02

Câmera e arquivos

Para termos um cadastro de alunos realmente completo, vamos capturar uma foto do aluno. Após isto, vamos gravar esta imagem no sistema de arquivos e o endereço no SQLite.

Um pouco do sistema de arquivos

O android possui as classes principais de manipulação de arquivos tais como `FileInputStream` e `FileOutputStream`. Porém, em vez de instanciá-las, pegamos referência a esses objetos através de métodos da `Activity`. Por exemplo, para ler:

```
FileInputStream stream = openFileInput("arquivo.txt");
```

`FileInputStream`, assim como os outros streams, possui apenas métodos de baixo nível para trabalhar com bytes. Para trabalhar com caracteres, pode-se usar o `FileReader` e o `BufferedReader`.

Para facilitar a manipulação de arquivos, podemos utilizar a classe `Scanner` em conjunto, que já parseia datas, números e pode utilizar expressões regulares. Ele recebe um `FileInputStream` como argumento, ou ainda um `File`, que também está presente no Android e funciona como a representação de um arquivo, mas não serve para manipular seu conteúdo.

Em outras palavras, o `java.io` aqui é muito próximo de tudo que vimos no curso FJ-11.

Caso você precise abrir arquivos que em tempo de compilação estejam em `res/raw/`, você pode utilizar `openRawResource(R.raw.arquivo)`.

Para criar e escrever em arquivos há o método análogo `openFileOutput`, que ainda recebe um segundo argumento com o modo de operação. No caso abaixo, o arquivo só poderá ser utilizado pela nossa aplicação:

```
FileOutputStream stream = openFileOutput("arquivo.txt", MODE_PRIVATE);
```

Existem opções para permitir outras aplicações lerem ou escreverem, além de abrir a saída para append, pois por padrão o arquivo será recriado caso existente.

Você também pode trabalhar com arquivos que não estejam na memória principal do seu dispositivo:

```
File directory = Environment.getExternalStorageDirectory();
File file = new File(directory, "nomeDoAluno.jpg");
```

`Environment.getExternalStorageDirectory()` seleciona uma área de memória externa, o diretório, possivelmente um *sdcard*. Para versões de API level acima de 8, você pode utilizar `getExternalFilesDir()` que a própria `Activity` possui.

Assim como no Java padrão, a classe `File` pode também representar um diretório, o que pode ser estranho ao ler o código.

No emulador, para conferir a criação do arquivo, vá ao menu *DDMS -> File Explorer -> sdcard*. Cuidado que um diretório externo pode não estar preparado! Pode não haver um cartão inserido, ou ele pode estar em modo de apenas leitura para a sua aplicação. Esse estado pode ser verificado através de `Environment.getExternalStorageState()`, comparando o resultado como `Environment.MEDIA_MOUNTED` e `Environment.MEDIA_MOUNTED_READ_ONLY`.

Câmera

A utilização da câmera pode ser feita de duas maneiras. Ou você "pede" para a câmera tirar a foto para você, ou você "pega" a câmera e refaz **todos** os controles, através de uma nova View, parecido com o que fizemos com o `WebView` para usar o browser com seus próprios controles.

O primeiro modo, logicamente é mais simples pois você não tem que saber exatamente como uma câmera funciona e também porque poderá usar todos os controles já existentes na câmera, como efeitos especiais e controles de luminosidade e até o flash.

Para chamar a câmera, vamos utilizar uma `Intent` do **MediaStore**. Acessando a constante `ACTION_MEDIA_CAPTURE` vamos invocar a chamada da câmera do sistema.

```
Intent camera = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivity(camera);
```

Com este trecho de código, já será possível visualizar a chamada da câmera na tela do simulador.

Mas onde podemos guardar o resultado da foto? Também temos controle sobre isso. Podemos passar um parâmetro específico com a URI que diz em que arquivo o resultado deverá ser gravado, antes de fazer o `startActivity` :

```
File file = new File(Environment.getExternalStorageDirectory(),
    "nomeDoAluno.jpg");
Uri outputFileUri = Uri.fromFile(file);
camera.putExtra(MediaStore.EXTRA_OUTPUT, outputFileUri);
```

Se preferir exibir a própria imagem em vez de gravá-la, você pode utilizar o componente `ImageView` , declarando-o em seu layout, e depois setar o `ImageBitmap` como abaixo:

A classe `BitmapFactory` é responsável por guardar um série de métodos estáticos para geração de `Bitmap` s. Veremos o código para isso a seguir.

- decodeByteArray - Com este método estático é possível criar um bitmap a partir de um array de bytes
- decodeFile - Este método cria um bitmap a partir de um arquivo
- decodeResource - É possível criar um Bitmap com um resource do seu projeto android
- decodeStream - Com um InputStream poderemos criar um Bitmap.

Mas atenção, para as versões mais antigas do android, anteriores à versão 2.1, existe um pequeno bug no uso `decodeStream` , e para redes telefônicas lentas (como a nossa), ele se perde.

```
ImageView iv = (ImageView) findViewById(R.id.ImageView) ;
iv.setImageBitmap(BitmapFactory.decodeFile("/sdcard/imagem.jpg")) ;
```

Capturando o resultado de uma activity chamada

Já aprendemos que podemos passar dados para outra activity: o `setData` carrega os dados principais, e podemos utilizar os métodos `putExtra` para passar mais valores, como num mapa.

Algumas vezes precisamos de algo a mais: depois que uma activity que iniciamos terminar, queremos saber qual foi o resultado da operação. Isso é, receber algum valor de volta, para, por exemplo, poder tomar uma decisão no fluxo da aplicação de acordo com o que ocorreu na outra tela.

No caso da câmera, o usuário pode recusar de tirar a foto, ter problemas com a câmera ou pode ser que tudo funcione normalmente e imagem seja capturada.

Para utilizar o recurso do Android que possibilita capturar um resultado de uma activity, precisamos também mandar um código de request, que a outra activity pode também utilizar como informação.

Então vamos chamar a activity de câmera aguardando um resultado. Vamos criar uma constante no sistema chamada `TIRAR_FOTO` , com um valor `101` , por exemplo. E vamos chamar o método `startActivityForResult(intent, TIRA_FOTO)` . Ele vai executar a câmera do Android e devolver a nossa opção num método de callback, o `onActivityResult(int requestCode, int resultCode, Intent data)` . O `requestCode` será `TIRAR_FOTO` no caso desse callback ser invocado por resposta ao nosso pedido de chamada de câmera. Caso estivéssemos esperando resposta de outras activities, diferenciaríamos através desse parâmetro.

O `resultCode` devolverá o valor que a activity da câmera desejar. Já existem dois resultados padrão que devem ser respeitados e são usados pelas activities do android: `RESULT_CANCELED` no caso de problemas ou cancelamento por parte do usuário e `RESULT_OK` se tudo ocorreu normalmente.

Caso a activity chamada fosse criada por nós, poderíamos devolver outros `resultCode` dependendo de cada situação.

Exercício

Nosso desafio será colocar uma foto do aluno no seu formulário. Mais ainda: queremos que, ao clicar na foto, a câmera seja ativada para tirar uma nova foto do aluno. Caso ainda não haja foto, uma imagem padrão deve ser exibida.