Media Player [Java]

En lo relacionado al contenido multimedia, Android admite la reproducción de diversos contenidos comunes como audio, video e imágenes en tu app. Se puede reproducir desde archivos multimedia almacenados en los recursos de tu app (**raw**), desde archivos del **sistema de archivos** o desde un flujo de datos que llega a través de una **conexión de red**. Para ello disponemos de diferentes API de **MediaPlayer**.

Al igual que con una Activity, la reproducción de un elemento multimedia se gestiona como una máquina de estados. Cada vez que interactuamos con el reproductor inicializando una canción (por ejemplo) estamos cambiando el estado del reproductor. Los estados son:

1) **Idle**. El reproductor está sin contenido que reproducir.
2) **Initialized**: El reproductor Media Player dispone de contenido para reproducir.
3) **Prepared**: El reproductor Media Player está listo para reproducir.
4) **Started**: La reproducción ha comenzado.
5) **Pause / Stop**: Parada o pausa de la reproducción.
6) **Completed**: La reproducción se ha completado.

El **MediaPlayer** ofrece una serie de métodos para cambiar el estado del reproductor. A continuación, vamos a ver los pasos para preparar y reproducir un elemento multimedia.

Preparando la app

Para reproducir un elemento multimedia, usaremos un componente llamado **VideoView**. Una vez referenciado desde nuestro Activity, podemos empezar a preparar el **MediaPlayer** para reproducir un elemento multimedia:

```
mediaPlayer = new MediaPlayer();
mediaController = new MediaController( context: this);
mediaController.setMediaPlayer(this);
mediaController.setAnchorView(findViewById(R.id.layout));
```

Al crear un reproductor **MediaPlayer** éste se encuentra por defecto en estado **Idle**.

A continuación, es necesario indicarle qué elemento multimedia tiene que reproducir. Utilizaremos el método **setDataSource ()**. Le podemos pasar contenido multimedia de 4 formas distintas:

- A través de un **identificador de recurso**. Estamos hablando de indicarle mediante una id un fichero de audio o vídeo almacenado en **raw**. Esto no es muy recomendable dado que estaríamos añadiendo un peso considerable a nuestra app.

```
mediaPlayer.setDataSource( context: this,
        Uri.parse( uriString: "android.resource://"
                + getPackageName() + "/" + R.raw.always));
```

- Mediante una **ruta local**. En este caso estaríamos accediendo a un archivo almacenado en el propio dispositivo móvil como, por ejemplo, en una tarjeta SD.

- Usando una **URI (Uniform Resource Identifier)**, que podría ser perfectamente una URL para reproducir contenido online (streaming).

- Desde un **Content Provider** (Proveedor de Servicio).

Una vez asignado el **DataSource** el reproductor **MediaPlayer** pasa al estado **Initialized**

El siguiente paso consiste en pasar al estado **Prepared**, para lo que tan solo hay que invocar al método **prepare ()** o emplear el método estático **create ()**. A partir de aquí, el **MediaPlayer** ofrece tres métodos para controlar la reproducción del elemento multimedia: **start (), pause ()** y **stop ()**.

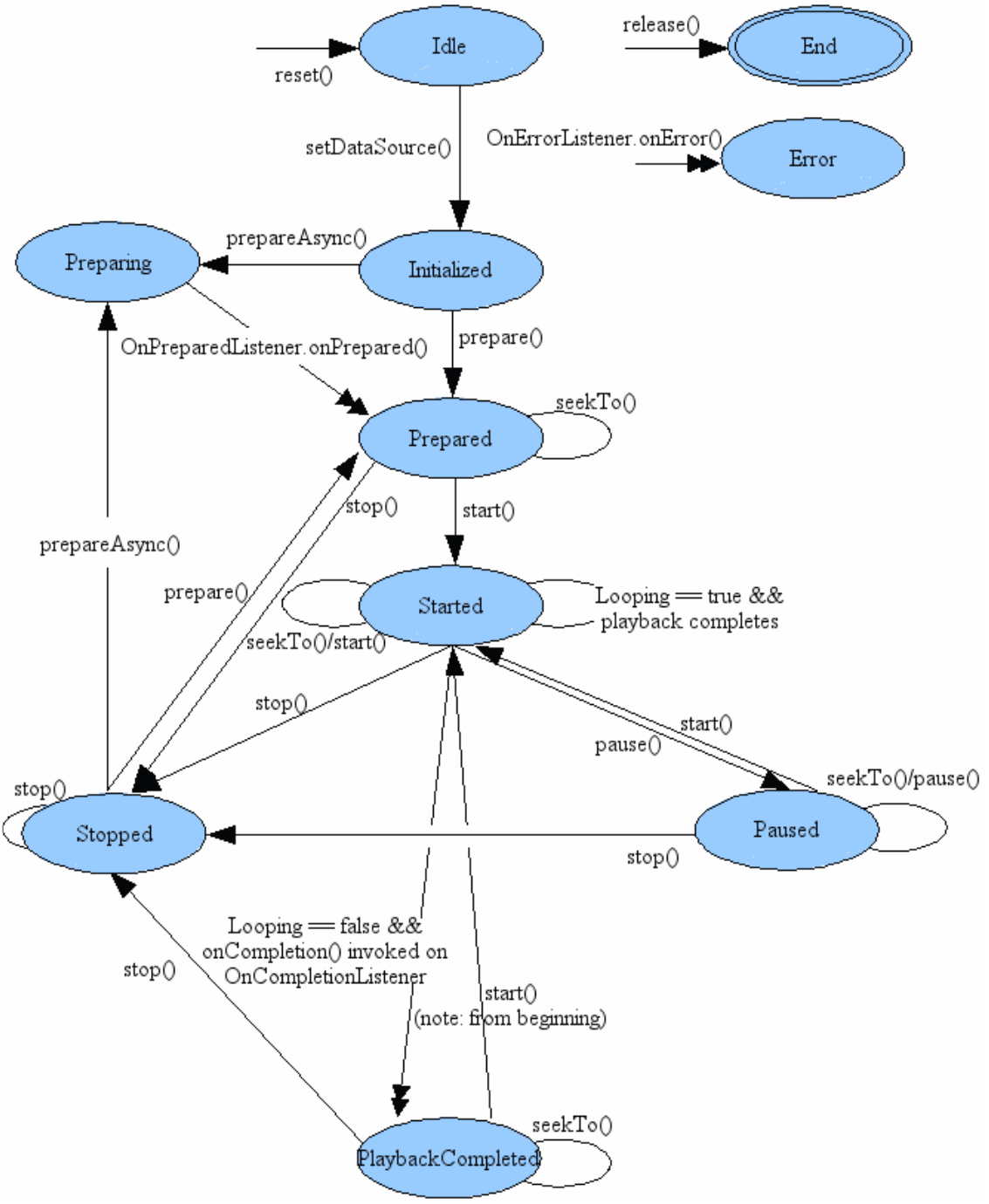
Obviamente, llamar al método start () pasa al reproductor **MediaPlayer** al estado **Started**

```
try {
    mediaPlayer.setDataSource( context: this,
            Uri.parse( uriString: "android.resource://"
                    + getPackageName() + "/" + R.raw.always));
    mediaPlayer.prepare();
    mediaPlayer.start();
} catch (IOException e) {
    Toast.makeText( context: this, "Ha ocurrido un error inesperado", Toast.LENGTH_LONG).show()
}
```

Controlando la reproducción

Cada uno de los métodos **start (), pause ()** y **stop ()** deja al reproductor en un estado diferente: **Started, Paused** o **Stopped**. Es necesario tener en cuenta que para volver al estado **Started** desde **Paused** tan solo hay que llamar a **start ()**; pero para volver desde **Stopped** debemos antes pasar por el estado **Prepared**, lo que requerirá llamar primero al método **prepare ()** y después a **start ()**.

¿Confuso? Bien, para eso existe este diagrama con todos los estados y métodos posibles.



Una vez que haya acabado la reproducción y no se necesite el **MediaPlayer**, es importante invocar al método **reléase ()** para liberar los recursos asociados a la reproducción.

Añadiendo controles estándar: la clase MediaController

Para facilitarnos el control de reproducción, Android incluye la clase **MediaController** que nos ofrece un interfaz con los botones habituales de un reproductor: play, pause, rewind, forward y una barra de progreso. También nos ofrece la interfaz típica de reproducción de vídeo o audio.

```
videoView = (VideoView) findViewById(R.id.idVideoView);
mediaController = new MediaController( context: this);
videoView.setMediaController(mediaController);
mediaController.setAnchorView(videoView);
videoView.setVideoURI(
        Uri.parse( uriString: "android.resource://" + getPackageName() + "/" + R.raw.bunny));
```

Es interesante ocultar el **MediaController** mientras se reproduce el elemento multimedia. Por defecto el **MediaController** se oculta pro sí solo cada 3 segundos, pero podríamos indicar otro comportamiento. Por ejemplo, podríamos aumentar el tiempo para cada estado del reproductor.

```
videoView.setOnPreparedListener( mp -> {
    mediaController.show( timeout: 20000);
    videoView.start();
});

videoView.setOnCompletionListener( mp -> {
    mediaController.show( timeout: 20000);
});
```

De la misma forma, podríamos querer recuperar el **MediaController** cuando el usuario toca la pantalla.

```
// Cuando el usuario toca la pantalla, mostrar el reproductor
public boolean onTouchEvent(MotionEvent event) {
    mediaController.show();
    return false;
}
```

Práctica 48

Realiza la práctica propuesta en el documento **11 - Audio y Video [Java].**

Práctica 49

Realiza la práctica propuesta en el documento **11 - Audio y Video [Kotlin].**