IT 114 - ADVANCED PROGRAMMING FOR INFORMATION TECHNOLOGY

Programming Assignment 3: Audio Player

Author: Jennifer Soh

GitHub Repository: https://github.com/00savethepandas/assignment-3.git

1. Formulating the Problem

1.1 Assignment Description

Design and implement a Java applet that plays an audio file. The player should provide three control buttons (Play, Loop, and Stop). The Play button will play the song, the Loop button will loop the song, and the Stop button will stop the song.

1.2 Verbalization

What is the goal?

To create an audio player applet with control buttons that will play, loop, and stop a song.

What are the givens?

The given is that access to a particular song is provided through a Java Applet GUI and the control types provided to the user (play, stop, or loop the song). The song will stop playing if the window is closed.

What are the unknowns?

The unknowns are what controls the user will select, be it play, stop, or loop.

1.3 Information Elicitation

Goal

Provide a GUI for the user to control the playing, looping, or stopping of a song through button event objects.

Givens

The givens are the ways in which the user can control the song. Buttons are provided so that the user can play, loop, or stop the song. The other givens are that the song will play whenever the play button is selected, will loop whenever the loop button is selected, and will stop playing whenever the stop button is selected. If any particular button is selected more than once successively, the function of the button will not change (ex. hitting the loop button twice won't make it stop looping).

Unknowns
The unknowns are what buttons the user will choose to press and in what order. The buttons will operate independently from each other.

2. Planning the Solution

2.1 Solution Strategy

Once the page is opened, all button controls for the player along with the song title, artist, and album will appear and the song automatically starts. The user can choose to loop the song, stop the song, and then play the song after stopping it.

2.2 Goal Decomposition

Sub-goal 1
Create buttons for play, stop, loop.

Sub-goal 2
Respond to user button clicks by executing their respective commands (stop, play, loop).

2.3 Resources

Relevant formulas
None.

Formula Derivation
Not applicable.

2.4 Data Organization and Description

Input (givens):

Name	Description	Origin	Used in Sub-goal #
playButton, loopButton, stopButton	Event Object for Event	User	1
*control*Button ActionListener	Responds to event	User	2

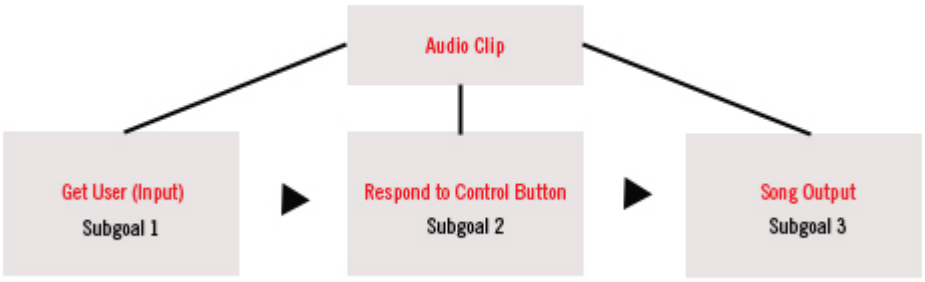
Output (unknowns):

Name	Description	Origin	Used in Sub-goal #
audioClip	Output that is being controlled.	Speaker	3

3. Designing the Solution

3.1 Structure Chart

First Level Decomposition



The first level decomposition shows the broad outline of the whole program. The steps are to get input from the User on the GUI (button click) to control the song that is playing. The program responds to the button action and the song is played, stopped or looped.

Goal Refinement

Sub-goal 1

Provide the user buttons to control the song.

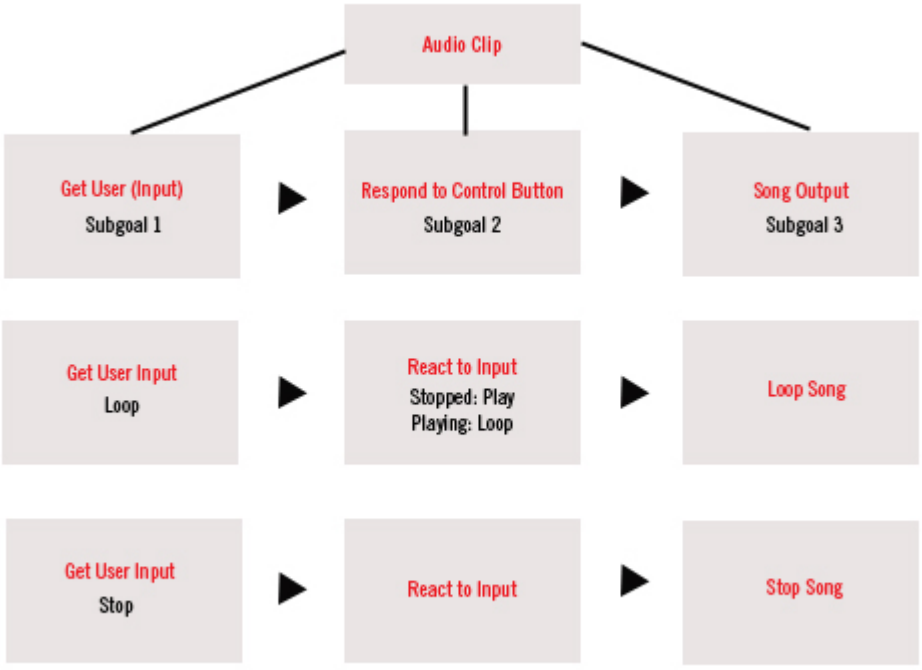
Sub-goal 2

Respond to the user input.

Sub-goal 3

Output or don't output the music.

Second Level Decomposition



The second level shows the repetition of the process for all three control buttons. The program will react to the input by playing the song, stopping the song, looping the song.

3.2 Module and Data Specifications

Global Variable: AudioClip type audioClip. Will hold the audio clip variable.

Class: audioPlayer

Name: audioPlayer extends JApplet and holds everything (constructor, variables, methods, panels etc).

Methods: start(), loop(), stop().

audioPlayer: Constructor that contains:

Variables:

- **buttonsPanel:** Panel that holds the buttons.
- **artistPanel:** Panel that holds all the artist information.
- **Play, loop, stop**: Creates buttons out of the images.
- **playButton, loopButton, stopButton:** GridLayout Panel holds labels for the image buttons.
- **playURL, loopURL, stopURL:** holds the URL for the play, loop and stop images.
- **audioURL**: holds the URL for the audio file.
- **audioClip:** AudioClip type object that allows the manipulation of the audio object.
- **artistName:** JLabel that holds the artist name.
- **songTitle:** JLabel that holds the song title.
- **album**: JLabel that holds the album title.

Method: start()
Variable: audioClip.
Logic: if the audioClip exists, play the audio clip.
Output: audio clip plays out to the speakers.

Method: loop()
Variable: audioClip.
Logic: if the audioClip exists, loop the audio clip.
Output: Audio Clip plays on loop outputting to speakers.

Method: stop()
Variable: audioClip.
Logic: if the audioClip exists, stop the audio clip.
Output: None, output stopped.

Data:

Name	Type	Structure
User input	Event	None

3.3 Algorithm

Logic

1.0 Create and render the image buttons
2.0 Listen for User input (clicks)
3.0 Respond to user clicks:
    3.1: action listeners respond
    3.2: Output song to speakers (loop, play)
    3.3: Stop song if stop is clicked.

Algorithm Description

There wasn't much of an algorithm involved. Because the program is just responding to user clicks and isn't recording information, there was no need to creation anything more than some simple conditional statements.

4. Translation

4.1 Source Code

```
// ****************************************************************
//   audioPlayer.java
//   Author: Jennifer Soh  ID: JS542
//   Compiler Used: JGrasp
//   Design and implement a Java applet that simulates an audio
//   player. The applet has three buttons labeled Play, Loop and
//   Stop that makes the song play, loop, or stop.
// ****************************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.applet.*;
import java.net.URL;

public class audioPlayer extends JApplet {
   private AudioClip audioClip;
   public audioPlayer() {
      // BUTTONS CONTAINER: buttonsPanel. Set to GridLayout
      JPanel buttonsPanel = new JPanel();
      buttonsPanel.setLayout(new GridLayout(1,4));

      // CREATES THE PLAY BUTTON.
      // 1. image url, 2. image icon, 3. button 4.add to buttonsPanel

      // Play Button
      URL playURL = getClass().getResource("img/play.png");
      ImageIcon play = new ImageIcon(playURL);
      JButton playButton = new JButton(play);
      buttonsPanel.add(playButton);

      // Loop Button
      URL loopURL = getClass().getResource("img/loop.png");
      ImageIcon loop = new ImageIcon(loopURL);
      JButton loopButton = new JButton(loop);
      buttonsPanel.add(loopButton);

      // Stop Button
      URL stopURL = getClass().getResource("img/stop.png");
      ImageIcon stop = new ImageIcon(stopURL);
      JButton stopButton = new JButton(stop);
      buttonsPanel.add(stopButton);

      // Get audio location. Store it in audioClip variable
      URL audioURL = getClass().getResource("amy.wav");
      audioClip = Applet.newAudioClip(audioURL);
```

```
        // Artist Information Panel set to GridLayout.
        JPanel artistPanel = new JPanel();
        artistPanel.setLayout(new GridLayout(3,0));
        JLabel artistName = new JLabel("Amy Winehouse");
        JLabel songTitle = new JLabel("Love is a Losing Game (Demo
Version)");
        JLabel album = new JLabel("Back to Black: B-Sides");

        // Adds the artist information labels to artistPanel
        artistPanel.add(artistName);
        artistPanel.add(songTitle);
        artistPanel.add(album);

        // Creates the main container for buttonPanel and artistPanel
        // Layout is set to BorderLayout.
        JPanel mainContainer = new JPanel();
        mainContainer.setLayout(new BorderLayout());
        mainContainer.add(artistPanel, BorderLayout.NORTH);
        mainContainer.add(buttonsPanel, BorderLayout.SOUTH);

        // add the mainContainer to the Frame
        add(mainContainer);

        // Adds an action listener to the playButton
        // ActionListener overrides the actionPerformed to start song.
        playButton.addActionListener(
              new ActionListener(){
                 @Override
                 public void actionPerformed(ActionEvent e){
                    start();
                 }
              });

        // Adds an action listener to the loop button.
        // ActionListener overrides action performed to loop song.
        loopButton.addActionListener(
              new ActionListener(){
                 @Override
                 public void actionPerformed(ActionEvent e){
                    loop();
                 }
              });

        // Adds an action listener to the stop button.
        // ActionListener overrides action performed to stop song.
        stopButton.addActionListener(
              new ActionListener(){
                 @Override
                 public void actionPerformed(ActionEvent e){
                    stop();
                 }
              });

    }

   // Defines the start method
    public void start(){
```

```
        if(audioClip != null) audioClip.play();
    }

    // Defines the loop method
    public void loop(){
        if(audioClip != null) audioClip.loop();
    }

    // Defines the stop method
    public void stop(){
        if(audioClip != null) audioClip.stop();
    }

}
```

5. Solution Testing

Test the program with following data domain:

I tested the program by clicking on each control button to see if it was working. I tested the play button by first testing the stop button to stop the song and then hit the play button to make it play again. I would test whether or not the song is looping by listening to the entire song after hitting the play button. It stops after the song concludes. I then tested the loop button but clicking on it and listening for the song to play again after the first play.

6. Testing: Output