

PLAYING SOUNDS IN A JAVA FX APPLICATION

This tutorial will explore how to play sounds in a JavaFX application. The sounds used and their application will reflect our build up to creating a computer game.

CREATING THE BASIC APPLICATION

I have set up a basic JavaFX application, with event listeners for up, left and right keys. These will correspond to the jump, left/right walk. If you need a reminder on how to do this then see the earlier tutorial.

I have also added an ImageView with a simple sprite. An animation timer is used to keep moving the sprite down until he hits a theoretical floor; the KeyEvent for up repositions the sprite in the "air". My basic start method is below. All sound and image files are in the zip archive.

```
public void start(Stage primaryStage) {

    Group root = new Group();
    final ImageView iv = new ImageView(new
        Image(getClass().getResourceAsStream("DinkeyTong.png")));
    iv.setLayoutX(150);
    iv.setLayoutY(150);

    Scene scene = new Scene(root, 300, 300);
    scene.setOnKeyPressed(new EventHandler<KeyEvent>() {

        @Override
        public void handle(KeyEvent event) {
            switch (event.getCode()){
                case LEFT:
                    break;
                case RIGHT:
                    break;
                case UP:
                    iv.setLayoutY(0);
                    break;
                default:
                    break;
            }
        }
    });

    root.getChildren().add(iv);

    primaryStage.setTitle("Playing Sounds");
    primaryStage.setScene(scene);
    primaryStage.show();

    new AnimationTimer(){
        @Override
        public void handle(long l) {
            if(iv.getLayoutY()<200){
                iv.setLayoutY(iv.getLayoutY()+1);
            }
        }
    }.start();
}
```

USING AUDIOCLIP TO PLAY SOUNDS

There are different options available for playing sound in JavaFX. The first that we will try is called AudioClip. It loads the raw, uncompressed audio into memory. The use of AudioClip has several advantages:

- It loads files with minimal latency so they are available to play almost immediately
- They are “fire and forget”. This means you can call it to play and the audio clip will play and stop itself
- AudioClips can be played simultaneously.

I have modified my event listener for the key events to call a method called playSound. This is passed a string corresponding to the action performed.

```
case LEFT:
    playSound("Left");
    break;
```

In order to play a sound, a Path to the file is needed. A switch statement can be used to determine the correct file path, and then the AudioClip can be played. In the case of this tutorial, the left and right sounds are the same: footsteps.wav. I have used fall through to simplify this. A case can be implemented for jump too.

```
private void playSound(String s) {
    URL path;
    AudioClip ac;

    switch (s) {
        case "Left": //fall through to right
        case "Right":
            path = getClass().getResource("footsteps.wav");
            ac = new AudioClip(path.toString());
            ac.play();
            break;
        default:
            //do nothing
    }
}
```

You should be able to hear sounds now when the arrow keys are pressed. Try pressing the same key repeatedly. There should be a bit of a bug here!

REFINING THE JUMP

A simple technique for solving the repeated sound when jumping is to use a flag. This can be set to true when jumping to prevent the sound from being played. This is my Boolean flag to determine whether I am jumping or not.

```
public class PlayingSound extends Application {

    boolean stillJumping = true;
```

I have also modified my event listener to test if my flag. If I am not jumping then it is fine to proceed with the jump action and set the flag to true.

```
case UP:
    if (!stillJumping) {
        iv.setLayoutY(0);
        stillJumping = true;
        playSound("Jump");
    }
    break;
```

A final modification involves resetting the flag to false once I hit the hypothetical ground.

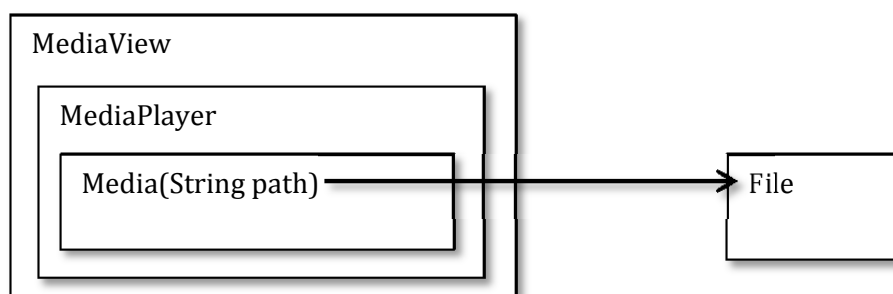
```
new AnimationTimer() {
    @Override
    public void handle(long l) {
        if(iv.getLayoutY() < 200) {
            iv.setLayoutY(iv.getLayoutY()+1);
            if(iv.getLayoutY() == 200) {
                stillJumping = false;
            }
        }
    }
}.start();
```

This should prevent a repeat jump action. It doesn't however stop the playSound from being called repeatedly while left or right are being pressed.

USING MEDIAPLAYER

MediaPlayer has an advantage over AudioClip in that it does not load the complete raw sound data into memory. It keeps the small amount of pre-prepared data in memory that it needs to play for a few seconds. This means that it is more memory efficient than AudioClip.

In order to make use of media player you need to specify the Media, use this in a MediaPlayer, which is accessed via a MediaView.



This looks something like the screenshot below. Note, I have made use of file:/ before writing the path to indicate that this is a file path and not a URL or other path.

```
Media m = new Media("file:/c:/JavaFXtutorials/PlayingSound/src/playingsound/bgLoop.aiff");
MediaPlayer player = new MediaPlayer(m);
MediaView mv = new MediaView(player);

root.getChildren().add(mv);
```

I can then configure the player and play the sound. `.setVolume()` ranges from 0 (muted) to 1.0 (a loud as possible). `.balance()` ranges from -1.0 (left channel) to 1.0 (right channel); this would let you apply effects based on the location of sprites.

```
player.setVolume(0.3);
player.setCycleCount(MediaPlayer.INDEFINITE);
player.play();
```

If you want to make use of features such as indefinite looping, you need to use the file path and not shortcuts such as `getClass().getResource("filename.extension")`. To repeat a media file a specified number of time, change the `.setCycleCount()` to the specified number, or `mediaplayer.INDEFINITE` to repeatedly loop.

MODIFYING THE FOOTSTEPS

If we think about what needs to be achieved with the footsteps, the sound file needs to loop until the walk key (LEFT or RIGHT) is released. It is possible to make an `AudioClip` loop indefinitely; the code could be changed to `ac.setCycleCount(AudioClip.INDEFINITE)`.

There are two ways of achieving this walking sound effect.

The first:

1. Declare a global flag to indicate whether the walk movement has been initiated.
2. Initialise this in the start method.
3. Modify the event listener for the walk keys to:
 - a. check if the flag is currently false
 - b. set the it to true
 - c. and then call `playSound`.
4. Add an action listener to listen for `onKeyReleased`.
5. When the walk keys are released, stop the sound clip and set the walk flag to false.

The second:

1. Declare a global `MediaPlayer` for playing footsteps.

```
public class PlayingSound extends Application {

    boolean stillJumping = true;
    MediaPlayer playFootsteps;
```

2. Initialise this in the start method to the footsteps sound effect, remembering to set it to play indefinitely. (Complete path not shown in screenshot.)

```
Media foot = new Media("file:/c:/JavaFXtutorials/PlayingS  
playFootsteps = new MediaPlayer(foot);  
MediaView mvFoot = new MediaView(playFootsteps);  
playFootsteps.setCycleCount(MediaPlayer.INDEFINITE);
```

3. Add an event listener for key released:

```
@Override  
public void handle(KeyEvent event) {  
    switch (event.getCode()) {  
        case LEFT:  
        case RIGHT:  
            playFootsteps.stop();  
            break;  
        default:  
            break;  
    }  
}
```

4. Modify the original event listener for key pressed to play the new MediaPlayer for footsteps.

SOURCES OF SOUND FILES:

Background loop: <http://freesound.org/people/JimiMod/sounds/217260/>

Footsteps: http://freesound.org/people/Iberian_Runa/sounds/243794/

Jump: <http://freesound.org/people/CommanderRobot/sounds/264828/>