Topic

Music Player

PO-YI LIN (Kevin)
YI-SHENG HSIAO (Ethan)
WEN-HSUAN HUANG(Emily)

Group:5

Contribution

Name	Student ID	Contribution
Po Yi Lin	D0965676	Coding, GUI design, paper
YI-SHENG HSIAO	D0965752	Flowchart,Function,Design,PowerPoint, paper, game mod design
WEN-HSUAN HUANG	D0970261	Flowchart, paper

ABSTRACT

Using Swing to write a music player in Java can be a challenging experience Need to understand some player logic and the application of Java grammar. In a graphical system, a windowing toolkit is usually responsible for providing a framework to make it relatively painless for a graphical user interface (GUI) to render the right bits to the screen at the right time. Both the AWT (abstract windowing toolkit) and Swing provide such a framework. In this report, we designed and report a simple music player for pausing, playing, switching songs, stopping, and saving music. A guessing song function is added to the design, which is different from ordinary music players. In our design, the music player only accepts WAV files. The purpose of this project is to allow you to enjoy music and guess songs with our music player. This project was implemented using the components from Java's awt and swing library in the Java programming language (Eclipse IDE).

Keywords: Eclipse IDE, Swing, GUI, AWT.

Contents

1. Introduction

1.1 Overview

Java is a general-purpose, concurrent, class-based, object-oriented computer programming the language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that code that runs on one platform does not need to be recompiled to run on another. Java applications are typically compiled to byte code (class file) that can run on any Java virtual machine (JVM) regardless of computer architecture. Java is, as of 2012, one of the most popular programming languages in use, particularly for client-server web applications, with a reported 10 million users. Java was originally developed by James Gosling at Sun Microsystems (which has since merged into Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them. Java can be used to write applications and applets. A Java application is similar to any other high-level language program: It can only be compiled and then run on the same machine. An applet is compiled on one machine, stored on a server in binary, and can be sent to another machine over the Internet to be interpreted by a Java-aware browser. Java comes with a large library of ready-made classes and objects. The key difference between Java 1.0 and 1.1 was in this library. Similarly, Java 2.0 has a much larger library for handling user interfaces (Swing by name) but only small changes to the core of the language.

1.2 Object-oriented Programming

Java supports object-oriented programming techniques that are based on a hierarchy of classes and well-defined and cooperating objects.
Classes: A class is a structure that defines the data and the methods to work on that data.
Objects: An instance is a synonym for an object. A newly created instance has data members and methods as defined by the class for that instance.
Inheritance and Polymorphism: One object-oriented concept that helps objects work together is inheritance. Inheritance defines relationships among classes in an object-oriented language. The relationship is one of parent to a child where the child or extending class inherits all the attributes (methods and data) of the parent class.

1.3 The Basic GUI Application

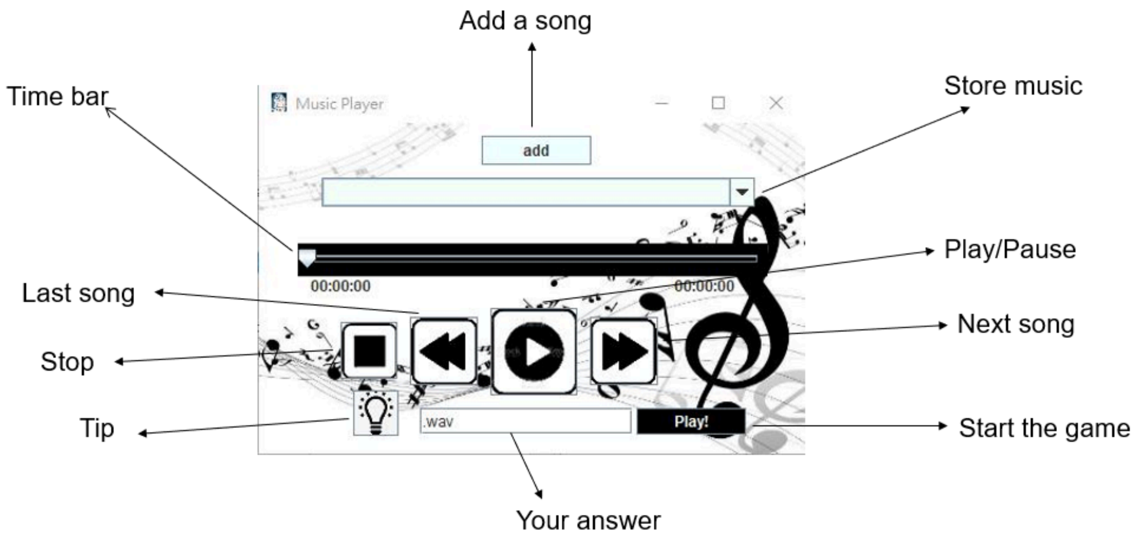
There are two basic types of GUI program in Java: stand-alone applications and (online) applets. An applet is a program that runs in a rectangular area on a Web page. Applets are generally small programs, meant to do fairly simple things, although there is nothing to stop them from being very complex. A GUI program offers a much richer type of user interface, where the user uses a mouse and keyboard to interact with GUI components such as windows, menus, buttons, checkboxes, text input boxes, scroll bars, and so on.
JFrame: In a Java GUI program, each GUI component in the interface is represented by an object in the program. It can be opened and closed.

JButton: It is used to create a labeled button that has platform-independent implementation. The application results in some action when the button is pushed. It inherits the AbstractButton class.
JComboBox: This shows a popup menu that shows a list and the user can select an option from that specified list. JComboBox can be editable or read-only depending on the choice of the programmer.

1.4 Motivation

There is much software that teaches us how to make a simple music player, and most of them are too basic, only start and stop this kind of simple command, so we want to use our way to build a useful music player and also add some fun to design music playback software that is different from other types. We want to make a music player who can play music and choose our favorite songs, repeat playback plus a music guessing game, and recognize more different songs.

2. How to use



2.1 Function

Time bar: See the current playing time.

Last song: Change to the previous song.

Next song: Switch to the next song.

Stop: End the current song.

Tip: Tell us the rules of the game.

Your answer box: Write the guessed song title.

Play! : Verify the answer (you can only start the game after the song is played).

Play/Pause: Pause or play the current song.

Store music list: Select songs from the list.

2.2 Produce

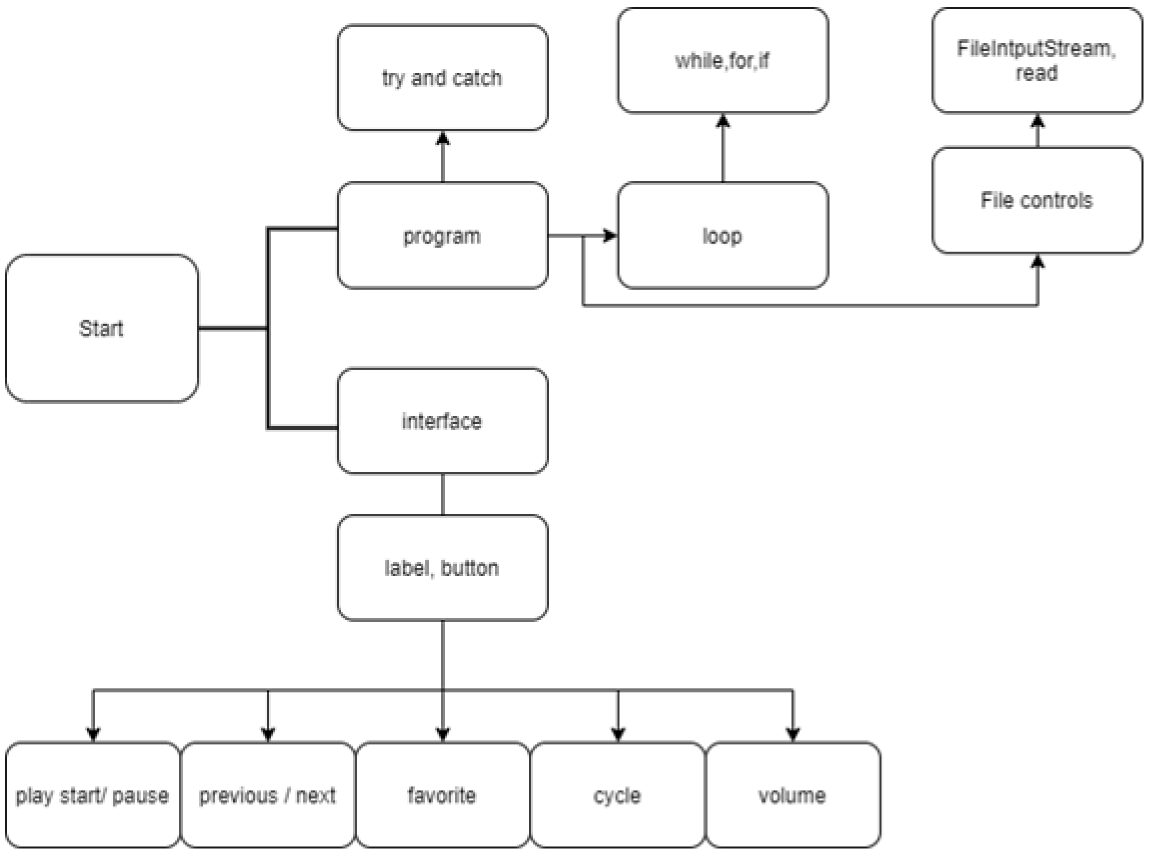
Step1: Use the add button to import .wav songs, the files will appear in the store music list.

Step2: Select a song and press the Play/Pause button to start playing.

Step3:Enter the song name in your answer box.

Step4: Press the Play! button.
Step5: The system will tell if the song title is correct.

2.3 Flowchart



3. Program

3.1 Add Button

Result: Add music to the player.
Code:

```
add.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                        browser.showOpenDialog(add);//Press the add button, the browser interface appears.

                seletedFile = browser.getSelectedFile();//The selected file is the file we selected in the browser.
                musics[index] = seletedFile.toString();//Put into ARRAY (start from zero) After that, the selection (number) from JCombobox becomes a variable in music[]. The two storage locations are different.
                list.addItem("Song." + (index + 1)+":"+seletedFile.getName());//Put into JCombobox (start from zero) After that, the selection (number) from JCombobox becomes a variable in music[]. The two storage locations are different.
                index++; } });//When we press the add button a second time, the audio file will be stored in [1] of the music array because the index is the audio file.
```

3.2 Playpause Button

Result: Execution result: play music, or stop music, or play again
Logic:
1. Change the variable counter to 1 or -1 to determine the code to be executed next time
2. Start by stopping and recording the time, and set the recording time to create the play and pause functions
Code:

```
playpause.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e1) {
           try {
                 if (counter == -1) {
                playpause.setIcon(new ImageIcon("pause.png"));
                counter = 1;
                clip.setMicrosecondPosition(nowFrame);//Set the clip setting to the value obtained by clip.getMicrosecondPosition()

                clip.start();//The music file starts to play
                timer.resumeTimer();
             } else if (counter == 1) {
                if (clip.isRunning()) {
                playpause.setIcon(new ImageIcon("play.png"));
```

```
                        nowFrame = clip.getMicrosecondPosition();//Record "nowFrame" as the
current time
                        clip.stop();//Stop the music

                        counter = -1;
                        timer.pauseTimer();

                    } else {//When the clip is no longer running

                    playpause.setIcon(new ImageIcon("pause.png"));
                        sound = new File(musics[list.getSelectedIndex()]);// Convert the selected
list to an integer to select the string in the array, and then convert the string to a file.
                        ais = AudioSystem.getAudioInputStream(sound);// Convert file to music
file
                        clip = AudioSystem.getClip();
                        clip.open(ais);//Put the music file into the clip mixer

                        clip.start();
                        sliderTime.setMaximum((int) clip.getMicrosecondLength()/ 1_000_000);
//Set the maximum value of Jslider to the number of seconds of music

                        timer = new PlayingTimer(labelTimeCounter, sliderTime);
                        timer.start();//play music

                        timer.setAudioClip(clip);
                        maxtime();//The class that converts the number of seconds to hours and
minutes

                        ans = sound.getName();
                    }
                } else if (counter == 0) {
                    playpause.setIcon(new ImageIcon("pause.png"));
                    sound = new File(musics[list.getSelectedIndex()]);
                    ais = AudioSystem.getAudioInputStream(sound);
                    clip = AudioSystem.getClip();
                    clip.open(ais);
                    clip.start();
                    sliderTime.setMaximum((int) clip.getMicrosecondLength()/ 1_000_000);
                    timer = new PlayingTimer(labelTimeCounter, sliderTime);
                    timer.start();
                    timer.setAudioClip(clip)
                    maxtime();
                    counter = 1;
                    ans = sound.getName();
                    counter3=1;
                }
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "erro");}} });
```

Detail:
1. The music does not run anymore, restart from counter==1 because the music stops automatically and must not be able to pause it, so the end must be at counter==-1
At -1, if only the counter becomes 0, there will be one more step
At 1, it is not easy to judge when it is zero.
2. Set the max value of slider time.
3. Ans gets the file name of our sound file and uses it for the game mod.
4. Combobox saves from 0.
5.slider only accepts int files.

3.3 Next Button

Result: The song will skip to the next or previous song of my choice and start playing.

Logic: As we assume num is the number of an array which we selected, changing the size of num to change the location of an array that contains music inside; therefore, we can control the front and back of the music.

Code:
```
next.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                    playpause.setIcon(new ImageIcon("pause.png"));
             try {
             clip.stop();
             num++;//Increase the value of num by 1 to change the selected array item

             sound = new File(musics[list.getSelectedIndex()+num]);
             ais = AudioSystem.getAudioInputStream(sound);
             clip = AudioSystem.getClip();
            clip.open(ais);
            clip.start();
            sliderTime.setMaximum((int) clip.getMicrosecondLength()/ 1_000_000);
            timer = new PlayingTimer(labelTimeCounter, sliderTime);
            timer.start();
            timer.setAudioClip(clip);
            maxtime();
            counter = 1;
            ans = sound.getName();
            } catch (Exception e1) {
              JOptionPane.showMessageDialog(null, "There is no any music");
              if(num!=0) {num--;//If an error occurs in num++, subtract 1 to restore it.
              timer.reset();
              timer.interrupt();}}}}});//Remake the PlayerTimer Class.
```

3.4 Stop Button

Result: Music, timeline, time text will end and zero.
Logic: Use timer.interrupt() to interfere with the PlayingTimer function so that the timer and timing text will come back to the Initial state.
Code:

```
 stop.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
                 playpause.setIcon(new ImageIcon("play.png"));
            clip.stop();
            counter = 1;
            timer.reset();
            timer.interrupt();} });
```

3.5 Game mod Button

Result: Determine your answer is right or wrong.
Logic: Using if, else if, else condition to separate different actions.
Code:

```
 gamemod.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            ans2 = asw.getText();//ans2 is what we got from asw, which is what we type
            if (ans2.equals(ans)) {//If the answer is the same as what we typed, boolean will run
true and run if, and there will be an interface to tell us that the answer is correct.

                 JOptionPane.showMessageDialog(null, "Congratulations, you guessed it
right!");
              } else if (ans == null || ans2 == null || ans2 ==".wav") {//If we don't play music
or we don't write answers, boolean will run true and run else if, and there will be an interface
that tells us to write answers or ask us to play music.
                 JOptionPane.showMessageDialog(null, "You need to write the answer or play
a song.");
              } else {//Other booleans will run false and run else, and there will be an interface
telling us that the error should be written once.
                 JOptionPane.showMessageDialog(null, "Wrong! Answer it again.");}} });
```

3.6 PlayingTimer Class

Result: Executed in the background and the millisecond conversion obtained is presented by the string and the value of Jslider.
Logic:
1.Use isrunning to determine boolean, and isPause to determine whether the program is running.
2. The time difference between the current system time of the call minus the system time in the beginning and the elapsed time during pause is the running time.
Code:

```
public void run() {
      isRunning = true;
      startTime = System.currentTimeMillis();//Set startTime to the current system time.
      while (isRunning) {
         try {
            Thread.sleep(100);//Delay 100 milliseconds.
            if (!isPause) {
               if (audioClip != null && audioClip.isRunning()) {
                  labelRecordTime.setText(toTimeString());//Set Jlable to the current music
time.
                  int currentSecond = (int) audioClip.getMicrosecondPosition() /
1_000_000;//Used to set the value of Jslider as the current music time.
                  slider.setValue(currentSecond);
               }
            } else {
               pauseTime += 100;
            }
         } catch (InterruptedException ex) {
            ex.printStackTrace();
            if (isReset) {
               slider.setValue(0);
               labelRecordTime.setText("00:00:00");
               isRunning = false;
               break;}}}}

private String toTimeString() {
      long now = System.currentTimeMillis();//Set now as the current system time.
      Date current = new Date(now - startTime - pauseTime);//Calculate the elapsed time.
      dateFormater.setTimeZone(TimeZone.getTimeZone("GMT"));
      String timeCounter = dateFormater.format(current);//Format the calculated time.
      return timeCounter;
   }
```

Detail:
1. Pause plus 100 means to add 100 milliseconds.
2.printStackTrace() is used to track and control exceptions and errors.
3.class extends Thread enables the program to run in the background so that the while will appear in the main program and cause it to fail to execute.
4.(audioClip != null && audioClip.isRunning()) is to not execute if the audio file has not been inserted.
5. The ratio between the total milliseconds and the set milliseconds can show the time axis.
6. The reason why boolean is set to class is that the two main classes can be connected.

3.7 JCombobox
Result: Select a piece of music and then end the music, and adjust the timeline to the initial state.
Code:
```
list.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            if(counter3!=0) {//Set it to be unable to execute before putting in music to avoid errors.
            playpause.setIcon(new ImageIcon("play.png"));
            clip.stop();
            counter = 0;
            timer.reset();
            timer.interrupt();
            num=0;}
        }
    });
```

Detail: “if(counter3!=0)” is used to prevent the first time selecting music because of “clip.stop();” causing a program error because there is no music yet.

4. Conclusion

4.1 Conclusion

Our report introduced the Java audio player sample application in Swing. The music file can only be a wav file, and we add a pause, start, previous song, next song, and timeline. We also add a guessing game to let everyone know songs they don’t know when listening to music. If we have a chance to improve our program we will add a timeline that can be pulled, add my favorites, it will be more like a music player.

4.2 Limitations

1. Only use wav files in the correct format
2. The case of previous and next music cannot be pressed back and forth more than 8 times
3. The pattern of the enlarged window will not be enlarged
4.Only 10 songs can be inserted.

4.3 Experience

From the beginning, we decided to do something more special and practical. In the beginning, the choice was “Snake”, a retro game, but later we found that we couldn't make it without completely using other people's code, so we gave up making games. After that, someone just happened to propose to make a music player, and we also think it is a good idea. First, we separately researched and collected information about GUI and audio files from scratch, and then we tried to make a few simple music players, and then reorganized the feasible solutions into our main program class 5 (representing the 5th attempt). When the program is finally completed, we start to add additional functions like games and timelines. Here we would like to thank our TA for solving many of our doubts and helping us understand the difficult timeline operation. The biggest difficulty I encountered was challenging the field I was not familiar with. Because I didn't know what was wrong. I could only open the previous file and compare which side of the program error occurred step by step.

Reference

1.Time Bar:
Author: Nam Ha Minh
Topic: Java audio player sample application in Swing
Watch Date: 2020/05/27
Website: https://www.codejava.net/coding/java-audio-player-sample-application-in-swing

2.Introduction:
Author: Mohamed Hazber
Topic: Final Report: Java Programming Language. A Simple project to Draw Paint (Java language)
Watch Date: 2020/06/13
Website:
https://www.researchgate.net/publication/306631299_Final_Report_Java_Programming_Language_A_Simple_project_to_Draw_Paint_Java_language