
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 ventanas¹:

- Ventana de reproducción, que permitirá visualizar vídeos² (cada ventana irá asociada a un fichero).
- Ventana de cámara, que mostrará el vídeo captado a través de la webcam.
- Ventana de imagen³, que mostrará instantáneas capturadas de los vídeos/cámara mostrados en las ventanas anteriores.

En el menú “Archivo” se incluirá la opción “Abrir vídeo”, que lanzará el correspondiente diálogo y creará una nueva ventana interna de reproducción. Además, se incluirá una barra de herramientas⁴ que contenga cuatro botones:

- Un botón para reproducir vídeo⁵. Al pulsar el botón, se reproduciría el video de la ventana seleccionada en ese momento.
- Un botón para parar la reproducción.
- Un botón correspondiente a la opción “Cámara”, que lanzará una ventana que muestre la secuencia que esté captando la webcam,
- Un botón asociado a la opción “Captura” que permitirá la captura de imágenes de la cámara o del vídeo.

Aunque el entorno oficial de Java para la reproducción vídeo es *Java Multimedia Framework* (JMF), hace tiempo que no se actualiza (no reconoce los formatos y códec más modernos); por este motivo, para llevar a cabo esta práctica usaremos una aproximación basada en bibliotecas de código abierto más actualizadas. En el apéndice de esta práctica se indica cómo se podría abordar usando la JMF, si bien se aconseja usar soluciones más actualizadas.

¹ El objetivo es que esta práctica amplíe las prácticas 8-13 realizadas anteriormente. 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 `VentanaInternaMultimedia` 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 vídeo se añadirá a continuación de las barras de herramientas ya existentes. En este caso, la barra de vídeo solo contendrá los botones de “cámara” y “captura”, ya que los de reproducción/parada ya estarán incorporados en la barra de sonido.

⁵ Si se continúa con las prácticas 8-13, se usarán como botones de reproducción/parada los mismos que ya se habían incluido en la práctica 13 para la reproducción de sonido. En este caso, habría que comprobar si hay una ventana de vídeo seleccionada, en cuyo caso se reproduciría/pararía su vídeo; en caso contrario, se reproduciría/pararía el sonido seleccionado en la lista de reproducción.

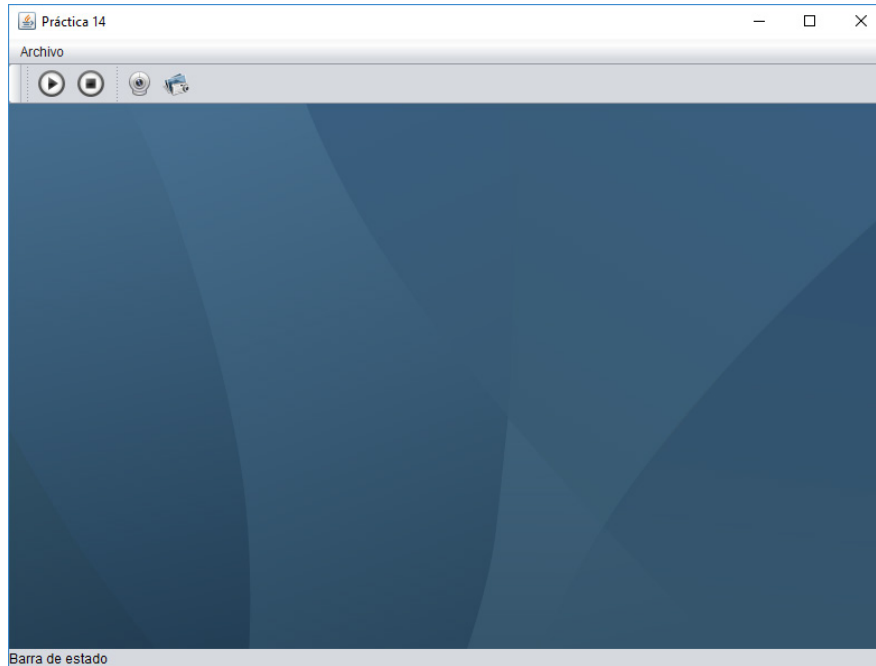


Figura 1: Aspecto de la aplicación

■ Acceso a cámara

En este primer bloque abordaremos el uso de la webcam a partir de la biblioteca de código abierto *Webcam Capture API* 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*⁶. 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*).

● Ventana cámara

La ventana de cámara mostrará la secuencia que esté captando la webcam, estando el área visual en el centro de la ventana. Se recomienda:

- 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^{7,8}:

⁶ SLF4J genera *warnings* cuando se lanza la cámara; no afectan a la captura, por lo que pueden ignorarse.

⁷ 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);
```

```

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);
}

```

- No hay que olvidar cerrar la webcam al cerrar la ventana; para ello habrá que llamar al método `close()` de la cámara cuando se lance el 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*, por lo que se aconseja definir dentro de la clase un método `getImage()` que llame al correspondiente método de la cámara y devuelva la instantánea. La imagen capturada será mostrada usando una ventana interna “imagen” como la diseñada en las prácticas 8-12.

■ 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

⁸ 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);`

- 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) jna-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, habría que instalar la máquina virtual de la arquitectura adecuada -véase apéndice- o un VLC compatible).
- 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 programa⁹; 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.

• 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 códec soportados por éste). Para controlar la reproducción y parada, se usarán los correspondientes botones de la barra de herramientas; concretamente, desde el “*actionPerformed*” asociado a estos botones habrá que (1) acceder a la ventana de video seleccionada y (2) mandarle el correspondiente mensaje para reproducir/parar el video asociado.

Algunas recomendaciones:

- Crear una clase `VentanaInternaVideo` (que herede de `JInternalFrame`) que contenga una variable del tipo `EmbeddedMediaPlayer` definido en la biblioteca VLCj. El constructor de dicha clase creará el objeto

⁹ Por ejemplo, antes del `java.awt.EventQueue.invokeLater(new Runnable()...`

EmbeddedMediaPlayer asociado a la ventana, para lo cual se recomienda pasar el fichero a reproducir como parámetro del constructor¹⁰:

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

private VentanaInternaVideo (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 VentanaInternaVideo getInstance(File f){
    VentanaInternaVideo v = new VentanaInternaVideo(f);
    return (v.vlcplayer!=null?v:null);
}
```

- Definir los métodos *play()* y *stop()* que lancen y detengan la reproducción respectivamente^{11,12}:

```
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 e incluir el código necesario para detener la reproducción¹³.
- Para gestionar los eventos generados por el reproductor, se aconseja definir un método que permita asociar un manejador al reproductor (en este caso, al *vlcplayer*):

¹⁰ 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.

¹¹ Nótese como en el método *play()* se usa el fichero vinculado al reproductor, de ahí que lo definiéramos como dato miembro.

¹² Estos métodos se llamarán desde la ventana principal (concretamente, cuando se pulsen los botones de reproducción y parada de la barra de herramientas).

¹³ También se aconseja hacer *vlcplayer=null*

```

public void addMediaPlayerEventListener
        (MediaPlayerEventListener ml) {
    if (vlcplayer != null) {
        vlcplayer.addMediaPlayerEventListener(ml);
    }
}

```

Recordemos que para manejar eventos hay que (1) definir una clase manejadora, en este caso implementando el interface *MediaPlayerEventListener*, (2) crear el objeto manejador y (3) enlazar el manejador con el generador, en este caso mediante la llamada a *addMediaPlayerEventListener*. Por ejemplo, si queremos que los botones de reproducción/parada se activen/desactiven en función de que se esté o no reproduciendo, los pasos anteriores habría que implementarlos en la clase *VentanaPrincipal*; concretamente, habría que definir la clase manejadora:

```

private class VideoListener extends MediaPlayerEventAdapter {
    public void playing(MediaPlayer mediaPlayer) {
        botonStop.setEnabled(true);
        botonPlay.setEnabled(false);
    }

    public void paused(MediaPlayer mediaPlayer) {
        botonStop.setEnabled(false);
        botonPlay.setEnabled(true);
    }

    public void finished(MediaPlayer mediaPlayer) {
        this.paused(mediaPlayer);
    }
}

```

crear el objeto manejador y enlazar el generador con el manejador¹⁴:

```

vi.addMediaPlayerEventListener(new VideoListener());

```

donde *vi* sería la ventana interna del reproductor.

■ Posibles mejoras para trabajar en casa...

Una vez realizada la práctica, se proponen una serie de. Si consideramos esta práctica como continuación de las 8-13, 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.

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

- Incluir una lista desplegable con la lista de cámaras disponibles; cuando el usuario seleccione la opción “Cámara”, se mostrará la captura de la webcam seleccionada.
- Añadir la opción de que el usuario pueda elegir la resolución de la webcam (de la lista de disponibles).
- Incorporar la captura de instantáneas del vídeo en reproducción; para ello se usará el mismo botón¹⁵ que se ha empleado para la captura de instantáneas desde la cámara.

¹⁴ Este código lo pondremos después de crear la ventana interna de reproducción (que, en nuestro caso, está vinculado a la opción “abrir” del menú archivo).

¹⁵ En el acción de dicho botón habrá que tener en cuenta que se la ventana activa puede ser de tipo “cámara” o “vídeo”.

■ Apéndice: Reproducción y captura usando JMF

El entorno *Java Multimedia Framework* (JMF) es, a día de hoy, 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 este apéndice se explica cómo abordar los objetivos de (1) reproducción y (2) captura de instantáneas usando este entorno.

● 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 JMF¹⁶, 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*).

- 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();  
|     }  
| }
```

¹⁶ En el caso del sistema Windows, lo instala por defecto en “C:\Archivos de programa \JMF2.1.1e\lib”

```

        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 webcam¹⁷. No obstante, dado que JMF no se actualiza desde hace tiempo, su uso para acceder a la webcam puede generar problemas¹⁸.

- 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.

¹⁷ 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).

¹⁸ 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*).