lists all the .wav files in that Folder.

```
private void displayFileList() {
      fileList = new JList<>(model = new DefaultListModel<String>());
      File path = new File(fileLocationTextField.getText());
      File[] listOfFiles = path.listFiles();
      if (listOfFiles != null) {
            for(File f : listOfFiles) {
                  if(f.getName().toLowerCase().endsWith(".wav"))
                        model.addElement(f.getName());
            }
      }
      fileListPanel.add(new JScrollPane(fileList),
BorderLayout.PAGE_START);
      fileList.addListSelectionListener(new ListSelectionListener() {

            @Override
            public void valueChanged(ListSelectionEvent e) {
                  if (!e.getValueIsAdjusting()) {
                        playFileName = (new
StringBuilder().append(fileLocationTextField.getText()).append("/").ap
pend(fileList.getSelectedValue().toString())).toString();
                        playPauseButton.setEnabled(true);
                        recordStopButton.setEnabled(true);
                        recordStopButton.setText("Record");
                        recordStopButton.setIcon(recordIcon);
                        playPauseButton.setText("Play");
                        playPauseButton.setIcon(playIcon);
                  }
            }
      });
}
```

Implement startPlaying() method

On clicking any of the files from the list, enables it to be played by the Audio Player. Playing an audio also spawns a thread, while the main thread waits for the program to stop or pause playing.

```
private void startPlaying() {
        isRecording = false;
        isPlaying = true;
        timer = new RecordTimer(recordTimeLabel);
        timer.start();
        playbackThread = new Thread(new Runnable() {

            @Override
            public void run() {
                try {
                    recordStopButton.setText("Stop");
                    recordStopButton.setIcon(stopIcon);
                    playPauseButton.setText("Pause");
                    playPauseButton.setIcon(pauseIcon);

                    recorder.play(playFileName);
                    timer.reset();

                } catch (UnsupportedAudioFileException ex) {
                    ex.printStackTrace();
                } catch (LineUnavailableException ex) {
                    ex.printStackTrace();
                } catch (IOException ex) {
                    ex.printStackTrace();
                }

            }
        });

        playbackThread.start();
}
```



Implement stopPlaying()

Stopping the playing audio, just resets everything to initial conditions and interrupting the playing Thread.

```
private void stopPlaying() {
      isRecording = false;
      isPlaying = false;
      timer.reset();
      timer.interrupt();
      recordStopButton.setText("Record");
      recordStopButton.setIcon(recordIcon);
      playPauseButton.setText("Play");
      playPauseButton.setIcon(playIcon);
      playPauseButton.setEnabled(false);
      recorder.stop();
      playbackThread.interrupt();
}
```

Step 6. Update the simple audio recording functionality

Upon implementing all the GUI functionalities, the simple audio recording program also needs to be updated with playback functionality and removing the console input recording functionality.

```
public void play(String audioFilePath) throws
UnsupportedAudioFileException,
IOException, LineUnavailableException {
      File audioFile = new File(audioFilePath);
      AudioInputStream audioStream = AudioSystem
                  .getAudioInputStream(audioFile);
      AudioFormat format = audioStream.getFormat();
      DataLine.Info info = new DataLine.Info(Clip.class, format);
      Clip audioClip = (Clip) AudioSystem.getLine(info);
      audioClip.addLineListener(this);
      audioClip.open(audioStream);
      audioClip.start();
      playCompleted = false;
      while (!playCompleted) {
            // wait for the playback completes
            try {
                  Thread.sleep(1000);
            } catch (InterruptedException ex) {
                  ex.printStackTrace();
                  if (isStopped) {
                        audioClip.stop();
                        break;
                  }
            }
      }
      audioClip.close();
}
```

Result and Observations

Initial project objective of an audio recorder with recording, saving and playing capabilities are met. Testing of the recorded audio in the handheld (portable) device is also successful. Design of the GUI is simple, easy to use and basic. Some of the observations along the course of this project are as follows.

- Recording an audio without threads is not a good recording as the function for recording has to be constantly interrupted for checking if there is an input for stopping the recording, thus resulting in a non-continuous audio.
- Initial audio format specification were 16KHz-sampling frequency, with Signed 16 bit wav format with stereo input for recording and saved in Big Endian format. Hardware unit for the handheld (portable) device CC3200 had audio specifications of playing the audio with mono channel, Unsigned 8 bit, saved in Little Endian format. We also observed that, an audio recorded at the sampling rate of 12KHz gave a clearer and audible sound while being played by the CC3200.
- Recording the audio using a small external microphone produced a good quality, less noisy audio, when compared to recording it using the in-built microphone of the computer.

Scope for Future Work

Although the initial functionality of the software is neat and complete, there are a few areas that can be improved with this software project. Here are some of the ideas.

- Implement pause functionality during recording and playing, separately.
- Display the list of files with better looks and feel, where its details are also displayed. For example, display the list with filename, icon, size of the file, length of the audio and a small description about the audio.
- Add a waveform of the audio being recorded/played to the GUI, to give a fancier look.
- Implement the file saving technique to be able to set alarm for future days instead of 24 hours.
- Add a help button to display instructions on how to use the audio recorder.

References

[1] Grover, Chris, and Radhika S. Grover. *Programming with Java: A Multimedia Approach*. Jones & Bartlett Publishers, 2011.