Hello

Hello! How can I assist you today?

Pls write me example for view waveworm (amplitude/time) of .wav file in javaFx

Sure, here's an example of how to display a waveform (amplitude over time) of a .wav file in JavaFX:

java

import javafx.application.Application;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.chart.LineChart;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.XYChart;

import javafx.scene.layout.BorderPane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javafx.scene.chart.XYChart.Data;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import java.io.File;

public class WaveformViewer extends Application {

private final static int WIDTH = 800;

private final static int HEIGHT = 600;

private final static String AUDIO\_FILE = "example.wav";

@Override

public void start(Stage stage) throws Exception {

// Load the audio file and get the sample data

AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(new File(AUDIO\_FILE));

int numBytes = (int) (audioInputStream.getFrameLength() \* audioInputStream.getFormat().getFrameSize());

byte[] audioBytes = new byte[numBytes];

audioInputStream.read(audioBytes);

// Create the X and Y axis for the waveform chart

final NumberAxis xAxis = new NumberAxis();

final NumberAxis yAxis = new NumberAxis(-1.0, 1.0, 0.1);

xAxis.setLabel("Time (seconds)");

yAxis.setLabel("Amplitude");

// Create the waveform chart and add the sample data

final LineChart<Number, Number> waveformChart = new LineChart<>(xAxis, yAxis);

XYChart.Series<Number, Number> series = new XYChart.Series<>();

int numSamples = audioBytes.length / 2;

for (int i = 0; i < numSamples; i++) {

double time = (double) i / audioInputStream.getFormat().getFrameRate();

short sample = (short) ((audioBytes[2\*i] & 0xff) | (audioBytes[2\*i+1] << 8));

double amplitude = (double) sample / 32768.0;

series.getData().add(new XYChart.Data<>(time, amplitude));

}

waveformChart.getData().add(series);

// Create the root layout and add the waveform chart

BorderPane root = new BorderPane();

root.setCenter(waveformChart);

// Create the scene and add the root layout

Scene scene = new Scene(root, WIDTH, HEIGHT, Color.WHITE);

// Show the stage

stage.setScene(scene);

stage.show();

}

public static void main(String[] args) {

launch(args);

}

}

This code uses the AudioSystem and Clip classes from the Java Sound API to load a .wav file and get the sample data as an array of bytes. It then creates a LineChart from the JavaFX chart API and adds the sample data to the chart as a series of XY data points. Finally, it creates a JavaFX Scene and shows it in a Stage.

Note that this code assumes that the .wav file is in the same directory as the Java source file and is named "example.wav". You will need to modify the AUDIO\_FILE constant to point to the actual .wav file that you want to view.

add some contoller of time by mouse for .wav play

Sure, here's an updated example that includes a time controller that allows the user to seek to a specific time in the .wav file by clicking on the waveform chart:

java

import javafx.application.Application;

import javafx.beans.value.ChangeListener;

import javafx.beans.value.ObservableValue;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.chart.LineChart;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.XYChart;

import javafx.scene.input.MouseEvent;

import javafx.scene.layout.BorderPane;

import javafx.scene.paint.Color;

import javafx.stage.Stage;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import java.io.File;

public class WaveformPlayer extends Application {

private final static int WIDTH = 800;

private final static int HEIGHT = 600;

private final static String AUDIO\_FILE = "example.wav";

private AudioInputStream audioInputStream;

private Clip clip;

private LineChart<Number, Number> waveformChart;

private double startTime;

@Override

public void start(Stage stage) throws Exception {

// Load the audio file and get the sample data

audioInputStream = AudioSystem.getAudioInputStream(new File(AUDIO\_FILE));

int numBytes = (int) (audioInputStream.getFrameLength() \* audioInputStream.getFormat().getFrameSize());

byte[] audioBytes = new byte[numBytes];

audioInputStream.read(audioBytes);

// Create the X and Y axis for the waveform chart

final NumberAxis xAxis = new NumberAxis();

final NumberAxis yAxis = new NumberAxis(-1.0, 1.0, 0.1);

xAxis.setLabel("Time (seconds)");

yAxis.setLabel("Amplitude");

// Create the waveform chart and add the sample data

waveformChart = new LineChart<>(xAxis, yAxis);

XYChart.Series<Number, Number> series = new XYChart.Series<>();

int numSamples = audioBytes.length / 2;

for (int i = 0; i < numSamples; i++) {

double time = (double) i / audioInputStream.getFormat().getFrameRate();

short sample = (short) ((audioBytes[2\*i] & 0xff) | (audioBytes[2\*i+1] << 8));

double amplitude = (double) sample / 32768.0;

series.getData().add(new XYChart.Data<>(time, amplitude));

}

waveformChart.getData().add(series);

// Add a mouse click event handler to the waveform chart to seek to the selected time

waveformChart.setOnMouseClicked(event -> {

if (event.getButton() == MouseEvent.BUTTON1) {

double time = startTime + event.getXAxis().getValueForDisplay(event.getX()).doubleValue();

clip.setMicrosecondPosition((long) (time \* 1000000));

}

});

// Create the root layout and add the waveform chart

BorderPane root = new BorderPane();

root.setCenter(waveformChart);

// Create the scene and add the root layout

Scene scene = new Scene(root, WIDTH, HEIGHT, Color.WHITE);

// Show the stage

stage.setScene(scene);

stage.show();

// Create the audio clip and add a listener to update the start time as the clip plays

clip = AudioSystem.getClip();

clip.open(audioInputStream);

clip.addLineListener(event -> {

if (event.getType() == javax.sound.sampled.LineEvent.Type.START) {

startTime = clip.getMicrosecondPosition() / 1000000.0;

}

});

// Add a listener to update the position of the waveform chart as the clip plays

Continue

scss

clip.addLineListener(event -> {

if (event.getType() == javax.sound.sampled.LineEvent.Type.UPDATE) {

double currentTime = clip.getMicrosecondPosition() / 1000000.0;

double chartWidth = waveformChart.getXAxis().getWidth();

double secondsPerPixel = (waveformChart.getXAxis().getUpperBound() - waveformChart.getXAxis().getLowerBound()) / chartWidth;

double startPosition = currentTime - (chartWidth \* secondsPerPixel / 2);

waveformChart.getXAxis().setLowerBound(startPosition);

waveformChart.getXAxis().setUpperBound(startPosition + chartWidth);

}

});

// Add a listener to pause the clip and reset the start time when the end is reached

clip.addLineListener(event -> {

if (event.getType() == javax.sound.sampled.LineEvent.Type.STOP && clip.getMicrosecondPosition() == clip.getMicrosecondLength()) {

clip.stop();

clip.setMicrosecondPosition(0);

startTime = 0;

}

});

// Add a listener to update the position of the time controller as the clip plays

final double timeControllerWidth = 5;

final double timeControllerHeight = 20;

final double timeControllerXOffset = timeControllerWidth / 2;

final double timeControllerYOffset = timeControllerHeight / 2;

Group timeController = new Group();

timeController.getChildren().add(new javafx.scene.shape.Rectangle(-timeControllerXOffset, -timeControllerYOffset, timeControllerWidth, timeControllerHeight));

timeController.setTranslateY(waveformChart.getHeight() + timeControllerYOffset);

root.getChildren().add(timeController);

clip.addLineListener(event -> {

if (event.getType() == javax.sound.sampled.LineEvent.Type.UPDATE) {

double currentTime = clip.getMicrosecondPosition() / 1000000.0;

double chartWidth = waveformChart.getXAxis().getWidth();

double secondsPerPixel = (waveformChart.getXAxis().getUpperBound() - waveformChart.getXAxis().getLowerBound()) / chartWidth;

double position = (currentTime - startTime) / secondsPerPixel;

timeController.setTranslateX(position);

}

});

// Add a listener to update the clip position when the time controller is dragged

timeController.setOnMouseDragged(event -> {

if (clip.isActive()) {

double position = startTime + waveformChart.getXAxis().getValueForDisplay(event.getX() - timeControllerXOffset).doubleValue();

clip.setMicrosecondPosition((long) (position \* 1000000));

}

});

// Add a listener to update the time controller position when the waveform chart is dragged

waveformChart.setOnMouseDragged(event -> {

if (clip.isActive()) {

double time = startTime + event.getXAxis().getValueForDisplay(event.getX()).doubleValue();

double chartWidth = waveformChart.getXAxis().getWidth();

double secondsPerPixel = (waveformChart.getXAxis().getUpperBound() - waveformChart.getXAxis().getLowerBound()) / chartWidth;

double position = (time - startTime) / secondsPerPixel;

timeController.setTranslateX(position);

}

});

// Start the audio clip

clip.start();

}

public static void main(String[] args) {

launch(args);

}

}

css

This example adds a time controller that is represented by a small rectangle at the bottom of the waveform chart. The user can drag the time controller to seek to a specific time in the .wav file, and the position of the time controller is updated as the clip plays. The position of the time controller can also be updated by dragging the waveform chart.

Continue

Here's a breakdown of the code:

1. The start() method is called when the JavaFX application starts. It sets up the UI elements and loads the .wav file.
2. The waveform chart is created using the LineChart class. The x-axis represents time, and the y-axis represents amplitude. The waveform data is plotted as a series of line segments.
3. The .wav file is loaded using the AudioSystem class. The Clip class is used to play back the audio data.
4. The LineListener interface is used to update the position of the waveform chart and the time controller as the clip plays. The UPDATE event is fired whenever the clip position changes. The STOP event is fired when the clip reaches the end.
5. The Group class is used to create the time controller. The Rectangle class is used to draw a small rectangle that represents the current position in the .wav file.
6. The position of the time controller is updated when the clip plays using the setTranslateX() method. The position is calculated based on the current time and the width of the waveform chart.
7. The position of the time controller can be updated by dragging it with the mouse. The setOnMouseDragged() method is used to listen for mouse events. The position of the time controller is updated by calculating the time based on the position of the mouse and the width of the waveform chart.
8. The position of the time controller can also be updated by dragging the waveform chart. The setOnMouseDragged() method is used to listen for mouse events. The position of the time controller is updated by calculating the time based on the position of the mouse and the width of the waveform chart.
9. The clip is started using the start() method.

I hope this example helps you to implement the desired functionality in your project. Let me know if you have any other questions or need further assistance.

Show sample example for play .wav file in javaFX with usability to time jump by mouse and rePlay .wav by key \*SPACE\*

Sure, here's an example of playing a .wav file in JavaFX and allowing the user to jump to a specific time by clicking on a waveform chart, as well as replaying the .wav file with the space bar:

java

import javafx.application.Application;

import javafx.beans.value.ChangeListener;

import javafx.beans.value.ObservableValue;

import javafx.scene.Group;

import javafx.scene.Scene;

import javafx.scene.chart.LineChart;

import javafx.scene.chart.NumberAxis;

import javafx.scene.chart.XYChart;

import javafx.scene.input.KeyCode;

import javafx.scene.layout.BorderPane;

import javafx.scene.paint.Color;

import javafx.scene.shape.Rectangle;

import javafx.scene.shape.StrokeLineCap;

import javafx.scene.shape.StrokeLineJoin;

import javafx.stage.Stage;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.Clip;

import javax.sound.sampled.DataLine;

import java.io.File;

public class AudioPlayer extends Application {

private static final int CHART\_HEIGHT = 200;

private static final int CONTROLLER\_HEIGHT = 10;

private static final int CONTROLLER\_WIDTH = 10;

private Clip clip;

private double duration;

private XYChart.Series<Number, Number> series;

private Rectangle controller;

private NumberAxis xAxis;

private double lastX = 0;

@Override

public void start(Stage primaryStage) throws Exception {

// Load the .wav file

File audioFile = new File("audio.wav");

AudioInputStream audioStream = AudioSystem.getAudioInputStream(audioFile);

clip = (Clip) AudioSystem.getLine(new DataLine.Info(Clip.class, audioStream.getFormat()));

clip.open(audioStream);

duration = clip.getMicrosecondLength() / 1000000.0;

// Create the waveform chart

NumberAxis yAxis = new NumberAxis(-1, 1, 0.2);

xAxis = new NumberAxis(0, duration, 1);

xAxis.setAutoRanging(false);

xAxis.setTickUnit(1);

xAxis.setMinorTickCount(0);

LineChart<Number, Number> chart = new LineChart<>(xAxis, yAxis);

chart.setAnimated(false);

chart.setCreateSymbols(false);

chart.setLegendVisible(false);

chart.setPrefHeight(CHART\_HEIGHT);

series = new XYChart.Series<>();

chart.getData().add(series);

// Create the time controller

Group controllerGroup = new Group();

controller = new Rectangle(CONTROLLER\_WIDTH, CONTROLLER\_HEIGHT);

controller.setFill(Color.WHITE);

controller.setStroke(Color.BLACK);

controller.setStrokeWidth(1);

controller.setStrokeLineJoin(StrokeLineJoin.ROUND);

controller.setStrokeLineCap(StrokeLineCap.ROUND);

controllerGroup.getChildren().add(controller);

// Create the main layout

BorderPane root = new BorderPane();

root.setCenter(chart);

root.setBottom(controllerGroup);

// Update the position of the time controller as the clip plays

clip.addLineListener(event -> {

if (event.getType() == javax.sound.sampled.LineEvent.Type.UPDATE) {

double x = clip.getMicrosecondPosition() / 1000000.0;

controller.setTranslateX(x / duration \* CHART\_HEIGHT);

} else if (event.getType() == javax.sound.sampled.LineEvent.Type.STOP) {

clip.setMicrosecondPosition(0);

controller.setTranslateX(0);

}

});

// Jump to a specific time by clicking on the waveform chart

chart.setOnMouseClicked(event -> {

double x = xAxis.getValueForDisplay(event.getX()).doubleValue();

clip.setMicrosecondPosition((long) (x \* 1000000.0));

controller.setTranslateX(event

hi dear

Hello! How can I assist you today?

i need class for play .wav file. This class must have constructor with Path to .wav and method for jump to custom time play. JavaFX

Sure, here is an example of a JavaFX class that plays a .wav file and provides a method to jump to a custom time:

java

import javafx.scene.media.Media;

import javafx.scene.media.MediaPlayer;

import java.io.File;

public class WavPlayer {

private final MediaPlayer mediaPlayer;

public WavPlayer(String path) {

Media media = new Media(new File(path).toURI().toString());

mediaPlayer = new MediaPlayer(media);

mediaPlayer.setAutoPlay(false);

}

public void play() {

mediaPlayer.play();

}

public void pause() {

mediaPlayer.pause();

}

public void jumpTo(double time) {

mediaPlayer.seek(mediaPlayer.getTotalDuration().multiply(time));

}

public void dispose() {

mediaPlayer.dispose();

}

}

This class takes the path to the .wav file in its constructor and initializes a MediaPlayer object to play the audio. The play() and pause() methods start and stop playback, respectively. The jumpTo() method takes a time (in seconds) and jumps to that point in the audio playback. The dispose() method releases the resources used by the MediaPlayer object.

why .wav file paused after some time?

There can be several reasons why a .wav file would pause after playing for some time. Here are a few possible explanations:

1. Buffering: If the .wav file is being streamed over a network or read from a slow disk, it may be buffering the audio data as it plays. If the buffer runs out of data, the playback may pause until more data can be loaded.
2. Memory usage: If the .wav file is large and/or the application is using a lot of memory, the system may start paging memory to disk, causing the application to slow down or pause while it waits for data to be loaded from disk.
3. CPU usage: If the system is running other CPU-intensive tasks or if the application is doing a lot of processing on the audio data, the playback may pause as the CPU is unable to keep up with the demand.
4. Code issues: There may be issues in the code that are causing the pause, such as memory leaks or thread locking.

To diagnose the issue, you can try to identify if the problem is specific to the .wav file you're playing or if it's a more general problem with your system or code. You can try playing the .wav file with a different media player to see if the problem persists, or you can try running your application on a different machine to see if the problem is related to your hardware. You can also use a profiler to identify any memory or CPU usage issues in your application.

can u try to solve this task with any another library?

Sure! There are several libraries in Java that can be used to play audio files. One popular option is the Java Sound API, which provides a set of classes for playing and manipulating audio data. Here's an example class that plays a .wav file using the Java Sound API:

java

import javax.sound.sampled.\*;

import java.io.File;

import java.io.IOException;

public class WavPlayer {

private AudioInputStream audioInputStream;

private SourceDataLine line;

private boolean isPlaying = false;

public WavPlayer(String path) {

try {

audioInputStream = AudioSystem.getAudioInputStream(new File(path));

AudioFormat format = audioInputStream.getFormat();

DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);

line = (SourceDataLine) AudioSystem.getLine(info);

line.open(format);

} catch (UnsupportedAudioFileException | LineUnavailableException | IOException e) {

e.printStackTrace();

}

}

public void play() {

if (!isPlaying) {

new Thread(() -> {

try {

line.start();

isPlaying = true;

byte[] buffer = new byte[4096];

int bytesRead = 0;

while ((bytesRead = audioInputStream.read(buffer)) != -1) {

if (!isPlaying) {

break;

}

line.write(buffer, 0, bytesRead);

}

} catch (IOException e) {

e.printStackTrace();

} finally {

stop();

}

}).start();

}

}

public void stop() {

isPlaying = false;

line.stop();

line.close();

}

public void jumpTo(double time) {

long bytesToSkip = (long) (time \* audioInputStream.getFormat().getFrameRate() \* audioInputStream.getFormat().getFrameSize());

try {

audioInputStream.skip(bytesToSkip);

} catch (IOException e) {

e.printStackTrace();

}

}

}

This class takes the path to the .wav file in its constructor and initializes an AudioInputStream and a SourceDataLine object to play the audio. The play() method starts playing the audio on a new thread, reading and writing the audio data to the SourceDataLine in chunks. The stop() method stops the playback by setting a flag to false and closing the SourceDataLine. The jumpTo() method takes a time (in seconds) and calculates the number of bytes to skip in the AudioInputStream to jump to that point in the audio playback.

and with someone another lib?

Certainly! Another popular library for playing audio in Java is the JavaFX Media library. Here's an example class that plays a .wav file using JavaFX Media:

java

import javafx.scene.media.Media;

import javafx.scene.media.MediaPlayer;

import javafx.util.Duration;

import java.io.File;

public class WavPlayer {

private Media media;

private MediaPlayer mediaPlayer;

private boolean isPlaying = false;

public WavPlayer(String path) {

media = new Media(new File(path).toURI().toString());

mediaPlayer = new MediaPlayer(media);

}

public void play() {

if (!isPlaying) {

mediaPlayer.play();

isPlaying = true;

}

}

public void stop() {

if (isPlaying) {

mediaPlayer.stop();

isPlaying = false;

}

}

public void jumpTo(double time) {

Duration duration = media.getDuration();

double millis = duration.toMillis();

double jumpMillis = millis \* time;

mediaPlayer.seek(Duration.millis(jumpMillis));

}

}

This class takes the path to the .wav file in its constructor and initializes a Media and a MediaPlayer object to play the audio. The play() method starts playing the audio, and the stop() method stops the playback by calling the stop() method on the MediaPlayer. The jumpTo() method takes a time (in seconds) and calculates the number of milliseconds to jump in the audio playback using the Duration class. It then calls the seek() method on the MediaPlayer to jump to the specified position in the audio.

after jump method its not playing

If the audio is not playing after using the jumpTo() method, it's possible that the MediaPlayer has been paused or stopped. When you call seek() on the MediaPlayer, it does not automatically start playing the audio from the new position. You'll need to call the play() method again to resume playback from the new position. Here's an updated version of the jumpTo() method that calls play() after seeking to the new position:

java

public void jumpTo(double time) {

Duration duration = media.getDuration();

double millis = duration.toMillis();

double jumpMillis = millis \* time;

mediaPlayer.seek(Duration.millis(jumpMillis));

mediaPlayer.play();

}

With this updated method, the MediaPlayer will resume playing from the new position after the jump.

still dont playing after JumpTo

If the audio is still not playing after calling jumpTo() and play(), there are a few other things you can try:

1. Make sure that the MediaPlayer is not in a PAUSED or STOPPED state. If it is, you'll need to call play() to start playback again.
2. Make sure that the MediaPlayer is not muted or volume is set to 0. You can check this by calling mediaPlayer.isMute() or mediaPlayer.getVolume().
3. Make sure that the Media object has finished loading before calling jumpTo() and play(). You can check this by registering an OnReady event handler on the Media object and calling jumpTo() and play() in the event handler.

Here's an updated version of the WavPlayer class that includes these changes:

java

import javafx.scene.media.Media;

import javafx.scene.media.MediaPlayer;

import javafx.scene.media.MediaPlayer.Status;

import javafx.util.Duration;

import java.io.File;

public class WavPlayer {

private Media media;

private MediaPlayer mediaPlayer;

private boolean isPlaying = false;

public WavPlayer(String path) {

media = new Media(new File(path).toURI().toString());

mediaPlayer = new MediaPlayer(media);

mediaPlayer.setOnReady(() -> {

// media has finished loading, ready to play

isPlaying = true;

mediaPlayer.play();

});

}

public void play() {

if (mediaPlayer.getStatus() == Status.PAUSED || mediaPlayer.getStatus() == Status.STOPPED) {

mediaPlayer.play();

isPlaying = true;

}

}

public void stop() {

if (isPlaying) {

mediaPlayer.stop();

isPlaying = false;

}

}

public void jumpTo(double time) {

if (mediaPlayer.getStatus() == Status.UNKNOWN || mediaPlayer.getStatus() == Status.HALTED) {

return;

}

Duration duration = media.getDuration();

double millis = duration.toMillis();

double jumpMillis = millis \* time;

mediaPlayer.seek(Duration.millis(jumpMillis));

if (mediaPlayer.getStatus() != Status.PLAYING) {

mediaPlayer.play();

}

}

}

This updated WavPlayer class registers an OnReady event handler on the Media object that starts playback when the media has finished loading. The play() method now checks the status of the MediaPlayer and starts playback if it's in a PAUSED or STOPPED state. The jumpTo() method also checks the status of the MediaPlayer and returns early if the status is UNKNOWN or HALTED. Finally, the jumpTo() method calls play() after seeking to the new position to ensure that playback resumes from the new position.

you have a bug, now i will explain)

I'm sorry to hear that there is still a bug in the code. Can you please provide more details on the issue you are encountering so that I can help you better?

method jumpTo takes % of file to start, not seconds of .wav file

I apologize for the confusion. If the jumpTo() method takes a percentage of the file to start instead of seconds of the .wav file, you can modify the jumpTo() method to take a double value between 0 and 1 instead of a time value in seconds. Here's an updated version of the jumpTo() method that takes a percentage value:

java

public void jumpTo(double percent) {

if (mediaPlayer.getStatus() == Status.UNKNOWN || mediaPlayer.getStatus() == Status.HALTED) {

return;

}

Duration duration = media.getDuration();

double millis = duration.toMillis();

double jumpMillis = millis \* percent;

mediaPlayer.seek(Duration.millis(jumpMillis));

if (mediaPlayer.getStatus() != Status.PLAYING) {

mediaPlayer.play();

}

}

This updated version of the jumpTo() method calculates the number of milliseconds to jump to based on the percentage value and the duration of the media file. It then seeks to the new position using the seek() method of the MediaPlayer. Finally, the method checks the status of the MediaPlayer and starts playback if it's not already playing.

переведи на русский width

Конечно, "width" на русском языке означает "ширина".

write class, whick will cut .wav file from t1 to t2. This class must have constructor with PathToFile and t1,t2 time. Also class must self destroued if some code will call it twice

Here's an example implementation of a class that cuts a .wav file from a starting time t1 to an ending time t2. The class has a constructor that takes the path to the .wav file, t1 and t2. It also has a cut() method that performs the cut and returns a new .wav file with the cut segment.

java

import java.io.File;

import java.io.IOException;

import javax.sound.sampled.AudioFileFormat;

import javax.sound.sampled.AudioFormat;

import javax.sound.sampled.AudioInputStream;

import javax.sound.sampled.AudioSystem;

import javax.sound.sampled.DataLine;

import javax.sound.sampled.LineUnavailableException;

import javax.sound.sampled.TargetDataLine;

import javax.sound.sampled.UnsupportedAudioFileException;

public class WavCutter {

private File inputFile;

private double t1;

private double t2;

private boolean isCut = false;

public WavCutter(File inputFile, double t1, double t2) {

this.inputFile = inputFile;

this.t1 = t1;

this.t2 = t2;

}

public File cut() throws IOException, UnsupportedAudioFileException {

if (isCut) {

throw new IllegalStateException("This WavCutter object has already been used to cut a file.");

}

isCut = true;

// Get the input audio stream and format

AudioInputStream inStream = AudioSystem.getAudioInputStream(inputFile);

AudioFormat format = inStream.getFormat();

// Calculate the frame size and starting and ending frame positions

int frameSize = format.getFrameSize();

long startFrame = Math.round(t1 \* format.getFrameRate()) \* frameSize;

long endFrame = Math.round(t2 \* format.getFrameRate()) \* frameSize;

// Create a new output file with the same format as the input file

File outputFile = File.createTempFile("wavcut", ".wav");

AudioSystem.write(new AudioInputStream(inStream, format, endFrame - startFrame), AudioFileFormat.Type.WAVE, outputFile);

// Close the input stream and return the output file

inStream.close();

return outputFile;

}

}

The class uses the javax.sound.sampled package to read and write the .wav files. The cut() method calculates the starting and ending frames based on the starting and ending times (t1 and t2) and the audio format of the input file. It then creates a new output file with the same audio format as the input file and writes the cut segment to the file. Finally, it closes the input stream and returns the output file.

The class also includes a check to prevent it from being used to cut a file more than once. If the cut() method is called again on the same WavCutter object, it will throw an IllegalStateException.

where i can find created temp file?

The temporary file created by File.createTempFile() is usually stored in the default temporary file directory for the system. The exact location of this directory can vary depending on the operating system, but it's typically in a location that's specific to the user or the system.

You can find the default temporary file directory for your system by calling the java.io.tmpdir system property. Here's an example that prints out the location of the default temporary file directory:

java

String tempDir = System.getProperty("java.io.tmpdir");

System.out.println("Default temporary file directory: " + tempDir);

When you create a temporary file with File.createTempFile(), the file will be automatically deleted when the JVM exits. However, you can also delete the file manually by calling the File.delete() method. For example:

java

File outputFile = cutter.cut();

// Do something with the output file...

// ...

// Delete the output file when you're done with it

outputFile.delete();

how can I call this temp file for use in other class like MediaPlayer?

You can use the toURI() method of the File class to get a URI that points to the temporary file. Then, you can use this URI to create a Media object and pass it to a MediaPlayer for playback.

Here's an example that shows how to do this:

java

// Assume that you have a WavCutter class that cuts a .wav file and returns a File object

WavCutter cutter = new WavCutter("path/to/input.wav", 10.0, 20.0);

File outputFile = cutter.cut();

// Create a URI that points to the output file

URI uri = outputFile.toURI();

// Create a Media object using the URI

Media media = new Media(uri.toString());

// Create a MediaPlayer and play the media

MediaPlayer mediaPlayer = new MediaPlayer(media);

mediaPlayer.play();

In this example, the WavCutter class cuts a section of the input file from 10 seconds to 20 seconds and returns a File object that points to the output file. The toURI() method is then used to create a URI that points to the output file, which is used to create a Media object. Finally, a MediaPlayer is created with the Media object and the media is played using the play() method.