02

Cadastro de Alunos

Para estudar os conceitos do curso e aplicá-los na prática, desenvolveremos incrementalmente um projeto para cadastrar alunos de uma instituição de ensino.

Precisamos de uma aplicação capaz de registrar alunos com suas notas com funcionalidades específicas como fazer ligações, navegação web, envio de dados para a web e envio/recebimento de SMS. Além disso, encontraremos os alunos num mapa e numa galeria.

Para começar, vamos:

1. Iniciar um novo projeto para cadastro de alunos.
2. Integrar alguns componentes de tela e navegação.

Vamos começar criando a funcionalidade de listar os alunos em uma tela. Para isso, iremos criar um novo projeto chamado **CadastroCaelum**. Neste novo projeto, tal como fizemos no anterior, vamos desmarcar a opção de criar um *Custom launcher icon*. Também iremos desmarcar a opção de criar uma `Activity`, iremos criá-la do zero.

Criar uma `Activity` do zero seguirá sempre o mesmo padrão. Do contrário, quando solicitamos ao ADT criar uma `Blank Activity` durante a criação de um novo projeto, dependendo da versão dele, esta Activity pode vir com códigos muito complexos para nós no momento.

Agora que criamos o projeto, vamos criar a tela para exibir uma listagem dos alunos. Crie uma nova classe Java com o nome `ListaAlunosActivity.java`. Esta é uma classe comum do Java, vamos fazê-la ser uma tela do Android herdando de `Activity`. Agora basta sobrescrever o método `onCreate` para que ele chame o mesmo método da classe mãe. Sua classe deve estar conforme a seguir:

```
public class ListaAlunosActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
}
```

Agora vamos criar o layout desta nova tela. Para criar um novo layout no Eclipse ADT você pode digitar `Ctrl+N` e buscar por *Android XML Layout File*. Queremos que nosso layout tenha apenas uma lista para exibir os alunos, para isso vamos usar a *tag* `ListView` dentro do `ViewGroup` `LinearLayout`. Faremos nossa lista ocupar todo o espaço da tela, para isso usamos as constantes `match_parent` nos atributos de altura e largura da lista.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ListView
        android:id="@+id/lista_alunos"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </ListView>
</LinearLayout>
```

É necessário também alterar o arquivo AndroidManifest.xml, informando a nova Activity criada e também informando que o seu `xml` será o `LAUNCHER` da aplicação.

```
<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".ListaAlunosActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
```

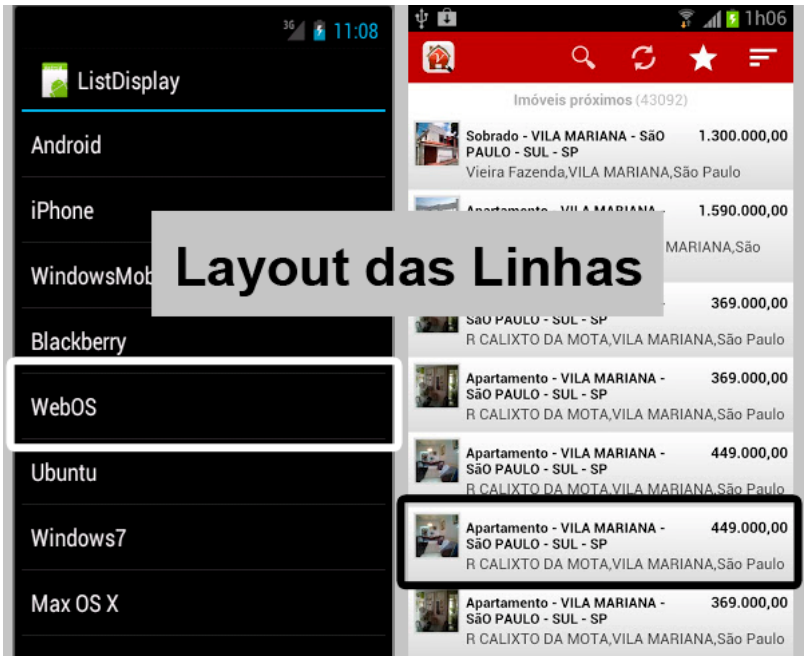
```
            </intent-filter>
        </activity>

    </application>
```

Agora precisamos manipular o `ListView` e colocar uma lista de nomes para serem apresentados na tela. Na `ListaAlunosActivity` podemos buscar o `ListView` criado e criar um `Array` de nomes:

```
public class ListaAlunosActivity extends Activity {
    private ListView listaAlunos;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.listagem_alunos);

        String[] alunos = {"Anderson", "Filipe", "Guilherme"};

        listaAlunos = (ListView) findViewById(R.id.lista_alunos);
    }
}
```

Agora precisamos fazer o `ListView` apresentar os nomes do `Array`. O problema é que o `Array` contém Strings e a tela do Android só aceita objetos do tipo `View`, precisamos fazer o trabalho de converter um objeto comum do `Java` em uma `View`. Soma-se a essa tarefa mais uma dificuldade: Como definir a aparência da linha do `ListView`?

Abaixo podemos observar dois exemplos de uso de `ListView`, um deles mais simples outro mais complexo:



Para conseguirmos atingir nosso objetivo e criar uma tela com um `ListView` populado vamos utilizar um `Adapter`, uma classe do `Android` especializada em **adaptar** objetos **Java** para a tela por meio da criação de um objeto do tipo `View`.

Um dos adapters disponíveis é o `ArrayAdapter` que é capaz de converter listas ou *arrays* em *views*. Em seu construtor o `ArrayAdapter` nos pede 3 argumentos:

- Um `Context`.
- O id do **layout da linha**. Acessado com o auxílio da classe `R`.
- A lista ou array de objetos que queremos exibir na `ListView`.

```
String[] alunos = {"Anderson", "Filipe", "Guilherme"};
int layout = android.R.layout.simple_list_item_1;
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, layout, alunos);
listaAlunos = (ListView) findViewById(R.id.lista_alunos);
```

Para que o `ListView` possua um adapter vamos associá-los usando o método `setAdapter`:

```
//código anterior
listaAlunos.setAdapter(adapter);
```

Exibindo mensagens

Pequenos avisos podem ser mostrados usando a classe `Toast`, que faz "brotar" um aviso na tela por um certo tempo. A duração pode ser `Toast.LENGTH_LONG (1)` ou `Toast.LENGTH_SHORT (0)`.

Quando um `Toast` é criado, uma mensagem fica pronta para ser mostrada na tela, na forma de um balãozinho de texto informativo. Ele só é mostrado, porém, quando chamamos o método `show()`.

```
Toast.makeText(context, "Meu texto de aviso", Toast.LENGTH_SHORT).show();
```

Diferentemente de um alerta comum, você pode continuar a utilizar sua aplicação normalmente enquanto o Toast é exibido. Ele não trava a tela.

Adicionando comportamento

Vamos agora implementar no nosso aplicativo a funcionalidade de, quando o usuário clicar em um aluno da lista, mostrarmos a posição na lista daquele aluno.

Para conseguirmos responder a qualquer evento que ocorra na nossa aplicação, usamos o conceito de *Listener*. Para fazermos nossa `ListView` responder a eventos de **cliques nos itens** precisamos adicionar um *listener* de **click no item da lista**, fazemos isso com o método `setOnItemClickListener` da `ListView` passando uma instância de `OnItemClickListener` que nos obriga a implementar o método `onItemClick`, conforme o código abaixo:

```
public class MinhaActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        //chamada a super, setContentView e outros..

        ListView minhaLista = (ListView) findViewById(R.id.identificador_da_lista);

        listaAlunos.setOnItemClickListener(new OnItemClickListener() {
            public void onItemClick(AdapterView<?> adapter, View view, int posicao, long id) {

                Toast.makeText(MinhaActivity.this, "Meu texto de aviso", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

Repare que na chamada ao `Toast.makeText(...)` passamos o primeiro parâmetro `Context` como `MinhaActivity.this`. Fazemos isso pois `this` refere-se a instância da **classe anônima** que é uma classe sem nome que implementa a interface `OnItemClickListener` e ela **não** herda de `Activity` que é o único contexto que conhecemos até o momento, portanto essa instância da classe anônima não pode ser usada como um `Context`.

Para implementar o clique longo basta usarmos o *listener* `OnItemLongClickListener` que nos obriga a implementar o método `onItemLongClick`.

Vamos fazer no clique longo exibir a mensagem **Aluno clicado é Fulano**.

Para pegar o nome do aluno podemos usar o `adapter` que conhece os dados que estão na lista. Nós o recebemos como parâmetro no método `onItemLongClick`. O `adapter` possui o método `getItemAtPosition` que recebe um inteiro que representa a posicão que queremos na lista; essa posição nós também recebemos como parâmetro no método `onItemLongClick`.

A seguir podemos ver a implementação do clique longo.

```
public class MinhaActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        //chamada a super, setContentView e outros..

        ListView minhaLista = (ListView) findViewById(R.id.identificador_da_lista);
        //lister de clique simples

        listaAlunos.setOnItemLongClickListener(new OnItemLongClickListener() {
```

```
            public boolean onItemLongClick(AdapterView<?> adapter, View view, int posicao, long id) {

                Toast.makeText(MinhaActivity.this, "Aluno clicado é: " + , Toast.LENGTH_SHORT).show();

                return false;
            }
        });
    }
}
```

Repare que diferentemente do *listener* de clique simples que possui retorno `void`, o clique longo retorna um `boolean`. Se executarmos a aplicação agora com os dois *listeners* e efetuarmos o clique longo na lista, a mensagem **Aluno clicado é Fulano** será exibida e, logo após, a mensagem **Posição é x** também será exibida; isso ocorre pois estamos retornando `false` no método `onItemLongClick`.

Este `boolean` responde a pergunta se o *listener* de clique longo irá **consumir o evento de clique sozinho** ou se irá compartilhar com os demais eventos que reagem a cliques. O clique longo ainda assim é um clique, ou seja, ele pode passar pelo clique longo e pelo clique simples além de outros métodos que o Android tem internamente.

Este comportamento é estranho, pois no clique longo queremos apenas uma mensagem: **Aluno clicado é Fulano**. Para obtermos este comportamento basta retornar `true` no método `onItemLongClick`, que indicará que ele irá consumir o evento de clique sozinho.