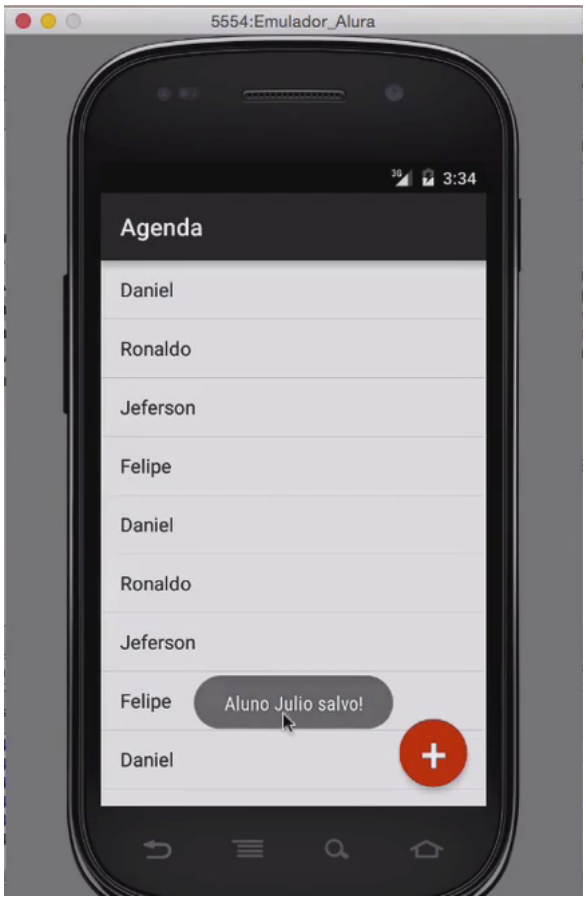
02

Adicionando alunos a nossa lista

Transcrição

Conseguimos armazenar as informações preenchidas no formulário e jogá-las no objeto "Aluno". Assim, quando acrescentamos um novo nome no formulário e damos um "check" o aplicativo reconhece a ação e dá uma resposta.



Agora, queremos que todos os novos formulários criados apareçam em nossa lista. Isto é, que o nome de "Paulo" esteja junto aos demais!

Vamos começar a abrir o caminho para um banco de dados, que possibilitará armazenar as informações. Para fazer isso poderíamos modificar nossa *Array*, na `ListaAlunosActivity.java` . Mas, se fizermos isso "a mão", sempre que desligarmos o celular qualquer novo aluno cadastrado será perdido.

Se tivermos as informações dos alunos em um arquivo ou banco de dados e a *Array* popular isso, não perderemos os alunos toda vez que a aplicação reiniciar. Portanto, para não perder mais as informações, vamos começar a colocar persistência no aplicativo!

Podemos inserir uma conexão para se ligar ao banco de dados tanto na `ListaAlunosActivity.java` quanto no `FormularioActivity.java` , mas as conexões nem sempre são códigos simples. A medida que formos acrescentando novas informações o código aumentaria e teríamos que espalhá-lo em diversos locais, o que complicaria um pouco o desenvolvimento.

Vamos fazer o que aprendemos anteriormente, que é isolar toda a parte da conexão em uma classe específica. Assim, sempre que quisermos manipular algo que diga respeito a um aluno, usaremos uma classe que nos ajude a fazer as

modificações necessárias. Ela também irá abstrair toda a parte de acesso ao banco de dados, operações, conexão, etc. O nome do objeto que abstrai tudo o que acabamos de comentar é "DAO".

Criaremos uma classe nova no *Android*, para isso, vamos na pasta "br.com.alura", clicamos com o botão direito e "New > Java Class". Abrirá uma janela pedindo um *Name*, vamos nomear de "AlunoDAO", damos um "Enter" e... Pronto! Criou o "AlunoDAO.java".

Queremos colocar ele em outro pacote, para isso, vamos de novo em "br.com.alura", clicamos com o botão direito e selecionamos "New > Package". Abrirá uma janela pedindo para que digitemos o *Package name*. O nome do pacote que vamos criar será "dao". Ele criará no canto esquerdo o "dao", dentro de "br.com.alura". Para trocar o "AlunoDAO" de lugar basta selecioná-lo e arrastá-lo para o novo *package*. Abrirá uma janela nova e nela é só pedir para refaturar o código selecionando o "Refactor".

Criamos a nova classe, "AlunoDAO.java" e dentro dela vamos fazer a parte de conexão de bancos de dados. Imagine que, a medida que o tempo for passando, nosso objetivo seja introduzir uma nova coluna na nossa lista, por exemplo, a coluna foto dos alunos. Nesse caso, teríamos que fazer diversas verificações para que tudo ocorresse bem! Teríamos que pensar no caso de como o *Android* avisaria as pessoas que já tinham a aplicação baixada para atualizá-la, etc.

As coisas ficariam complicadas! Assim, vamos facilitar um pouco utilizando uma classe do *Android* que faz tudo isso por nós.

Vamos na aba 'AlunoDAO.java'. O banco de dados que vamos utilizar no *Android* é o `SQLite`, uma versão simplificada do `sql`. Então, vamos transformar nossa classe em um `SQLiteOpenHelper`, para fazer isso digitaremos `SQLiteOpenHelper` na mesma linha do `public class AlunoDAO`. Vamos importar o `SQLiteOpenHelper` usando em cima dela o atalho "Alt+Enter".

Essa classe vai nos ajudar em todas as operações relativas ao banco de dados.

Mas, perceba que o que acabamos de digitar estará em vermelho, pois, existe um pequeno erro. O *Android* ainda requer um construtor uma vez que o `SQLiteOpenHelper` pede, obrigatoriamente, alguns parâmetros que são fornecidos, justamente, pelo construtor. Clicando em cima do `SQLiteOpenHelper` damos um "Alt+Enter" e pedimos para que ele "Implement methods". Na hora que dermos um "Enter" ele vai pedir para que implementemos dois métodos, que são obrigatórios, o `onCreate` e o `onUpgrade`. Basta dar um "Ok".

![Mostrando a janela do immplement methods]https://s3.amazonaws.com/caelum-online-public/Android+Studio/andl4_2+mostrando+metodos+para+importar.png"> (https://s3.amazonaws.com/caelum-online-public/Android+Studio/andl4_2+mostrando+metodos+para+importar.png">)

Nossa tela ficará da seguinte maneira:

```
public class AlunoDAO extends SQLiteOpenHelper{
    @Override
    public void onCreate (SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Só que, ao fazer isso, o `SQLiteOpenHelper` continuará em vermelho, agora damos mais um "Alt+ Enter" e pedimos para ele gerar o seguinte construtor: `Create constructor matching super`. Uma nova janela será aberta e ele vai sugerir dois construtores, nós vamos utilizar apenas o primeiro, que selecionamos, e daremos "ok"

![mostranda a janela que é aberta e que sugere os construtores]https://s3.amazonaws.com/caelum-online-public/Android+III/Android+I+Super+class+Constructor.png"> (https://s3.amazonaws.com/caelum-online-public/Android+III/Android+I+Super+class+Constructor.png">)

O que ele irá gerar é um construtor automático:

```
Context context, String name, SQLiteDatabase.CursorFactory factory, int version
```

O código ficará assim:

```
public class AlunoDAO extends SQLiteOpenHelper{
    public AlunoDAO (Context context, String name, SQLiteDatabase.CursorFactory factory, int ver
        super(context, name, factory, version);
    }

    @Override
    public void onCreate (SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Veremos o que está sendo pedido no 'super'. Primeiro, o construtor pediu um contexto, que é a identificação para o *java*. Ele pede em seguida um *name*, mas com isso não precisamos nos preocupar, pois temos a opção de usar qualquer nome. Vamos tirar o primeiro parâmetro que colocamos no `AlunoDAO`, apagando o `String name`.

Ele irá pedir um `factory`, se quisermos customizar alguns parâmetros, mas como nem sempre precisamos fazer essas modificações, vamos apagar e substituir o `factory` por um `null` e apagaremos, por consequência o `SQLiteDatabase.CursorFactory factory`.

Por fim, ele pergunta qual é a versão do banco de dados. Como é a primeira, escreveremos apenas um `1` e apagamos o `int version`. No `super` ficará entre os parênteses o `context, "Agenda", null, 1` e no `public` apenas `Context context`. Agora, temos tudo para que o construtor funcione. Ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper {
    public class AlunoDAO (Context context) {
        super(context, "Agenda", null, 1);
    }

    @Override
    public void onCreate (SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
    }
}
```

Perceba que ainda existem dois métodos. O primeiro é o `onCreate`, que é usado assim que o banco de dados é criado.

Para introduzir algo dentro do banco de dados, a primeira coisa a fazer é colocar uma tabela. Para criar uma tabela em `sql` vamos introduzir uma instrução, a `String sql`, e um comando para criar a tabela, o `CREATE TABLE`, e um nome para a tabela, `Alunos`. Descreveremos entre os parênteses os nomes das colunas da tabela.

Colocaremos na primeira coluna um número de identificação, o `id INTEGER PRIMARY KEY`. Em seguida, inserimos em todos os campos do formulário o nome, o endereço, o telefone, o site e a nota. O dado de nome será acompanhado de `TEXT`, pois é um texto, portanto, não pode ser um campo sem nada. Aos demais campos acrescentaremos que são do tipo `TEXT` e a nota é do tipo `REAL`. Vamos fechar o parênteses e colocar um `;` para dizer que nossa instrução acabou e mais um `;` devido a linha `java`. Ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper {
    public class AlunoDAO (Context context) {
        super(context, "Agenda", null, 1);
    }

    @Override
    public void onCreate (SQLiteDatabase db) {
        String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT NOT NULL, endereço
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Depois da linha da `String`, damos um "Enter" e na próxima, para executar as instruções, acrescentamos o `db.execSQL(sql);` - usando a variável `db` que o *Android* passa como parâmetro. O método fica da seguinte maneira:

```
    @Override
        public void onCreate (SQLiteDatabase db) {
            String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT NOT NULL, ende
            db.execSQL(sql);
    }
```

Vamos passar para o próximo método, o `onUpgrade`. Lembre-se que podemos ter duas situações, ou um novo banco é criado ou ele precisa ser atualizado. Toda vez que mudarmos algo no banco de dados vamos ter que alterar a versão para 2, 3, 4, etc.

Uma vez que o banco de dados é alterado e modificamos o número da versão é o `onUpgrade` que avisa que ela foi modificada e precisa ser atualizada. Assim, adicionamos algo que faça passar da versão 1 para a versão 2 e que avisa sobre a atualização. Como não temos uma atualização do banco de dados não precisaremos introduzir nada mas, vamos inserir um caminho para facilitar isso no futuro.

Vamos adicionar uma instrução para jogar fora a tabela, digitaremos o comando `DROP TABLE IF EXISTS` e o nome da tabela, no caso, "Alunos". Teremos, `String sql = DROP TABLE IF EXISTS Alunos` . Vamos pedir também para que ela seja executada, através de `db.execSQL(sql)` .

Por que estamos fazendo isso?

Imagine que tivéssemos escrito o método `onCreate` errado, ao em vez de "nome", tivéssemos digitado "nom" e tivéssemos subido a aplicação. Essa seria nossa versão 1 e ao perceber o erro, voltaríamos para alterar isso. Mas, arrumando o *Android*, ele não chama nem o `onCreate` , nem o `onUpgrade` , pois a aplicação não mudou sua versão. Como erros iguais esse são comuns é bom deixar a instrução `DROP TABLE IF EXISTS` , pois dessa forma a tabela velha será substituída pela arrumada.

Também vamos pedir, na linha de baixo de `db.execSQL(sql)` , para chamar o `onCreate` de novo, utilizando o `onCreate(db)` . Sempre que fizermos alguma modificação, vamos alterar a versão (no caso do número 1 para 2, 3, 4, etc.) e a tabela antiga vai ser "jogada fora" e substituída por uma versão mais atualizada. Ficaremos com o seguinte método:

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String sql = "DROP TABLE IF EXISTS Alunos";
        db.execSQL(sql);
        onCreate(db);
    }
```

Por enquanto é isso que faremos! Estamos apenas preparando caminho para que o banco de dados comece a salvar os alunos. Na tela final ficaremos com:

```
public class AlunoDAO extends SQLiteOpenHelper {
    public AlunoDAO (Context context) {
        super(context, "Agenda", null, 1);
    }

     @Override
        public void onCreate (SQLiteDatabase db) {
            String sql = "CREATE TABLE Alunos (id INTEGER PRIMARY KEY, nome TEXT NOT NULL, endei
            db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        String sql = "DROP TABLE IF EXISTS Alunos";
        db.execSQL(sql);
        onCreate(db);
    }
}
```