# Dispositivos Móveis

## Aula 12 - Multimídia

# Apresentação

Olá! Hoje estaremos estudando os principais aspectos de multimídia no Android e como utilizá-los. Na primeira parte da aula estudaremos o MediaPlayer e alguns outros aspectos relacionados à reprodução de áudio e vídeo em aplicações Android. Na sequência, estudaremos como utilizar a câmera para fotografar e filmar dentro de nossa aplicação e como poderemos utilizar o resultado obtido. Essa é uma aula bastante divertida, pois trata de aspectos que podem dar uma nova cara à aplicação. Vamos a ela!

**Vídeo 01** - Apresentação

## Objetivos

Ao final desta aula, você deverá ser capaz de:

- Reproduzir áudio e vídeo dentro de sua aplicação.
- Utilizar a câmera em sua aplicação.

# Reprodução de Mídia

O Android permite aos desenvolvedores incluir suporte a reprodução de diversos formatos diferentes de áudio e vídeo em suas aplicações. Para tornar isso possível da maneira mais fácil, o Android disponibiliza a API MediaPlayer, que é capaz de tocar áudio e vídeo que esteja guardado nas pastas da aplicação, no sistema de arquivos do dispositivo ou até mesmo em um endereço conhecido, na internet. Esses arquivos podem ser reproduzidos em formatos diversos. Mais informações sobre os tipos suportados, bem como sobre o MediaPlayer podem ser encontrados no endereço http://developer.android.com/guide/appendix/media-formats.html (último acesso em 05 jun. 2015).

Para começar a utilizar o MediaPlayer em nossa aplicação, precisamos, mais uma vez, começar adicionando ao Manifest as permissões necessárias, caso existam. Para utilizar o MediaPlayer localmente e sem nenhuma interrupção aos serviços do Android, não é necessário configurar nenhuma permissão, porém, caso se deseje acessar uma mídia da internet, é necessário adicionar a permissão de internet ao Manifest. Essa permissão é a **android.permission.INTERNET**.

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

Outra funcionalidade que é utilizada muitas vezes quando se está reproduzindo uma mídia, principalmente um vídeo, é a interrupção do escurecimento da tela. É importante que, ao reproduzir um vídeo, sua aplicação mantenha a tela acesa o tempo inteiro, para que o usuário possa acompanhar o vídeo, sem precisar interagir com o dispositivo para evitar o escurecimento da tela.

Para que seja possível interromper esse serviço, é necessário declarar a preferência **android.permission.WAKE_LOCK**. Essa preferência avisa ao usuário que sua aplicação poderá suspender o escurecimento da tela e, com isso, gastar uma quantidade maior de energia, ao longo do tempo de execução da aplicação.

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

Conhecidas as principais permissões necessárias, vejamos agora como funciona o MediaPlayer.

---

## MediaPlayer

A classe MediaPlayer é a classe mais importante quando se trata de reprodução de mídias no Android. Através dessa classe é possível, com pouca configuração, reproduzir em sua aplicação diversos formatos de áudio e vídeo, vindos diretamente dos recursos da aplicação, do sistema de arquivos e até mesmo da rede. Para ver uma lista completa dos formatos suportados pelo MediaPlayer do Android, acesse o link disponível na seção **Leitura Complementar**. Vamos agora ver como reproduzir arquivos de áudio e vídeo em cada uma das três maneiras que citamos.

A primeira maneira de reproduzir arquivos de áudio ou vídeo é através de um arquivo incluso dentro dos recursos da aplicação. Esse arquivo deve ser um arquivo de formato suportado pelo Android e estar colocado dentro da pasta raw, dos Resources. Como vimos na aula sobre a estrutura de pastas do Android, os arquivos inseridos nessa pasta não sofrem nenhum tipo de alteração e são passados adiante no formato que são. Uma vez que tenhamos colocado o arquivo nessa pasta, basta criar uma instância do MediaPlayer para reproduzi-lo e então iniciar a reprodução. A **Listagem 1** indica o código necessário para fazer essa inicialização.

Esse código deve ser adicionado em seu método que executará a mídia. Caso queira tocá-la desde o início da aplicação, é possível adicioná-lo ao onCreate, por exemplo.

```
1 Context mContext = getApplicationContext();
2 MediaPlayer mediaPlayer = MediaPlayer.create(mContext, R.raw.sa);
3 mediaPlayer.start();
```

**Listagem 1** - Reproduzindo um arquivo dos recursos utilizando o MediaPlayer

É importante lembrar que arquivos inseridos dentro da pasta raw serão considerados parte de sua aplicação e com isso serão adicionados ao APK gerado e poderão fazer com que este se torne bem grande, dependendo do que seja colocado.

Como você já deve estar imaginando, para parar de tocar a mídia que está sendo reproduzida pelo MediaPlayer, basta chamar o método *stop.* Outros comandos são necessários para preparar o arquivo antes da reprodução, porém, no caso específico da reprodução de arquivos que estão inclusos nos recursos de sua aplicação, a utilização do método *create* faz toda essa preparação para você e já cria uma instância pronta para ser utilizada. Para o caso de a reprodução de mídias vindas do sistema de arquivos do Android, alguns passos adicionais são necessários, como veremos adiante.

O primeiro passo parar conseguir carregar um arquivo do sistema de arquivos é descobrir onde o arquivo está localizado. A reprodução do arquivo depende de um URI que é responsável por indicar o caminho até o arquivo a ser tocado. Depois de iniciar o URI que localizará o arquivo a ser reproduzido, mais alguns passos devem ser seguidos para que o mesmo possa ser tocado, como podemos ver na **Listagem 2**.

Mais uma vez, basta adicionar esse código ao método que será responsável por executar a mídia.

```
1  Uri myUri = Uri.parse("Alguma URI");
2  MediaPlayer mediaPlayer = new MediaPlayer();
3
4  mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
5
6     try {
7       mediaPlayer.setDataSource(getApplicationContext(), myUri);
8       mediaPlayer.prepare();
9     } catch (IllegalArgumentException  e) {
10    } catch (SecurityException e) {
11    } catch  (IllegalStateException e) {
12    } catch (IOException e) {
13 }
14
15 mediaPlayer.start();
```

**Listagem 2** - Reproduzindo um arquivo local a partir de uma URI

Vemos na **Listagem 2** diversos passos além dos que fizemos na **Listagem 1**, para abrir um arquivo de recurso. Isso acontece porque novas variáveis são adicionadas ao processo quando precisamos buscar um arquivo em um caminho específico do aparelho e não um caminho dentro de nossos recursos. Vamos discutir linha a linha o que é necessário para reproduzir mídia dessa maneira.

Inicialmente, precisamos definir a URI onde acessaremos o arquivo. Uma vez definida a URI, precisamos de uma instância do MediaPlayer. Criamos uma normalmente e então passamos a configurá-la para garantir a reprodução correta do arquivo. A primeira configuração que devemos fazer é relacionada ao tipo de *stream* que estaremos utilizando em nosso MediaPlayer. Diversos tipos de reprodução são disponíveis e podem ser consultados na classe AudioManager. A constante STREAM_MUSIC é a responsável por indicar que o arquivo será uma música.

Após configurar o tipo do *stream* de áudio, devemos configurar o local que nos fornecerá a mídia que estaremos reproduzindo. Como vemos na **Listagem 2**, precisamos passar a esse método apenas o contexto da aplicação e o URI que indica onde está o arquivo a ser carregado. É importante notar que esse método retorna algumas exceções que devem ser tratadas ou passadas adiante. Normalmente, a melhor solução é avisar ao usuário do problema ocorrido e voltar a um estado estável para a aplicação prosseguir.

---

Com o arquivo que será executado configurado e carregado, podemos, então, preparar a execução dele para o usuário final. Isso é feito através do método *prepare.* Esse método é responsável por carregar o que for necessário do arquivo, ler os dados e deixá-lo pronto para ser reproduzido. Mesmo no caso de o arquivo estar no sistema de arquivos, é importante notar que o *prepare* precisa ser chamado e pode demorar algum tempo para ser completamente executado, de acordo com o arquivo que esteja sendo preparado.

Uma vez que o MediaPlayer tenha finalizado a preparação do arquivo, o próximo passo é, finalmente, reproduzi-lo. Para fazer isso, da mesma maneira que fizemos anteriormente, precisamos apenas chamar o método *start*.

A partir daí, sua aplicação estará executando a mídia que tenha sido carregada. Para parar, o método *stop* deve ser chamado. É importante notar que uma vez que a mídia tenha sido parada, ela deve ser novamente preparada para que volte a ser executada.

Outros métodos do MediaPlayer também podem ajudar a atingir a reprodução desejada. Consulte-os no autocompletar da IDE ou no site do Android Developers.

Agora que já sabemos reproduzir arquivos vindos do sistema de arquivos do dispositivo, vamos ver, na **Listagem 3**, as mudanças necessárias no código para que possamos reproduzir mídia vinda diretamente da internet.

```
String url = "http://site.para.stream";
MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);

try {
     mediaPlayer.setDataSource(url);
     mediaPlayer.prepare();
   } catch (IllegalArgumentException e) {
   } catch (IllegalStateException e) {
   } catch (IOException e) {
}

mediaPlayer.start();
```

**Listagem 3** - Carregando arquivos da internet para reprodução

A **Listagem 3** nos mostra que para carregar e executar arquivos da rede poucas mudanças são necessárias, graças à praticidade nos fornecida pela API do MediaPlayer. As duas mudanças notáveis no código são que agora o DataSource é setado para um URL e não para um URI, e também passamos a não precisar mais a SecurityException. Fora isso, temos exatamente o mesmo procedimento.

O que é importante notar aqui, quando fazemos o carregamento da mídia através da rede, é o tempo que será necessário para que o *prepare* seja executado. Perceba que fatores como sobrecarga da rede podem afetar drasticamente o tempo necessário para baixar o arquivo e, com isso, pode ser que o programa chegue a travar nesse comando. Na próxima seção, veremos a maneira correta de evitar isso. Também perceba a necessidade de que o URL passado seja direto para um arquivo de um tipo que possa ser carregado pelo Android e que o URL permita ao usuário fazer o *download* progressivamente. Passando essas limitações, basta carregar o arquivo como mostrado na **Listagem 3** e tocá-lo.

---

## Preparando Mídia de Maneira Assíncrona

Como vimos anteriormente, reproduzir mídia utilizando o MediaPlayer não é uma tarefa complicada. Apesar disso, existem alguns cuidados que devemos tomar quando estamos lidando com arquivos. No caso do MediaPlayer, o maior desses

problemas está no método *prepare.* Como citamos anteriormente, mesmo que se trate de um arquivo local, o método pode levar um tempo longo para executar, fazendo com que a *thread* que o está executando fique parada por um tempo, se dedicando exclusivamente ao processamento desse método. Percebemos, assim, que se ele for chamado na *thread* principal, a aplicação ficará bloqueada por um tempo, fazendo com que o usuário perceba uma aplicação lenta (mesmo que seu *prepare* leve apenas poucos segundos) e também criando o risco da aplicação gerar um ANR (janela de erro indicando que a *Activity* não está respondendo). Para evitar esses riscos, devemos sempre executar o prepare em uma *thread* separada.

Apesar de ser possível fazer isso da maneira convencional, criando uma nova *thread*, executando o método nela, tratando a comunicação etc, também é possível utilizar um mecanismo já incluso no MediaPlayer e que também é bem mais simples do que seguir pelo caminho anterior. Ao criar um MediaPlayer, é possível adicionar a ele um PreparedListener, através do método setOnPreparedListener*,* que recebe um OnPreparedListener. Após implementar corretamente o onPreparedListener, podemos substituir a chamada do *prepare* pela chamada ao prepareAsync.

Ao utilizar o prepareAsync, o MediaPlayer cria uma nova *thread* para você, em que toda a preparação do arquivo selecionado será executada e, então, uma vez que o arquivo esteja pronto, a *thread* retornará imediatamente para o onPreparedListener que esteja configurado no MediaPlayer que executou o prepareAsync. Com isso, toda a preparação necessária pode ser executada sem bloquear a *thread* principal, fazendo com que seu programa possa ser executado com fluidez e ainda assim execute de maneira correta a reprodução da mídia, seja ela de arquivo ou da web, onde isso se torna ainda mais crítico.

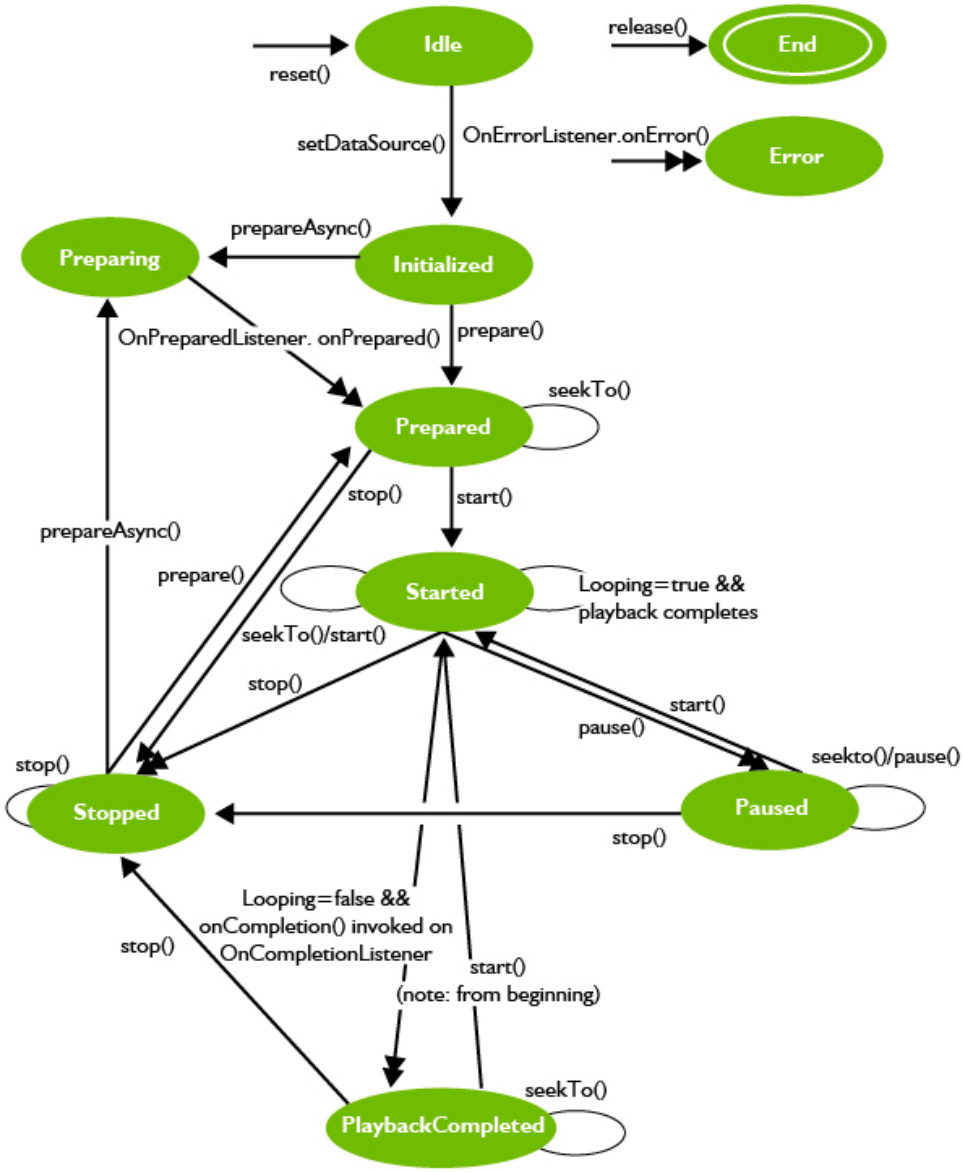
---

## Tratando os Estados do MediaPlayer

Quando estamos criando aplicações que se utilizam do MediaPlayer, devemos sempre ter em mente que o MediaPlayer funciona baseado em estados. É importante ter isso implementado corretamente em todos os pontos da aplicação, pois comandos aplicados em estados errados do MediaPlayer são a principal causa de *bugs* em aplicações que se utilizam dessa API. Um exemplo claro disso seria executar um *start* antes de ter o MediaPlayer configurado para a execução do arquivo. O MediaPlayer não teria o que inicializar e isso geraria um problema

interno. O diagrama de estados mostrado na **Figura 1** mostra de maneira completa os estados do MediaPlayer e como atingir cada um deles. Vejamos a **Figura 1** antes de passar brevemente pelos estados mais importantes.

**Figura 01** - Diagrama de estados do MediaPlayer.




**Fonte**: Android Developers

Como podemos ver, o diagrama completo é bem complexo e possui diversas alterações que podem ser geradas entre os estados. Nesta aula, estudaremos apenas as principais transições. Primeiramente, ao criar ou reiniciar o MediaPlayer, vemos que ele está no estado Idle. Esse estado é incapaz de fazer qualquer ação. Após ser inicializado e entrar nesse estado, devemos configurar a fonte da mídia a

ser reproduzida pelo MediaPlayer. Essa configuração gerará a mudança para o estado Initialized. Esse estado indica que o MediaPlayer está inicializado. Para utilizarmos essa instância precisamos ainda preparar a mídia que será executada. Como vimos, isso acontece através do comando *prepare* ou, preferencialmente, prepareAsync*.* Caso o comando assíncrono seja o escolhido, o MediaPlayer passará por um estado de preparação antes de chamar o onPreparedListener e entrar no estado preparado. Caso a versão síncrona seja chamada, o MediaPlayer irá executar a preparação na *thread* que está e irá direto ao estado preparado.

Uma vez no estado preparado, o MediaPlayer está pronto para ser utilizado e pode receber diversos comandos de interação com a mídia. O principal deles, como já vimos, é o *start*, responsável por começar a execução da mídia preparada. Fora ele, ainda temos a possibilidade de utilizar o comando seekTo, que irá mover a reprodução para um ponto específico da mídia. Quando o comando *start* for executado, o estado passará a iniciado e o MediaPlayer suportará comandos como *pause* e *stop*. O último ponto importante a se notar em relação a esse diagrama é a necessidade de se preparar novamente o arquivo antes de voltar a executá-lo, caso este seja parado pelo método *stop* . Tentar voltar à reprodução diretamente, sem passar pela preparação, após o *stop*, é uma das grandes fontes de problemas quando estamos utilizando essa API

---

## Liberando o MediaPlayer

O MediaPlayer, até mesmo pela sua função, consome recursos diversos do sistema. Cada nova instância dele que é criada consome mais desses recursos, podendo chegar a um ponto que isso gera instabilidade ao sistema. Por esse motivo, é importante que sempre que o MediaPlayer deixe de ser utilizado, ele seja liberado.

Como estamos estudando aqui maneiras de implementar o MediaPlayer em uma *Activity* e não em um *Service* (que veremos nas próximas aulas), não faz nenhum sentido que guardemos uma cópia do MediaPlayer se a *Activity* for parada além, é claro, do caso em que o MediaPlayer não vai mais ser utilizado dentro da *Activity*. Em ambos esses casos, devemos liberar os recursos do MediaPlayer. Para fazer isso, devemos, no método onStop de nossa *Activity*, ou quando o MediaPlayer se tornar inútil, utilizar o método *release* para que ele possa liberar os recursos utilizados pelo MediaPlayer e, na sequência, tornar a instância do MediaPlayer nula,

fazendo com que o MediaPlayer seja igual a *null.* Ao tomar essas duas precauções, os recursos do MediaPlayer serão liberados e a instância dele que tiver em memória será limpa pela máquina virtual, já que não estará mais sendo utilizada.

É importante notar que, principalmente no caso de parar o MediaPlayer no onStop, devemos reiniciar o MediaPlayer desde o primeiro passo no caso de voltarmos a utilizá-lo. Isso deve ser feito no método onStart de sua *Activity*. Com isso, poderemos garantir que no caso de o usuário virar o aparelho, por exemplo, o MediaPlayer seja recriado e o antigo seja deletado. Caso queira manter a reprodução, deve-se salvar o ponto atual de reprodução antes do encerramento e, após a reinicialização, utilizar, após o *prepare*, o método seekTo para posicionar a reprodução onde se deseja. Encerramos assim os nossos estudos sobre reprodução de mídia.

**Vídeo 02** - Tocando Mídias da Biblioteca de Músicas

## Atividade 01

1. Crie um novo projeto e copie uma música para a pasta raw, dentro dos recursos de seu projeto. Na *Activity* principal, crie uma instância do MediaPlayer e reproduza a música copiada.

## Câmera

Outra funcionalidade bastante interessante que o Android disponibiliza aos seus desenvolvedores é a possibilidade de utilizar a câmera do dispositivo em sua aplicação. Existem aparelhos Android que não possuem câmera, porém, a grande maioria deles tem acesso a uma ou até duas câmeras. Isso dá aos desenvolvedores mais caminhos para trabalhar com o seu aplicativo.

Assim como vimos na parte sobre MediaPlayer, aplicações que utilizam a câmera precisam declarar isso no Manifest da aplicação. No caso da câmera, ao declarar a utilização dela no Manifest, é possível configurar ou não se ela é necessária para a aplicação. Se for apenas uma funcionalidade extra, é possível dizer que ela não é necessária e permitir que dispositivos que não possuem câmera instalem a sua aplicação. Porém, caso a câmera seja indispensável a sua aplicação, a declaração dela no Manifest faz com que o Google Play não permita aparelhos que não possuem uma câmera instalar seu aplicativo. Para declarar que a câmera é utilizada em sua aplicação, a seguinte linha deve ser adicionada no seu Manifest:

```
1 <uses-feature android:name = "android.hardware.camera" />
```

Com isso, o Google Play é capaz de filtrar e mostrar sua aplicação apenas para aparelhos que tem acesso a câmeras. Caso a câmera seja uma funcionalidade extra de sua aplicação, explicite isso na declaração feita no Manifest, da seguinte maneira:

```
1 <uses-feature android:name="android.hardware.camera" android:required="false" />
```

Em relação à permissões, a primeira que deve ser adicionada é a permissão de utilização da câmera do dispositivo, através da linha:

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

Também é importante notar que, caso sua aplicação vá gravar um vídeo pela câmera, ao invés de fotos, é necessário requisitar também a permissão da gravação de áudio, pois ele estará junto ao vídeo. Para fazer isso, adicionamos ao Manifest a seguinte linha:

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

Caso sua aplicação vá salvar as imagens e vídeos capturados, também precisamos requerer permissão para tal. Isso garante que só escreverá na memória do dispositivo quem for autorizado a tal. Para pedirmos essa permissão, deveremos incluir no Manifest a linha a seguir:

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

Com as declarações corretas feitas no Manifest, podemos prosseguir à classe que utilizará os serviços de câmera do aparelho. Durante esta aula, estudaremos como lançar os serviços de câmera do Android e como utilizar o resultado obtido (seja câmera ou vídeo) em nossa aplicação.

---

## Utilizando o Intent para Captura de Imagens

A maneira mais simples de se capturar imagens e vídeos em aplicações Android é a utilização da aplicação padrão do Android para essas funcionalidades. A aplicação padrão está configurada para receber Intents de tipos conhecidos, os mesmos que veremos agora.

O primeiro passo que devemos atentar é a utilização da propriedade MediaStore.EXTRA_OUTPUT, que pode ou não ser passada à aplicação de câmera do Android. Caso seja configurada, essa propriedade especifica ao Android o nome do arquivo a ser gerado, bem como o local em que ele deve ser salvo. Quando não utilizada, a ausência dessa propriedade indica ao Android para salvar a imagem com o nome padrão no local padrão. Assim, perde-se um pouco do controle sobre o que o Android fará com a imagem obtida. Apesar de opcional, é sempre bom configurar essa propriedade. Vejamos agora na **Listagem 4** o código necessário para tirarmos uma foto dentro de nossa aplicação.

```
private static final int CAPTURAR_IMAGEM = 100;
private Uri fileUri;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

    File mediaStorageDir = new File(
        Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), "NomeDaPasta");

    if (!mediaStorageDir.exists()){
        mediaStorageDir.mkdirs();
    }

    String timeStamp =
        new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());

    File file = new File(mediaStorageDir.getPath() + File.separator +
        "IMG_"+ timeStamp + ".jpg");

    fileUri = Uri.fromFile(file);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);

    startActivityForResult(intent, CAPTURAR_IMAGEM);
}
```

**Listagem 4** - Capturando uma imagem e salvando a um local conhecido

Vamos analisar o código apresentado. Primeiramente, criamos um Intent passando como ação a constante MediaStore.ACTION_IMAGE_CAPTURE. Com esse Intent, podemos requisitar ao Android o serviço de câmera dele para capturar imagens. Uma vez que temos o Intent, o próximo passo é definir onde o arquivo será salvo.

Para definir isso, solicitamos ao sistema uma pasta dentro de seu diretório publico de imagens com o nome que quisermos. Em seguida, se essa pasta não existir, nós a criamos. Com isso, já temos o local onde a imagem será salva. Precisamos agora do nome que o arquivo terá. Para isso, pegamos o caminho indicado pela pasta selecionada e adicionamos a ele o código IMG_, o timeStamp da data em que a foto foi tirada e a extensão .jpg. Na sequência, criamos um URI a partir do arquivo declarado e adicionamos esse URI ao Intent já criado como a propriedade EXTRA_OUTPUT que vimos anteriormente. Com isso, o Intent está

pronto para ser enviado e aí o enviamos por um resultado. Adiante, na seção "Tratando o resultado do Intent", veremos como tratar o resultado gerado pela Activity da câmera.

Após executar o código exibido na **Listagem 4**, a sua aplicação lançará a Activity de câmera do Android e ela será responsável por tratar tudo relacionado a câmera, como *zoom*, ajustes no contraste, *flash*, caso disponível, entre outras coisas. Uma vez que o usuário tenha tirado a foto e escolhido armazená-la, a Activity da câmera retornará para a Activity que a chamou e o resultado deverá ser tratado no método *onActivityResult*. Antes de vermos como lidar com esse velho conhecido, vamos ver as mudanças necessárias para capturar vídeo no lugar de fotos.

---

## Utilizando o Intent para Captura de Vídeo

Para capturar vídeos utilizando a Activity padrão do Android não é muito diferente de capturar fotos. A **Listagem 5** nos mostrará o código necessário para atingir o mesmo resultado e depois analisaremos as diferenças entre os dois métodos de captura.

```
private static final int CAPTURAR_VIDEO = 200;
private Uri fileUri;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);

    File mediaStorageDir = new File(
        Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), "NomeDaPasta");

    if (!mediaStorageDir.exists()){
        mediaStorageDir.mkdirs();
    }

    String timeStamp =
    new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
    File file = new File(mediaStorageDir.getPath() + File.separator +
        "VID_"+ timeStamp + ".mp4");

    fileUri = Uri.fromFile(file);
    intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
    intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY, 0);
    intent.putExtra(MediaStore.EXTRA_DURATION_LIMIT, 30);
    intent.putExtra(MediaStore.EXTRA_SIZE_LIMIT, (1024 * 1024));
    startActivityForResult(intent, CAPTURAR_VIDEO);
}
```

**Listagem 5** - Capturando um vídeo e salvando em local conhecido

Podemos perceber no código que poucas mudanças são necessárias para capturar vídeo, no lugar de fotos. A primeira mudança notável é a mudança na *action* do Intent. Agora passamos a ACTION_VIDEO_CAPTURE. Além disso, mudamos o formato e o prefixo do arquivo a ser salvo, indicando agora que o mesmo é um vídeo e mantendo o timeStamp como nome. As últimas linhas que mostram uma diferença importante e dizem respeito aos extras que são colocados no Intent.

O primeiro extra novo que vemos no Intent é o EXTRA_VIDEO_QUALITY. Esse atributo pode receber o valor 0, indicando que o vídeo deve ser gravado na menor qualidade possível, que também é a que ocupa menos espaço. Já o segundo extra, o EXTRA_DURATION_LIMIT, indica, em segundos, o tempo máximo que o vídeo poderá ter antes de ser finalizado. Por fim, o EXTRA_SIZE_LIMIT indica, em *bytes*, o tamanho máximo que o vídeo poderá atingir antes de ser parado. Com todos esses extras configurados, podemos iniciar a Activity da câmera normalmente, através do Intent criado.

## Tratando o Resultado do Intent

Uma vez lançada a Activity de captura de imagem ou vídeos, o usuário deve utilizá-la normalmente até chegar a um resultado, que será retornado para o método *onActivityResult*. A **Listagem 6** mostra a forma que esse método terá, para que se tenha uma ideia de como o tratamento do resultado deve ser feito.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
 if (requestCode == CAPTURAR_IMAGEM) {
      if (resultCode == RESULT_OK) {
          // Imagem capturada com sucesso e salva no URI especificado na chamada
          Toast.makeText(this, "Imagem salva em:\n" +
              data.getData(), Toast.LENGTH_LONG).show();
      } else if (resultCode == RESULT_CANCELED) {
          // Captura cancelada pelo usuário
      } else {
          // Captura falha
      }
 }

 if (requestCode == CAPTURAR_VIDEO) {
      if (resultCode == RESULT_OK) {
          // Vídeo capturado com sucesso e salvo no URI especificado na chamada
          Toast.makeText(this, "Vídeo salvo em:\n" +
              data.getData(), Toast.LENGTH_LONG).show();
      } else if (resultCode == RESULT_CANCELED) {
          // Captura cancelada pelo usuário
      } else {
          // Captura falha
      }
 }
}
```

**Listagem 6** - Estrutura do onActivityResult para o Intent da câmera

Vemos na **Listagem 6** que, ao retornar, como em qualquer outra Activity, será possível determinar qual foi o código da requisição (configurado no lançamento do Intent, nas **Listagens 5** e **6**) e também qual foi o resultado. A Activity da câmera retorna como resultado RESULT_OK caso tudo tenha corrido bem, RESULT_CANCELED caso o usuário tenha cancelado a requisição e, caso um erro tenha ocorrido, outro valor. Como os comentários mostram, uma utilização da imagem ou vídeo deve ser implementada dentro de cada um desses casos, ou uma alguma mensagem deve ser mostrada. Um exemplo do que podemos fazer, uma vez

que sabemos o local onde o vídeo foi salvo, é reproduzir este utilizando o MediaPlayer. Com isso, finalizamos nossa seção sobre câmeras e também nossa aula.

**Vídeo 03** - Usando Câmera Frontal e Traseira

## Atividade 02

1. Crie um Activity que possui um ImageView e um botão que, ao ser clicado, deve iniciar a captura da imagem e depois exibi-la no ImageView.

### Dica!

Veja o método setImageBitmap(Bitmap bm) do ImageView e o BitmapFactory.decodeFile(String path) para criar um objeto Bitmap a partir de caminho de arquivo.

# Leitura Complementar

Para melhorar o seu conhecimento sobre o uso de recursos de multimídia e do dispositivo de câmera no Android, leia os dois links abaixo do site Android Developers.

- Media Player. Disponível em: <http://developer.android.com/guide/topics/media/mediaplayer.html >. Acesso em: 02 jun. 2015.

- Camera. Disponível em: <http://developer.android.com/guide/topics/media/camera.html>. Acesso em: 02 jun. 2015.

# Resumo

Nesta aula aprendemos a reproduzir diversas mídias dentro de nossa aplicação, bem como utilizar recursos de câmera. Na parte da reprodução de mídia vimos como construir o MediaPlayer, como prepará-lo para as diversas fontes disponíveis e como executá-lo assincronamente para evitar travamentos na *thread* principal. Por fim, estudamos os estados que envolvem o MediaPlayer e aprendemos também a liberá-lo para evitar desperdício de memória. Em seguida, aprendemos como lidar com a câmera através da criação de Intents para serviços que o Android disponibiliza tanto para vídeo como para imagens. Vimos quais são os pontos importantes nesses Intents e como especificar o caminho onde as imagens devem ser salvas. Por fim, vimos o esqueleto de como tratar os resultados gerados pela câmera para que possamos utilizar de maneira adequada as imagens e vídeos em nossa aplicação.

# Autoavaliação

1. Descreva os passos necessários para reproduzir uma música hospedada em um site da Internet que permite *download* progressivo.
2. Por que é importante fazer a preparação assíncrona da mídia a ser reproduzida?
3. Qual é a Action responsável por lançar o Intent de câmera para imagens? É possível não configurar o EXTRA_OUTPUT? O que aconteceria?
4. Quais os três extras que podem ser configurados no Intent de vídeo, além do EXTRA_OUTPUT? Quais suas unidades e para que servem?

# Referências


**Android Developers**. 2012. Disponível em: <http://developer.android.com>. Acesso em 02 jun. 2015.

DIMARZIO, J. *Android: A Programmer's Guide.* McGraw-Hill, 2008. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 23 ago. 2012.

LECHETA, RICARDO R. **GOOGLE ANDROID - APRENDA A CRIAR APLICAÇOES**. 2. ed. São Paulo: NOVATEC, 2010.

LECHETA, RICARDO R. **GOOGLE ANDROID PARA TABLETS**. São Paulo: NOVATEC, 2012.

MEIER, R. *Professional Android 2 Application Development*. John Wiley e Sons, 2010. Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC>. Acesso em: 23 ago. 2012.