Reproducción multimedia con Java

Sin añadir nuevas bibliotecas (desaprobado) a la aplicación, es muy sencillo que reproduzca audio en formato **WAV o MIDI**, aunque podría añadirse para reproducir formatos *AIFC, AIFF, AU o SND* con el mismo código que para reproducir sonido WAV, pues son los tipos de audio admitidos en la actualidad por el **API de Java Sound** (*javax.sound*).

El API de Java Sound (paquete ***javax.sound***) proporciona dos modos de reproducir audio: usando ***Clip*** o ***SourceDataLine***, cada una con sus ventajas e inconvenientes. Sin embargo, dada la popularidad del formato MP3, y pese a los problemas con las licencias, existe la posibilidad de reproducir MP3 por medio de bibliotecas como JavaFX (ahora gestionado por OpenJFX) o bibliotecas de terceros.

I. Reproducción de audio MP3 (y otros) con Java FX

Aunque el API **Java Sound no incorpora la posibilidad de reproducir MP3 de forma nativa** por temas de licencia, existen muchos modos de hacerlo, incluso con bibliotecas de clase estándar y/o libres.

Durante mucho tiempo se requería el uso de bibliotecas de clases de terceros que había que incluir en nuestros proyectos (se trata de descargar el *.jar e incorporarlo a las bibliotecas del proyecto o incluir las bibliotecas dentro del propio IDE para incluirla en todos los proyectos).

Una de esas bibliotecas de clases Java que permite ejecutar MP3 es **JLayer**:

http://www.javazoom.net/javalayer/sources.html

Sin embargo, Oracle también soportó de manera oficial la reproducción de MP3 por medio del **framework** de Java **JavaFX** (atractiva alternativa a Swing para aplicaciones visuales). El tratamiento de archivos MP3 se incorporó desde la versión de JavaFX 2.0, por medio de las clases **Media** (*javafx.scene.media.Media*) y **MediaPlayer** (*javafx.scene.media.MediaPlayer*), que representan los recursos de media y el control de reproducción de esos medios, respectivamente.

https://es.wikipedia.org/wiki/JavaFX

Aunque inicialmente JavaFX venía incorporado dentro del JDK, Java SE, entre las versiones 8 y 10, fue separado en la versión JDK 11 e incorporado al proyecto OpenJFX:
https://openjfx.io/
https://openjdk.java.net/projects/openjfx/

OpenJFX puede descargarse desde la página https://openjfx.io/ o emplear versiones anteriores del JDK. Sin embargo, en un proyecto Maven sólo se precisa añadir la correspondiente dependencia:

```
<dependencies>
  <!-- https://mvnrepository.com/artifact/org.openjfx/javafx-media -->
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-media</artifactId>
    <version>16-ea+7</version>
  </dependency>
</dependencies>
```

Así, una vez la biblioteca está en nuestro proyecto, puede reproducirse un archivo MP3 de modo sencillo con el siguiente código:

```
String cancion = "gavotte.mp3";
Media media = new Media(new File(cancion).toURI().toString());
MediaPlayer mediaPlayer = new MediaPlayer(media);
mediaPlayer.play();
```

A continuación se detallan los pasos para reproducir un archivo de audio (MP3, AAC, PCM o Vídeo H.264/AVC):

1. Crea un **objeto de tipo *Media*** (*javafx.scene.media.Media*) apuntado al recurso multimedia (archivo) correspondiente.

   Media tiene un **único constructor**, que es el único modo de especificar el archivo fuente del medio:

   public Media(String source)

   *Recoge un String que debe representar **una URI válida e inmutable**. Las URL válidas pueden ser: **HTTP**, **HTTPS**, **FILE** o **JAR**. Si la URL no es válida lanza una excepción de tipo **IllegalArgumentException**.*

   Si el archivo usa un protocolo no bloqueante, como FILE, puede producirse una excepción del tipo MediaException (si no es accesible), que también se lanza si el formato no es admitido.

   El formato de la URI admitida viene especificado en la RFC-2396, tal y como lo requiere java.net.URI.

2. Crea un objeto ***MediaPlayer*** a partir del objeto Media anterior.

   Tiene un único constructor que recoge el objeto de tipo Media, y es el único

modo de asociar un objeto Media al ***MediaPlayer***:

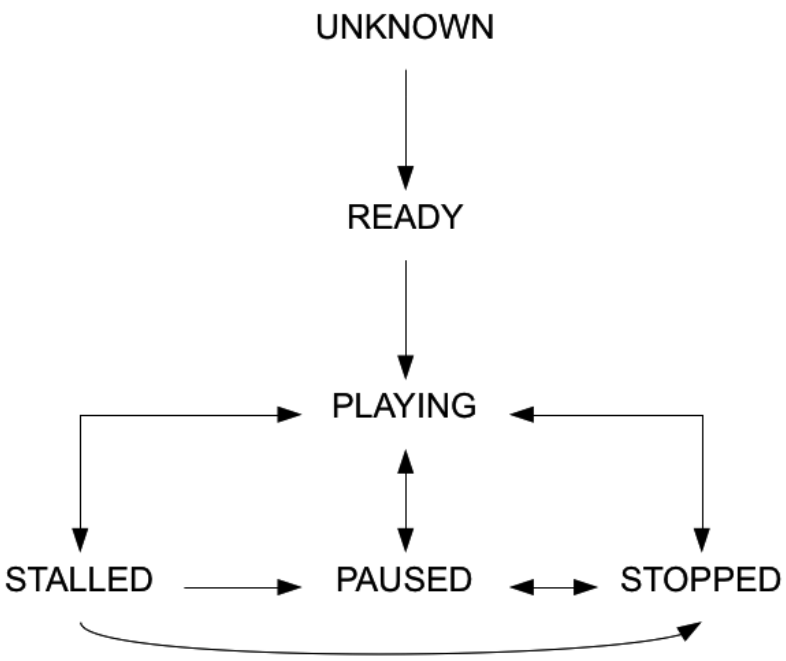
public MediaPlayer(Media media)

Una vez ha sido creado no puede cambiarse.

3. Invoca al método *play()* de MediaPlayer.
   Si previamente estaba en pausa, seguirá ejecutándose en ese punto. Si estaba parado empezará ejecutarse desde el punto de inicio. Se puede consultar el estado (STATUS de la ejecución).

   - Un Objeto Media puede ser compartido por muchos MediaPlayer.
   - Además de con el método "play()", puede usarse el método "***setAutoPlay(true)***" para que lo reproduzca tan pronto le sea posible.
   - Los estados de reproducción están definidos en **MediaPlayer.Status**.
   - Un archivo de sólo audio puede reproducirse usando la clase *javafx.scene.media.**AudioClip***: https://openjfx.io/javadoc/15/javafx.media/javafx/scene/media/AudioClip.html).
     Recomendada para ejecuciones de baja latencia o audios breves.

Estado por el que pasa un *MediaPlayer*:



Ejemplo completo (con *MediaView*, un elemento visual del reproductor con Java FX):

```
String source;
Media media;
MediaPlayer mediaPlayer;
MediaView mediaView;
try {
   media = new Media(source);
   if (media.getError() == null) {
      media.setOnError(new Runnable() {
      public void run() {
          // Handle asynchronous error in Media object.
      }
       });
     try {
      mediaPlayer = new MediaPlayer(media);
      if (mediaPlayer.getError() == null) {
          mediaPlayer.setOnError(new Runnable() {
         public void run() {
             // Handle asynchronous error in MediaPlayer object.
         }
          });
          mediaView = new MediaView(mediaPlayer);
          mediaView.setOnError(new EventHandler<MediaErrorEvent>() {
         public void handle(MediaErrorEvent t) {
             // Handle asynchronous error in MediaView.
         }
          });
      } else {
          // Handle synchronous error creating MediaPlayer.
      }
       } catch (Exception mediaPlayerException) {
      // Handle exception in MediaPlayer constructor.
       }
   } else {
       // Handle synchronous error creating Media.
   }
} catch (Exception mediaException) {
// Handle exception in Media constructor.
}
```

Media de JavaFX admite varios tipos de codificación (audio o vídeo):

Codificación	Tipo	Descripción
AAC	Audio	*Compresión de audio Advanced Audio Coding*
MP3	Audio	*Audio Raw MPEG-1, 2, y 2.5; layers I, II, y III; admite todas las combinaciones de frecuencias de muestreo y bit rates. Al archivo debe contener al menos 3 frames MP3*
PCM	Audio	Sin compresión, *raw audio samples*
H.264/AVC	Vídeo	*Compresión de vídeo H.264/MPEG-4 Part 10/AVC (Advanced Video Coding)*

Los contenedores de archivo indican cómo se almacenan los archivos de audio, vídeo u otro tipo dentro del archivo. Cada tipo de contenedor está asociado a uno o más tipo MIME, una extensión de archivo, y firmas de archivo (los valores iniciales de la cabecera del archivo). Java FX Media admite los siguientes contenedores (tipo de archivo):

Cont.	Descripción	Cod. Vídeo	Cod. Audio	Tipo MIME	Ext.
AIFF	*Audio Interchange File Format*	-	PCM	audio/x-aiff	.aif .aiff
HLS (*)	*MP2T HTTP Live Streaming (audiovisual)*	H.264/AVC	AAC	application/vnd.apple.mpegurl audio/mpegurl	.m3u8
HLS (*)	*MP3 HTTP Live Streaming (sólo audio)*		MP3	application/vnd.apple.mpegurl audio/mpegurl	.m3u8
MP3	*MPEG-1, 2, 2.5 Flujo de audio raw con metadatos ID3v2.3 o v2.4*	-	MP3	audio/mpeg	.mp3
MP4	*MPEG-4 Part 14*	H.264/AVC	AAC	video/mp4, audio/x-m4a video/x-m4v	.mp4 .m4a .m4v
WAV	*Waveform Audio Format*	-	PCM	audio/x-wav	.wav

(*) HSL es tanto un protocolo como un contenedor.

También existen bibliotecas *Open Source* o libres para reproducción de MIDI, partituras, etc.. como JFugue:

http://www.jfugue.org/

II. Interface Clip

Con la interface ***Clip***, que representa un tipo de línea de datos de audio que puede ser **precargado para ser reproducido**, en vez de usar *streaming* en tiempo real. Por ello, puede **reproducirse desde cualquier punto o hacer bucles para ser reproducidos repetidamente**.

Loa clips pueden obtenerse a partir de un ***Mixer*** que admite líneas y los datos de audio se cargan en un *clip* cuando son abiertos. Pueden reproducirse y pararse con los métodos *start()* y *stop()*. Para volver al principio hay que parar el *clip* y llamar al método *setFramePosition(0)* o *setMicrosecondPosition(0);*

Debería usarse *Clip* (*javax.sound.sampled.Clip*) cuando se quiere **ejecutar un sonido que no sea en tiempo real, como un archivo de sonido corto. El archivo se carga en memoria antes de ser reproducido**, lo que da **control total sobre su reproducción**.

- Reproducir desde cualquier posición mediante: ***setMicrosecondPosition****(long) o* ***setFramePosition****(int).*
- Reproducir en bucle parte o todo el sonido: ***setLoopPoints****(int, int) y* ***loop****(int)*.
- Saber la duración del sonido antes de ser reproducido: ***getFrameLength****() o* ***getMicrosecondLength****()*.
- Parar y reproducir desde la posición actual: ***stop****()* y ***start****()*.

Sin embargo, **no es eficiente para reproducir grandes tamaños de archivos de sonido** porque consumen mucha memoria. Además, el método *start()* ejecuta el sonido, pero no bloquea el hilo de ejecución en curso, por lo que **se requiere implantar la interface *LineListener* para saber cuándo termina de ejecutarse el sonido**.

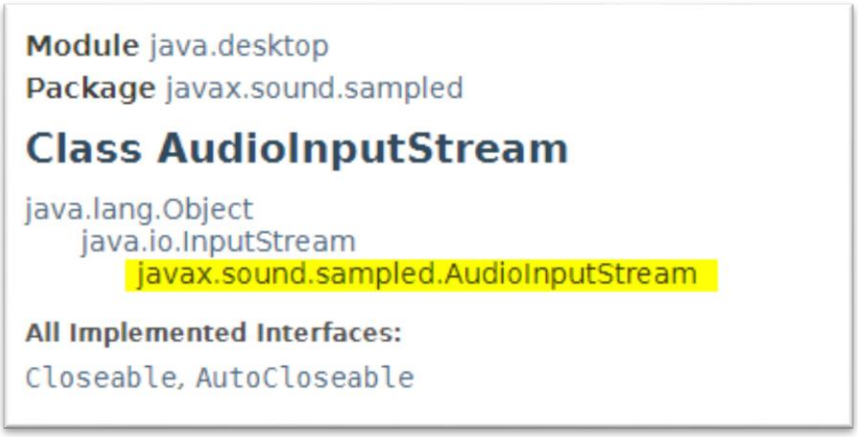
Cómo reproducir sonido con Clip

Hay varios modos de crear un Clip de audio. El siguiente código muestra un ejemplo de hacerlo de manera sencilla:

```
Clip clip = AudioSystem.getClip();
AudioInputStream inputStream =
    AudioSystem.getAudioInputStream(flujo de entrada a un fichero);
clip.open(inputStream);
```

clip.start();

1) Crear un objeto (flujo de entrada de audio) de tipo ***AudioInputStream*** al archivo de audio (WAV,…):
File *archivoAudio* = ***new File****(rutaArchivoAudio);*
AudioInputStream ***audioIS*** = ***AudioSystem.getAudioInputStream****(archivoAudio);*


Module java.desktop
Package javax.sound.sampled
Class AudioInputStream
java.lang.Object
java.io.InputStream
javax.sound.sampled.AudioInputStream
All Implemented Interfaces:
Closeable, AutoCloseable

2) Obtener el formato de audio y crear un objeto de tipo ***DataLine.Info***:
AudioFormat *formato* = ***audioIS.getFormat****();*
DataLine.Info *info* = *new* ***DataLine.Info****(****Clip.class****,* *formato);*

3) Obtener el Clip de audio:
Clip *audioClip* = ***(Clip)*** ***AudioSystem****.getLine(info);*

4) **Abrir el flujo de tipo** *AudioInputStream* **y reproducirlo**:
*audioClip****.open****(****audioIS****);*
*audioClip****.start****();*

5) Cerrar el Clip y el flujo:
*audioClip****.close****();*
audioIS.close*();*

III. Interface SourceDataLine

El uso de *SourceDataLine* (*javax.sound.sampled.SourceDataLine*) es recomendado cuando se quiere **reproducir archivos de sonido grandes que no pueden ser precargados en memoria o para datos de sonido en *streaming* en tiempo real**, así como reproducir sonidos que se están capturando.

- Es eficiente para **archivos grandes o sonido en streaming**.

- Es posible controlar que los datos de sonido sean escritos en buffers de reproducción de líneas de audio.
- A diferencia de la interface Clip, **no se necesita gestionar los eventos con la interface *LineListener* para saber cuándo se completa la reproducción del sonido**.

Sin embargo:

- No se puede iniciar la reproducción en una posición arbitraria.
- No se puede ejecutar en bucles parte del sonido (loop).
- No se puede parar y reanudar la reproducción en el medio.
- No se puede conocer la duración del sonido antes de ser ejecutado.

Cómo reproducir sonido con SourceDataLine

1) Como con la interface Clip, es **necesario obtener una referencia a un objeto que implante la interface *SourceDataLine***:

```
File archivoAudio = new File(rutaArchivoAudio);
AudioInputStream audioIS = AudioSystem.getAudioInputStream(archivoAudio);
```

2) Obtener el formato de audio y crear un objeto de tipo ***DataLine.Info***:

```
AudioFormat formato= audioIS.getFormat();
DataLine.Info info = new DataLine.Info(SourceDataLine.class, formato);
```

3) Obtener objeto de línea de audio de tipo *SourceDataLine* (igual que Clip):

```
SourceDataLine lineaDeAudio = (SourceDataLine) AudioSystem.getLine(info);
```

4) Abrir la línea de audio ***SourceDataLine*** y reproducirla:

```
lineaDeAudio.open(formato);
lineaDeAudio.start();
```

5) Como se trata de un flujo, hay que **leer un bloque de bytes** de tipo ***AudioInputStream*** y enviarlo al buffer de reproducción de *SourceDataLine* hasta que llegue al final:

```
final int TAM_BUFFER = 4096; // 4 KB de buffer

byte[] bytesSonido = new byte[TAM_BUFFER];
int bytesLeidos = -1; // la marca de fin de lectura es -1 devuelta por el método "read(...)"

while ((bytesLeidos = audioIS.read(bytesSonido)) != -1) {
```

```
    lineaDeAudio.write(bytesSonido, 0, bytesLeidos);
}
```

6) Cerrar y liberar recursos:

```
lineaDeAudio.drain();
lineaDeAudio.close();
audioIS.close();
```