PRÁCTICA 14

Reproducción y captura de vídeo

Descripción de la práctica

El objetivo de esta práctica es realizar una aplicación que permita (1) reproducir medios continuos, (2) acceder a la cámara y (3) capturar imágenes instantáneas. El aspecto visual de la aplicación será el mostrado en la Figura 1, donde distinguimos tres tipos de ventanas[1]:

- Ventana de reproducción, que permitirá visualizar vídeos[2] (cada ventana irá asociada a un fichero). Esta ventana, además, incluirá botones de control que permitan reproducir/parar el vídeo..
- Ventana de cámara, que mostrará el vídeo captado a través de la webcam.
- Ventana de imagen[3], que mostrará instantáneas capturadas de los vídeos/cámara mostrados en las ventanas anteriores.

En el menú "Archivo" se incluirá la opción "Abrir video", que lanzará el correspondiente diálogo y creará una nueva ventana interna de reproducción que permita reproducir el fichero abierto. Además, se incluirá una barra de herramientas[4] que contenga dos botones: uno correspondiente a la opción "Cámara", que lanzará una ventana que muestre la secuencia que esté captando la webcam, y otro asociado a la opción "Captura" que permitirá la captura de imágenes de la cámara o del vídeo (véase Figura 1).

Para llevar a cabo esta práctica usaremos dos aproximaciones: (1) una primera basada en la API del *Java Multimedia Framework* y (2) otra segunda basada en bibliotecas de código abierto más actualizadas. Realmente bastaría usar solo una de las dos aproximaciones, ya que ambas abordan los mismos objetivos de reproducción, uso de webcam y captura; de hecho, y pensando en desarrollar una práctica actualizada en lo que a formatos y códecs se refiere, lo más adecuado sería trabajar directamente con el segundo enfoque (bibliotecas de código abierto). No obstante, con el objetivo de probar la JMF, se plantea el doble enfoque en esta práctica.

Concretamente, seguiremos la siguiente secuencia:

- Reproducción y captura de instantáneas usando *JMF*[5]
- Uso de webcam basándose en la *Webcam Capture API*
- Reproducción y captura de instantáneas usando *VLCj*

---

[1] La práctica se puede hacer continuando la prácticas 8-13. En este caso, el hecho de que haya diferentes tipos de ventanas internas, cada una correspondiente a una clase distinta, hay que tenerlo en cuenta a la hora de usar el método `getSelectedFrame`; por ejemplo, si se va a aplicar una operación sobre una imagen hemos de comprobar que la ventana activa es del tipo adecuado (es decir, que contiene una imagen). En las prácticas 8-13 se asumía que todas las ventanas internas eran del mismo tipo, por lo que se hacía directamente el casting y se accedía al lienzo y su imagen; tras esta práctica, esto ya no tiene por qué ser siempre así (el método *`getSelectedFrame`* puede devolver ventanas de diferentes tipos, por lo que hay que tenerlo en cuenta antes de hacer el casting). Para abordar este punto, se aconseja crear una superclase `VentanaInterna` de la que hereden diferentes subclases correspondientes a diferentes tipos de ventanas

[2] Gracias al uso de JMF y/o VLC, también permitirá reproducir audio sin necesidad de código adicional.

[3] Esta ventana corresponde a la ventana interna para mostrar imágenes desarrollada en las prácticas 8-12.

[4] En caso de continuar con las prácticas 8-13, la barra de herramientas de video se añadirá a continuación de las barras de herramientas ya existentes.

[5] Permitirá ver/oír vídeos, si bien el comportamiento dependerá de si se ejecuta usando plataforma de 32 o 64 bits, pudiéndose dar el caso de que se vea pero no se oiga (o viceversa). Además, estará restringido a formatos y códecs anteriores a 2003.

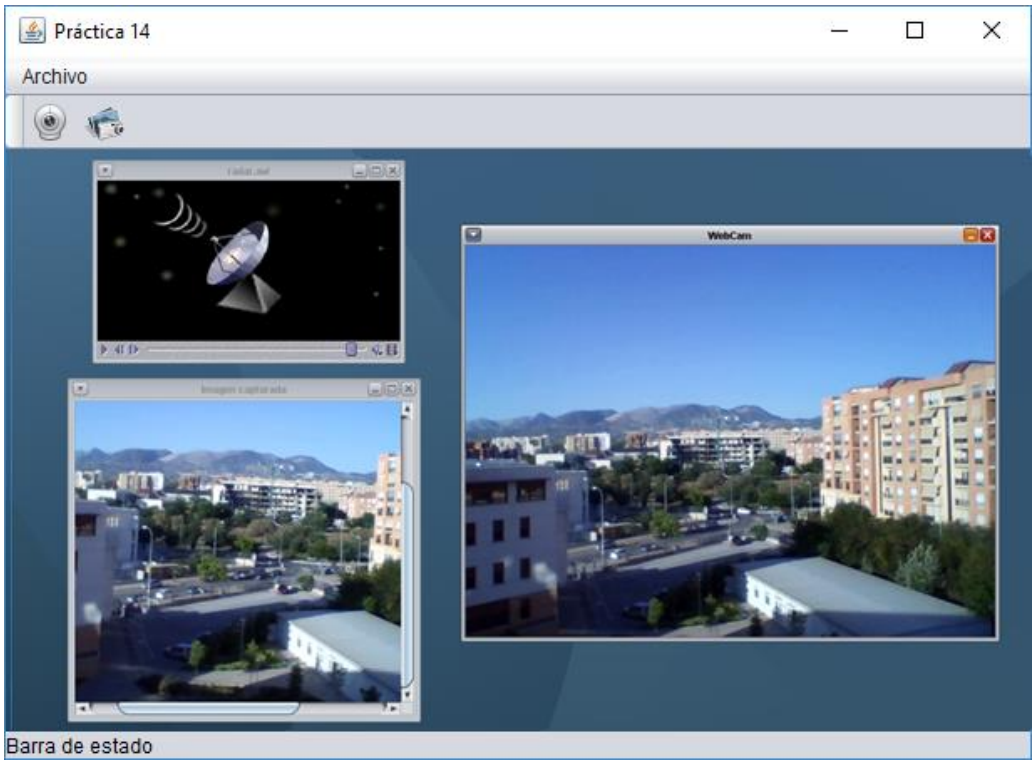

Figura 1: Aspecto de la aplicación

Java Multimedia Framework (JMF)

En este primer bloque probaremos el entorno JMF que, a día de hoy, es la API oficial de Java para la gestión de medios continuos. El principal problema de esta biblioteca es que no se actualiza desde el año 2003, lo que implica numerosos desfases en los formatos y códecs reconocidos, así como en el reconocimiento de dispositivos de captura, o en el desarrollo bajo plataformas de 64 bits. No obstante, y dada su oficialidad, en esta primera parte de la práctica se propone abordar los objetivos de (1) reproducción y (2) captura de instantáneas usando este entorno (se descarta su uso para la gestión de la webcam).

Instalación JMF

- En primer lugar, hay que instalar el JMF. Para ello, podemos optar por dos alternativas:

  a. Bajarnos el JMF de la página oficial de Java e instalarlo. En este caso, la instalación incluirá los ficheros .jar asociados a el JMF[6], así como la aplicación *JMStudio* (demo que usa JMF y que es útil, entre otras cosas, para saber qué dispositivos de captura hay instalados y su nombre).

  b. Bajarnos directamente de la web de la práctica los ficheros .jar (en este caso, no tendríamos la aplicación *JMStudio*)

  Independientemente de la opción elegida, habrá que incorporar los ficheros .jar al proyecto NetBeans (a través de *Propiedades → Biblioteca → Añadir JAR*).

[6] En el caso del sistema Windows, lo instala por defecto en "*C:\ Archivos de programa \JMF2.1.1e\lib*"

- Por otro lado, recordemos que el JMF no funciona correctamente con el JDK de 64 bits, por lo que deberemos de ejecutar nuestra aplicación con un JDK de 32 bits. En caso de que nuestro sistema sea de 64 bits, tendremos que:

  - Bajarnos el JDK de 32 bits e instalarlo en nuestro sistema.

  - En el proyecto NetBeans, incluir la opción de ejecutar el proyecto usando el JDK de 32 bits recién instalado. Para ello, en la sección *Propiedades* → *Biblioteca* → *Plataforma Java*, seleccionaremos la opción de "Administrar plataformas" y añadiremos una nueva indicándole como ubicación del JDK la correspondiente al JDK de 32 bits instalado.

Ventana de reproducción JMF

La ventana de reproducción permitirá reproducir tanto vídeo como sonido (los formatos y códecs soportados por JMF). La zona inferior de la ventana mostrará un panel de control y, en el caso del vídeo, la zona central mostrará el área de visualización.

Algunas recomendaciones:

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

```
private Player player = null;
```

```
private VentanaInternaJMFPlayer(File f) {
  initComponents();
  String sfichero = "file:" + f.getAbsolutePath();
  MediaLocator ml = new MediaLocator(sfichero);
  try {
    player = Manager.createRealizedPlayer(ml);
    Component vc = player.getVisualComponent();
    if(vc!=null)add(vc, java.awt.BorderLayout.CENTER);
    Component cpc = player.getControlPanelComponent();
    if(cpc!=null)add(cpc, java.awt.BorderLayout.SOUTH);
    this.pack();
  }catch(Exception e) {
    System.err.println("VentanaInternaJMFPlayer: "+e);
    player = null;
  }
}
```

  La creación del `Player` puede generar excepciones que impliquen la no realización del mismo, en cuyo caso no debería de lanzarse la ventana. El uso de un constructor estándar implicaría siempre la creación de la ventana interna, por lo que se aconseja la definición de un método `getInstance(File f)` que llame internamente al constructor (que se declararía privado) y que, en caso de error en la creación del `Player`, devuelva `null`:

```
public static VentanaInternaJMFPlayer getInstance(File f){
  VentanaInternaJMFPlayer v = new VentanaInternaJMFPlayer(f);
  return (v.player!=null?v:null);
}
```

- Definir los métodos `play()` y `stop()`, que lancen y detengan la reproducción respectivamente, así como el método `close()` que cierre el `Player`. Por ejemplo, para el caso del `play()`:

```
public void play() {
  if (player != null) {
    try {
      player.start();
    } catch (Exception e) {
      System.err.println("VentanaInternaJMFPlayer: "+e);
    }
  }
}
```

- Gestionar el evento "`InternalFrameClosing`" asociado al cierre de la ventana interna y llamar al método `close()`.

Si se quisiera usar JMF para gestionar la webcam, el enfoque sería similar al indicado para la reproducción, sin más que cambiar el localizar y asociarlo a la webcam[7]. No obstante, dado que JMF no se actualiza desde hace tiempo, su uso para acceder a la webcam puede generar problemas[8], por lo que en esta práctica optaremos por otras alternativas de código abierto

• Captura de instantáneas con JMF

Mediante el uso del botón situado en la barra de herramientas, el usuario podrá capturar imágenes de la cámara o del vídeo que se esté reproduciendo; concretamente, lo hará de la ventana que esté activa, siempre y cuando sea una ventana de tipo "reproducción" o "cámara". Para ello se usará el código visto en teoría (método `getFrame(Player)`). La imagen capturada será mostrada usando una ventana interna "imagen" como la diseñada en las prácticas 8-12[9].

Cámara usando la *Webcam Capture* API

En este segundo bloque abordaremos el uso de la webcam a partir de la biblioteca de código abierto disponible en https://github.com/sarxos/webcam-capture.

• Instalación

- En primer lugar, hay que instalar las bibliotecas de esta API; concretamente, hay que incorporar tres ficheros a nuestro proyecto: (1) webcam-capture-0.3.10.jar, (2) bridj-0.6.2.jar y (3) slf4j-api-1.7.2.jar. Para ello, podemos optar por dos alternativas:

  a. Bajarnos la biblioteca de la web oficial.

  b. Bajarnos directamente de la web de la práctica los ficheros .jar

  Independientemente de la opción elegida, habrá que incorporar los ficheros .jar al proyecto NetBeans (a través de *Propiedades* → *Biblioteca* → *Añadir JAR*).

---

[7] En este caso, crearíamos una clase `VentanaInternaJMFCamara` que tuviese una variable de tipo `Player`. El constructor de dicha clase crearía el objeto `Player` asociado a la cámara: el código sería similar al constructor de la clase `VentanaInternaJMFPlayer`, pero en este caso el localizador del medio tendrá que estar vinculado a la cámara web, por lo que no habrá un fichero asociado (véanse transparencias de teoría).

[8] El acceso a la cámara web puede dar problemas, ya sea porque no identifica alguna de las cámaras instaladas (no saliendo en la lista de cámaras web disponibles) o porque, aun detectándolas, no puede conectarse (generando la excepción `java.io.IOException: Could not connect to capture device`). En este caso, se aconseja ejecutar el programa "*JMF Registry*", que se instala con el JMF, y comprobar que hay dispositivos de captura instalados (en caso negativo, darle al botón de detectar); también se aconseja ejecutar el programa "*JMStudio*" y probar si hay acceso a la cámara web (menú *File* → *Capture*).

[9] Nótese que sobre esta imagen se podrían aplicar todas las operaciones desarrolladas en las prácticas 9-12.

Ventana cámara

La ventana de cámara mostrará la secuencia que esté captando la webcam, estando el área visual en el centro de la ventana. Las recomendaciones en este caso siguen la línea de las indicadas para la ventana cámara basada en JMF, si bien en este caso usaremos las rutinas ofrecidas por la nueva API:

- Crear una clase `VentanaInternaCamara` (que herede de `JInternalFrame`) que contenga una variable del tipo `Webcam` definido en la biblioteca recién incorporada. El constructor de dicha clase creará el objeto `Webcam` asociado a la ventana[10,11]:

```
private Webcam camara = null;
```

```
private VentanaInternaCamara() {
  initComponents();
  camara = Webcam.getDefault();
  if (camara != null) {
    WebcamPanel areaVisual = new WebcamPanel(camara);
    if (areaVisual!= null) {
      getContentPane().add(areaVisual, BorderLayout.CENTER);
      pack();
    }
  }
}
```

Puede que no haya acceso a una webcam, en cuyo caso no debería de lanzarse la ventana. El uso de un constructor estándar implicaría siempre la creación de la ventana interna, por lo que se aconseja la definición de un método `getInstance()` que llame internamente al constructor (que se declararía privado) y que, en caso de error en la creación de la webcam, devuelva `null`:

```
public static VentanaInternaCamara getInstance(){
  VentanaInternaCamara v = new VentanaInternaCamara();
  return (v.camara!=null?v:null);
}
```

- Al igual que en el caso de la aproximación basada en JMF, definir un método `close()` que cierre la webcam[12] y llamarlo desde el manejador asociado al evento "`InternalFrameClosing`" de la ventana interna.

Captura de instantáneas

La *Webcam Capture* API permite la captura de instantáneas de forma sencilla:

```
BufferedImage img = camara.getImage();
```

En nuestro caso, la cámara estará vinculada a una ventana de tipo `VentanaInternaCamara`[13]. La imagen capturada será mostrada usando una ventana interna "imagen" como la diseñada en las prácticas 8-12.

---

[10] Este código creará un objeto webcam usando una resolución por defecto (normalmente, la más pequeña). Existe la posibilidad de preguntar por la lista de resoluciones disponibles, pudiendo así activar aquella de mayor resolución. El siguiente código, que se incluiría después de haber creado el objeto cámara, activaría la máxima resolución disponible:

```
Dimension resoluciones[] = camara.getViewSizes();
Dimension maxRes = resoluciones[resoluciones.length-1];
camara.setViewSize(maxRes);
```

[11] Por defecto, el tamaño del área visual se va adaptando al tamaño de la ventana, si bien mantiene el ratio. Si se quiere mantener el tamaño original fijo, se haría mediante el siguiente código (que se incluiría después de haber creado el área visual): `areaVisual.setFitArea(false);`

[12] Mediante la llamada al método `close()` de la clase `Webcam`.

Reproducción usando VLCj

VLC Media Player es un software de reproducción multimedia, libre y de código abierto, desarrollado por el proyecto *VideoLAN*. La biblioteca VLCj, también de código abierto, ofrece la posibilidad de llamar al código nativo de VLC y crear un reproductor embebido en un panel Swing de Java. La principal ventaja de este enfoque es que tenemos acceso a formatos y códecs actualizados de una forma sencilla; el inconveniente es que depende de librerías nativas, por lo que es condición imprescindible tener instalado VLC en nuestro ordenador.

Instalación

- En primer lugar hay que instalar el VLC en nuestro sistema. Para ello, lo descargaremos de la web oficial de VLC y lo instalaremos
- En segundo lugar, hay que instalar las bibliotecas de la VLCj; concretamente, hay que incorporar cuatro ficheros a nuestro proyecto: (1) vlcj-3.8.0.jar, (2) jna-3.5.2.jar, (3) platform-3.5.2.jar y (4) slf4j-api-1.7.2.jar. Para ello, podemos optar por dos alternativas:
  a. Bajarnos la biblioteca de la web oficial de VLCj.
  b. Bajarnos directamente de la web de la práctica los ficheros .jar

  Independientemente de la opción elegida, habrá que incorporar los ficheros .jar al proyecto NetBeans (a través de *Propiedades* → *Biblioteca* → *Añadir JAR*).
- Para más detalles, y en caso de problemas, se aconseja leer el tutorial de la web oficial, y más concretamente, el relativo a la instalación.

A la hora de usar la VLCj…

- Para poder usar VLCj es obligatorio ejecutar nuestra aplicación usando una máquina virtual de JAVA con la misma arquitectura (32 o 64 bits) que la del VLC instalado. En principio, lo normal es que hayamos descargado el VLC para la arquitectura de nuestro sistema y que, además, esa sea la arquitectura de nuestra máquina virtual. De no ser así, hemos de hacer que estén coordinados, para lo cual se recuerda lo indicado en el segundo punto de la sección "Instalación JMF"
- Nuestra aplicación ha de poder acceder a las bibliotecas nativas de VLC instaladas en nuestro sistema. Si se ha realizado una instalación usando las carpetas por defecto, una forma rápida de vincular las librerías VLC a nuestra aplicación es:

```
boolean ok = new NativeDiscovery().discover();
```

  Este código habría que incluirlo en el método `main()` de nuestro programa[14]; el método devuelve un booleano indicando si se ha podido, o no, acceder a las bibliotecas de VLC.
- Si bien lo anterior debería de funcionar en un entorno estándar, puede que el acceso a las librerías nativas no sea tan inmediato y genere algún problema. De nuevo, en caso de errores, se se aconseja leer el tutorial de la web oficial, y más concretamente, el relativo a la instalación.

---

[13] Se aconseja tener un método `getCamera()` en `VentanaInternaCamara` que devuelva la cámara asociada a la ventana (dicha cámara será la usada para capturar la instantánea).

[14] Por ejemplo, antes del `java.awt.EventQueue.invokeLater(new Runnable()…`

- **Ventana de reproducción**

La ventana de reproducción permitirá reproducir tanto vídeo como sonido, en este caso usando las bibliotecas nativas de VLC (y todos los formatos y códecs soportados por éste). La zona inferior de la ventana deberá incluir un panel con botones para reproducir/parar (véase Figura 2); en este caso, y al contario que el JMF, en VLCj no existe un panel de control predefinido con botones de reproducción/parada, por lo que tendremos que (1) incorporarlo nosotros en nuestra zona de diseño y (2) gestionar los eventos correspondientes. Concretamente, la ventana de reproducción tendrá un panel en su zona inferior con dos botones que permitirán la reproducción y parada del vídeo asociado a esa ventana.

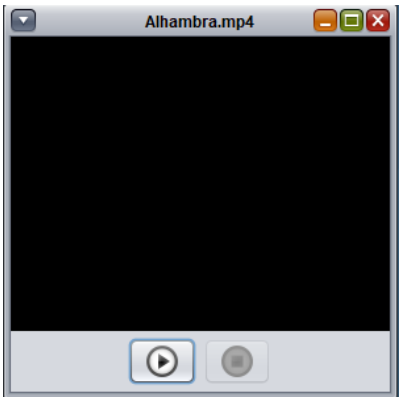

Figura 2: Ventana de reproducción usando VLCj

Algunas recomendaciones:

- Crear una clase `VentanaInternaVLCPlayer` (que herede de `JInternalFrame`) que contenga una variable del tipo `EmbeddedMediaPlayer` definido en la biblioteca VLCj. El constructor de dicha clase creará el objeto `EmbeddedMediaPlayer` asociado a la ventana, para lo cual se recomienda pasar el fichero a reproducir como parámetro del constructor[15]:

```
private EmbeddedMediaPlayer vlcplayer = null;
private File fMedia;
```

```
private VentanaInternaVLCPlayer (File f) {
  initComponents();
  fMedia = f;
  EmbeddedMediaPlayerComponent aVisual =
                              new EmbeddedMediaPlayerComponent();
  getContentPane().add(aVisual,java.awt.BorderLayout.CENTER);
  vlcplayer = aVisual.getMediaPlayer();
}
```

Y, como en casos anteriores, se aconseja la definición de un método `getInstance(File f)`:

```
public static VentanaInternaPlayerVLC getInstance(File f){
  VentanaInternaPlayerVLC v = new VentanaInternaPlayerVLC(f);
  return (v.vlcplayer!=null?v:null);
}
```

- Definir los métodos `play()` y `stop()` que lancen y detengan la reproducción respectivamente[16]:

[15] En este caso, dada la forma de trabajar de la VLCj, también es necesario definir una variable miembro que almacene el fichero asociado a la ventana de reproducción.

[16] Nótese como en el método *play()* se usa el fichero vinculado al reproductor, de ahí que lo definiéramos como dato miembro.

```
public void play() {
  if (vlcplayer != null) {
    if(vlcplayer.isPlayable()){
      //Si se estaba reproduciendo
      vlcplayer.play();
    } else {
      vlcplayer.playMedia(fMedia.getAbsolutePath());
    }
  }
}

public void stop() {
  if (vlcplayer != null) {
    if (vlcplayer.isPlaying()) {
      vlcplayer.pause();
    } else {
      vlcplayer.stop();
    }
  }
}
```

- Gestionar el evento "*InternalFrameClosing*" asociado al cierre de la ventana interna y llamar al método *stop()*[17].

- Para gestionar los eventos generador por el reproductor hay que definir una clase manejadora. Por ejemplo, si queremos controlar cuando se acaba la reproducción[18]:

```
private class VideoListener extends MediaPlayerEventAdapter {
  @Override
  public void finished(MediaPlayer mediaPlayer) {
     //Código
  }
}
```

Además, es necesario (1) crear el objeto manejador y (2) enlazar el generador con el manejador[19]:

```
vlcplayer.addMediaPlayerEventListener(new VideoListener());
```

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 y vídeo. En este caso, el filtro del diálogo incluiría tanto tipos de archivo de imágenes como de sonido y vídeo.

Una mejora de mayor calado implicaría incorporar una barra de herramientas de video que centralizase las acciones de reproducción/parada, en la misma línea que se proponía para el audio en la práctica 13 (nótese que, con la implementación actual, cada ventana tiene sus propios botones de reproducción/parada). De hecho, podría centralizarse en dichos botones la reproducción/parada tanto de audio como de vídeo; en este caso, al diseño mejorado de la práctica 13 sólo habría que añadirle dos nuevos botones a los ya existentes: el de webcam y el de captura de instantáneas.

[17] También se aconseja poner hacer *vlcplayer=null*
[18] En este ejemplo nos interesa controlar la parada para hacer que el botón de parada quede pulsado.
[19] Este código lo pondremos en el constructor, después de haberle asignado valor a *vlcplayer*