PRÁCTICA 13

Sonido

Descripción de la práctica

El objetivo de esta práctica es realizar una aplicación que permita reproducir y grabar audio usando la *Java Sound API*. El aspecto visual de la aplicación será el mostrado en la Figura 1, donde tendremos una barra de herramientas para el audio con los siguientes elementos[1]:

- Lista de reproducción, que mostrará la relación de audios que se pueden reproducir (al abrir un nuevo fichero de audio éste se incluiría en la lista; de igual forma, se añadirán los sonidos grabados desde la aplicación).
- Un botón para reproducir sonido. Al pulsar el botón, se reproduciría el audio que esté seleccionado en ese momento en la lista de reproducción.
- Un botón para parar la reproducción o la grabación.
- Un botón para grabar audio, de forma que al pulsarlo se iniciará el proceso de grabación, que terminará cuando se pulse el botón de parada. El sonido se grabará en el fichero indicado por el usuario y se añadirá a la lista de reproducción.

En el menú se incluirá la opción "Archivo" que tendrá a su vez dos opciones: "Abrir audio" y "Grabar audio". La primera deberá lanzar el correspondiente diálogo y añadir el fichero seleccionado a la lista de reproducción.

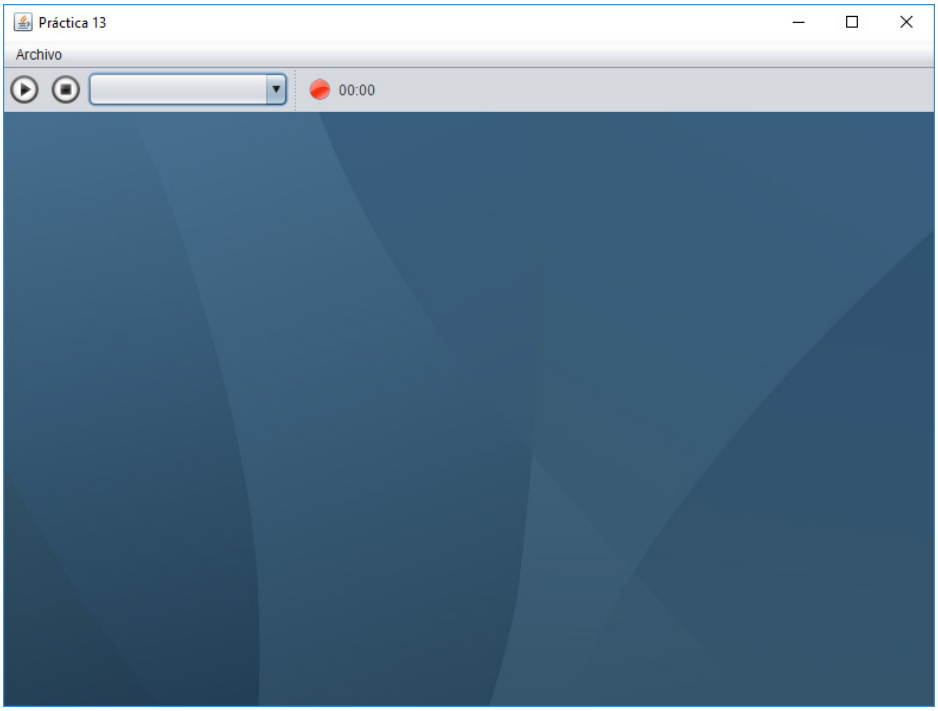

Figura 1: Aspecto de la aplicación

[1] La práctica se puede hacer continuando la prácticas 8-12. En ese caso, la barra de herramientas de audio se añadirá a continuación de la barra de herramientas ya existente (que incluía la funcionalidad de dibujo).

Lista de reproducción

La lista de reproducción será una lista desplegable (`JComboBox`) cuyos elementos serán objetos de tipo[2] `File`. En el menú "Archivo" se incluirá la opción "Abrir audio", que lanzará el diálogo abrir para que el usuario selecciones un fichero de sonido; el fichero seleccionado deberá añadirse a la lista de reproducción.

Al mostrar los elementos de una lista desplegable, por defecto se llama al método `toString` asociado a la clase de los elementos de dicha lista. En el caso de la clase `File`, el método `toString` devuelve el nombre del fichero con la ruta completa. Para que sólo aparezca el nombre corto, sería necesario sobrescribir el método `toString`; una forma sencilla de hacerlo es mediante el uso de clases anónimas de la siguiente forma:

```
File f = new File( parámetros ){
  @Override
  public String toString(){
    return this.getName();
  }
};
```

En nuestro caso, el parámetro que se le pasará al constructor será la ruta del fichero seleccionado por el usuario. El objeto `File` creado se añadirá a la lista desplegable mediante el método[3] `addItem`.

```
listaReproduccion.addItem(f);
```

Reproducir sonido

Al pulsar el botón de reproducción situado en la barra de herramientas, se reproduciría el sonido que esté seleccionado en ese momento en la lista de reproducción. Para ello haremos uso del paquete `sm.sound` adjunto a esta práctica y de su clase `SMPlayer`. Así, asociada a nuestra aplicación tendremos un reproductor[4]:

```
SMClipPlayer player = null;
```

que crearemos cuando se pulse el botón de reproducir:

```
File f = (File)listaReproduccion.getSelectedItem();
if(f!=null){
  player = new SMClipPlayer(f);
  if (player != null) {
    player.play();
  }
}
```

Por otro lado, al pulsar el botón de parada deberá de detenerse la reproducción (en este caso se hará uso del método `stop()` de la clase `SMPlayer`)[5].

---

[2] En NetBeans, para indicar el tipo de elementos del *JComboBox* se puede usar la sección de propiedades; concretamente, en "*Código→Parámetros del tipo*". Esto implicará que la variable se declarará como *JComboBox<File>*.

[3] Si queremos que el elemento añadido aparezca seleccionado, habrá que seleccionarlo mediante la llamada al método *setSelectedItem*.

[4] Para este ejemplo, usaremos la reproducción basada en *Clip* de la clase *SMClipPlayer*

[5] Nótese que, con esta implementación, el sonido se reproducirá desde el principio cada vez que se pulse el botón de reproducción. Una posible mejora sería incorporar la opción de la pausa.

Grabar sonido

Al pulsar el botón de grabación situado en la barra de herramientas, deberá de comenzar el proceso de grabación de sonido. Dado que el sonido se deberá de grabar en un fichero, podría optarse por (1) preguntar el nombre del fichero al principio del proceso o (2) cuando el usuario pulse el botón de parada: para esta práctica preguntaremos al principio del proceso (más sencillo)[6].

En este caso haremos uso de la clase `SMRecorder` para crear un grabador de audio vinculado al fichero seleccionado. Al igual que para el caso de la reproducción, asociada a nuestra aplicación tendremos un grabador:

```
SMSoundRecorder recorder = null;
```

que crearemos una vez seleccionado el fichero; así, asumiendo que se ha lanzado el diálogo de guardar (`dlg`), la grabación empezaría tras la llamada al método `record()`:

```
File f = dlg.getSelectedFile();
recorder = new SMSoundRecorder(f);
  if(recorder!=null){
    recorder.record();
  }
```

La grabación finalizará cuando se pulse el botón de parada; para ello, en el evento acción asociado a dicho botón habrá que llamar al método `stop()` para detener la grabación. Nótese que dicho botón se usará tanto para detener la reproducción (sección anterior) como para finalizar la grabación. Esta circunstancia deberá de tenerse en cuenta a la hora de implementar el método asociado a la selección de dicho botón. Una vez finalizada la grabación, el nuevo audio se incorporará a la lista de reproducción.

Gestión de eventos

Tanto en el caso de la reproducción como el de la grabación, es posible asociarle al `SMPlayer` y al `SMRecorder` un manejador de eventos[7]. Para ello, primero habrá que crear el objeto manejador que implemente el interfaz `LineListener` y activarlo mediante la llamada al método `addLineListener` de las clases `SMPlayer` y `SMRecorder`[8]. Esto permitiría, entre otros, conocer cuando acaba o empieza la reproducción/grabación. Por ejemplo, el siguiente código define una clase manejadora de eventos `LineEvent`:

```
class ManejadorAudio implements LineListener {
  @Override
  public void update(LineEvent event) {
    if (event.getType() == LineEvent.Type.START) {
    }
    if (event.getType() == LineEvent.Type.STOP) {
    }
    if (event.getType() == LineEvent.Type.CLOSE) {
    }
  }
}
```

[6] Lo habitual en los programas de grabación es la segunda opción. Implicaría guardar el sonido en un fichero temporal y luego renombrarlo y reubicarlo (o eliminarlo) en función de lo que indicase el usuario en el diálogo de guardar.

[7] El manejador realmente está asociado a los objetos `Line` que hay definidos en las clases `SMPlayer` y `SMRecorder`.

[8] La llamada al método `addLineListener` implicará internamente la llamada al método `addLineListener` que asocia el manejador al objeto `Line` definido en `SMPlayer` y `SMRecorder`.

Para asociar un manejador de la clase anterior a un objeto reproductor/grabador, se usaría el siguiente código[9]:

```
player.addLineListener(new ManejadorAudio());
```

con *player* un objeto *SMPlayer* o *SMRecorder*.

En este caso, resulta especialmente útil controlar el momento en el que empieza o se detiene la reproducción y/o grabación Por ejemplo, si quisiéramos deshabilitar el botón de reproducción mientras se está reproduciendo y volver a habilitarlo cuando se termine:

```
if (event.getType() == LineEvent.Type.START) {
      botonPlay.setEnabled (false);
}
if (event.getType() == LineEvent.Type.STOP) {
      botonPlay.setEnabled (true);
}
```

Posibles mejoras para trabajar en casa…

Una vez realizada la práctica, se proponen una serie de mejoras para darle mayor funcionalidad y mejorar el interfaz. Si consideramos esta práctica como continuación de las 8-12, una primera mejora sería:

- Usar una única opción "Abrir" que gestione la lectura tanto de imágenes como de audio (considerando internamente las diferentes casuísticas). En este caso, el filtro del diálogo incluiría tanto tipos de archivo de imágenes como de sonido

Centrándonos en mejoras asociadas al proceso de reproducción y grabación, se proponen las siguientes ampliaciones:

- Incluir un botón de pausa que permita pausar y reanudar la reproducción (que continuaría por donde estaba, no desde el principio como en el ejemplo de la práctica). Lo recomendable sería que se usara el mismo botón de reproducción para ambas tares: al pulsar el botón de reproducción por primera vez, éste cambiaría su icono al de pausa (y, con ello, su función); si se pulsa el de pausa, cambiaría su icono al de reproducción (y, con ello, su función, que en este caso sería la de reanudar). Cuando se pulse el botón de parada, o cuando llegue el sonido al final, volverá a activarse el icono de reproducción (con su funcionalidad inicial)[10].

- Incluir en la barra de herramientas, o en la barra de estado, información sobre la duración del audio y el tiempo de reproducción transcurrido (que se actualice cada segundo). Esta información puede ser mostrada, por ejemplo, en forma de etiqueta (como en el caso de la Figura 1)[11]

- Incluir un temporizador en el que empiece a correr el tiempo cuando el usuario pulse el botón de grabación (véase Figura 1)[12].

- En la grabación, preguntar el nombre del fichero al final del proceso de grabación (incluyendo la opción de cancelar)[6].

---

[9] Este código se incluiría después de la creación del objeto reproductor/grabador. El método *addLineListener* está definido en las clases *SMClipPlayer*, *SMSoundPlayer* y *SMSoundRecorder* (es decir, en aquellas clases en los que haya una línea asociada). Por este motivo, si la variable se ha declarado de tipo *SMPlayer* o *SMRecorder*, será necesario hacer el casting (o declarar la variable de la subclase usada).

[10] Para conseguir esta "multifunción" en un mismo botón, es necesario gestionar los eventos generados por el reproductor.

[11] La actualización del tiempo de reproducción, necesita gestionarse mediante una hebra.

[12] Implica el uso de hebras.