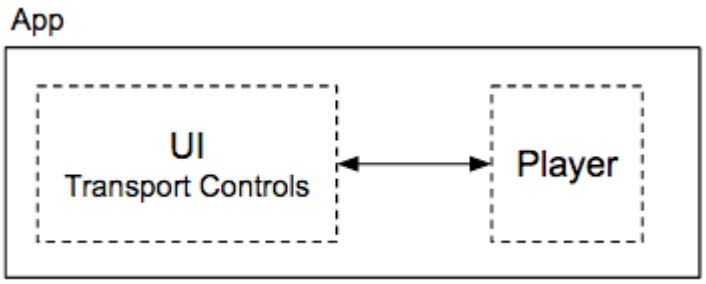
Introducción - Video

Primer Parte
Primera tarea

Una aplicación multimedia que reproduce audio o vídeo suele tener dos partes:

- Un reproductor que, dada una fuente de medios, descarga y decodifica ese medio, y lo representa como video y / o audio.

- Una interfaz de usuario (UI) con controles de transporte para ejecutar el reproductor y, opcionalmente, mostrar el estado del reproductor.



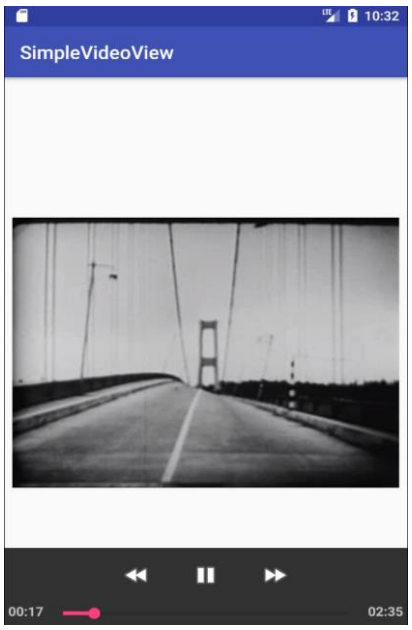
La arquitectura de una aplicación multimedia puede volverse mucho más complicada, especialmente si el propósito de la aplicación es reproducir audio o vídeo.
Android proporciona una arquitectura general para aplicaciones de audio y vídeo, y un gran número de clases relacionadas con los medios
La forma más sencilla posible de reproducir un clip de vídeo en la aplicación es
usar la clase **VideoView** para reproducir el vídeo y la clase **MediaController** como interfaz de usuario para controlarlo.
En este lección se crea una aplicación sencilla que reproduce un clip de vídeo que está incrustado en los recursos de la aplicación o disponible en Internet.

Actividad

- Compile una aplicación llamada **SimpleVideoView** que reproduzca un clip de vídeo mediante las clases **VideoView** and **MediaController**
- Conserve la posición de reproducción de la reproducción de vídeo en todos los cambios de estado de actividad.
- Reaccione al final de la reproducción de vídeo con un detector de eventos.
- Controla el período de tiempo en tu aplicación donde se prepara el contenido multimedia (almacenamiento en búfer o decodificación) y proporciona un mensaje útil al usuario.
- Reproduzca archivos multimedia como archivos incrustados en la aplicación o transmitidos desde Internet.

Descripción general de la aplicación

En esta lección se construyes la aplicación **SimpleVideoView** desde cero.
SimpleVideoView reproduce un archivo de vídeo en su propia vista en tu aplicación:



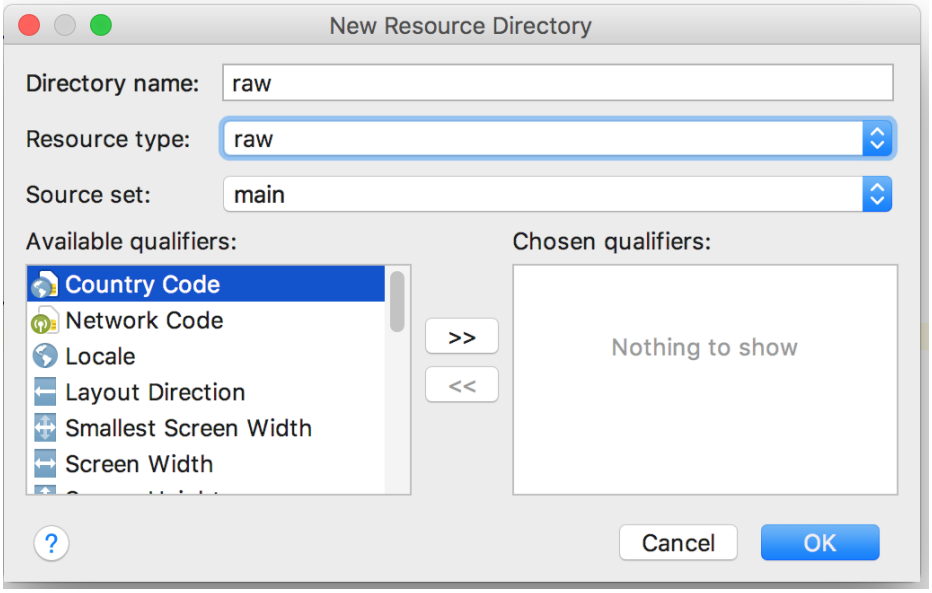
La aplicación **SimpleVideoView** incluye un conjunto familiar de controles multimedia.
Los controles le permiten

- reproducir, pausar, retroceder, avanzar rápidamente y usar el control deslizante de progreso para saltar hacia adelante o hacia atrás a lugares específicos del video.

Primero se reproduce un archivo de video incrustado con los recursos de la aplicación. Luego En la segunda tarea, modifica la aplicación para almacenar en búfer y reproducir un archivo de vídeo desde Internet.

Importante:
Se inicia el proyecto con una aplicación Vacía.

1 – Crear un nuevo directorio llamado "**raw**" en el proyecto:
ir a ……..”**New Resource Directory**”



Mover el archivo de video al directorio creado.

2- Mover el archivo de video "**suvideo.mp4**" al directorio ("**/raw**") creado anteriormente en su proyecto.
3. En la actividad principal (**activity_main.xml**), borrar el TextView existente y remplazarlo con un elemento VideoView con los siguiente atributos:

```
<VideoView
android:id="@+id/videoview"
android:layout_width="0dp"
android:layout_height="0dp"
android:layout_margin="8dp"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintDimensionRatio="4:3"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"/>
```

Consideraciones:
1- Cuando las restricciones (constraints) **layout_width** and l**ayout_height** are **0dp** , el tamaño y la posición de elemento VideoView se calcuran en forma dinamica basandose en las otras restricciones.
2- El atributo **layout_constraintDimensionRatio** indica que cuando la aplicación calcula el tamaño de VideoView, la relación entre el ancho y la altura debe ser 4:3

Esta restricción mantiene la relación de aspecto del video igual y evita que
la vista se estire demasiado en cualquier dirección (dependiendo de cómo se gire el dispositivo).

Implementar la Actividad Principal (MainActivity)

1. En **MainActivity.java**, agregue una constante con el nombre del archivo de muestra del video y una variable miembro para contener la instancia de VideoView.

```
private static final String VIDEO_SAMPLE = "mivideo";
private VideoView  mVideoView;
```

```
// recordar:  mVideoView es el nombre de la variable
// recordar:  VIDEO_SAMPLE es el nombre de la constante
```

2- También en MainActivity , crear un metodo privado llamado **getMedia()** que toma un string y retorna una Uri

La implementación se ve así:

```
private Uri getMedia(String mediaName) {
return Uri.parse("android.resource://" + getPackageName() +
"/raw/" + mediaName);
}
```

// recordar: "/raw/" es el directorio que creo en su proyecto al principio

Este método convierte el nombre del archivo multimedia de muestra (una cadena) en un objeto URI completo que representa la ruta a la muestra en los recursos de su aplicación.

Tenga en cuenta que aunque el nombre de archivo real en el directorio de recursos de la aplicación es **mivideo.mp4** , el nombre de la cadena y el nombre del recurso resultante no incluyen la extensión.

3. En el metodo **onCreate()** , obtenga una referencia a VideoView en el diseño:

```
mVideoView = findViewById(R.id.videoview);
```

4.Crear un nuevo metodo privado llamado que no acepta argumentos y no retorna nada.

```
private void initializePlayer() {
}
```

5. Dentro de **initializePlayer**(), use los metodos **getMedia**() y **setVideoUri**() para establecer la URI de medios que reproducirá VideoView.

```
Uri videoUri = getMedia(VIDEO_SAMPLE);
mVideoView.setVideoURI(videoUri);
```

6. Llamar al metodo **start()** de VideoView para empezar la reproduccion.

```
mVideoView.start();
```

7. Crear un metodo privado llamado **releasePlayer()**, y llamar al metodo **stopPlayback()** de VideoView .

Esto detiene la reproducción del video y libera todos los recursos retenidos por la vista de video.

```
private void releasePlayer() {
mVideoView.stopPlayback();
}
```

Implementar métodos de ciclo de vida de actividad

La reproducción multimedia utiliza muchos más recursos del sistema que la mayoría de las aplicaciones. Es fundamental que su aplicación libere completamente esos recursos cuando no los esté usando, incluso si la aplicación está en pausa en segundo plano.

- Implemente el método de ciclo de vida de la actividad (Activity) **onStop()** para liberar los recursos de medios cuando la aplicación se detiene.
- Implemente el metodo **onStart()** para inicializar los recursos cuando se inicia la aplicación de nuevo.

1. Sobrescribir el metodo **onStart()** y llamar al metodo **initializePlayer()** .

```
@Override
protected void onStart() {
super.onStart();
initializePlayer();
}
```

2. Sobrescribir el metodo **onStop()** y llamar al metodo **releasePlayer()** .

```
@Override
protected void onStop() {
super.onStop();
releasePlayer();
}
```

3. Sobrescribir el metodo **onPause()** y agregar una prueba para versiones de Android anteriores a N (inferiores a 7.0, API 24).
Si la aplicación se ejecuta en una versión anterior de Android, pause VideoView aquí.

```
@Override
protected void onPause() {
super.onPause();
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.N) {
mVideoView.pause();
}
}
```

Consideraciones:
a- Esta prueba es necesaria porque el comportamiento de **onPause**() y **onStop**() cambió en Android N (7.0, API 24).

b- En versiones anteriores de Android, **onPause**() era el final del ciclo de vida visual de su aplicación, y podría comenzar a liberar recursos cuando la aplicación estaba en pausa.
c- En las versiones más recientes de Android, su aplicación puede estar en pausa pero aún visible en la pantalla, ya que con modo multiventana o imagen en imagen (PIP).

- En esos casos, es probable que el usuario quiera el video para continuar reproduciendo en segundo plano.
- Si el video se está reproduciendo en múltiples ventanas o PIP modo, entonces es onStop() que indica el final del ciclo de vida visible de la aplicación y su video la reproducción debería detenerse en ese momento.
- Si solo deja de reproducir su video en onStop() , como en el paso anterior, entonces en dispositivos más antiguos puede haber unos segundos en los que, aunque la aplicación ya no esté visible en la pantalla, la la pista de audio del video continúa reproduciéndose mientras onStop() se pone al día.

Esta prueba para versiones anteriores de Android pausa la reproducción real en onPause() para evitar que el sonido se reproduzca después de la aplicación ha desaparecido de la pantalla.

4. Compile y ejecute la aplicación.

Segunda tarea

Cuando se inicia la aplicación, el archivo de video se abre y se decodifica, comienza a reproducirse y se reproduce al fin. No hay forma de controlar la reproducción de medios, por ejemplo, usando pausa, reproducir, avanzar rápido avanzar o retroceder.

Agregue estas capacidades en la siguiente tarea…

Nota:
si prueba la aplicación SimpleVideoView en un emulador, use un AVD que admita API 23 o más.
Las versiones anteriores del emulador admiten menos tipos de formatos de video y pueden también sufren un rendimiento degradado durante la reproducción.
Si ejecuta la aplicación en un dispositivo físico que ejecuta una versión de Android anterior a la API 23, no deberías tener ninguno de estos problemas.

Añadir un MediaController

Una aplicación que reproduce video pero no proporciona una forma para que el usuario controle ese video es menos que útil.
La plataforma Android proporciona una forma de controlar los medios usando **la vista MediaController**, que se encuentra en el **paquete android.widget**.

Una vista de **MediaController** combina:
los elementos de interfaz de usuario de control de medios más comunes (botones para reproducir, pausar, avanzar rápido y rebobinar, así como una barra de búsqueda o progreso) con la capacidad de controlar un reproductor multimedia subyacente, como un VideoView.

Importante:
Para usar una vista de **MediaController**, no la define en su diseño como lo haría con otras vistas.

- En su lugar, lo creas mediante programación en el método **onCreate()** de tu aplicación y luego lo adjuntas a un reproductor multimedia.
- El controlador flota sobre el diseño de su aplicación y le permite al usuario comenzar,
  detener, avanzar, rebobinar y buscar dentro del video.

En la figura de arriba:

1. Un VideoView, que incluye un MediaPlayer para decodificar y reproducir video, y un SurfaceView para mostrar el video en la pantalla
2. Una vista de MediaController, que incluye elementos de interfaz de usuario para controles de transporte de video (reproducir, pausar, avance rápido, rebobinado, control deslizante de progreso) y la capacidad de controlar el video

En esta tarea, agrega un MediaController a la aplicación SimpleVideoView.

Pasos
1. Localiza el método **onCreate()** en MainActivity .
   - Debajo de la asignación de la variable **mVideoView**, cree un nuevo objeto **MediaController** y use **setMediaPlayer()** para conectar el objeto a la vista de video.

```
controlador MediaController = nuevo MediaController(this);
controlador.setMediaPlayer(mVideoView);
```

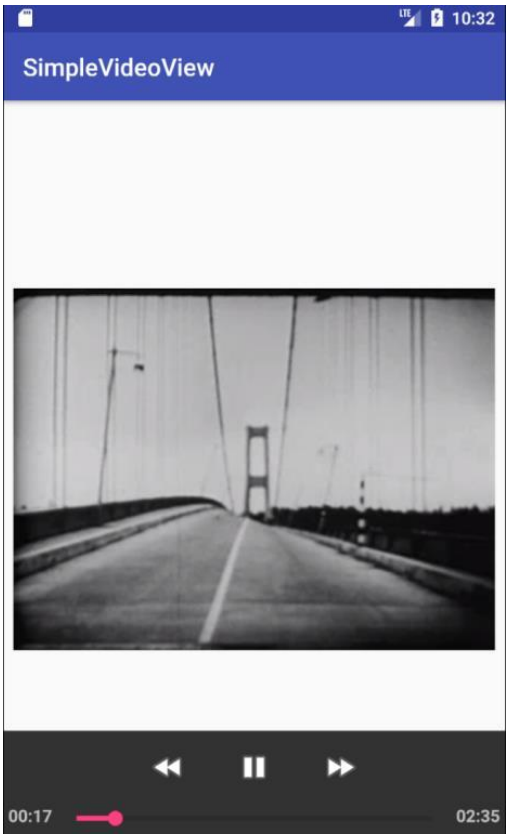
Nota:
Cuando agregue la clase MediaController, asegúrese de que importe android.widget.MediaController y no android.media.session.MediaController .

2. Use **setMediaController()** para hacer la conexión inversa, es decir, para decirle a VideoView que el MediaController se utilizará para controlarlo:

```
mVideoView.setMediaController(controller);
```

3. Compile y ejecute la aplicación.

Detalle:
- Como antes, el video comienza a reproducirse cuando se inicia la aplicación.
- Toca **VideoView** para que aparezca **MediaController**.
- A continuación, puede utilizar cualquiera de los elementos en ese controlador para controlar la reproducción multimedia.
- El MediaController desaparece por sí solo después de tres segundos.



Conservar la posición de reproducción durante todo el ciclo de vida de la actividad

Cuando se llama al método **onStop**() y su aplicación pasa a segundo plano o se reinicia porque de un cambio de configuración, la clase VideoView libera todos sus recursos y no conserva el estado de reproducción de video, como por ejemplo la posición actual.

- Esto significa que cada vez que se inicia la aplicación o pasa a primer plano, el vídeo se vuelve a abrir y se reproduce desde el principio.

Para obtener una línea de base para la comparación, ejecute la aplicación, si aún no se está ejecutando. Con su aplicación en el primer plano, intente las siguientes tareas:

- Gira el dispositivo.
- Toque el botón Inicio y luego use el selector de tareas para traer la aplicación de vuelta al primer plano.
- Si su dispositivo o emulador ejecuta Android 7.0 (API nivel 24) o superior, mantenga presionada la tecla conmutador de tareas para habilitar el modo multiventana.

Observe en cada uno de estos casos cómo el video se reinicia desde el principio.
En esta tarea, realiza un seguimiento de la posición de reproducción actual, en milisegundos, a lo largo del ciclo de vida de la aplicación. Si se liberan los recursos de video, el video puede reiniciarse donde lo dejó.

Pasos:
1. En MainActivity, agregue una variable de miembro para mantener la posición actual del video (inicialmente 0). La la posición de reproducción se registra en milisegundos desde 0.

```
private int mCurrentPosition = 0;
```

2. Agregue una variable miembro para mantener la clave para la posición de reproducción en el paquete de estado de la instancia:

```
private static final String PLAYBACK_TIME = "play_time";
```

3. En initializePlayer() , después de setVideoUri() pero antes de start() , verifique si la posición actual es mayor que 0, lo que indica que el video se estaba reproduciendo en algún momento. Usar el método seekTo() para mover la posición de reproducción a la posición actual.

```
if (mPosiciónActual > 0) {
mVideoView.seekTo(mPosiciónActual);
} else {
// Saltar a 1 muestra el primer cuadro del video.
mVideoView.seekTo(1);
}
```

Si la posición actual es 0, el video aún no se ha reproducido. Use **seekTo**() para configurar la reproducción posición a 1 milisegundo. Esto mostrará el primer cuadro del video en lugar de una pantalla negra.

4. Anule el método **onSaveInstanceState**() para guardar el valor de **mCurrentTime** en el bundle de estado de la instancia (el bundle). Obtenga la posición de reproducción actual con el método getCurrentPosition().

```
@Override
protected void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState);
outState.putInt(PLAYBACK_TIME, mVideoView.getCurrentPosition());
}
```

Nota:
Cuando se crea una Activity se llama al método onCreate, que recibe un Bundle. **La primera vez que se crea la Activity el Bundle es null**, no recibe nada

5. En **onCreate**() , verifique la existencia del paquete de estado de la instancia y actualice el valor de mCurrentTime con el valor de ese paquete. Agregue estas líneas antes de crear el controlador de medios.

```
if (savedInstanceState != null) {
mCurrentPosition = savedInstanceState.getInt(PLAYBACK_TIME);
}
```

6. Compile y ejecute la aplicación. Repita las tareas de referencia y observe cómo se guarda la posición de reproducción en cada caso.

Agregar una devolución de llamada onCompletion()

La clase VideoView (y el MediaPlayer que contiene) le permite definir **listeners** para varios eventos relacionados con la reproducción de medios. Por ejemplo:

- El medio ha sido preparado (almacenado en búfer, decodificado, sin comprimir, etc.) y está listo para jugar.
- Se ha producido un error durante la reproducción multimedia.
- La fuente de medios ha informado información como un retraso en el almacenamiento en búfer durante la reproducción, por ejemplo, cuando los medios se transmiten a través de Internet.
- El medio ha terminado de reproducirse.

Los listeners de los eventos de preparación y finalización son los **listeners** más comunes para implementar en una aplicación multimedia. Todavía no ha necesitado manejar la preparación de medios en el aplicación **SimpleVideoView**, porque el videoclip está incrustado en la aplicación y es bastante pequeño, por lo que se reproduce rápidamente. Es posible que este no sea el caso para videoclips más grandes o para aquellos que reproduzca directamente desde el Internet.
Volverá a preparar los medios en una tarea posterior.
Cuando finaliza la reproducción de medios, se produce el evento de finalización y la devolución de llamada llamando al metodo **onCompletion**().
En esta tarea, implementa un **Listener** para eventos onCompletion() para mostrar un **Toast** cuando la reproducción finaliza.

1. Al final del método **initializePlayer()**, cree un nuevo **MediaPlayer.OnCompletionListener**, anule el método **onCompletion**() y use **setOnCompletionListener**() para agregar ese llistener a VideoView.

```
mVideoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
@Override
public void onCompletion(MediaPlayer mediaPlayer) {
// Implementation here.
}
});
```

2. Dentro del método **onCompletion**(), agregue un Toast para mostrar el mensaje "**Reproducción completada**".
3. Después del Toast, agregue una llamada a **seekTo**() para restablecer la reproducción y el MediaController al comienzo del clip.

```
mVideoView.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
@Override
public void onCompletion(MediaPlayer mediaPlayer) {
Toast.makeText(MainActivity.this, "Playback completed",
Toast.LENGTH_SHORT).show();
mVideoView.seekTo(1);
}
});
```

De manera predeterminada, cuando los medios terminan de reproducirse, tanto VideoView como MediaController muestran el Estado final de los medios.
El código que se muestra arriba restablece tanto el reproductor como el controlador al inicio del video para que el video se pueda reproducir de nuevo.

4. Compile y ejecute la aplicación. Toque el video para mostrar MediaController y mueva el control deslizante casi hasta el final. Cuando el video termina de reproducirse, aparece el Toast y el controlador se restablece al principio del clip.