Lezione 21

Programmazione Android

- Architettura multimediale
  - Riproduzione di audio e video
  - Cattura di audio e video
  - Fotografia

Architettura multimediale

Riproduzione

Architettura multimediale

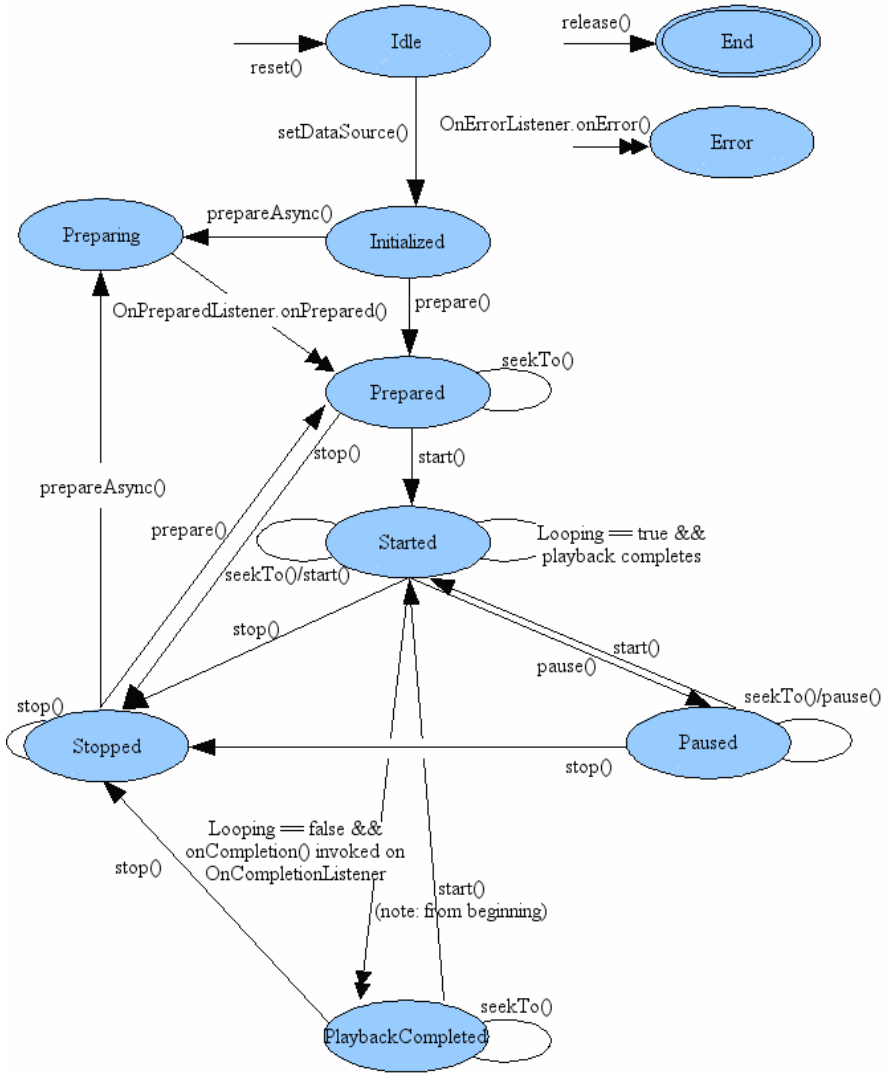
- Android offre un framework sofisticato per il supporto ad audio, foto e video
- Classi principali:
  - MediaPlayer – riproduce media
  - MediaRecorder – registra media
- Moltissime altre funzioni
  - Post-processing, transcoding, face recognition, streaming, audio routing, ecc.
- Trovate i vari componenti in **android.media.***

Riprodurre media

- La riproduzione di media richiede di **allocare** e **configurare** un MediaPlayer descrivendo:
  - Da dove ottenere i dati da riprodurre
  - Su quale dispositivo riprodurli
    - Eventualmente, controllando i dettagli: finestra da usare per il video, mixing dell'audio, ecc.
- Occorre poi **controllare** la riproduzione
  - Direttamente dall'utente: es: pulsanti Play/Pause/Stop
  - Da programma: es: effetti sonori di un videogioco
- E infine, **liberare** le risorse

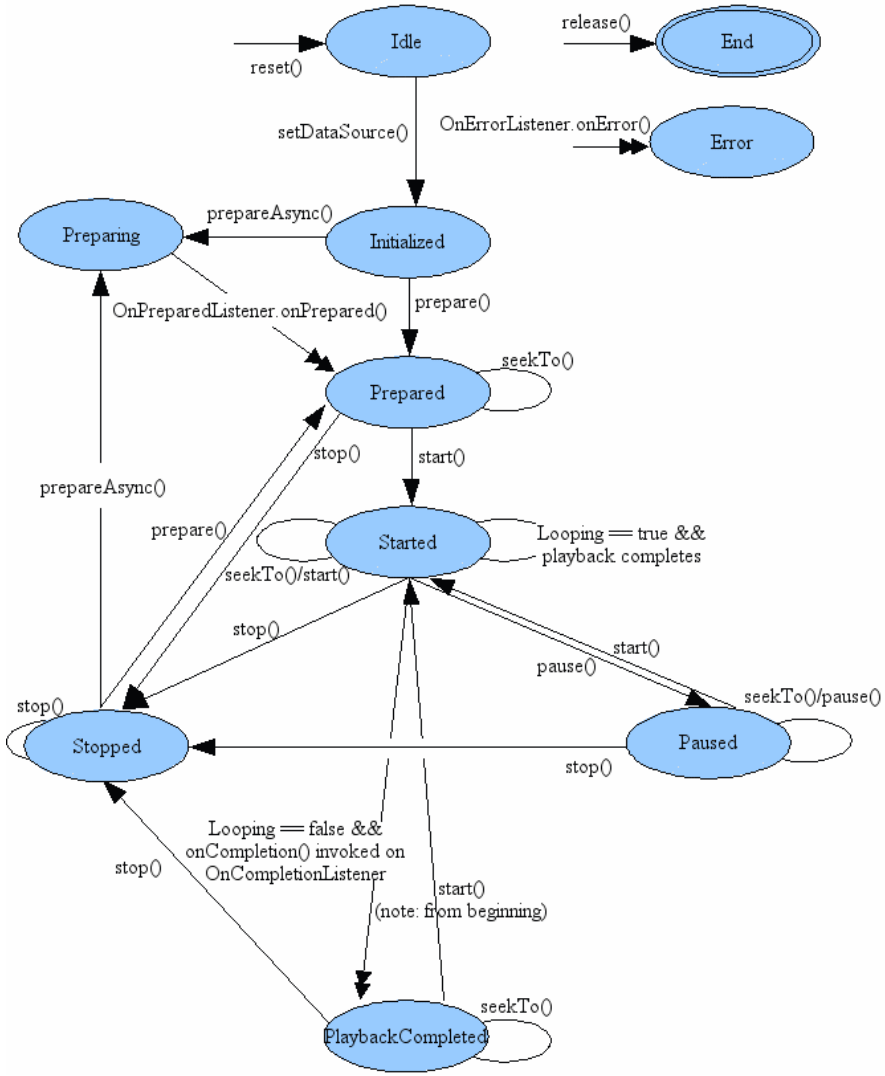
Stati del MediaPlayer

- Il MediaPlayer è un automa con vari stati
- Inizialmente in *Idle*, attende la configurazione poi è *Initialized*
- Segue il caricamento dei buffer o l'inizializzazione dello streaming (*Preparing*); quando tutto è pronto è *Prepared*



Stati del MediaPlayer

- A questo punto, segue il ciclo classico del playback
  - *Started-Paused-Stopped*
  - A fine media, va in *Playback completed*
- Se si verificano errori, può passare nello stato *Error*
- Dopo il rilascio delle risorse va in *End* (stato terminale)



Formati supportati

- Il MediaPlayer incorpora decoder per molti formati comuni
- In casi “strani”, si possono realizzare conversioni al volo da altri formati
- Formati “nuovi” supportati solo su Android recenti
- Audio
  - AAC, 3GP, FLAC, MP3, MIDI, Vorbis, PCM/WAV
- Immagini
  - JPEG, GIF, PNG, WebP
- Video
  - H263, H264, MPEG4, VP8 (WebM)

Sorgenti di dati

- File locali
  - Tramite path o FileDescriptor
- Risorse (raw)
  - Tramite R.raw. …
- URI locali
  - content:// …
  - Da ContentProvider
- URL remote
  - Statiche: http:// …
  - Streaming: rtsp:// …

Esempio: play da risorsa

- Nel caso più semplice, è sufficiente *creare* un'istanza di MediaPlayer e avviare la riproduzione

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.wagner);
mp.start();
```

- La risorsa dovrà essere un file in res/raw/wagner.*ext* (secondo il formato)
- La chiamata a start() si occupa anche di *preparare* la riproduzione
  - Qui non è necessario chiamare prepare()

Preparare i contenuti

- Mentre la creazione è un'operazione istantanea, e l'**avvio** della riproduzione è un evento, la **preparazione** può richiedere parecchio tempo
  - Lettura del file/risorsa, identificazione del formato, preparazione dei buffer, …
  - Apertura di una connessione TCP/IP, negoziazione della sessione, trasferimento dei dati, …
- Per questo motivo, la prepare() non deve **mai** essere fatta nel thread della UI

Preparare i contenuti

- La prepare() va quindi chiamata su un altro thread

- A mano:
  - Direttamente, con Thread
  - Indirettamente, con AsyncTask
- Una volta tornati da prepare(), si notifica il thread principale

- Pattern comune, offerto da MediaPlayer
  - Configuro la sorgente
  - Registro un listener
  - Chiamo prepareAsync() (che torna subito)
  - Quando il MediaPlayer è *prepared,* viene invocato un metodo del listener

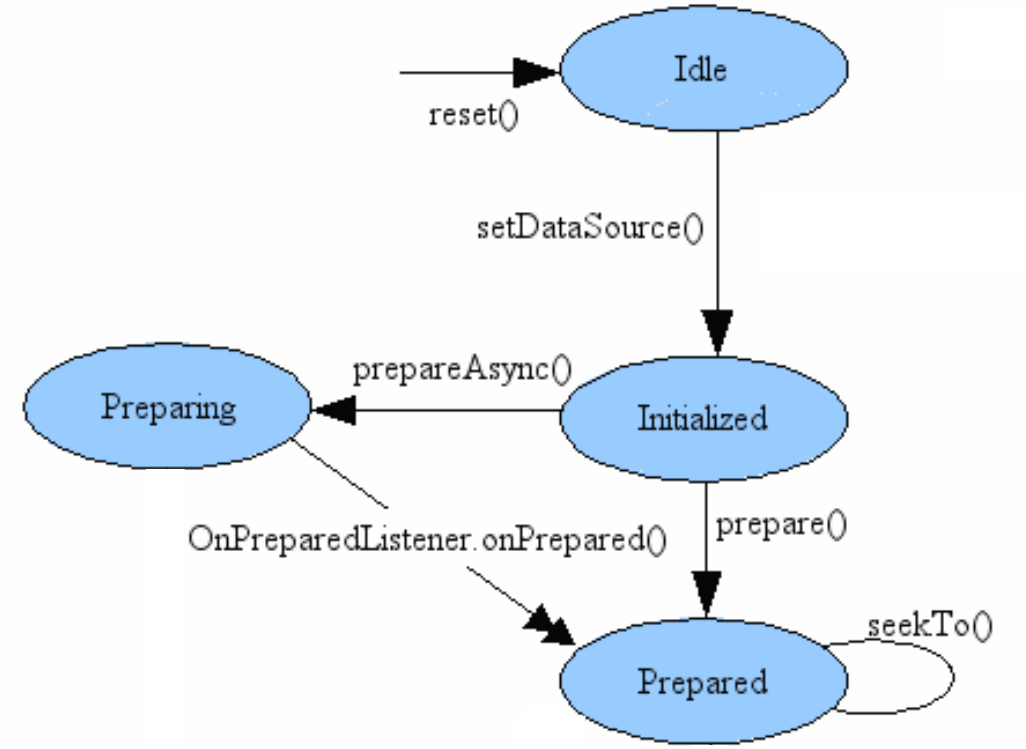
Esempio: play da rete

- Riprodurre audio da uno stream HTTP

```
MediaPlayer mp = new MediaPlayer();
mp.setAudioStreamType(AudioManager.STREAM_MUSIC);
mp.setDataSource("http://www.pirates.org/wagner.mp3");
mp.setOnPreparedListener(this);
mp.prepareAsync();
```

```
void OnPrepared(MediaPlayer mp)
{
    mp.start();
    // qui posso anche abilitare l'UI
}
```

Rispettate lo Stato!



- La maggior parte dei metodi del MediaPlayer sono **legali** solo in certi stati
- Per esempio
  - prepare() solo in Initialized
  - Start() solo in Prepared
  - Ecc.

Rilasciare un MediaPlayer

- Ogni istanza di MediaPlayer occupa una gran quantità di memoria e blocca altre risorse
  - Buffer audio/video
  - Socket per le connessioni di rete
- È **fondamentale** rilasciarle dopo l'uso!
  - Sempre: considerate un try/catch/finally

```
mp.release();
mp = null;
```

Rilascia risorse

Aiuta il GC a rilasciare la memoria

Riprodurre video

- Il caso di riproduzione di video è simile a quello per l'audio, con le ovvie differenze
- Occorre indicare una destinazione su schermo per il video, in uno di due modi:
  - **setSurface(Surface s)**
  - **setDisplay(SurfaceHolder sh)**
- Il SurfaceHolder è tipicamente ottenuto da una SurfaceView inserita nel layout
  - **sh = ((SurfaceView)findViewById(R.layout.video)).getHolder();**

Il Wake Lock

- Anche durante la riproduzione di audio o video, Android si sente libero di andare in modalità “sleep” (a basso consume energetico) fermando tutto
- Spesso questo non è desiderato
  - Riproduzione di musica in background: si vuole spegnere lo schermo, ma continuare il play
  - Riproduzione di video: si vuole mantenere acceso lo schermo e continuare il play
- L'applicazione può chiedere un **wake lock**

Il Wake Lock

- Il wake lock è un concetto globale, ma il MediaPlayer fornisce metodi di utilità per ottenerlo
- mp.setScreenOnWhilePlaying(b)
- mp.setWakeMode(ctx,flags)

Preferito per il video

Flag Value	CPU	Screen	Keyboard
PARTIAL_WAKE_LOCK	On*	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

Il Wake Lock

- In alternativa, si può indicare che il dispositivo non deve mai andare in sleep se è visualizzata una certa Activity
- Staticamente, nel file di layout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:keepScreenOn="true">
    ...
</RelativeLayout>
```

- Dinamicamente, in Java

```
getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
…
getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

Il Wake Lock

- **OVVIAMENTE** tenere lo schermo o la CPU sempre accesi senza un eccellente motivo, è una pessima idea
  - Assicuratevi di rilasciare il wake lock appena possibile
- Per gestioni più sofisticate potete usare il servizio di sistema PowerManager

```
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wl =
                    pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "TAG");

wl.acquire();
   …
wl.release();
```

Il Wake Lock

- Se la vostra applicazione manipola i wake lock, avrete bisogno di chiedere il permesso all'utente
- Come al solito, in AndroidManifest.xml:

```
<uses-permission
    android:name="android.permission.WAKE_LOCK"
/>
```

- Gli errori nella gestione dei wake lock (specie quelli espliciti) sono la principale fonte d'affari per i venditori di batterie extra!
  - “Non capisco, la batteria mi dura quattro ore...”

MediaController

- Per **controllare** un MediaPlayer si può usare il widget di sistema **MediaController**

Viene associato a un'altra View e offre la UI per controllare la riproduzione

MediaController

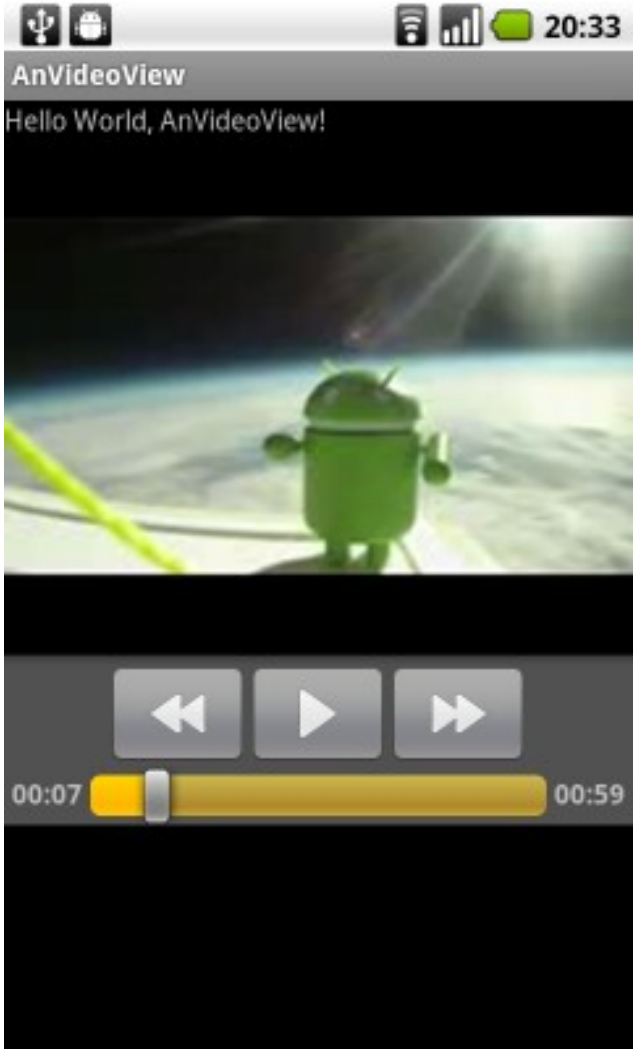
- Tipicamente, si prepara un MediaPlayer mp come abbiamo visto prima
- Si implementa un MediaPlayerControl mpc
  - Metodi che ne descrivono la configurazione: canPause(), canSeekForward/Backward(), ecc.
  - Metodi per le operazioni: start(), pause(), ecc. agiscono su mp ed eventualmente fanno altro
- Si inizializza un MediaController mc
- Lo si collega al MediaPlayer con mc.setMediaPlayer(mpc)
- Lo si collega a una vista con setAnchorView(v)
  - Di solito v è la SurfaceView connessa al mp

MediaController

- Una volta inizializzato, il MediaController si occupa di:
  - Comparire in overlay su v quando l'utente tappa lo schermo (scompare dopo qualche secondo)
  - Mantenere sincronizzato lo stato della UI con quello del mp
  - Passare i comandi dell'utente tramite UI al mpc che poi li passerà al mp
- Casi d'uso tipici: guardare un film sul tablet o un video su web

VideoView

- Ancora più ad alto livello, è disponibile il widget **VideoView**
- Incapsula praticamente tutta la gestione del video in una singola view
  - Uso assai facilitato
  - Limitate possibilità di configurazione
  - Interazione programmatica tramite mp come al solito



Esempio: VideoView

- App per mostrare un video da YouTube
  (es. tratto da android-coding.blogspot.it)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"                >
        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:text="@string/hello"                        />
        <LinearLayout
                android:orientation="vertical"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"          >
                <VideoView
                android:id="@+id/myvideoview"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"                 />
        </LinearLayout>
</LinearLayout>
```

File di layout
main.xml

Esempio: VideoView

Codice dell'Activity
EsVideoView.java

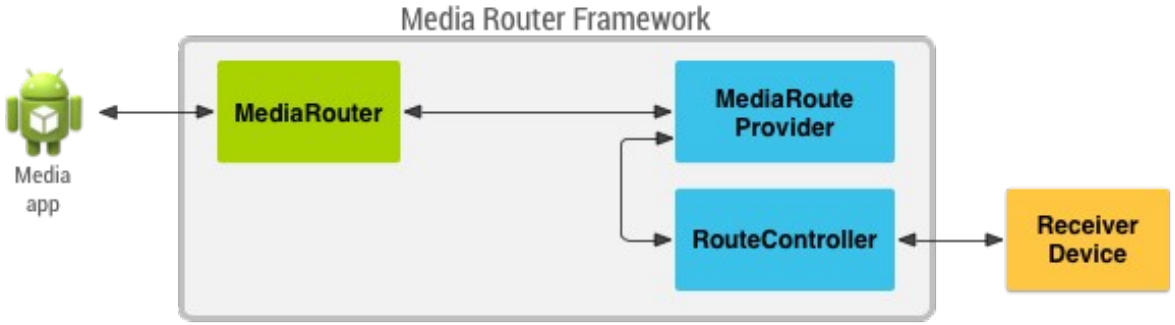
```
package …
import …
public class EsVideoView extends Activity
{
	String src =
"rtsp://v5.cache1.c.youtube.com/CjYLENy73wIaLQnhycnrJQ8qmRMYESARFEIJbXYtZ29vZ2xl
SARSBXdhdGNoYPj_hYjnq6uUTQw=/0/0/0/video.3gp";
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);
		VideoView myVideoView = (VideoView)findViewById(R.id.myvideoview);
		myVideoView.setVideoURI(Uri.parse(src));
		myVideoView.setMediaController(new MediaController(this));
		myVideoView.requestFocus();
		myVideoView.start();
	}
}
```

Alcuni altri temi

- La gestione dei media su Android è più potente di quanto abbiamo mostrato
- Fra i temi che non affrontiamo:
  - Gestire più flussi audio in contemporanea
    - Classe **AudioManager**
    - Es: il player musicale abbassa il volume quando arriva una telefonata
    - Es: il volume viene portato a un livello di sicurezza se si scollegano le cuffiette e si passa all'altoparlante

Alcuni altri temi

- La gestione dei media su Android è più potente di quanto abbiamo mostrato
- Non abbiamo però tempo per vederla a fondo...
  - Fare streaming di contenuti su altri dispositivi
    - Es: ChromeCast
    - Classi **MediaRouter** e **MediaRouterProvider**



Esempio rivisitato

- Una vecchia conoscenza: il RandomMusicPlayer
- Possiamo ora analizzarne più in dettaglio la parte multimediale
  - Vedi codice

Architettura multimediale

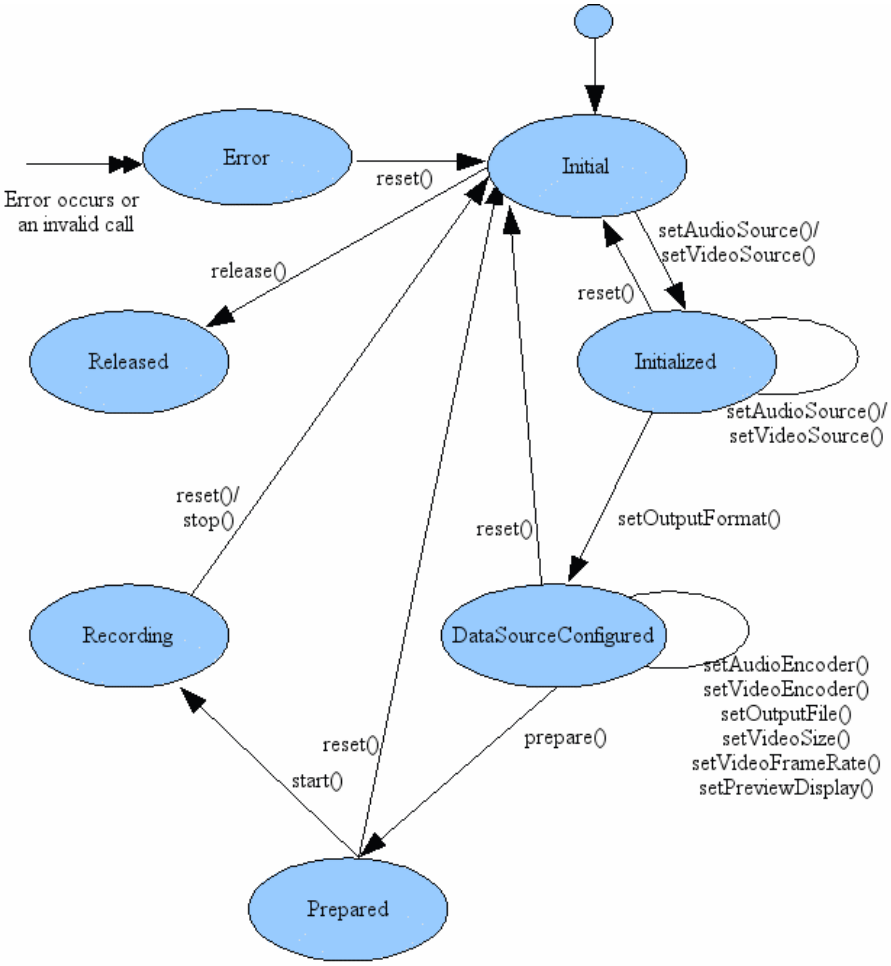
Registrazione

MediaRecorder

- L'analogo del MediaPlayer per la registrazione è una classe di nome **MediaRecorder**
- Anche in questo caso, si crea un'istanza, la si configura, e la si usa per registrare
- La configurazione è un po' più laboriosa
  - Il MediaPlayer poteva estrarre parametri dai dati della sorgente (in base al formato audio/video)
  - Per il MediaRecorder, i parametri vanno impostati dal programmatore
    - Che decide che tipo di file creare!

Stato del MediaRecorder

- Anche il MediaRecorder ha una macchina a stati
- Dallo stato *Initial*, si passa a *Initialized* impostando una **sorgente** per la registrazione
- Da lì, si passa a *DataSourceConfigured* impostando il formato di output
- A questo punto, si chiama prepare() e si va in *Prepared*
- Solo a questo punto si può iniziare la registrazione, passando in *Recording*
- Al termine della registrazione, si torna in *Initial*



MediaRecorder state diagram

Sorgenti

- La sorgente da cui registrare (audio o video) si imposta chiamando setAudioSource() e/o setVideoSource()
- Costanti in MediaRecorder.AudioSource

CAMCORDER	Microphone audio source with same orientation as camera if available, the main device microphone otherwise
DEFAULT	Default audio source
MIC	Microphone audio source
REMOTE_SUBMIX	Audio source for a submix of audio streams to be presented remotely.
VOICE_CALL	Voice call uplink + downlink audio source
VOICE_COMMUNICATION	Microphone audio source tuned for voice communications such as VoIP.
VOICE_DOWNLINK	Voice call downlink (Rx) audio source
VOICE_RECOGNITION	Microphone audio source tuned for voice recognition if available, behaves like DEFAULT otherwise.
VOICE_UPLINK	Voice call uplink (Tx) audio source

Sorgenti

- La sorgente da cui registrare (audio o video) si imposta chiamando setAudioSource() e/o setVideoSource()
- Costanti in MediaRecorder.VideoSource
  - Sono definiti solo due valori:
    - CAMERA – una fotocamera incorporata
    - DEFAULT – la CAMERA di cui sopra :)
  - È però possibile scegliere una fotocamera, se ne è disponibile più di una, con setCamera(Camera c)
    - **Camera** è la classe che controlla i dettagli "fotografici", scatta le foto, ecc.

Formati di output

- La varietà di formati supportati dipende dalla versione di Android
- Impostato con setOutputFormat()
- Definiti da costanti in MediaRecorder.OutputFormat
- Al momento:
  - AAC_ADTS, AMR_NB, AMR_WB, MPEG_4, THREE_GPP, DEFAULT

Encoding

- Analogamente, è possibile impostare gli encoder audio e video
- Per l'audio: setAudioEncoder()
  - Costanti definite in MediaRecorder.AudioEncoder
    - AAC, AAC_ELD, AMR_NB, AMR_WB, HE_AAC, DEFAULT
- Per il video: setVideoEncoder()
  - Costanti definite in MediaRecorder.VideoEncoder
    - H263, H264, MPEG_4_SP, DEFAULT

Esempio

```
MediaRecorder mr = new MediaRecorder();
mr.setAudioSource(MediaRecorder.AudioSource.MIC);
mr.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mr.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
mr.setOutputFile(PATH_NAME);
mr.prepare();
mr.start();
…
mr.stop();
mr.reset();
mr.release();
mr=null;
```

Il **mr** deve essere creato in un thread che ha un Looper associato (in pratica: il thread della UI).

Come al solito, passi come prepare() possono richiedere del tempo, e può essere importante eseguirli in modalità asincrona.

Una limitazione importante

- Disgraziatamente, l'output del MediaRecorder deve essere un file “vero”
  - Definito da un FileDescriptor – l'equivalente Java di un file descriptor del kernel Linux
- Questo impedisce di inviare direttamente i dati prodotti in un buffer di memoria, o verso un socket, ecc.
- Motivazione: tutto il MediaRecorder è implementato in codice nativo (in C), per efficienza, non in Java

Approccio alternativo

- È naturalmente possibile chiedere ad altre app di fare la registrazione di un video tramite intent:

```
Intent vi = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
if (vi.resolveActivity(getPackageManager()) != null) {
   startActivityForResult(vi, 1);
}
```

- E poi prendere il risultato:

```
@Override
protected void onActivityResult(int req, int res, Intent resi) {
    if (req == 1 && res == RESULT_OK) {
        Uri video = resi.getData();
        myVideoView.setVideoURI(video);
    }
}
```

Architettura multimediale

Fotografia

Accesso alla camera

- Nella maggior parte dei casi, non è opportuno accedere direttamente alla fotocamera
  - Meglio inviare un Intent dicendo che si vuole scattare una foto, e lasciare che parta l'app di fotocamera preferita dall'utente
  - Action predefinite:
    - MediaStore.ACTION_IMAGE_CAPTURE
    - MediaStore.ACTION_VIDEO_CAPTURE
  - Si inviano con startActivityForResult()
  - Nella onActivityResult() si ricevono i dati della foto scattata o del video ripreso dall'altra app

Accesso alla camera

- L'alternativa consiste nello sviluppare una propria app di fotografia “incorporata” nell'app principale
- Generalmente, non è una buona idea: troppo complicato
- Se lo si vuole davvero fare, sarà necessario:
  - Scoprire le caratteristiche hardware delle fotocamere disponibili sul dispositivo
  - Collegare una camera a una SurfaceView per avere una preview
  - Implementare una UI per impostare parametri fotografici e scattare le foto
  - Ricevere i dati “grezzi” dalla fotocamera e/o convertirli in un formato umano
  - Salvare i dati da qualche parte

Accesso alla camera
v1 — API < 21

- La classe principale da utilizzare è **Camera**
- Fornisce metodi statici per enumerare le fotocamere disponibili
- Si ottiene un'istanza con Camera.open(), poi si lavora con quella
- Metodi per configurare i dettagli fotografici
- Il metodo takePicture() “scatta” la foto
- I dati vengono passati (come array di byte) al callback onPictureTaken() in maniera asincrona

Alcune features

Feature	API Level	Description
Face Detection	14	Identify human faces within a picture and use them for focus, metering and white balance
Metering Areas	14	Specify one or more areas within an image for calculating white balance
Focus Areas	14	Set one or more areas within an image to use for focus
White Balance Lock	14	Stop or start automatic white balance adjustments
Exposure Lock	14	Stop or start automatic exposure adjustments
Video Snapshot	14	Take a picture while shooting video (frame grab)
Time Lapse Video	11	Record frames with set delays to record a time lapse video
Multiple Cameras	9	Support for more than one camera on a device, including front-facing and back-facing cameras
Focus Distance	9	Reports distances between the camera and objects that appear to be in focus
Zoom	8	Set image magnification
Exposure Compensation	8	Increase or decrease the light exposure level
GPS Data	5	Include or omit geographic location data with the image
White Balance	5	Set the white balance mode, which affects color values in the captured image

Video Snapshot	14	Take a picture while shooting video (frame grab)
Time Lapse Video	11	Record frames with set delays to record a time lapse video
Multiple Cameras	9	Support for more than one camera on a device, including front-facing and back-facing cameras
Focus Distance	9	Reports distances between the camera and objects that appear to be in focus
Zoom	8	Set image magnification
Exposure Compensation	8	Increase or decrease the light exposure level
GPS Data	5	Include or omit geographic location data with the image
White Balance	5	Set the white balance mode, which affects color values in the captured image
Focus Mode	5	Set how the camera focuses on a subject such as automatic, fixed, macro or infinity
Scene Mode	5	Apply a preset mode for specific types of photography situations such as night, beach, snow or candlelight scenes
JPEG Quality	5	Set the compression level for a JPEG image, which increases or decreases image output file quality and size
Flash Mode	5	Turn flash on, off, or use automatic setting
Color Effects	5	Apply a color effect to the captured image such as black and white, sepia tone or negative.
Anti-Banding	5	Reduces the effect of banding in color gradients due to JPEG compression
Picture Format	1	Specify the file format for the picture
Picture Size	1	Specify the pixel dimensions of the saved picture

Accesso alla camera
v2 — API ≥ 21

- Package android.hardware.camera2
- Una camera è vista come una *pipeline* di elementi
- **CameraManager** gestisce le fotocamere

```
CameraManager cm = (CameraManager)getSystemService(CAMERA_SERVICE);
```

- Vari metodi per ottenere l'elenco e le caratteristiche delle fotocamere

```
String[] cids = cm.getCameraIdList();
CameraCharacteristics cch = cm.getCameraCharacteristics(cids[0]);
```

- L'app apre una *connessione* con la camera
  - Modello già visto: automa + messagge queue + callback

```
cm.openCamera(cid, callback, handler)
```

Accesso alla camera
v2 — API ≥ 21

- Tutta l'interazione avviene tramite la callback, i cui metodi ricevono un **CameraDevice**:
  - onOpened(cd) – camera aperta e pronta per l'uso
  - onClosed(cd) – camera chiusa (in seguito a cd.close())
  - onDisconnected(cd) – connessione caduta
  - onError(cd, error) – si è verificato un errore
- L'oggetto cd ha poi tutti i metodi che potete immaginarvi
  - Per esempio, …

```
createReprocessableCaptureSessionByConfigurations(I
nputConfiguration inputconfig,
List<OutputConfiguration> outputs,
CameraCaptureSession.StateCallback callback,
Handler handler)
```

Accesso alla camera

Jetpack — alpha!

- Parte di Jetpack, al momento in alpha
  - Integrazione con i componenti architetturali di Jetpack
  - Fornisce supporto a estensioni vendor-specific
  - Retrocompatibile fino ad Android 5.0 (API 21)
  - Supportata solo su modelli specifici:
    - Huawei (HDR, Portrait): Mate 20 series, P30 series, Honor Magic 2, Honor View 20
    - Samsung (HDR, Night, Beauty): Galaxy S10 series

Accesso alla camera

Jetpack — alpha!

- Parte di Jetpac
  - Integrazione c
  - Fornisce supp
  - Retrocompatib
  - Supportata sol
    - Huawei (HDR, Portra
    - Samsung (HDR, Nigh
  - … o forse no!
    - “wait and see”

Esercizio finale

- Scrivere una semplice applicazione che consenta di registrare micro-filmati da 10 secondi, salvandoli sulla memoria condivisa, e di riprodurli
- Opzionalmente
  - Inviare i micro-filmati ad altri utenti nelle vicinanze, (hint: la prossima volta vedremo wi-fi direct)
  - Consentire di visualizzare i micro-filmati ricevuti
  - Approccio “effimero” - i micro-filmati ricevuti vengono cancellati immediatamente dopo la visione
    - Ma sono mantenuti in memoria fino alla visione