PRÁCTICA 12

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 el escritorio podrá tener dos tipos de ventanas[1]:

- Ventana de reproducción, que tendrá dos botones asociados a la reproducción del sonido y a la parada del mismo respectivamente (cada ventana irá asociada a un fichero de audio).
- Ventana de grabación, que tendrá dos botones: uno asociado al inicio de la grabación y otro a la finalización de la misma

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 crear una nueva ventana interna de reproducción que permita reproducir el fichero abierto. La opción "Grabar audio" lanzará, en primer lugar, el diálogo de guardar para indicar el fichero donde guardar la grabación; posteriormente creará una nueva ventana interna de grabación para llevar a cabo el proceso.

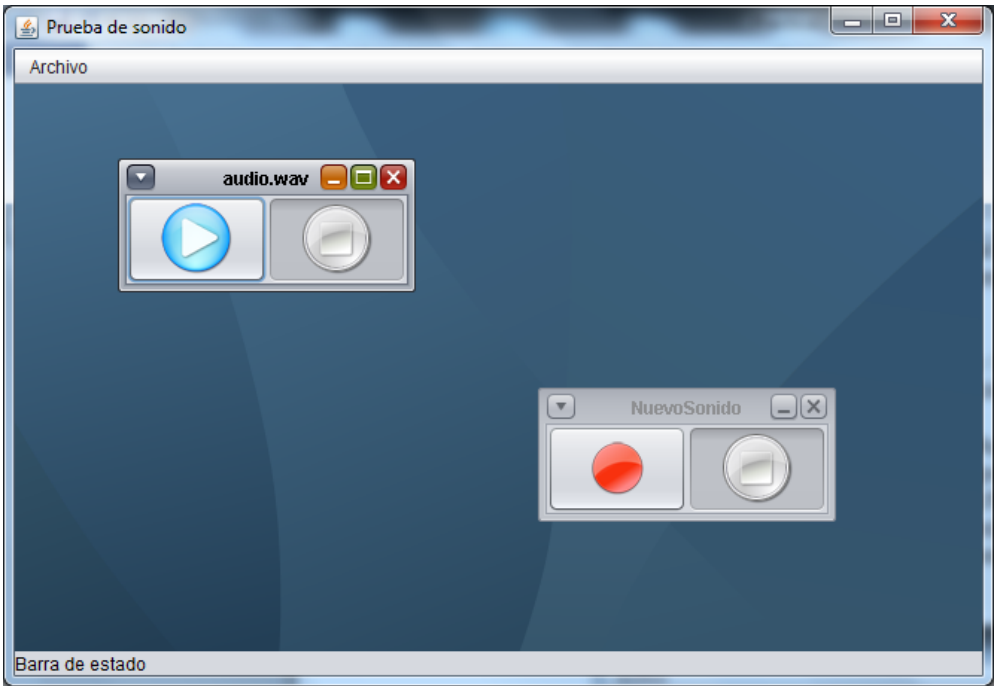


Figura 1: Aspecto de la aplicación

[1] En caso de continuar con la práctica 11, además de los dos nuevos tipos de ventanas de audio incorporadas en esta práctica, tendremos también las ventanas internas de imágenes. De ser así, hay que tener en cuenta que el método `getSelectedFrame()` puede devolver diferentes tipos de ventanas internas (imagen, reproductor o grabador).

Ventana de reproducción

La ventana de reproducción tendrá dos botones (del tipo “dos posiciones” y agrupados) que permitirán la reproducción y parada del sonido asociado a esa ventana. Para desarrollar esta ventana haremos uso del paquete `sm.sound` adjunto a esta práctica y de su clase `SMPlayer`.

Algunas recomendaciones:

- Crear una clase `VentanaInternaReproductor` que herede de `JInternalFrame` y que contenga una variable de tipo `SMPlayer`. El constructor de dicha clase creará el objeto `SMPLayer` asociado a la ventana, para lo cual se recomienda pasar el fichero a reproducir como parámetro del constructor:

```
SMPlayer player;
```

```
public VentanaInternaReproductor(File f) {
  initComponents();
  player = new SMClipPlayer(f);
}
```

- La selección de los botones implicará la llamada a los métodos `play()` y `stop()`:

```
private void botonPlayActionPerformed(ActionEvent evt) {
  if(player!=null)  player.play();
}
```

```
private void botonStopActionPerformed(ActionEvent evt) {
  if(player!=null) player.stop();
}
```

- Al cerrar la ventana de reproducción, asegurarse de llamar al método `stop()` en caso de que se esté reproduciendo.

Ventana de grabación

La ventana de grabación tendrá dos botones (del tipo “dos posiciones” y agrupados) que permitirán el inicio de la grabación y su posterior parada. Para desarrollar esta ventana haremos uso del paquete `sm.sound` y de su clase `SMRecorder`. En este caso, se seguirán las mismas recomendaciones que para la ventana de reproducción, pero adaptadas al caso del reproductor:

- Definir una clase `VentanaInternaGrabador` que herede de `JInternalFrame` y que contenga una variable de tipo `SMRecorder` (el constructor de dicha clase creará el objeto `SMRecorder` asociado a la ventana, para lo cual se recomienda pasar el fichero a reproducir como parámetro del constructor).
- La selección de los botones implicará la llamada a los métodos `record()` y `stop()`.

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[2]. Para ello, primero habrá que crear el objeto manejador que implemente el interfaz `LineListener` y activarlo mediante la llamada al

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

método `setLineListener` de las clases `SMPlayer` y `SMRecorder`[3]. 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) {
      //TODO
    }
    if (event.getType() == LineEvent.Type.STOP) {
      //TODO
    }
    if (event.getType() == LineEvent.Type.CLOSE) {
      //TODO
    }
  }
}
```

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

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

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

Ejercicio: Usar la gestión de eventos para cambiar el icono de los botones según el estado de la reproducción/grabación, de modo que se muestre en color "inactivo" cuando la operación no proceda (por ejemplo, mostrar con color inactivo el botón reproducir cuando se esté reproduciendo).

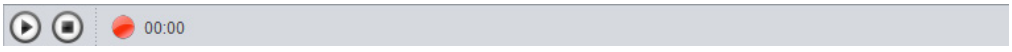
Posibles mejoras para trabajar en casa…

Una vez realizada la práctica, se proponen una serie de mejoras para darle mayor funcionalidad y mejorar el interfaz. En primer lugar, y manteniendo el actual diseño de las ventanas de reproducción/grabación, una primera mejora sería:

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

Una mejora de mayor calado implicaría incorporar una barra de herramientas de audio que centralizase las acciones de reproducción/grabación. Nótese que, con la implementación actual, cada ventana tiene sus propios botones de reproducción/parada o grabación/parada. Esta mejora implicaría:

- Usar los mismos botones para gestionar la reproducción/grabación de todas las ventanas internas de audio (esto implicaría descartar las ventanas `VentanaInternaReproductor` y `VentanaInternaGrabador` en su diseño actual). Estos botones estarían en una barra de herramientas como, por ejemplo:

  Los botones play/stop manejarán la reproducción de la ventana de audio activa (habrá tantas ventanas como audios se hayan abierto), mientras que el botón "*rec*" lanzará el proceso de grabación (y, en este caso, no habrá ventana interna asociada).

- La ventana de reproducción habría que rediseñarla, ya que en este nuevo enfoque no tendría sentido que incluyese botones de reproducir/parar. Para esta mejora, la

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

ventana tendría que mostrar información específica del audio que se está reproduciendo, como, por ejemplo, la duración o el "progreso" de la reproducción. La siguiente figura muestra un posible diseño, en el que se indica la duración del audio (parte derecha), evolución de la reproducción en una barra de progreso (parte central) y tiempo de reproducción actual (parte izquierda)[4]:

- En este caso, no tendríamos ventana de grabación (todo se gestionaría desde la barra de herramientas). Se propone incluir un temporizador en el que empiece a correr el tiempo cuando el usuario pulse el botón de grabación[5]; el botón se cambiará a modo "stop", de forma que la grabación parará cuando el usuario lo pulse[6]. Además de lo anterior, será necesario preguntarle al usuario el nombre del fichero donde se guardará el audio; se puede optar por preguntar al principio del proceso (como en la práctica) o hacerlo al final del proceso, en la línea de las aplicaciones profesionales (se recomienda esta última).

[4] La actualización de la barra de progreso, así como del tiempo de reproducción, necesita gestionarse mediante una hebra.
[5] Implica el uso de hebras.
[6] Otra opción sería tener dos botones en la barra de herramientas: una para iniciar la grabación y otro para detenerla.