04

Buscando aluno no banco

Transcrição

Inserimos um método que adiciona novos alunos ao banco de dados, mas os nomes presentes na lista advêm de `ListaAlunosActivity.java`. Lembra-se que escrevemos os nomes "a mão" na `String`?

Em vez de trazer os alunos do código, vamos buscá-los a partir do banco de dados!

Para auxiliar nessa tarefa adicionamos, abaixo da linha do `setContentView`, o `dao` que abstrairá o que desejamos. Instanciamos o `dao` digitando `AlunoDAO dao = new AlunoDAO`, importamos a classe com um "Alt+Enter" e como ela pede um contexto, acrescentamos um `this`. Ficaremos com, `AlunoDAO dao = new AlunoDAO(this)`. Vamos acrescentar na linha de baixo, também, um `dao.buscaAlunos` que trará todos os alunos do banco de dados.

Como o "lista alunos" deve devolver uma coleção de alunos, completaremos o `dao.buscaAlunos` com `List<Aluno> alunos` e ficaremos com `List<Aluno> alunos = dao.buscaAlunos`. A classe `Aluno` e `List` aparecerá em vermelho, dessa maneira, é preciso importar com um "Alt+Enter". Ainda, é preciso fechar o `dao`, para isso, digitaremos `dao.close`. Como a *Array* não é mais necessária podemos apagar sua linha. Falta alterar a `Adapter`, substituiremos a `String` por `Aluno` e faremos essa alteração em ambos os lados. Ficaremos com o seguinte:

```
public class ListaAlunosActivity extends AppCompatActivity  {

    @Override
    protected void onCreate(Bundle  savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_lista_alunos);
    AlunoDAO dao = new AlunoDAO(this);
    List<Aluno> alunos = dao.buscaAlunos();
    dao.close();

    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);
    ArrayAdapter<Aluno> adapter = new ArrayAdapter<ALuno>(this,  android.R.layout.simple_list_it
    listaAlunos.setAdapter(adapter);

    Button novoAluno = (Button) findViewById(R.id.novo_aluno);
    novoAluno.setOnClickListener((v) -> {
        Intent intentVaiProFormuIario = new  Intent(ListaAlunosActivity.this,  FormularioActivit
        startActivity(intentVaiProFormulario);
    }):
    }
}
```

Falta implementar um método de busca de alunos, para tanto, na linha da `List`, acima do `buscaAlunos`, damos um "Alt+Enter". Será criado um método abaixo do `SQLiteDatabase`. Como no `buscaAlunos` queremos realizar uma busca, adicionaremos na linha abaixo do `public class List<Aluno> buscaAluno` uma *String*: `String sql = "SELECT * FROM Aluno;"`. Nessa `String` indicamos as colunas que devemos receber e o *Select* e o *From* indicam qual é a tabela. Na linha de baixo inserimos um banco de dados e como é preciso realizar a leitura adicionamos o `getReadableDatabase` e `SQLiteDatabase db = getReadableDatabase`. Poderíamos inserir um `execSQL`, mas ele não devolveria nada para manipular.

Resolvemos esse problema inserindo a `rawQuery`, uma instrução `sql`, parâmetros e passando o `null`. Ficaremos com `db.rawQuery(sql, null)`. O `rawQuery` devolve um resultado que a primeira vista pode parecer meio estranho. Esse resultado, entretanto, é um `Cursor` que importaremos ("Alt+Enter"). Ficaremos com:

```
public void insere(Aluno aluno) { SQLiteDatabase db = getWritableDatabase();

ContentValues dados = new ContentValues();
dados.put("nome", aluno.getNome());
dados.put("endereco", aluno.getEndereco());
dados.put("telefone", aluno.getTelefone());
dados.put("site", aluno.getSite());
dados.put("nota", aluno.getNota()):

db.insert("Alunos", null, dados);
}

public List<Aluno> buscaAlunos()  {
String sql = "SELECT * FROM Alunos;";
SQLiteDataBase db = getReadableDatabase();
Cursor c = db.rawQuery(sql, null);

    return null;
}
```

O `Cursor` é uma espécie de ponteiro que mostra os resultados da busca. A princípio o ponteiro aponta para uma linha vazia, dessa maneira é preciso que ele se mova para a próxima linha. O `c.moveToNext` faz isso e acrescentamos ele na linha de baixo do `Cursor`. O `moveToNext`, além de mover o `Cursor` para a próxima linha também nos avisa se as linhas acabam, isto é, ele nos devolve um *boolean*. E, se desejamos percorrer todos os resultados da busca, podemos inserir o `moveToNext` em uma `while`.

Falta extrairmos os resultados dessa linha, portanto, ao final do `moveToNext` damos um "Enter" e usamos o método `c.getString`. Esse método pede o índice da coluna que queremos recuperar, o que podemos simplesmente perguntar ao `Cursor` e fazemos isso digitando `String nome = c.getString(c.getColumnIndex("nome"))`. Poderíamos repetir isso várias vezes adaptando apenas o campo. Mas, não estamos querendo gerar uma lista de alunos?

Vamos instanciar uma Lista que chamaremos "alunos", ela deve ficar acima do `moveToNext`. Escolheremos uma implementação, no caso, a `List<Aluno> alunos = new ArrayList<Alunos>`. Como cada linha do resultado representa um novo aluno, acrescentamos na linha de baixo do `moveToText` um `Aluno aluno = new aluno` e ao em vez de colocar cada um em uma `String` diferente, adicionamos apenas uma `setNome` para "aluno". Teremos, `aluno.setNome(c.getString(c.getColumnIndex("nome")))` e ao todo o seguinte:

```
public List<Aluno> buscaAlunos()  {
  String sql = "SELECT * FROM Alunos;";
  SQLiteDataBase db = getReadableDatabase();
  Cursor c = db.rawQuery(sql, null);

      List<Aluno> alunos = new ArrayList<Aluno>();
      while (c.moveToNext()) {
          Aluno aluno = new Aluno();
          aluno.setNome(c.getString(c.getColumnIndex("nome")));
      }
```

```
        return null;
    }
```

Introduzindo o `setNome` falamos que o nome do aluno deverá coincidir com a coluna "nome". Devemos fazer o mesmo com o restante dos campos que queremos resgatar e, para isso, basta selecionarmos a linha que acabamos de escrever e copiá-la e colá-la cinco vezes. Alterando apenas o campo "nome" e o `Id` por endereço, telefone, site e nota.

Atenção!!!: O campo `Id` não é uma String e sim um `Long`, então, digitaremos, `aluno.setId(c.getLong(c.getColumnIndex("id")))`. E atente-se também ao campo nota que é uma `Double`.

Falta adicionar um `add` para que de fato o aluno seja acrescentado em nossa lista. Portanto, ao final de tudo digitamos `alunos.add(aluno)`. E como queremos retornar um aluno, digitaremos `return aluno`. Para finalizar, o *Android* precisa saber quando terminamos a busca para liberar esse recurso do `Cursor`. Para dar o aviso evocamos o método `close`. Acrescentaremos, `c.close()`. Ficaremos com:

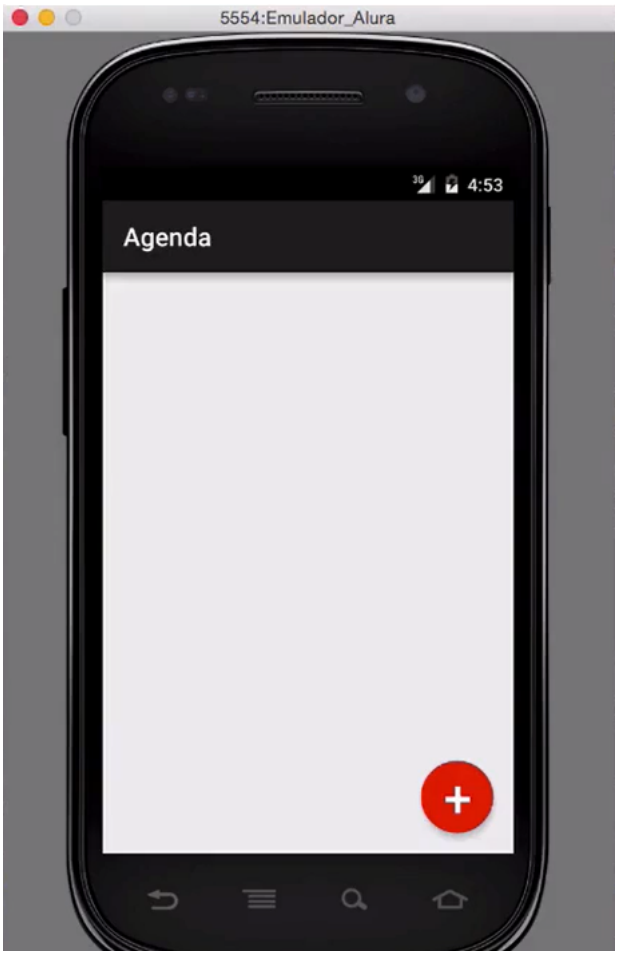
```
public List<Aluno> buscaAlunos()  {
  String sql = "SELECT * FROM Alunos;";
  SQLiteDataBase db = getReadableDatabase();
  Cursor c = db.rawQuery(sql, null);

      List<Aluno> alunos = new ArrayList<Aluno>();
      while (c.moveToNext()) {
          Aluno aluno = new Aluno();
          aluno.setId(c.getLong(c.getColumnIndex("id")));
          aluno.setNome(c.getString(c.getColumnIndex("nome")));
          aluno.setEndereco(c.getString(c.getColumnIndex("endereco")));
          aluno.setTelefone(c.getString(c.getColumnIndex("telefone")));
          aluno.setSite(c.getString(c.getColumnIndex("site")));
          aluno.setNota(c.getDouble(c.getColumnIndex("nota")));

          alunos.add(aluno);

      }
      c.close();

      return alunos;

  }
```

Voltando na `Activity`, repare como ela está. Antes a `ArrayAdapter` tinha uma `String` e agora tem Alunos:

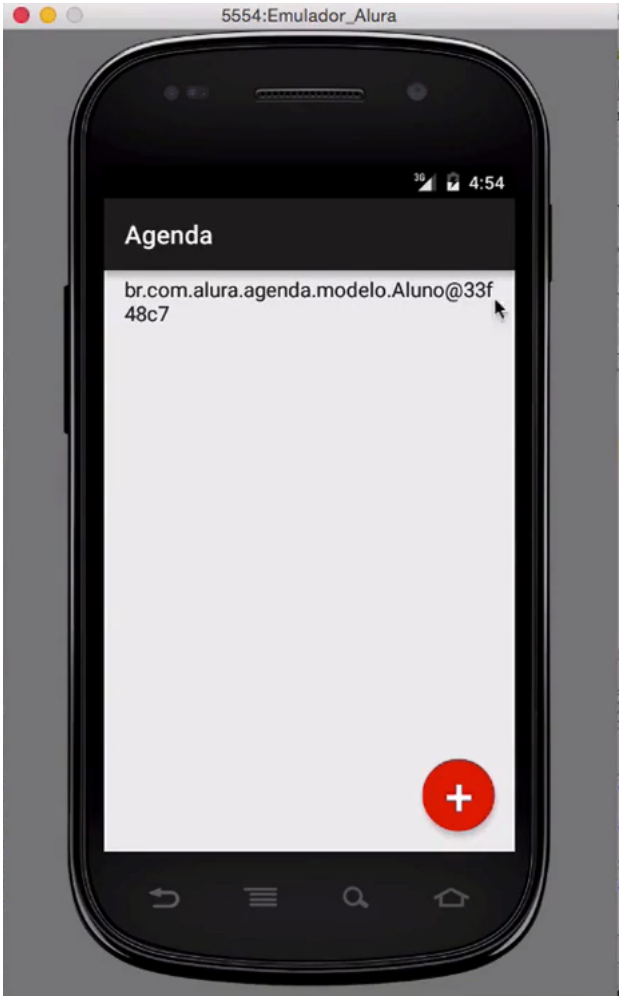
```
public class ListaAlunosActivity extends AppCompatActivity {

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

    AlunoDAO dao = new AlunoDAO(this);
    List<Aluno> alunos = dao.buscaAlunos(); dao.close();

    ListView listaAlunos = (ListView) findViewById(R.id.lista_alunos);
    ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>(this, android.R.layout.simple_list_it
```

```
        listaAlunos.setAdapter(adapter);

        Button novoAluno = (Button) findViewById(R.id.novo_aluno);
        novoAluno.setOnClickListener((v) -> {
            Intent intentVaiProFormulario = new Intent(ListaAlunosActivity.this,  FormularioActivity
        }
    }
```

Vamos salvar a aplicação e rodá-la:



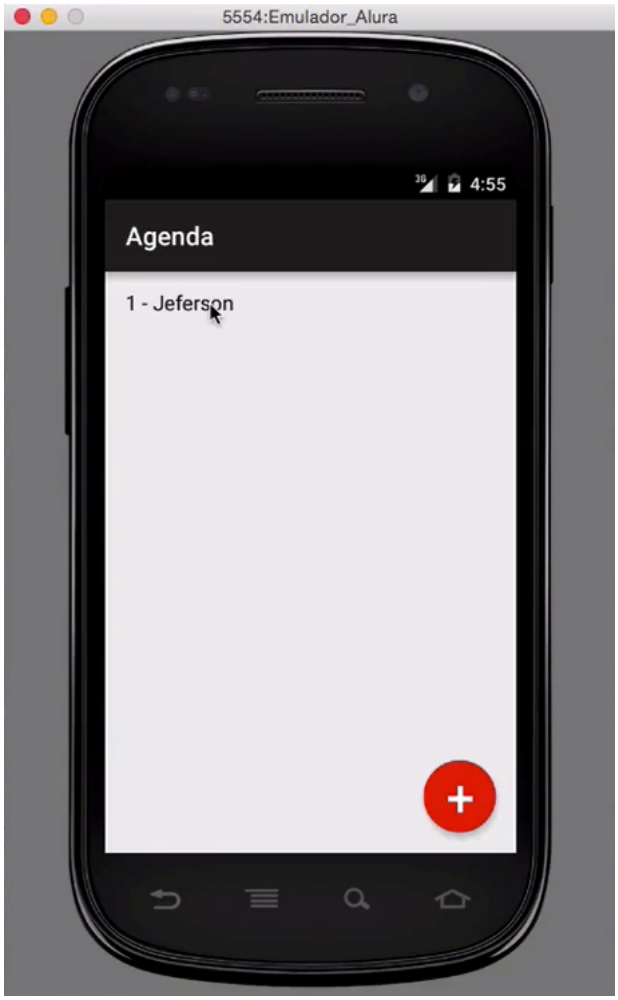
O banco de dados está vazio, mas acrescentando um novo aluno ele não será adicionado à lista. O que ele mostra é uma espécie de código na tela, uma referência. O Java não sabe como mostrar o *object*, por isso ele traz a referência:



Como não existe um `toString` no `Aluno.java` ele usa a implementação padrão do *object* e temos o que aparece na tela do celular. Para que algo seja mostrado é preciso sobrescrever o `toString` no `Aluno.java` . Ao final da tela, depois de `setNome` , digitamos `toString` e damos um "Enter" e isso será sobrescrito. Apagamos o `super.toString` e falamos o que deve ser devolvido, no caso, o nome do aluno através do `getNome` . Teremos o seguinte, `return getId() + " - " + getNome()` . Ficaremos com:

```
@Override
public String toString()  {
    return getId() + " - " + getNome();
}
```

Vamos rodar de novo e ver o que acontece?



Agora, toda vez que sairmos da aplicação nosso aluno continuará armazenado! Entretanto... nossa lista não é atualizada automaticamente! Assim, inserindo um segundo aluno, ele não aparece na lista , apenas depois de sairmos e voltarmos a aplicação.

Resolveremos esse problema na sequência!