02

Completando o cadastro

Já possuímos a capacidade de listar e inserir alunos em nossa aplicação. Vamos agora prepará-la para apagar e alterar os dados de aluno!

Deletando um aluno

Agora desejamos deletar um aluno de nossa listagem, podemos criar um método em nosso `DAO` que deleta um aluno passado como argumento:

```
public void deletar(Aluno aluno) {
    String[] args = {aluno.getId().toString()};
    getWritableDatabase().delete(TABELA, "id=?", args);
}
```

Quando vamos invocar esse método?

Vamos pensar na usabilidade de nossa aplicação! Quando clicamos no item da lista de alunos somos redirecionados para o formulário de cadastro para fazer a alteração do aluno clicado.

Normalmente em sistemas operacionais o clique simples em um item na tela significa a seleção do item. Quando desejamos outras opções de ações em um item normalmente o que fazemos é clicar nele com o botão direito do mouse e esperamos que um menu de opções apareça para que possamos escolher qual ação desejamos.

No Android não temos botão direito mas esse papel nos aparelhos é realizado pelo clique longo. Podemos fazer com que o clique longo em um item cause a abertura de um menu de opções semelhante ao dos demais sistemas operacionais. O detalhe é que as opções disponíveis podem variar em função do item clicado, devido a esse fato, esse tipo de menu no Android é chamado do **Menu de Contexto** ou ***Context Menu***.

Para avisarmos ao `Android` que uma view possui um menu de contexto associado precisamos registrá-la com o método `registerForContextMenu(sua_view_aqui)` da classe `Activity` .

Em nossa aplicação queremos o clique longo na `ListView` que contém a listagem. Vamos então registrar nossa lista:

```
public class ListaAlunos extends Activity {

    private ListView lista;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.listagem_alunos);

        lista = (ListView) findViewById(R.id.lista);
        //...

        registerForContextMenu(lista);
    }

    //...
}
```

Mas quais serão os itens que aparecerão no menu de contexto quando clicarmos na lista?

Quando fazemos um clique longo em uma `View` registrada para `Menu de Contexto` , o `Android` invoca o método `onCreateContextMenu` da `Activity` que estiver rodando. Para customizar o conteúdo do menu de contexto utilizaremos esse método de maneira muito parecida com a criação dos itens do `Options Menu` :

```
public void onCreateContextMenu(ContextMenu menu, View view,
                    ContextMenuInfo menuInfo) {

    menu.add("Ligar");
    menu.add("Enviar SMS");
    menu.add("Achar no Mapa");
```

```
    menu.add("Navegar no site");
    menu.add("Deletar");
    menu.add("Enviar E-mail");
}
```

Em uma mesma tela várias `Views` podem ser registradas para `Menus de Contexto`. Nesse caso talvez precisássemos criar menus diferenciados para cada uma delas. Pensando exatamente nesse problema é que o método `onCreateContextMenu` recebe uma `View` passada como argumento. Podemos verificar qual `View` foi criada e programaticamente criar menus diferentes em função da `View` clicada.

O comportamento de clique no item do menu de contexto pode ser criado no ato de criação desse item. Quando adicionamos programaticamente um item no menu de contexto é criado um `MenuItem` ao qual podemos associar um `Listener`:

```
//...
menu.add("Navegar no site");

MenuItem deletar = menu.add("Deletar");
deletar.setOnMenuItemClickListener(new OnMenuItemClickListener() {
    @Override
    public boolean onMenuItemClick(MenuItem item) {
        //... comportamento do clique em deletar
        return false;
    }
});
```

É possível usar um XML para criar os itens do menu. Poderíamos fazer a mesma abordagem que vimos com o `MenuInflater`, declarando um XML com seus itens, nomeando cada um com um id em particular:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/ligar"
        android:title="Ligar"/>
    <item
        android:id="@+id/sms"
        android:title="Enviar SMS"/>
    <item
        android:id="@+id/mapa"
        android:title="Achar no Mapa"/>
    <item
        android:id="@+id/site"
        android:title="Navegar no Site"/>
    <item
        android:id="@+id/deletar"
        android:title="Deletar"/>
    <item
        android:id="@+id/email"
        android:title="Enviar E-mail"/>
</menu>
```

No `onCreateContextMenu` basta fazer também `getMenuInflater().inflate(R.menu.menu_contexto, menu)`.

Para lidarmos com o clique no item do menu de contexto podemos sobrescrever o método `onContextItemSelected` e, de acordo com o `id` que demos para o item do menu podemos ter um comportamento diferenciado.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.deletar:
        //Qual aluno vamos deletar???
        break;
    }

    return super.onContextItemSelected(item);
}
```

O problema, no caso da remoção do aluno, é que não sabemos qual item da lista foi clicado quando o menu de contexto aparece.

Menu de Contexto (Context Menu)

Já possuímos o método `deleta` em nosso `AlunoDAO` e já criamos um menu de contexto para o clique no item da `ListView` que contém os alunos cadastrados.

Agora basta descobrirmos o aluno que foi selecionado para remoção.

Sabemos que o usuário fará um clique longo no item da `ListView` para disparar a criação do `Menu de Contexto`. Podemos alterar a implementação do `OnItemLongClickListener` para guardarmos o aluno selecionado em um atributo de nossa `ListaAlunosActivity`. No método `onCreate` da `ListaAlunosActivity`:

```
listaAlunos.setOnItemLongClickListener(new OnItemLongClickListener() {
    @Override
    public boolean onItemLongClick(AdapterView<?> adapter, View view,
        int posicao, long id) {

        //Guarda aluno escolhido em atributo
        alunoSelecionado = (Aluno) adapter.getItemAtPosition(posicao);
        AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this);
        dao.deletar(alunoSelecionado);
        return false;
    }
});
```

Não podemos esquecer de recarregar a lista para que o elemento deletado não apareça na tela. Para isso, podemos **extrair este código para um método** `carregaLista` e chamá-lo ao final da remoção do aluno.

```
public class ListaAlunosActivity extends Activity {
    public void onCreate(Bundle b) {
        //...

        listaAlunos.setOnItemLongClickListener(new OnItemLongClickListener() {
            @Override
            public boolean onItemLongClick(AdapterView<?> adapter, View view,
                int posicao, long id) {

                //Guarda aluno escolhido em atributo
                alunoSelecionado = (Aluno) adapter.getItemAtPosition(posicao);
                AlunoDAO dao = new AlunoDAO(ListaAlunosActivity.this);
                dao.deletar(alunoSelecionado);
                dao.close();
                carregaLista();
                return false;
            }
        });
    }

    //...

    private void carregaLista() {
        AlunoDAO dao = new AlunoDAO(this);
        List<Aluno> alunos = dao.getLista();

        ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>();
        lista.setAdapter(adapter);
    }
}
```

Alterando um aluno

Agora que já podemos excluir um aluno, vamos criar a possibilidade de alterar os dados de um aluno existente. Pensando na usabilidade, o ideal é que o clique em um aluno da lista, abra o formulário com as informações do aluno preenchidas, prontas para serem alteradas, conforme imagem a seguir.

Ok, abrir uma nova tela no Android nós já sabemos, basta criamos uma `Intent`, porém o desafio nesta funcionalidade é enviar um aluno de uma `Activity` para outra.

Para isso, basta colocarmos o aluno, ou qualquer outra informação "pendurada" na `Intent`. Para "pendurar" coisas na `Intent` vamos usar o método `putExtra` que recebe duas informações: uma `String` sendo a chave/apelido do que se quer enviar e o valor/objeto que se quer enviar.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    //...
    lista.setOnItemClickListener(new OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> adapter, View view, int posicao,
                long id) {

            Aluno alunoParaSerAlterado = (Aluno) adapter.getItemAtPosition(posicao);

            Intent irParaOFormulario = new Intent(ListaAlunosActivity.this, FormularioActivity.class);
            irParaOFormulario.putExtra("alunoSelecionado", alunoParaSerAlterado);

            startActivity(irParaOFormulario);
        }
    });

    /...
}
```

No entanto, para passarmos informações de uma `Activity` para outra no Android, essas informações precisam ser serializadas, ou seja, nosso aluno deve poder ser serializado pelo Android. Para isso, basta fazermos a classe `Aluno` implementar a interface `Serializable`.

```
public class Aluno implements Serializable {

    //...
}
```

Agora que conseguimos enviar o aluno para o formulário, só falta, no formulário, recuperar este aluno e preencher os campos na tela com as informações dele. Para este último passo, vamos usar o nosso especialista em tirar e colocar as informações na tela, `FormularioHelper`. No entanto, só vamos preencher os campos do formulário se **veio um aluno**.

Para recuperarmos o aluno no formulário vamos usar o método `getIntent` e recuperar a `Intent` que iniciou o formulário, pois é lá que está nosso aluno, e depois usar o método `getSerializableExtra` da `Intent` que recebe como parâmetro justamente a chave/apelido que demos anteriormente, ao realizar o envio.

```
public class FormularioActivity extends Activity {

    //...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...
        helper = new FormularioHelper(this);

        final Aluno alunoParaSerAlterado = (Aluno) getIntent().getSerializableExtra("alunoSelecionado");

        if (alunoParaSerAlterado != null) {
            helper.colocaAlunoNoFormulario(alunoParaSerAlterado);
                        // Colocando aqui a alteração no nome do botão ao carregar a tela o nome do botão já está alterado.
            botao.setText("Alterar");
        }
    }
}
```

Com o formulário preenchido, o usuário pode alterar as informações e depois clicar em **Salvar**. Falta apenas implementarmos o clique no botão. Ao clicar em **Salvar**, devemos recuperar as informações do formulário pois algo foi alterado, colocar estas informações em um aluno, e atualizar o aluno com base no id dele.

Após tudo isso, devemos **voltar** para a tela anterior (no caso, a `ListaAlunosActivity`) com a lista atualizada. Para **voltar** de forma programática, usamos o método `finish` .

```
public class FormularioActivity extends Activity {
    //...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        //...

        final Button botao = (Button) findViewById(R.id.botao);
        botao.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                Aluno aluno = helper.pegaAlunoDoFormulario();
                AlunoDAO dao = new AlunoDAO(FormularioActivity.this);

                if (alunoParaSerAlterado != null) {
                    aluno.setId(alunoParaSerAlterado.getId());
                                        //Esse código precisei colocar dentro do onCreate, senão o botão só mudou o nome para alterar no mome
                    //botao.setText("Alterar");
                    dao.atualizar(aluno);
                } else {
                    dao.salvar(aluno);
                }
                dao.close();

                finish();
            }
        });
    }
}
```