Players

Servidores de Información Multimedia
2º Ingeniero Técnico de Telecomunicación – Imagen y Sonido

Departamento de Ingeniería Telemática
Universidad Carlos III de Madrid

2 Índice

3 Introducción

- La JMF utiliza los Player para presentar los medios.
- La reproducción de los medios se puede controlar programáticamente o bien a través de los componentes visuales de interacción con los controles (panel de control, control de ganancia).
- Si se tienen diferentes fuentes de medios se usará un Player para la reproducción de cada una de ellas. Si se quiere la reproducción sincronizada de estas fuentes de medios se puede hacer que un Player controle la reproducción de los otros.
- Dejaremos el estudio de los procesadores, que son una clase que hereda de Player, para el siguiente tema. Los procesadores permiten controlar el proceso de trabajo con los medios antes de su reproducción, almacenamiento o para actuar como una nueva fuente de datos.

4 Creación de un Player

- Para crear un reproductor haremos uso de la clase Manager como indicábamos en el tema anterior.
- Para crear un Player en el estado *Unrealized* se llama a:
  - Manager.createPlayer
- Para crear un Player en el estado *Realized* se llama a:
  - Manager.createRealizedPlayer
- Los métodos de creación de un *Player* requieren de una de estas tres opciones para especificar el medio a reproducir:
  - URL
  - MediaLocator
  - DataSource

5 Componentes visuales de un Player

- Los dos componentes visuales principales de un Player son:
  - Ventana de visualización del video (se le denomina componente visual)
  - Panel de control para controlar la reproducción de los medios.
- Para visualizar el componente visual se siguen estos dos pasos:
  - Obtener el componente visual llamando a **getVisualComponent**.
  - Añadir este componente a la interfaz visual (el componente visual sigue el AWT)
- Para visualizar el panel de control:
  - Llamar el método **getControlPanelComponent** para obtener el componente.
  - Añadir este componente a la interfaz visual (el componente visual sigue el AWT)

6 Componentes visuales de un Player (cont)

- Un reproductor (Player) puede tener más componentes visuales. Uno típico es el control de ganancia. Para mostrarlo en pantalla haremos:
  - Llamar el método *getGainControl* para obtener el *GainControl* del Player. Si el Player devuelve null, no soporta el interfaz *GainControl*.
  - Llamar a *getControlComponent* en el *GainControl* devuelto.
  - Añadir este componente a la interfaz visual (el componente visual sigue el AWT).
- Pueden existir otros controles específicos con sus componentes visuales. Por ejemplo un Player puede permitir cambiar el brillo y el contraste de la señal que se está mostrando. Para usar estos componentes se puede recurrir al método *getControls* del Player y para cada Control obtener su componte visual. Veamos a modo de ejemplo el siguiente código:

```
Control[] controls = player.getControls();
for (int i = 0; i < controls.length; i++) {
    if (controls[i] instanceof CachingControl) {
      cachingControl = (CachingControl) controls[i];
} }
```

7 Mostrando el estado de la pre-descarga del medio

- El CachingControl, si se implementa en el Player, permite visualizar en pantalla el porcentaje descargado del medio antes de poder reproducirse.
- Si queremos utilizar este control en nuestra aplicación o en nuestro Applet debemos hacer los siguientes pasos:
  - Implementar el interfaz *ControllerListener* y escuchar eventos de tipo *CachingControlEvents* en el método controllerUpdate (de la interfaz ControllerListener).
  - La primera vez que se recibe un CachingControlEvent:
    - Llamar getCachingControl en el evento que se le pasa al método controllerUpdate anterior para obtener la referencia al control de Caché implementado por el Player.
    - Llamar getControlComponent (o getProgressBarComponent) en el CachingControl para obtener el componente visual (la barra de progreso).
    - Añadir la barra de progreso al espacio de presentación de nuestra aplicación o Applet
  - Cada vez que se recibe un CachingControlEvent, mirar si se ha completado la descarga. Cuando getContentProgress devuelve el mismo valor que getContentLength quitar el componente visual del espacio de visualización.

8 Ejemplo de código para barra de estado

- Veamos un Applet que muestra la barra de estado (parcial):

```
public class SimplePlayerApplet extends Applet implements ControllerListener
   {
   Component progressBar = null;
   …
   public synchronized void controllerUpdate(ControllerEvent event) {
    if (event instanceof CachingControlEvent) {
      CachingControlEvent e = (CachingControlEvent) event;
      CachingControl cc = e.getCachingControl();
      if (progressBar == null) {
        if ((progressBar = cc.getControlComponent()) != null) {
       panel.add(progressBar);
       panel.setSize(progressBar.getPreferredSize());
       }
    }
   }
}
```

9 Estableciendo la tasa de reproducción

- La tasa de reproducción es la relación de proporcionalidad entre cada unidad de tiempo del medio y cada unidad del tiempo base del reproductor.
- Una tasa de 2 por ejemplo indicaría que se está reproduciendo el medio a doble velocidad.
- En teoría la tasa puede ser un valor real cualquiera. En la práctica cada reproductor impondrá sus limitaciones:
  - Por ejemplo, no tiene sentido reproducir hacia atrás un medio que está siendo difundido por la red en multicast.
- El método de la clase Player para establecer el valor de la tasa de reproducción es setRate.
- El Player también nos deja establecer el instante de tiempo a partir del cual se desea reproducir el medio mediante setMediaTime (este método recibe un objeto de tipo Time).
  - ¿Recordáis como establecer un tiempo de parada del medio?

10 Preparación de un Player antes de la reproducción

- Antes de reproducir, un Player debe pasar por los estados de Realizing y de Prefetching.
- El método realize del Player mueve el estado a *Realizing* state y realiza las tareas asociadas (averigua el tamaño de la fuente de medios por ejemplo). El método prefetch mueve el estado del Player a *Prefetching* e inicia el llenado de los buffers de alimentación del decodificador. Los métodos realize y prefetch son asíncronos y retornan instantáneamente. Cuando el Player completa la operación requerida lanza un RealizeCompleteEvent o PrefetchCompleteEvent.
- Un Player en el estado *Prefetched* está preparado para la reproducción y la latencia de inicio no puede reducirse más. Llamar al método setMediaTime puede hacer que el Player retorne al estado *Realized* lo que incrementará su reproducción.
- Hay que tener en cuenta que un Player en *Prefetched* consume recursos del sistema.
- Para saber cuánto tiempo necesita un Player desde su estado actual hasta la reproducción del medio se tiene el método: getStartLatency

11 Arrancando y parando una presentación

- Para iniciar una presentación Hay que:
  - Especificar el tiempo desde dónde se quiere iniciar la reproducción del medio invocando el método setMediaTime.
  - Iniciar la reproducción invocando start en el Player.
- Si se llama a start y el reproductor ya está reproduciendo se lanza un evento de tipo StartEvent.
- Como la interfaz Player extiende la interfaz Controller que a su vez extiende la interfaz Clock, el Player hereda el método syncStart que se usa cuando se necesita reproducir síncronamente el medio con el de otro Player.
- Para detener la reproducción de un medio se puede producir una de estas 4 situaciones:
  - Se llama al método stop del Player
  - Se alcanza un punto de stop en la reproducción (se puede indicar el punto de stop con la función setStopTime del Player)
  - Se llega al final del medio
  - Cuando la tasa de recepción del medio es demasiado lenta para la reproducción del mismo a la tasa especificada

12 Liberando recursos del Player

- El método deallocate se utiliza para que el Player libere el máximo número de recursos (buffers por ejemplo).
- El método deallocate solo se puede invocar si el Player está parado. Si el Player está reproduciendo y se llama a deallocate se lanza un ClockStartedError.
- La invocación de deallocate sobre un Player que ha superado la fase de *Realized* le lleva nuevamente a este estado.
- La invocación de deallocate en la fase de realización del Player le lleva nuevamente al estado *Unrealized*
- Si el reproductor no se va a utilizar más en el futuro se llama al método close que libera todos los recursos y cierra el Player.

13 Obteniendo los parámetros del Player

- Un Player nos da información de su configuración mediante la invocación de una serie de métodos:
  - getRate nos devuelve la tasa de reproducción del Player. El valor devuelto es un float (el valor 1 indica reproducción normal).
  - getMediaTime nos da el valor del tiempo actual de reproducción del medio. Nos devuelve un objeto de tipo Time.
    - Hay que notar que no hay una relación 1 a 1 entre el tiempo del medio y el marco de la imagen pues un marco (frame) de la imagen puede ocupar varios tiempos.
  - Player.getTimeBase().getTime() nos permite recuperar el tiempo base de reproducción del medio.
  - mapToTimeBase (método que el Player hereda de la interfaz Clock) se usa para mapear un tiempo de medio a un tiempo base si el reproductor está reproduciendo.
  - getDuration (método heredado de la interfaz Duration) nos devuelve la duración del medio si su reproducción se hiciese a tasa 1.

14 Captura de eventos de medios

- La aplicación multimedia que quiera responder a los eventos de medios que lanzan los controladores (clase Controller) de la cual heredan tanto Players como Processors, debe implementar la interfaz ControllerListener y registrarse en el Controller como escuchador de eventos (ControllerEvents) con el método addControllerListener.
- La interfaz ControllerListener solo cuenta con el método controllerUpdate que se invoca cada vez que el Controller lanza un evento (ControllerEvent). Sin embargo existen muchos tipos de ControllerEvents con lo cual el procesado de eventos dentro del método controllerUpdate puede ser complejo.
- Para facilitar esta tarea se tiene la clase ControllerAdapter que es una clase ya codificada que implementa ControllerListener pero que no hace nada con los eventos que recibe. Define un método para cada tipo de ControllerEvent. Nosotros podemos extender ControllerAdapter e implementar sólo aquellos métodos cuyos eventos queremos tratar:

```
player.addControllerListener(new ControllerAdapter() {
    public void endOfMedia(EndOfMediaEvent e) {
      Controller controller = e.getSource();
      controller.stop();
      controller.setMediaTime(new Time(0));
      controller.deallocate(); }
})
```

15 Sincronización de medios

- Si queremos sincronizar la reproducción de varios Players lo más sencillo es hacer que uno de los Players asuma el control de la sincronización.
- Para ello la interfaz Player cuenta con dos métodos:
  - addController que añade un controlador (un Player por ejemplo) a la lista de controladores que van a ser sincronizados
  - removeController
- Al invocar addController ocurren 3 cosas básicamente:
  - El Controller añadido toma la base de tiempos del Player al que se añade.
  - La duración del Player es la más larga de las duraciones.
  - El Player calcula las latencias para empezar de cada uno de los medios de cara a su reproducción sincronizada.
- Todos los eventos de cualquiera de los controladores se lanzarán a través del Player donde se han añadido de forma que se simplifica mucho el procesado de eventos.
- Los métodos de cambio de estado del Player se propagan a los controladores registrados (por ejemplo, un start() en el Player llama a su vez a los start() de cada controlador registrado).

16 Propagación de invocaciones (sincronización)

Function	Stopped Player	Started Player
`setMediaTime`	Invokes `setMediaTime` on all managed `Controllers`.	Stops all managed `Controllers`, invokes `setMediaTime`, and restarts `Controllers`.
`setRate`	Invokes `setRate` on all managed `Controllers`. Returns the actual rate that was supported by all `Controllers` and set.	Stops all managed `Controllers`, invokes `setRate`, and restarts `Controllers`. Returns the actual rate that was supported by all `Controllers` and set.
`start`	Ensures all managed `Controllers` are *Prefetched* and invokes `syncStart` on each of them, taking into account their start latencies.	Depends on the `Player` implementation. `Player` might immediately post a `StartEvent`.
`realize`	The managing `Player` immediately posts a `RealizeCompleteEvent`. To be added, a `Controller` must already be realized.	The managing `Player` immediately posts a `RealizeCompleteEvent`. To be added, a `Controller` must already be realized.
`prefetch`	Invokes `prefetch` on all managed `Controllers`.	The managing `Player` immediately posts a `PrefetchCompleteEvent`, indicating that all managed `Controllers` are *Prefetched*.
`stop`	No effect.	Invokes `stop` on all managed `Controllers`.
`deallocate`	Invokes `deallocate` on all managed `Controllers`.	It is illegal to call `deallocate` on a *Started* `Player`.
`setStopTime`	Invokes `setStopTime` on all managed `Controllers`. (`Player` must be *Realized*.)	Invokes `setStopTime` on all managed `Controllers`. (Can only be set once on a *Started* `Player`.)
`syncStart`	Invokes `syncStart` on all managed `Controllers`.	It is illegal to call `syncStart` on a *Started* `Player`.
`close`	Invokes `close` on all managed `Controllers`.	It is illegal to call `close` on a *Started* `Player`.

17 Ejemplo: PlayerApplet

```
import java.applet.*;
import java.awt.*;
import java.net.*;
import javax.media.*;

public class PlayerApplet extends Applet implements ControllerListener {
   Player player = null;
   public void init() {
      setLayout(new BorderLayout());
      String mediaFile = getParameter("FILE");
      try {
         URL mediaURL = new URL(getDocumentBase(), mediaFile);
         player = Manager.createPlayer(mediaURL);
         player.addControllerListener(this);
      }
      catch (Exception e) {
         System.err.println("Got exception "+e);
      }
   }
   public void start() {
      player.start();
   }
   public void stop() {
      player.stop();
      player.deallocate();
   }
   public void destroy() {
      player.close();
   }
   public synchronized void controllerUpdate(ControllerEvent event) {
      if (event instanceof RealizeCompleteEvent) {
         Component comp;
         if ((comp = player.getVisualComponent()) != null)
            add ("Center", comp);
         if ((comp = player.getControlPanelComponent()) != null)
            add ("South", comp);
         validate();
      }
   }
}
```

```
<APPLET CODE=ExampleMedia.PlayerApplet
WIDTH=320 HEIGHT=300>
 <PARAM NAME=FILE VALUE="sample2.mpg">
 </APPLET>
```


18 El MediaPlayer bean de la JMF

- Para facilitarnos la vida, la JMF tiene ya implementada la clase:
  - javax.media.bean.playerbean.MediaPlayer
- Encapsula un reproductor.
- Para usarlo hacemos:
  - Construimos el bean:
    - MediaPlayer mp1 = new javax.media.bean.playerbean.MediaPlayer();
  - Indicamos el medio a reproducir:
    - mp1.setMediaLocation(new java.lang.String("file:///E:/jvideo/media/Sample1.mov"));
  - Empezamos la reproducción:
    - mp1.start();
  - Paramos la reproducción:
    - mp1.stop();

19 Propiedades del MediaPlayer

- Como bean, exporta una serie de propiedades mediante métodos get y set.
  - Ex: void setMediaLocation (java.lang.String location)

Property	Type	Default	Description
Show control panel	Boolean	Yes	Controls whether or not the video control panel is visible.
Loop	Boolean	Yes	Controls whether or not the media clip loops continuously.

Media location	String	N/A	The location of the media clip to be played. It can be an URL or a relative address. For example:
			`file:///e:/video/media/Sample1.mov`
			`http://webServer/media/Sample1.mov`
			`media/Sample1.mov`

Show caching control	Boolean	No	Controls whether or not the download-progress bar is displayed.
Fixed Aspect Ratio	Boolean	Yes	Controls whether or not the media's original fixed aspect ratio is maintained.
Volume	int	3	Controls the audio volume.

20 Players para RTP – un breve bosquejo

- Para reproducir medios que están siendo mandados por la red usando RTP se usa un MediaLocator empezando por rtp.
- Veamos un ejemplo (veremos más detalles en capítulos posteriores):

```
String url= "rtp://224.144.251.104:49150/audio/1";

MediaLocator mrl= new MediaLocator(url);

if (mrl == null) {
    System.err.println("Can't build MRL for RTP");
    return false;
}

// Create a player for this rtp session
try {
    player = Manager.createPlayer(mrl);
} catch (NoPlayerException e) {
    System.err.println("Error:" + e);
    return false;
} catch (MalformedURLException e) {
    System.err.println("Error:" + e);
    return false;
} catch (IOException e) {
    System.err.println("Error:" + e);
    return false;
}
```

Cuestiones de refuerzo

- ¿Qué elemento define JMF para presentar medios: **`Player`**, `Processor` o **`DataSink`**?
- ¿Qué dos métodos de la clase Manager se pueden utilizar para crear un Player?
- ¿Con qué tres tipos de medios se suele especificar un medio multimedia?
- Diga cuales son los dos tipos de componentes visuales principales que soporta un Player (pista: uno es la ventana de visualización del video)
  - Indique cómo se accede al componente visual
  - Indique cómo se accede al panel de control visual
- A parte de los componentes visuales principales, ¿Qué otros pueden existir? ¿Es *GainControl* uno de ellos?
- Diga cómo se pueden buscar los controles que maneja un Player
- ¿Para qué sirve el *CachingControl* ?
- Explique cómo funciona el ejemplo del *CachingControl* de la transparencia 8. ¿Qué hace?
- ¿Se puede cambiar siempre la velocidad de reproducción de un video? Justifique su respuesta ¿Qué método emplearía?
- Diga cuál es la diferencia entre el estado de *Realized* y *Prefeched* de un Player

Cuestiones de refuerzo (II)

- ¿Para qué sirve el método **`getStartLatency`** de un Player? Explicite un caso en el cual lo utilizaría.
- ¿Qué pasa cuando llamamos a **`start`** sobre un reproductor que ya está en funcionamiento?
- ¿Qué cuatro situaciones detienen el funcionamiento de un **`Player`**?
- ¿Cuál es el motivo por el cual existe un método **`deallocate()`** en un Player? Explique su utilidad.
- ¿Se puede invocar al método **`deallocate()`** sobre un **`Player`** en ejecución?
- ¿Cuál es la diferencia entre el método **`deallocate()`** y **`close`**() de un **`Player`**?
- Diga el método del Player que se encarga de saber a que velocidad está reproduciéndose
- Diga para qué sirve el método **`getDuration`** del **`Player`**
- ¿Cuál es la diferencia existente a la hora de capturar eventos con **`ControllerListener`** y **`ControllerAdapter`**? ¿Cuál es más sencilla de utilizar?

Cuestiones de refuerzo (III)

- En un Player, ¿para qué sirve la sincronización de medios?
  - ¿cómo se utiliza?
- Las acciones invocadas sobre un Player (**`start`**, **`stop`**, **`syncStart`**, **`close`**) ¿Se propagan a los **`Players`** que están suscritos?
- Dibuje un diagrama explicativo de cómo se ejecuta el código de la clase **`PlayerApplet`** (transparencia 17)
- Cuál es el objetivo de que JMF implemente la siguiente clase:
  - javax.media.bean.playerbean.MediaPlayer
- ¿Qué pasos evita tener que dar?
- A la vista de lo presentado en clase, ¿es necesario que esta clase maneje los eventos de jmf para mostar el panel de control? ¿Cómo solventa este problema entonces?
- Escriba el código necesario para reproducir el siguiente medio:
  - rtp://it.uc3m.es:49150/labsimitis/presentation

Autoría

- Mario Muñoz Organero
- Pablo Basanta Val

  + Cuestiones cortas