



# Programação para Android

Aula 08: Persistência de dados - SQL

# Persistência de dados

- ▶ Na maioria das aplicações precisamos ter algum tipo de persistência de dados.
- ▶ Para guardar informações de forma persistente no Android podemos usar os seguintes recursos:
  - ▶ *Shared Preferences*: Armazena dados primitivos privados em pares chave-valor.
  - ▶ *Internal Storage*: Armazena dados privados na memória do dispositivo.
  - ▶ *External Storage*: Armazena dados públicos no cartão de memória.
  - ▶ *SQLite Databases*: Armazena dados estruturados num banco de dados privado.
  - ▶ *Network Connection*: Armazena dados na web no seu servidor de rede.

# Parte 04: SQLite Databases

# Persistência de dados - SQLite Databases

- ▶ O Android fornece suporte completo ao banco de dados SQLite. Qualquer banco de dados será acessível pelo nome por qualquer classe da aplicação, mas não fora da aplicação.
- ▶ O SQLite é uma biblioteca em linguagem C, open source, que implementa um banco de dados SQL embutido, ou seja, não tem processo servidor separado, lendo e escrevendo diretamente no arquivo de banco de dados no disco.
- ▶ O SQLite está disponível em todos os dispositivos Android, utilizá-lo não requer qualquer configuração ou administração de banco de dados, precisamos somente definir os comandos SQL para criar e atualizar o banco.
- ▶ Depois disso, o banco de dados é gerenciado automaticamente para você pela plataforma Android.

# Persistência de dados - SQLite Databases

- ▶ Quando criado, o banco de dados por padrão é armazenado no diretório `/data/data/<nome-do-pacote-utilizado>/databases/nome-do-arquivo.sqlite`.
- ▶ Existem várias formas de acessar uma base de dados SQLite, neste caso, para efeitos didáticos e de simplicidade implementaremos nosso acesso a dados através da criação de métodos específicos na classe `MainActivity`.
- ▶ Uma outra forma muito utilizada de acesso a dados é estender a classe `SQLiteOpenHelper` que contém uma série de métodos específicos para acesso a dados.

# Persistência de dados - SQLite Databases

## Definição da base de dados

- ▶ Nosso banco de dados chamará **dados\_telefone** e contará com uma tabela chamada **contato**, com os seguintes campos:
  - ▶ `_id INTEGER PRIMARY KEY AUTOINCREMENT`
  - ▶ `nome varchar(100)`
  - ▶ `telefone varchar(10)`
- ▶ O primeiro passo será criar um objeto do tipo `SQLiteDataBase`. Este objeto será de acesso global a toda a classe é através dele que criaremos e usaremos um banco de dados do SQLite.

```
public class MainActivity extends Activity
{
    SQLiteDatabase db;
```

# Persistência de dados - SQLite Databases

## Definição da base de dados

- ▶ O segundo passo será construir um método privado que será responsável pela criação do banco de dados e da tabela contato.
- ▶ Para o desenvolvimento deste método usaremos o objeto **db** (SQLiteDataBase). Inicialmente usamos o comando **openOrCreateDataBase** para abrir um banco de dados existente ou criar um novo.

```
//Cria ou abre um banco de dados  
db = openOrCreateDatabase("dados_telefone.db", Context.MODE_PRIVATE,  
null);
```

# Persistência de dados - SQLite Databases

## Definição da base de dados

- Em seguida usamos um objeto do tipo `StringBuilder` construir um comando SQL, neste caso o comando de criação das tabelas.

```
//Objeto usado para construir a STRING do comando SQL a ser executado
//Neste caso a string SQL conterá o comando de criação de tabelas
StringBuilder sql = new StringBuilder();
//Obrigatoriamente tem que ter uma coluna id no banco SQL Lite
sql.append("CREATE TABLE IF NOT EXISTS contato(");
sql.append("_id INTEGER PRIMARY KEY AUTOINCREMENT,");
sql.append("nome varchar(100),");
sql.append("telefone varchar(10) );");
```

- OBS: `sql.append` tem a finalidade de concatenar os comandos escritos.



# Persistência de dados - SQLite Databases

## Definição da base de dados

- ▶ Posteriormente acionaremos um comando para executar uma *query* armazenada em uma *string (sql)*. O método `execSQL` executa o comando `sql` definido na *query*.
- ▶ Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação

```
try
{
    //executa um comando SQL, neste caso a StringBuilder SQL
    db.execSQL(sql.toString());
}
catch(Exception ex)
{

    Toast.makeText(getApplicationContext(), "Error = " + ex.getMessage(),
    Toast.LENGTH_LONG).show();
}
```

# Persistência de dados - SQLite Databases

## Definição da base de dados

- Por fim, chamaremos o método criado(criarBancoDados) no método *onCreate* da MainActivity

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

```
    criarBancoDados();
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - INSERIR

- Antes de implementar o código relacionado a inserção de dados, construiremos uma interface gráfica composta por 2 campos (EditText) que será responsável por receber os dados que posteriormente serão armazenado no banco de dados. Além dos campos para entrada de dados, usaremos 2 botões para a realização das ações (INSERIR e MOSTRAR)



The screenshot shows an Android application interface with a black header bar containing the Android logo and the title "Telefone". Below the header, there are two text input fields. The first field is labeled "Nome:" and contains the placeholder text "Digite seu nome aqui". The second field is labeled "Telefone:" and contains the placeholder text "Digite seu telefone aqui". Below the input fields, there are two buttons: "Inserir" and "Mostrar".

# Persistência de dados - SQLite Databases

## Manipulação de dados - INSERIR

- Para criação do método INSERIR utilizaremos o mesmo princípio apresentando anteriormente. Criaremos um método private chamado INSERIR.

```
private void Inserir()  
{  
}
```

- Inicialmente nosso método buscará os dados digitados pelo usuário nos campos do tipo EditText definidos no layout da interface gráfica.

```
EditText txtNome,txtTelefone;  
String nome, telefone;
```

```
txtNome = (EditText)findViewById(R.id.txtNome);  
txtTelefone = (EditText)findViewById(R.id.txtTelefone);
```

```
nome = txtNome.getText().toString();  
telefone = txtTelefone.getText().toString();
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - INSERIR

- ▶ Em seguida usamos um objeto do tipo `StringBuilder` construir um comando SQL, neste caso o comando `INSERT` para registrar os dados na tabela.

```
StringBuilder sql = new StringBuilder();  
sql.append("INSERT INTO contato(nome,telefone) VALUES (");  
sql.append("'" + nome + "'");  
sql.append(",");  
sql.append("'" + telefone + "'");  
sql.append(")");
```

- ▶ OBS: No comando `INSERT` não é necessário incluir o campo `_id`, uma vez que este é um campo autoincrement, ou seja, gerado sequencialmente pelo próprio SQLite.

# Persistência de dados - SQLite Databases

## Manipulação de dados - INSERIR

- ▶ Posteriormente acionaremos um comando para executar uma *query* armazenada em uma *string* (*sql*). O método `execSQL` executa o comando `sql` definido na *query*.
- ▶ Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação

```
try
{
    db.execSQL(sql.toString());
    Toast.makeText(getBaseContext(), "OK - Inserido",
        Toast.LENGTH_SHORT).show();
}
catch(SQLException ex)
{
    Toast.makeText(getBaseContext(), sql.toString() + "Erro = " +
        ex.getMessage(), Toast.LENGTH_LONG).show();
}
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - INSERIR

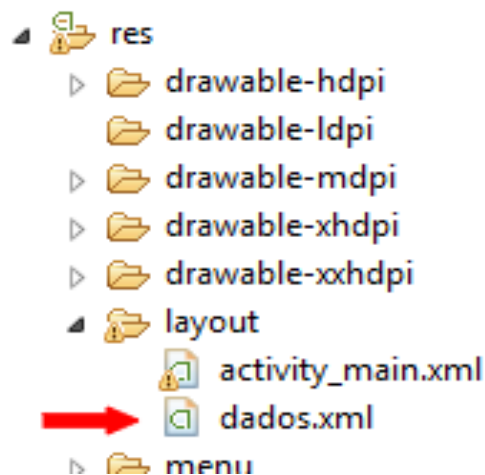
- ▶ Por fim, para a chamada do método INSERIR criaremos um listener do tipo `OnClick` para o botão inserir, de forma que o usuário ao clicar(tocar) no botão inserir o código de inserção seja realizado.

```
View.OnClickListener inserir = new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        Inserir(); //CHAMADA DO MÉTODO INSERIR  
  
    }  
};
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- ▶ Os dados a serem exibidos serão apresentados como itens de um **ListView**, sendo que um **item de lista** pode ser um texto simples (um **TextView**) ou , por exemplo, dois **TextView**, um contendo o nome de uma pessoa e outro o seu telefone.
- ▶ Para apresentação de dados iremos criar um **ListView** customizado, ou seja, criaremos um layout específico para apresentação dos dados. Este layout deverá ser criado dentro do diretório **res -> layout** e chamará **dados**.





# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- O layout será composto por duas TextView para apresentação dos dados(nome, telefone). Como este layout não se trata de uma activity, não faz-se necessário registra-lo no arquivo AndroidManifest.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/tvId"
        android:visibility="invisible"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/tvNome"/>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/tvTelefone"/>
</LinearLayout>
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- Da mesma forma dos códigos anteriores, definimos um método PRIVATE para obter os dados armazenados nas tabelas do banco de dados.

```
private void Buscar()  
{  
}
```

- Em seguida usamos um objeto do tipo StringBuilder construir um comando SQL, neste caso o comando SELECT para retornar dados na tabela. Usaremos um cursor para armazenar os dados obtidos.

```
StringBuilder sql = new StringBuilder();  
sql.append("SELECT * FROM contato");
```

```
Cursor dados = db.rawQuery(sql.toString(), null);
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- Posteriormente faremos a conexão com os dados armazenados no Cursor e os campos criados no layout da ListView, ou seja, a TextView nome deverá receber os dados da coluna nome, e a TextView telefone deverá receber os dados da coluna telefone.
- Para isto usaremos um SimpleCursorAdapter. Antes de realizar a conexão dos dados através do Adapter, criaremos 2 vetores. O primeiro do tipo INT que armazenará os ID dos campos definidos no layout da ListView; e o segundo com os nomes dos campos da consulta que serão exibidos.

```
//Array com os ID dos campos do layout dados  
int[] to = {R.id.tvId,R.id.tvNome,R.id.tvTelefone};  
//Array com o nome dos campos da tabela que serão mostrados  
String[] from = {"_id","nome","telefone"};
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- ▶ O SimpleCursorAdapter fará a ligação entre os campos retornados da consulta e os campos (TextView) definidos no layout da ListView.

```
try
{
SimpleCursorAdapter ad = new SimpleCursorAdapter(getBaseContext(), R.layout.dados, dados,
from, to, 0);

ListView lvDados;
lvDados = (ListView) findViewById(R.id.lvDados);

lvDados.setAdapter(ad);
}
catch(Exception ex)
{
Toast.makeText(getBaseContext(), sql.toString() + " Erro = " + ex.getMessage(),
Toast.LENGTH_LONG).show();
}
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- Por fim, para a chamada do método Buscar criaremos um botão com o nome Mostrar e um listener do tipo OnClickListener para o botão mostrar, de forma que o usuário ao clicar-lo o código de inserção seja realizado.

```
View.OnClickListener mostrar = new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
  
        Buscar(); // Chamada do método BUSCAR  
  
    }  
};
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - BUSCAR

- Apresentação dos resultados após clicar no botão Mostrar



The screenshot shows an Android application interface with a black header bar containing an Android robot icon and the title "Telefone". Below the header, there are two text input fields. The first is labeled "Nome:" and contains the placeholder text "Digite seu nome aqui". The second is labeled "Telefone:" and contains the placeholder text "Digite seu telefone aqui". Below these fields are two gray buttons: "Inserir" and "Mostrar". At the bottom of the screen, there is a list of data entries, each consisting of a name and a phone number, separated by horizontal lines. The entries are: "CBA 321", "ghi 789", and "def 456".

Nome	Telefone
CBA	321
ghi	789
def	456

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- ▶ Para criação do código referente aos comandos EDITAR e EXCLUIR, construiremos uma nova activity, de forma que o usuário ao clicar(tocar) em um dos itens exibidos na ListView seja carregado uma nova janela com os dados selecionado e as opções de EDITAR e EXCLUIR.
- ▶ Como se trata de uma nova activity é necessário definir o Layout e em seguida criar uma classe para associar ao layout e também registrá-la no arquivo AndroidManifest.xml
- ▶ A activity que receberá os dados para edição e/ou exclusão será carregada a partir do Listener *OnItemClickListener*
- ▶ Neste listener será buscado os dados selecionados e enviados para a nova Activity que exibirá os dados com as opções de EDITAR e EXCLUIR

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Programação do Listener relacionado ao clique do item selecionado na ListView.

```
OnItemClickListener itemClicado = new OnItemClickListener() {  
  
    @Override  
    public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,  
    long arg3) {  
        // TODO Auto-generated method stub  
  
        //A listview criada tem vários itens e é necessário saber  
        //o item selecionado e buscar o id, nome e telefone relacionado ao item  
        //O comando getChildAt pega os filhos da listview clicada.  
        //Arg2 refere-se ao item selecionado  
        View view = lvDados.getChildAt(arg2);
```



# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Continuação da programação do Listener relacionado ao clique do item selecionado na ListView

```
String id,nome,telefone;
```

```
TextView tvId = (TextView) view.findViewById(R.id.tvId);  
id = tvId.getText().toString();
```

```
TextView tvNome = (TextView) view.findViewById(R.id.tvNome);  
nome = tvNome.getText().toString();
```

```
TextView tvTelefone = (TextView)  
view.findViewById(R.id.tvTelefone);  
telefone = tvTelefone.getText().toString();
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Continuação da programação do Listener relacionado ao clique do item selecionado na ListView

```
//cria uma intenção para carregar uma nova activity  
Intent dados = new  
Intent(MainActivity.this,DadosActivity.class);  
//envia os dados para a activity que será carregada  
dados.putExtra("id", id);  
dados.putExtra("nome", nome);  
dados.putExtra("telefone", telefone);  
//inicia a activity  
startActivity(dados);  
}  
};
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- A Activity criada para receber os dados e mostrar as opções de EDITAR e EXCLUIR deve conter os widgets: TextView, EditText e Button. Sua apresentação visual é mostrada a seguir:



The screenshot displays an Android application window with a black title bar containing an Android icon and the text "Telefone". Below the title bar, there is a form with two text input fields. The first field is labeled "Nome:" and contains the text "CBA". The second field is labeled "Telefone:" and contains the text "321". Below these fields are two buttons: "Atualizar" (Update) and "Excluir" (Delete). The buttons are gray with black text.

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- No método onCreate abrimos o banco de dados, associamos os widgets do layout e pegamos os dados enviado pela activity anterior.

```
//Cria ou abre um banco de dados
db = openOrCreateDatabase("dados_telefone.db", Context.MODE_PRIVATE, null);

//associa os campos criados na activity
tvIdEditar = (TextView)findViewById(R.id.tvIdEditar);
txtNomeEditar = (EditText)findViewById(R.id.txtNomeEditar);
txtTelefoneEditar = (EditText)findViewById(R.id.txtTelefoneEditar);

//cria uma nova intenção para buscar os dados enviados pela activity anterior
Intent valores = getIntent();
//pega os valores enviados da activity anterior e preenche os campos

tvIdEditar.setText(valores.getStringExtra("id"));
txtNomeEditar.setText(valores.getStringExtra("nome"));
txtTelefoneEditar.setText(valores.getStringExtra("telefone"));
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Para criação do método INSERIR utilizaremos o mesmo princípio apresentando nos comandos INSERIR e MOSTRAR. Criaremos um método private chamado EDITAR.

```
private void Editar()  
{  
}
```

- Inicialmente nosso método buscará os dados digitados pelo usuário nos campos do tipo EditText definidos no layout da interface gráfica.

```
//associa os campos criados na activity  
tvIdEditar = (TextView)findViewById(R.id.tvIdEditar);  
txtNomeEditar = (EditText)findViewById(R.id.txtNomeEditar);  
txtTelefoneEditar = (EditText)findViewById(R.id.txtTelefoneEditar);  
  
//pega os valores de cada campo e coloca em variáveis  
id = tvIdEditar.getText().toString();  
nome = txtNomeEditar.getText().toString();  
telefone = txtTelefoneEditar.getText().toString();
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Em seguida usamos um objeto do tipo `StringBuilder` construir um comando SQL, neste caso o comando `UPDATE` para atualizar os dados na tabela.

//cria uma string SQL para atualizar o contato selecionado

```
StringBuilder sql = new StringBuilder();  
sql.append("UPDATE contato SET nome = '" + nome + "'");  
sql.append(", telefone = '" + telefone + "'");  
sql.append(" WHERE _id = " + id );
```

```
Toast.makeText(getApplicationContext(), sql, Toast.LENGTH_LONG).show();
```

- OBS: No comando `UPDATE` é necessário incluir o campo `_id`, uma vez que deve-se apenas atualizar o registro solicitado.

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- ▶ Posteriormente acionaremos um comando para executar uma *query* armazenada em uma *string* (*sql*). O método `execSQL` executa o comando `sql` definido na *query*.
- ▶ Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação

```
try
{
db.execSQL(sql.toString());
Toast.makeText(getBaseContext(), "OK - Editado", Toast.LENGTH_SHORT).show();
//volta para a janela anterior
Intent Main = new Intent(DadosActivity.this, MainActivity.class);
startActivity(Main);
}
catch(SQLException ex)
{
Toast.makeText(getBaseContext(), sql.toString() + "Erro = " + ex.getMessage(),
Toast.LENGTH_LONG).show();
}
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EDITAR

- Por fim, para a chamada do método EDITAR criaremos um listener do tipo `OnClick` para o botão EDITAR, de forma que o usuário ao clicar(tocar) no botão EDITAR o código de atualização dos dados seja realizado.

```
View.OnClickListener editar = new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        Editar();  
    }  
};
```



# Persistência de dados - SQLite Databases

## Manipulação de dados - EXCLUIR

- Para criação do método EXCLUIR utilizaremos o mesmo princípio apresentando nos comandos anteriores. Criaremos um método private chamado EXCLUIR.

```
private void Excluir()  
{  
}
```

- Inicialmente nosso método buscará o \_id relacionado ao item a ser excluído.

```
String id;
```

```
//associa os campos criados na activity  
tvIdEditor = (TextView)findViewById(R.id.tvIdEditor);
```

```
//pega o Id e coloca na variável id  
id = tvIdEditor.getText().toString();
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EXCLUIR

- ▶ Em seguida usamos um objeto do tipo `StringBuilder` construir um comando SQL, neste caso o comando `DELETE` para excluir os dados na tabela.

```
//cria uma string SQL para excluir o contato selecionado  
StringBuilder sql = new StringBuilder();  
sql.append("DELETE FROM contato WHERE _id = " + id);
```

- ▶ OBS: No comando `DELETE` é necessário incluir o campo `_id`, uma vez que deve-se apenas excluir o registro solicitado.

# Persistência de dados - SQLite Databases

## Manipulação de dados - EXCLUIR

- ▶ Posteriormente acionaremos um comando para executar uma *query* armazenada em uma *string* (*sql*). O método `execSQL` executa o comando `sql` definido na *query*.
- ▶ Usaremos a estrutura `try...catch` para maior consistência de nossa aplicação

```
try
{
    db.execSQL(sql.toString());
    Toast.makeText(getBaseContext(), "OK - Excluído", Toast.LENGTH_LONG).show();
    //volta para a janela anterior
    Intent Main = new Intent(DadosActivity.this, MainActivity.class);
    startActivity(Main);
}
catch(SQLException ex)
{
    Toast.makeText(getBaseContext(), sql.toString() + "Erro = " + ex.getMessage(),
    Toast.LENGTH_LONG).show();
}
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EXCLUIR

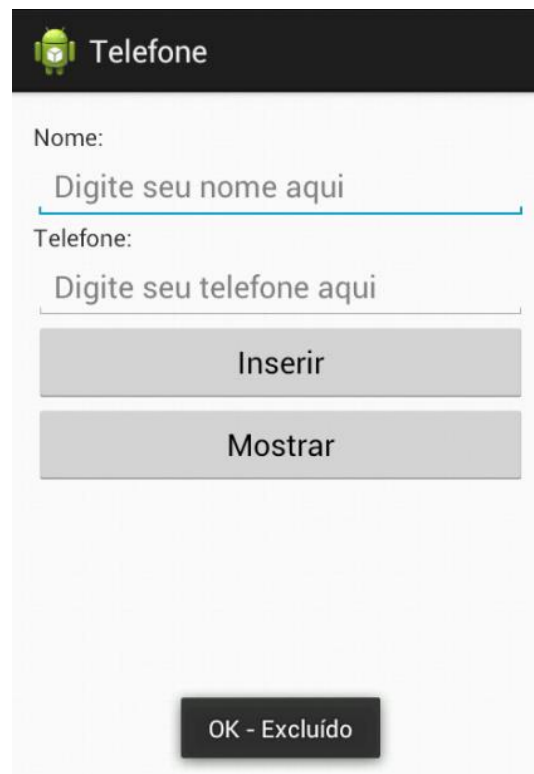
- Por fim, para a chamada do método EXCLUIR criaremos um listener do tipo `OnClick` para o botão EXCLUIR, de forma que o usuário ao clicar(tocar) no botão EXCLUIR o código de exclusão dos dados seja realizado.

```
View.OnClickListener excluirl = new View.OnClickListener() {  
  
    @Override  
    public void onClick(View v) {  
        // TODO Auto-generated method stub  
        Excluir();  
    }  
};
```

# Persistência de dados - SQLite Databases

## Manipulação de dados - EXCLUIR

### ► Resultado da exclusão



The screenshot shows an Android application interface with a black header bar containing an Android icon and the title "Telefone". Below the header, there are two text input fields. The first field is labeled "Nome:" and contains the placeholder text "Digite seu nome aqui". The second field is labeled "Telefone:" and contains the placeholder text "Digite seu telefone aqui". Below these fields are two gray buttons: "Inserir" and "Mostrar". At the bottom of the screen, there is a black button with the text "OK - Excluído".

# Na próxima aula...

- Persistência de dados: networking connection - acessando a uma base de dados de um servidor

