

RECYCLEVIEW

E

BANCO DE DADOS

RECYCLEVIEW

É muito comum aplicativos terem listas para apresentarem seu conteúdo de forma eficiente. E se mal implementada pelo desenvolvedor pode trazer descontentamento para o usuário, o seu cliente final.

RecyclerView → é uma “evolução” da **ListView** e da **GridView**, componentes presentes desde da primeira versão do Android para se fazer listas e grades de conteúdo.

Nas próximas aulas, vamos aprender como trabalhar com uma RecyclerView e com banco de dados “**SQLite**” do Android para manipular pequenas informações no celular.

RECYCLEVIEW

RecyclerView nada mais é que um contêiner que recebe um volume de dados, para exibir ao usuário em forma de lista de itens. Ele possui mais eficiência e simplifica a exibição e o tratamento de maiores volumes de dados.

Componentes necessários para se trabalhar com o RecyclerView:

- **Adapter**: usado para fornecer todas as novas View (Vistas)
- **ViewHolder**: local onde irá conter todas as referencias dos objetos de raiz para cada item
- **LayoutManager**: define o comportamento da RecyclerView na sua forma de apresentar os dados. (Horizontal ou Vertical)

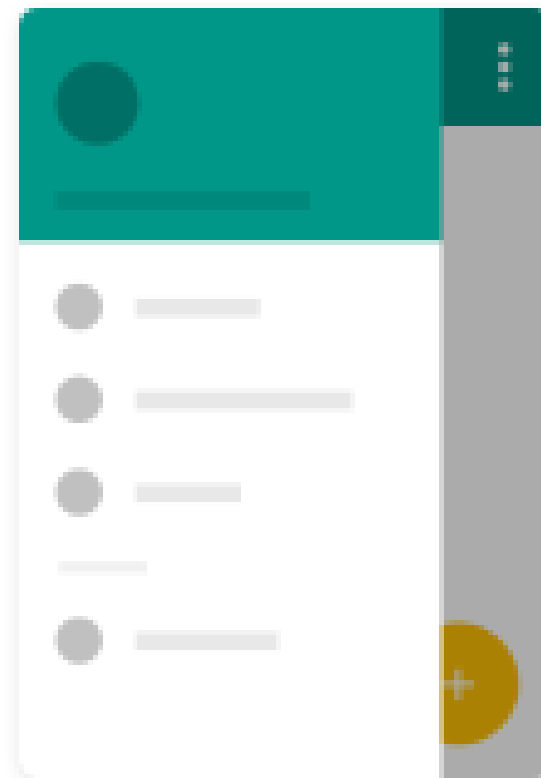
CRUD e SQLITE

CRUD → É um acrônimo para Create, Read, Update e Delete, as quatro operações elementares com bancos de dados relacionais.

SQLite → É o banco de dados compacto mais utilizado no mundo e que já vem com suporte nativo na plataforma Android, como banco de dados local nos smartphones.

CRUD e SQLITE

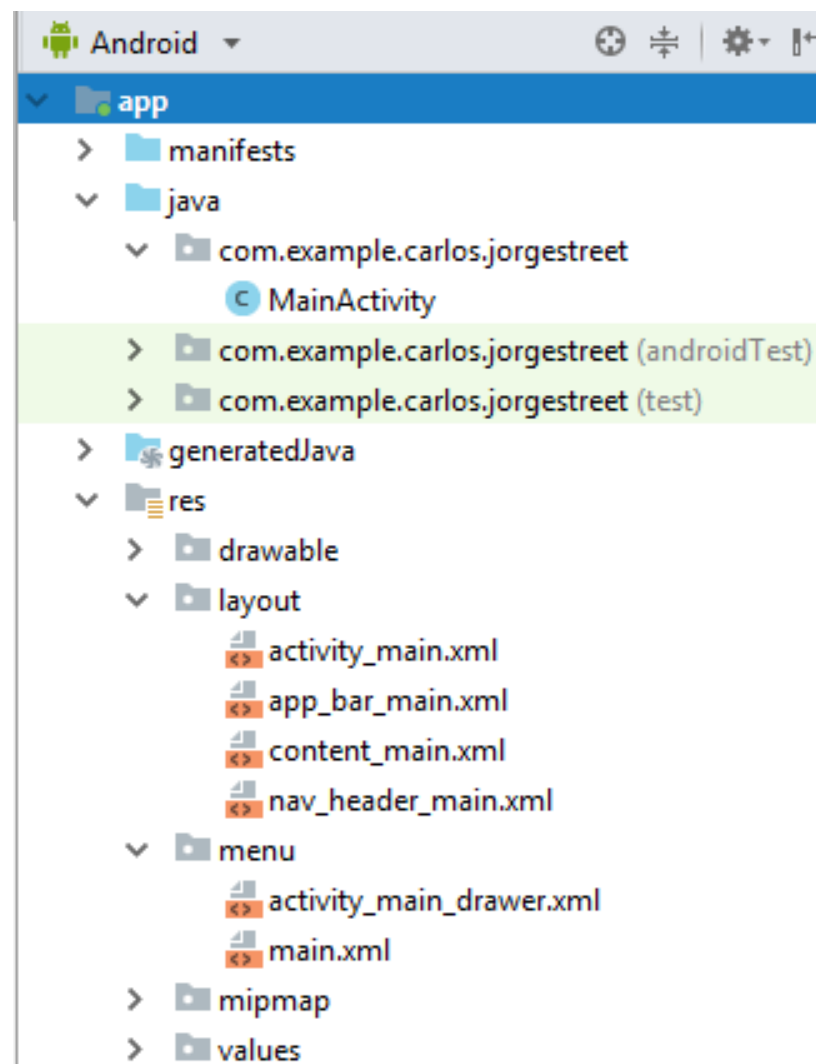
Layout para aula → Usaremos o layout de menu lateral (Navigation Drawer Activity)



Navigation Drawer Activity

Criando e explorando o projeto

- Crie um novo projeto no Android Studio como o nome de JorgeStreet
- Escolha como Activity Principal: Navigation Drawer Activity

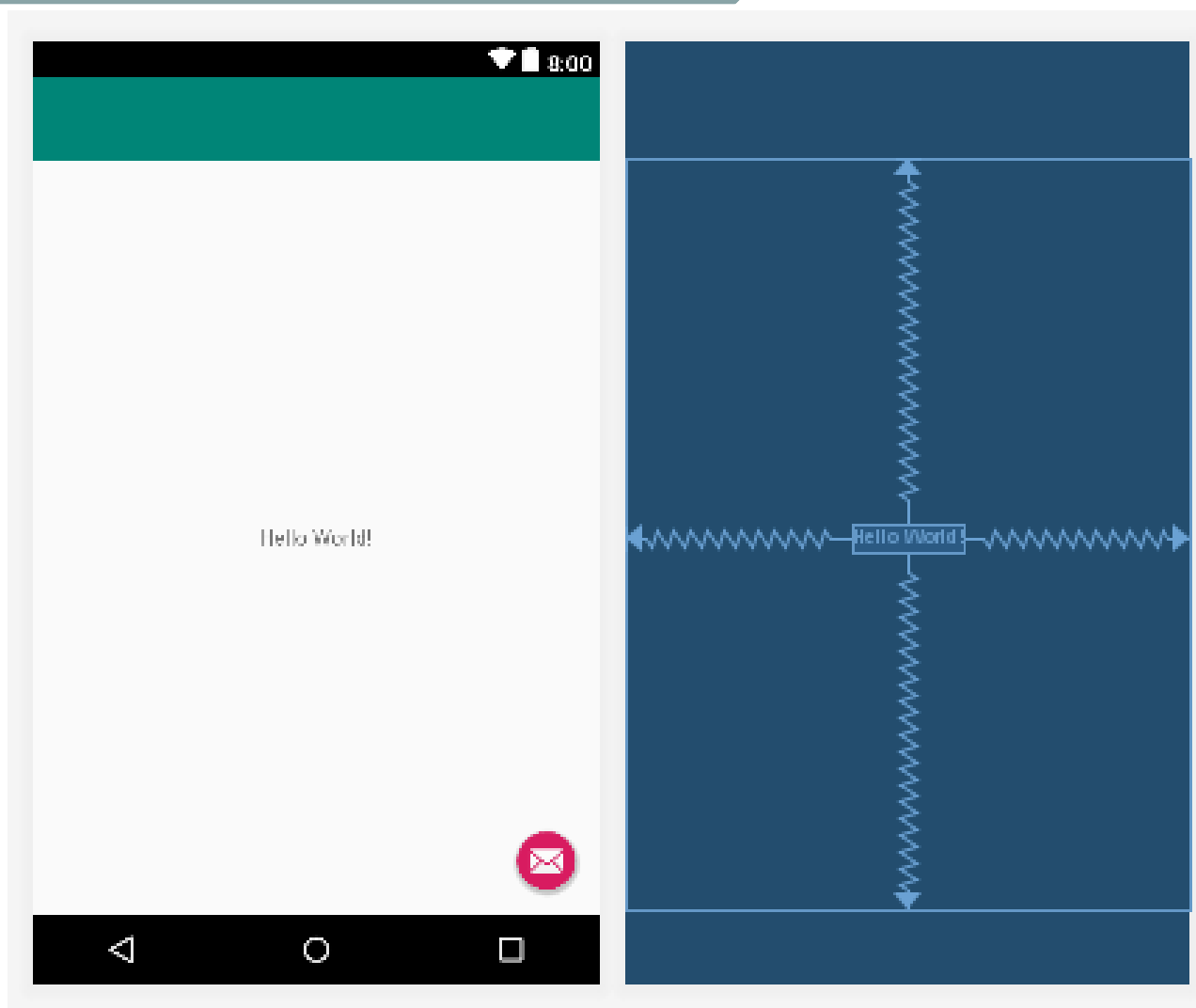


Criando e explorando o projeto

Se abrirmos a MainActivity veremos que ela já possui um evento onCreate que, alias, estamos acostumados em códigos anteriores. (ciclo de vida)

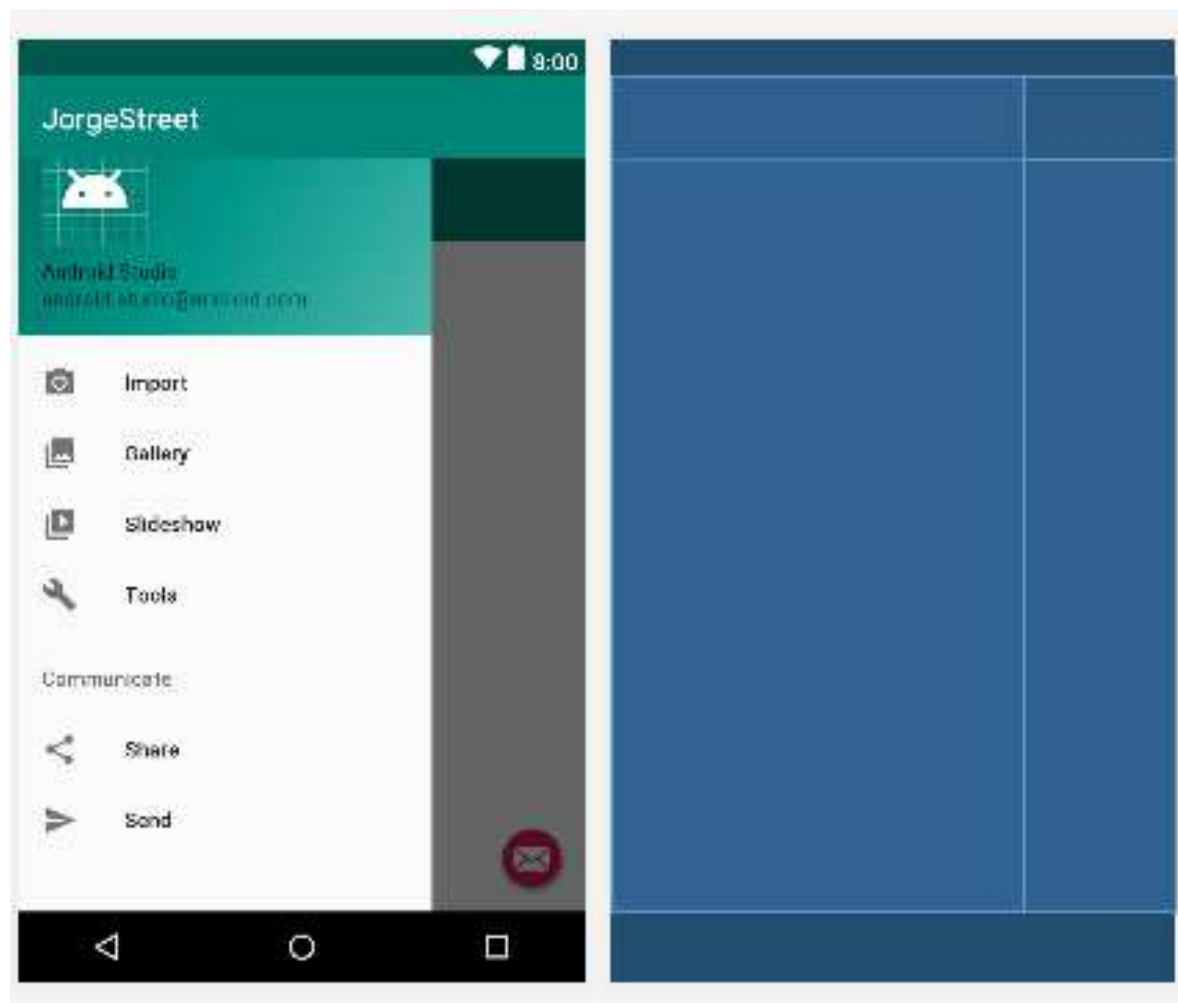
Nesse método definimos o arquivo XML de layout que é o activity_main.xml, uma toolbar que ficará no topo da activity e um botão flutuante (Floating Button) que quando clicado vai disparar uma mensagem genérica na Snackbar (uma barra inferior, tipo um Toast mais moderno)

Criando e explorando o projeto



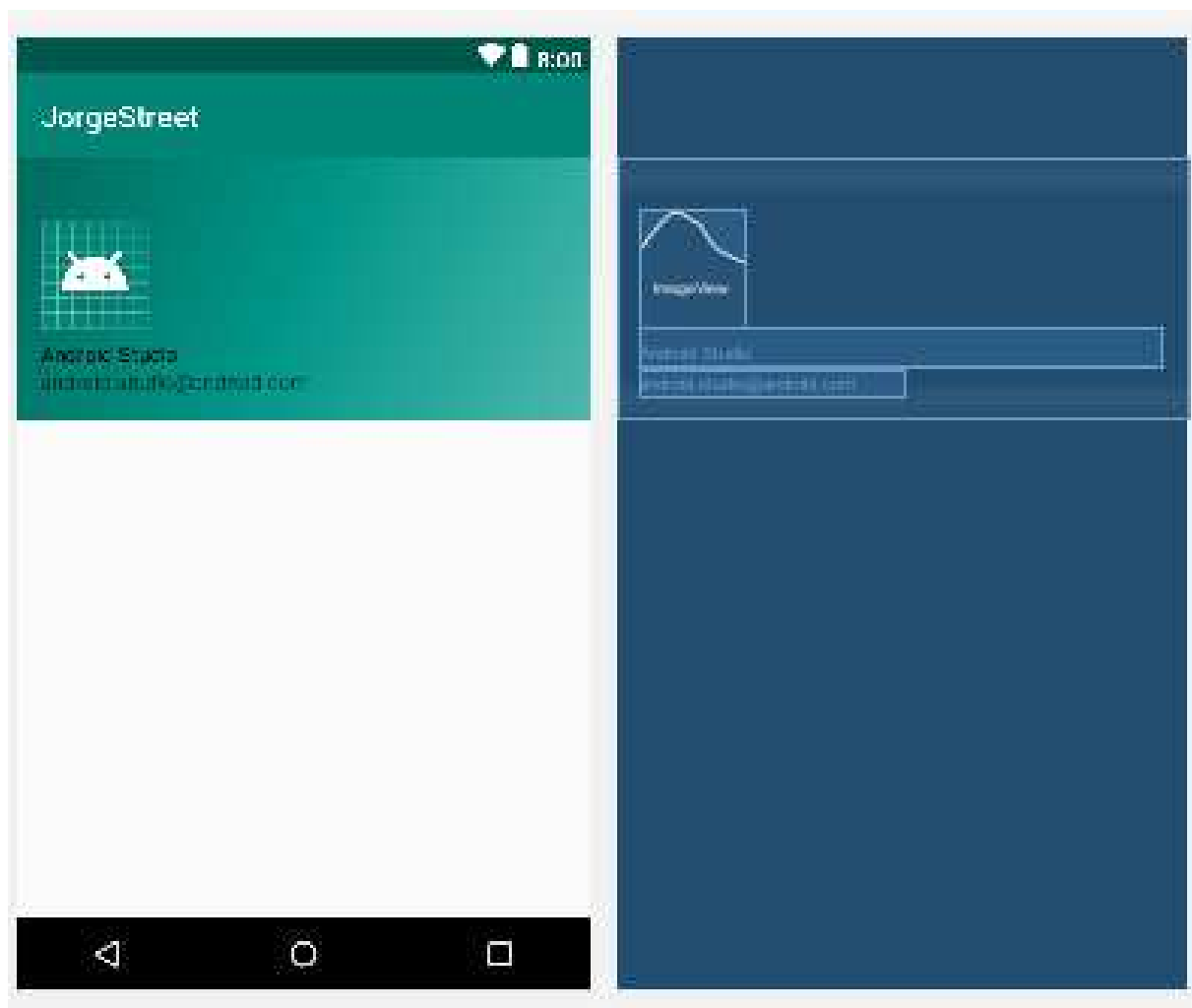
Criando e explorando o projeto

Vamos ter também um layout do menu lateral, onde o usuário vai interagir com o sistema.



Criando e explorando o projeto

Um outro layout para que possamos inserir o ícone do projeto toda vez que abrirmos o menu lateral



Trabalhando o projeto

- Vamos procurar um figura na internet para simbolizar nosso projeto
- Copie essa imagem para a pasta do projeto conforme abaixo:

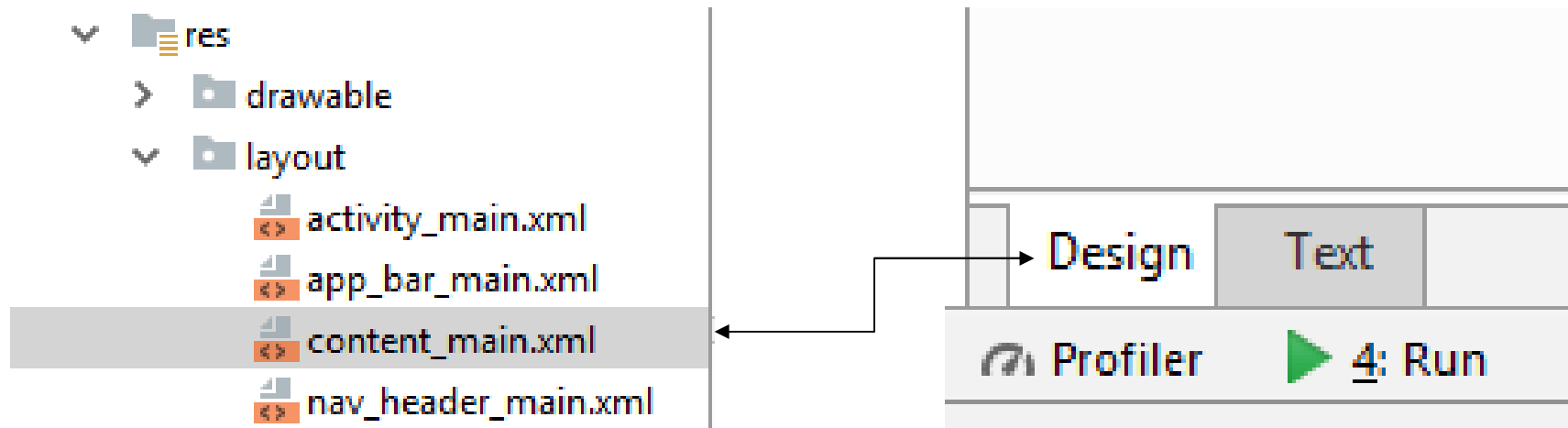
C:\Users\Carlos\Documents\AndroidStudioProjects\JorgeStreet\app\src\main\res\drawable

figura



Trabalhando o projeto

→ Selecione o layout **contente_main** na classe R e clique em Design

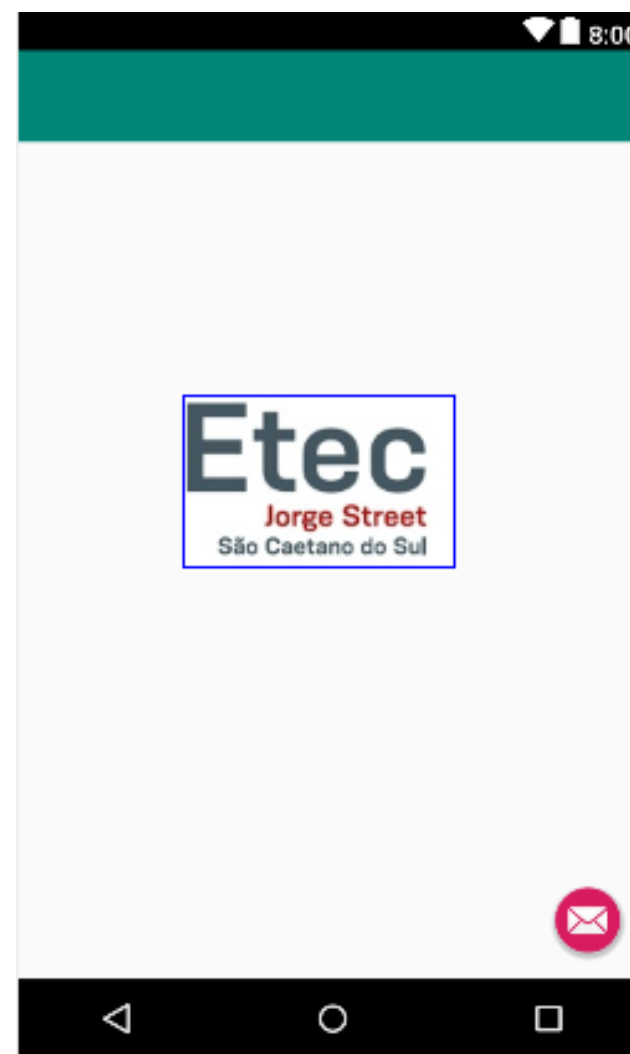
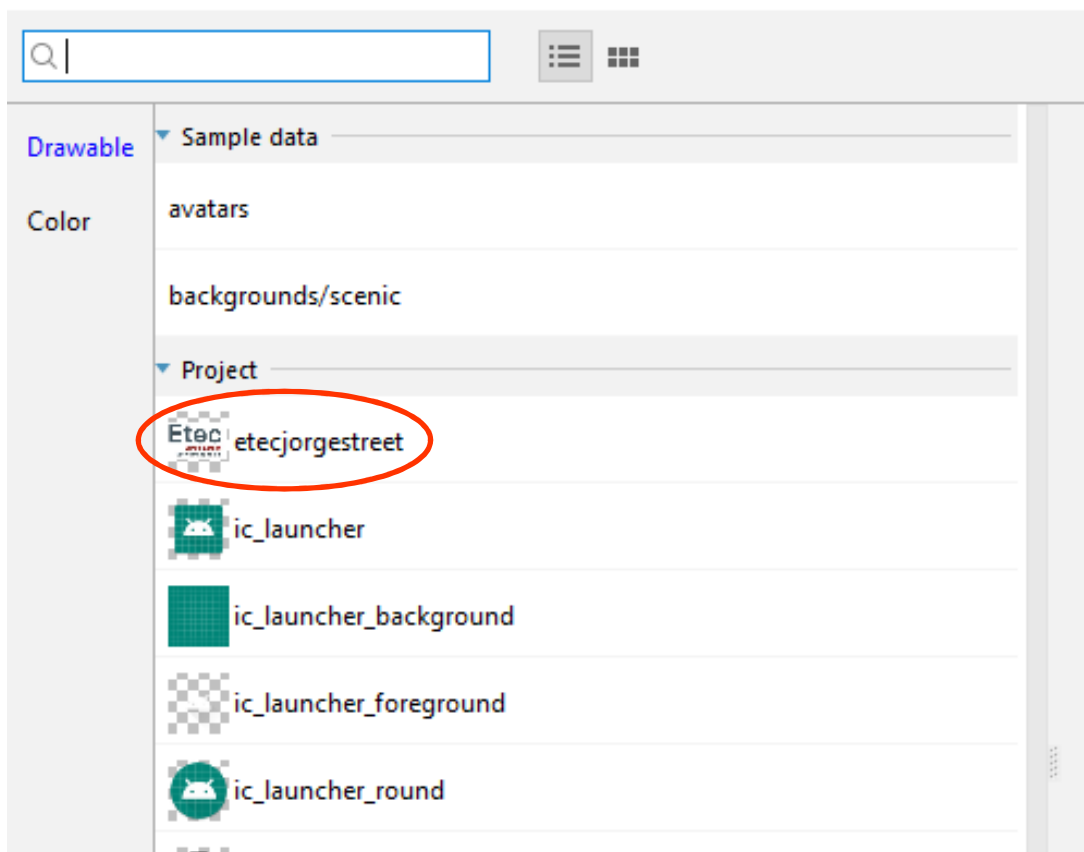


→ Delete o componente **TextView** padrão que já vem no layout assim que você criou o projeto.

→ Na **palette** escolha a opção **Common** e selecione o componete **ImageView** e coloque no layout

Trabalhando o projeto

→ Clique em Project e busque a figura copiada

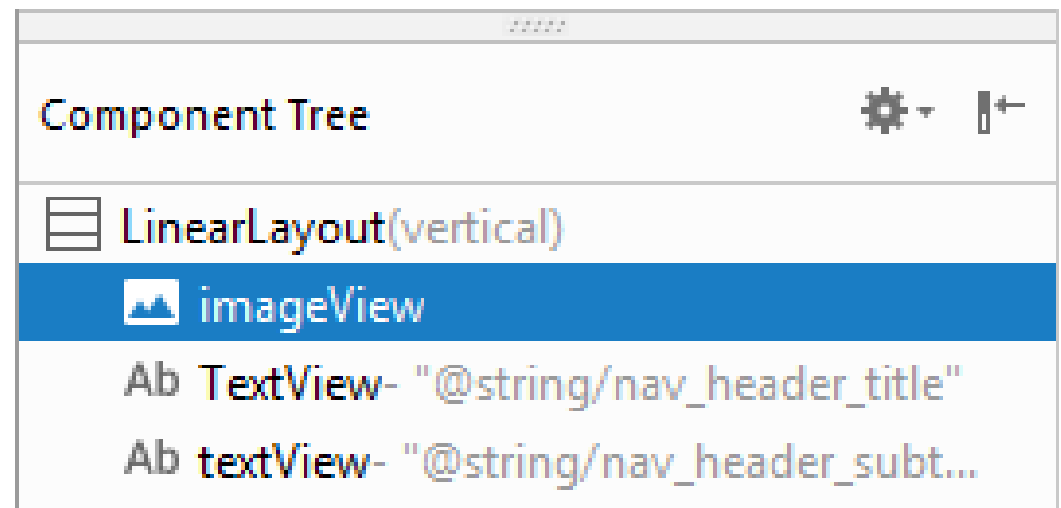
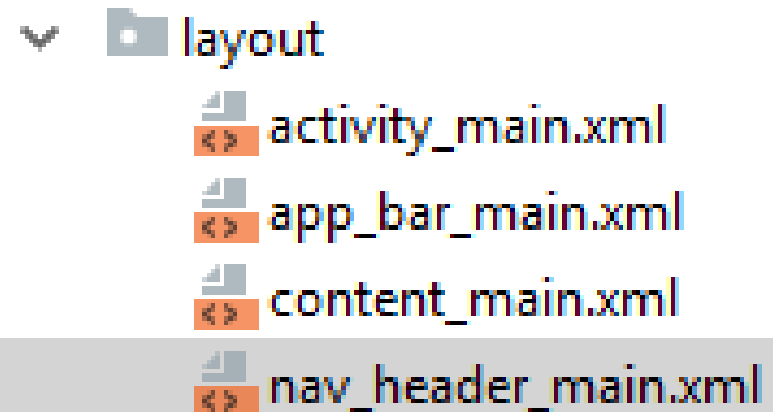


Trabalhando o projeto

→ Vamos trocar o ícone do
nosso projeto no menu

→ Selecionar o layout
“nav_header_mai.xml”

→ Selecione o componente
imageView no Component Tree



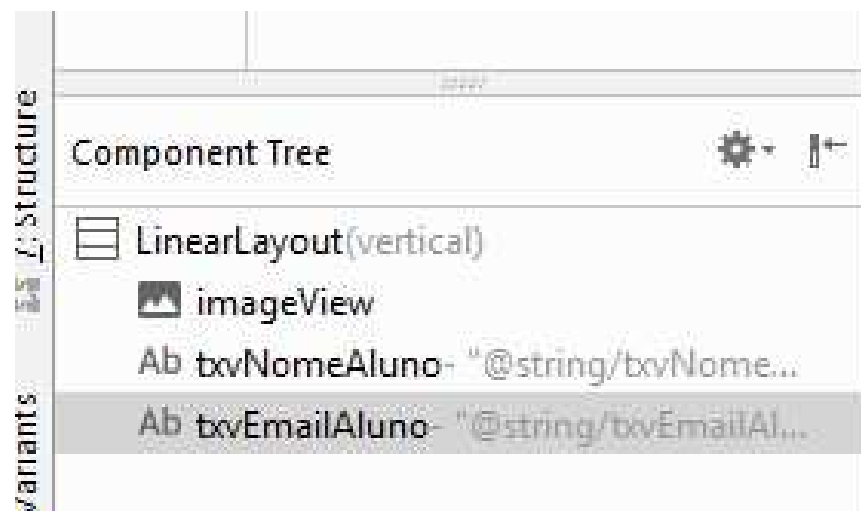
Trabalhando o projeto

- **Selecione o ImageView e clique em srsCompat**
- **Selecione o ícone do projeto e clique “ok”**



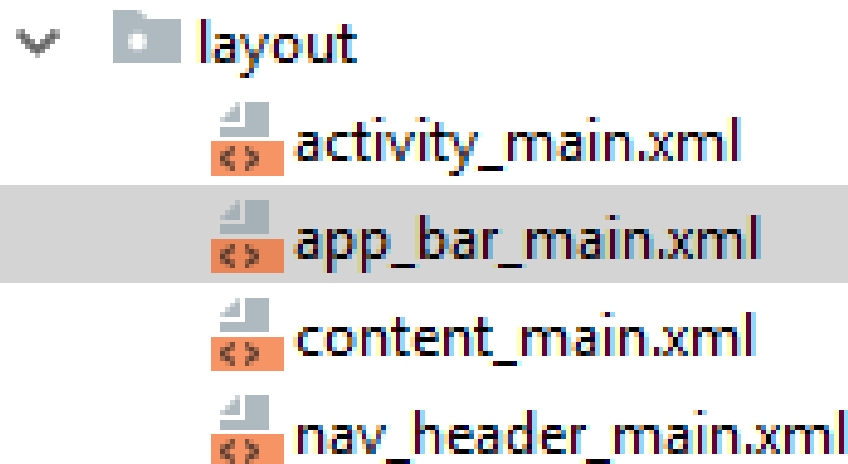
Trabalhando o projeto

→ Vamos alterar os dois TextView “Nome do aluno” e “E-mail do aluno”

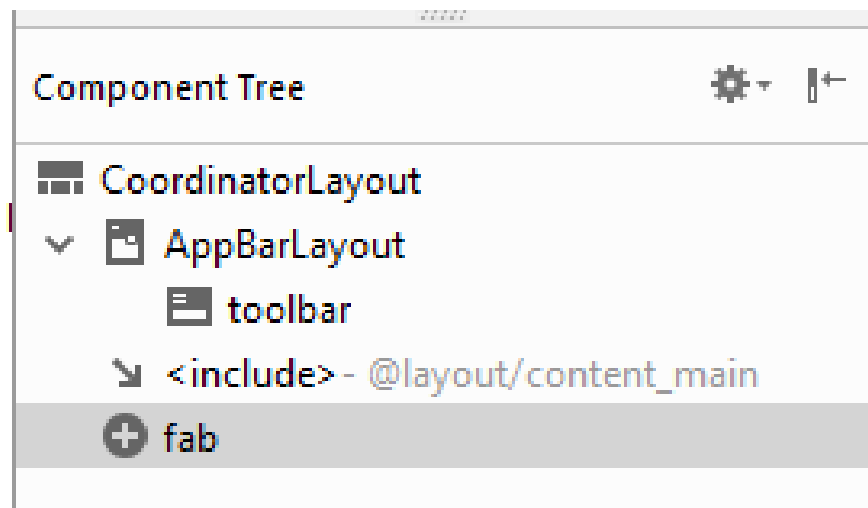


Trabalhando o projeto

- Trocando o ícone do botão flutuante
- Selecione o layout “app_bar_main.xml”
- Selecione o componente `floatingActionButton` de nome “fab” no componet tree



Trabalhando o projeto



Trabalhando o projeto

→ Uma forma de alterar o ícone pode ser:

→ arquivo xml

→ por atributos “tela lateral”

→ Vamos efetuar essa alteração pelo arquivo xml, para aprendermos as duas formas de alteração

Trabalhando o projeto

→ Após selecionado o componente no layout, clique em Text, acesso ao arquivo xml do layout

```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom|end"  
    android:layout_margin="16dp"  
    app:srcCompat="@android:drawable/ic_dialog_email" />
```

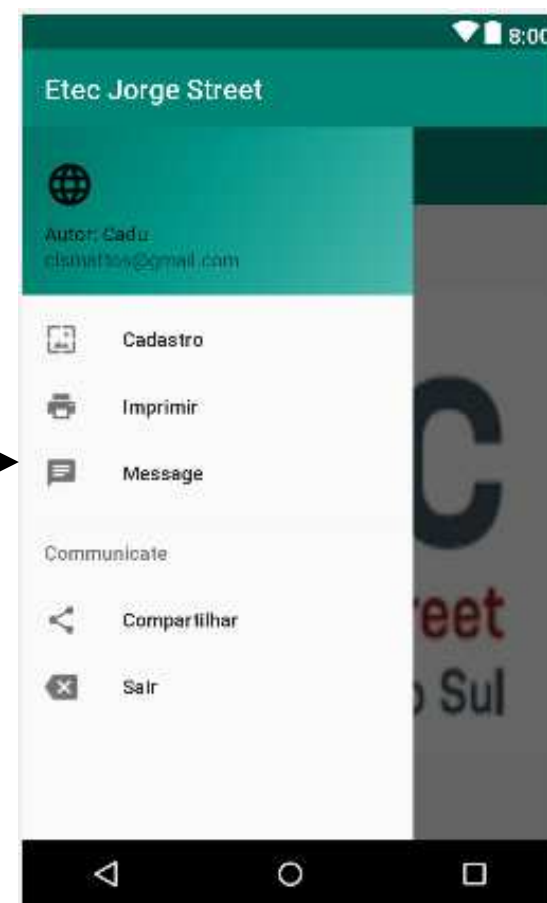
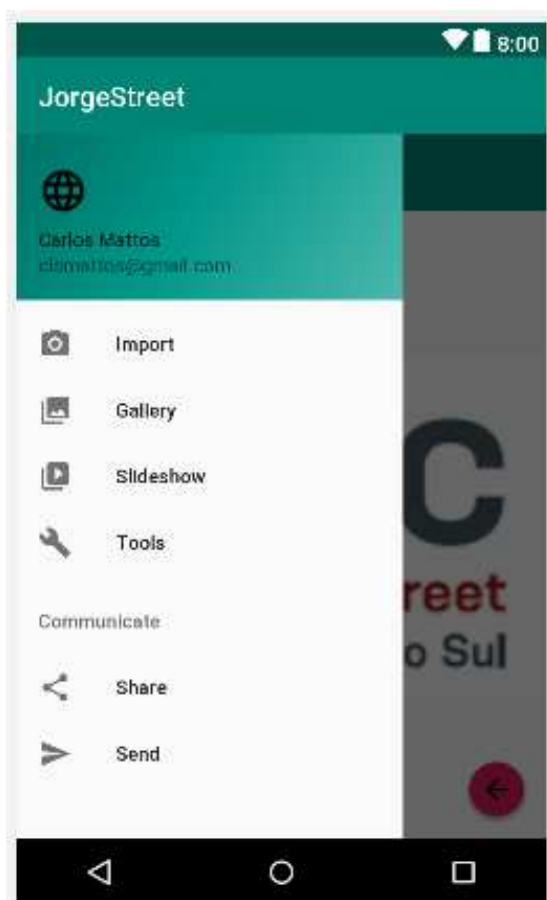
Trabalhando o projeto

→ Vamos trocar o ícone do e-mail “ic_dialog_email” em **app:srcCompat** para o nosso ícone de “back_Space”, que no meu caso “voltar”

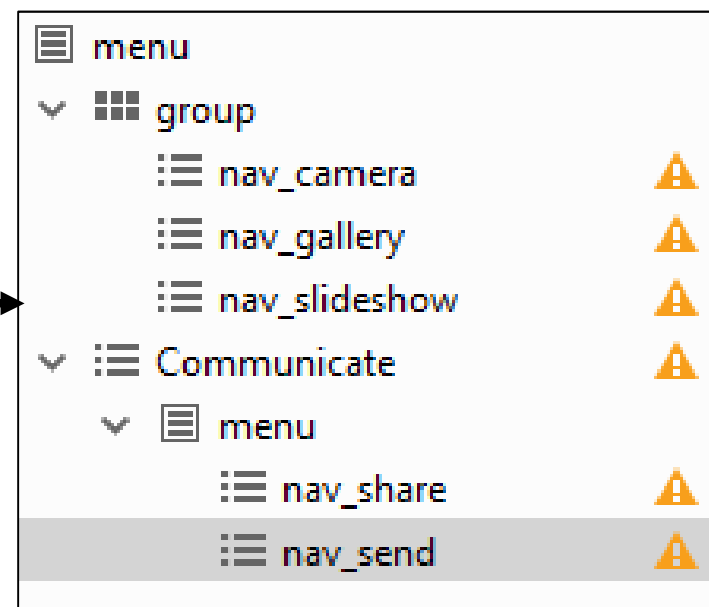
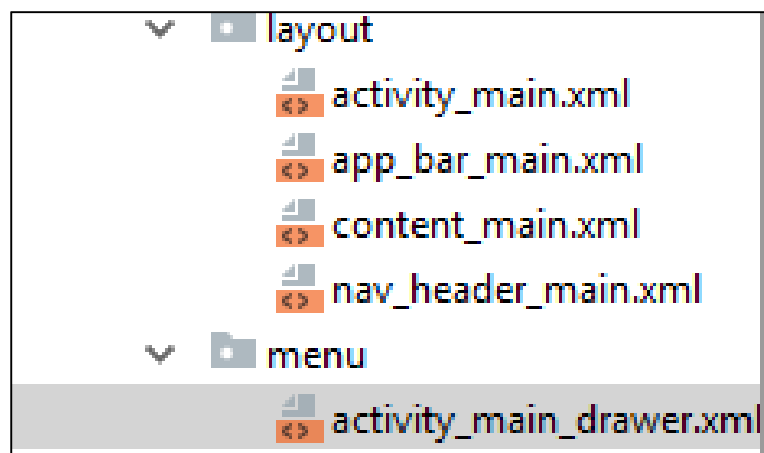
```
<android.support.design.widget.FloatingActionButton  
    android:id="@+id/fab"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="bottom|end"  
    android:layout_margin="16dp"  
    app:srcCompat="@drawable/voltar" />
```

Trabalhando o projeto

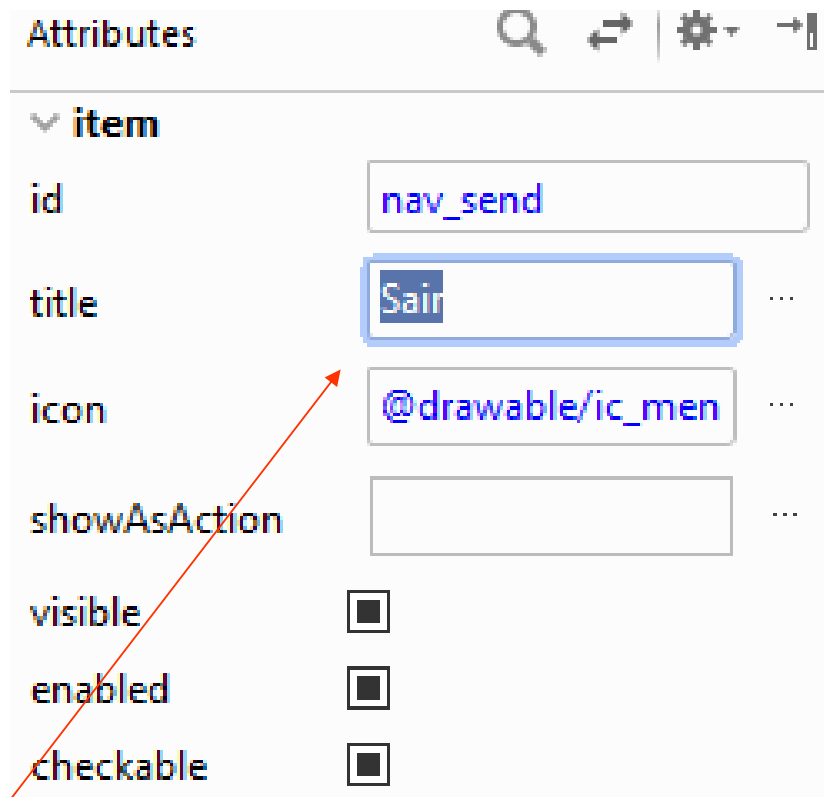
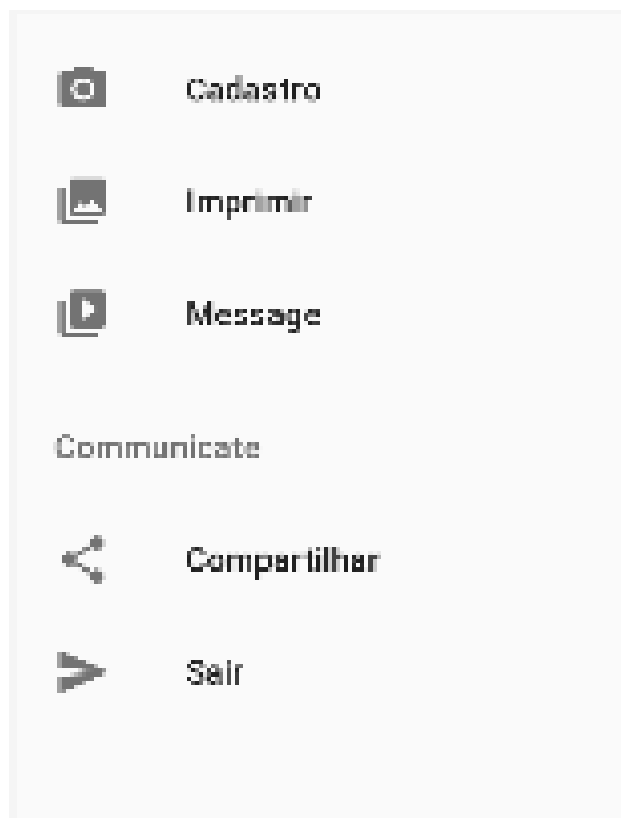
→ Fazer a alteração do menu principal



Trabalhando o projeto



Trabalhando o projeto



→ Para cada item troque o título

Trabalhando o projeto

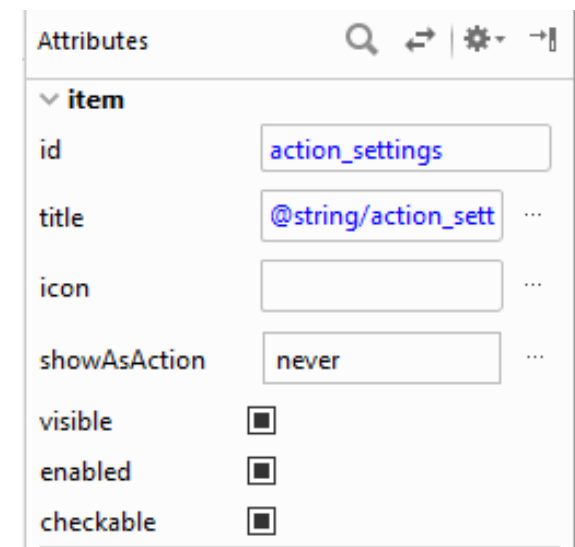
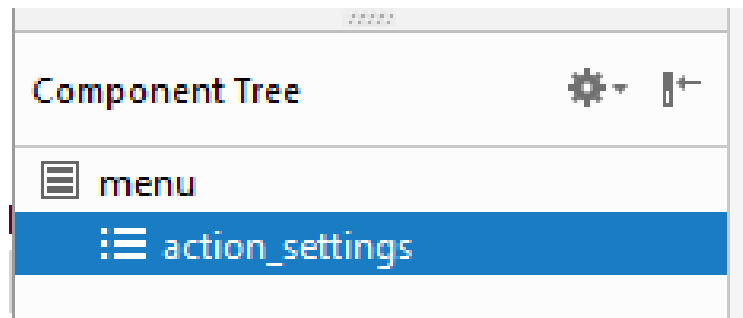
→ Faça o mesmo processo para alterar os ícones das opções do menu

The diagram illustrates the process of changing menu icons. On the left, a menu is shown with the following items: Cadastro, Imprimir, Message, Communicate, Compartilhar, and Sair. On the right, the 'Attributes' panel for an 'item' is displayed. A red arrow points from the 'Communicate' item in the menu to the 'icon' attribute in the panel. The 'icon' attribute is currently set to '@drawable/ic_men'.

Attributes	
▼ item	
id	nav_camera
title	Cadastro ...
icon	@drawable/ic_men ...
showAsAction	...
visible	<input type="checkbox"/>
enabled	<input type="checkbox"/>
checkable	<input type="checkbox"/>

Trabalhando o projeto

- Trocando no menu Setting para Sair do sistema, fixando um ícone “Exit”
- Selecionar o layout do main.xml na pasta de Menu da classe R



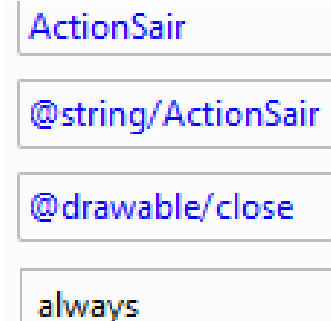
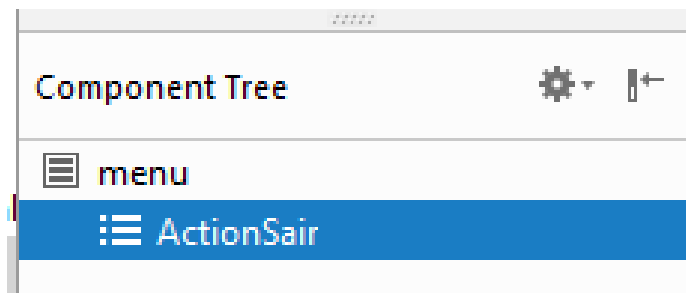
Trabalhando o projeto

→ Troque o nome do id da action para “ActionSair”

→ Alterar o ícone com o ícone de “close”

→ Mude o título para “Sair Sistema”

→ No item showAsAction mude para “Always”



Trabalhando o projeto

Agora na parte da “logica” do programa, temos que mudar os itens do menu principal para que quando o usuário clicar, o sistema atenda a solicitação.

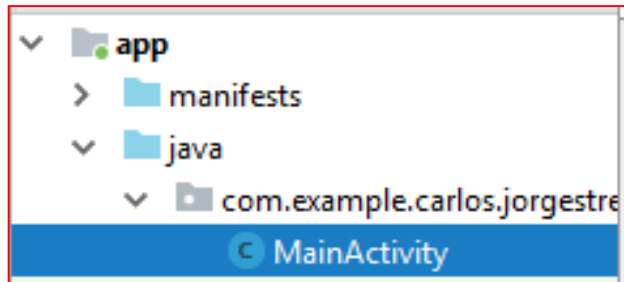
→ Selecione a pasta “java” do projeto

→ Selecione a classe MainActivity

→ Vamos alterar o método `onNavigationItemSelectedListenerSeletecd(MenuItem item) {...}`

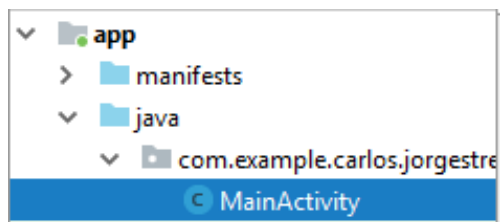
→ Mudar a identificação da classe R

Trabalhando o projeto



```
public boolean onNavigationItemSelected(MenuItem item) {  
    // Handle navigation view item clicks here.  
    int id = item.getItemId();  
  
    if (id == R.id.nav_camera) {  
        // Handle the camera action  
    } else if (id == R.id.nav_gallery) {  
  
    } else if (id == R.id.nav_slideshow) {  
  
    } else if (id == R.id.nav_manage) {  
  
    } else if (id == R.id.nav_share) {  
  
    } else if (id == R.id.nav_send) {  
  
    }  
}
```

Trabalhando o projeto



```
public boolean onNavigationItemSelected(MenuItem item) {  
    // Handle navigation view item clicks here.  
    int id = item.getItemId();  
  
    if (id == R.id.nav_cadastro) {  
        // Handle the camera action  
    } else if (id == R.id.nav_impressao) {  
  
    } else if (id == R.id.nav_message) {  
  
    } else if (id == R.id.nav_compartilhar) {  
  
    } else if (id == R.id.nav_sair) {  
  
    }  
}
```

Trabalhando o projeto

Bom, agora que modificamos todos os ícones e também modificamos o menu principal, temos que criar uma nova activity que vai ser a ponte entre o menu principal e a atividade de inserção de dados.

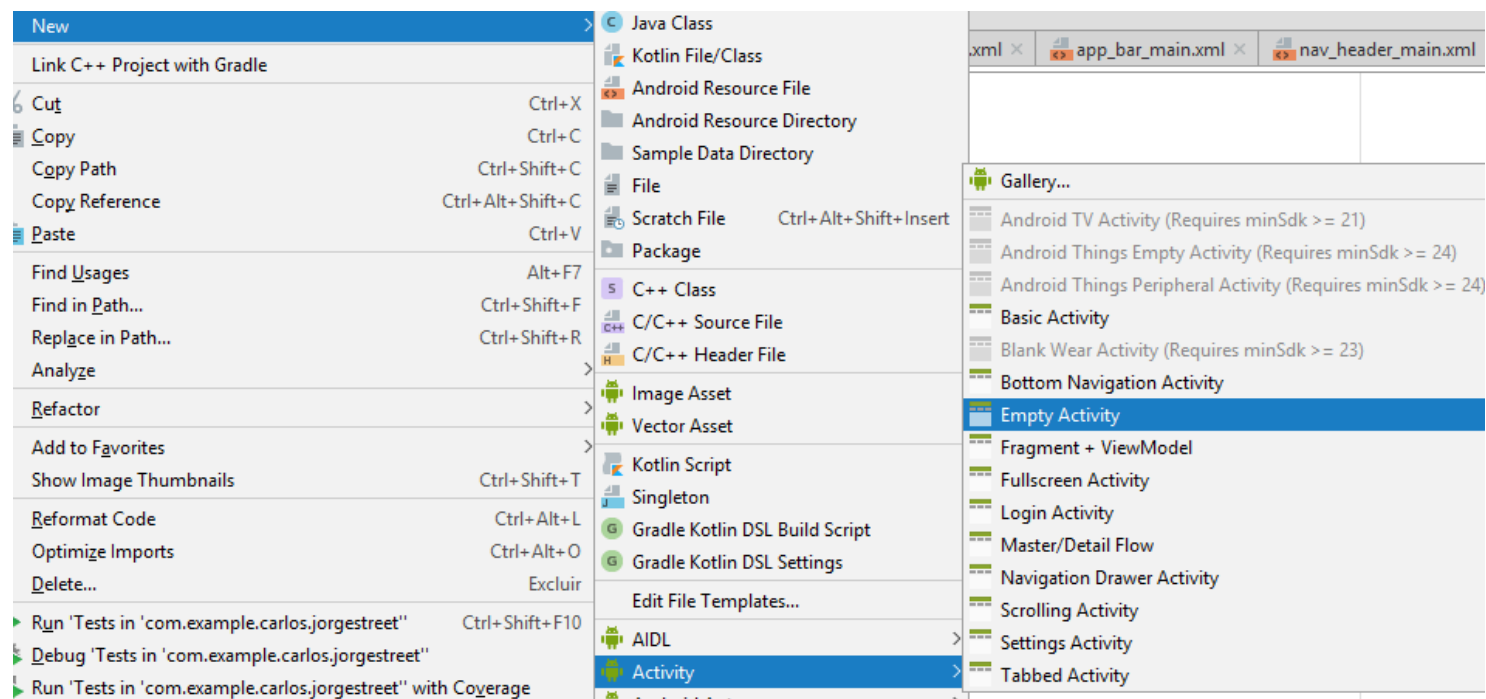
Essa nova activity terá uma figura (logotipo do projeto) e dois botões:

→ Inserir Dados

→ Consultar Dados

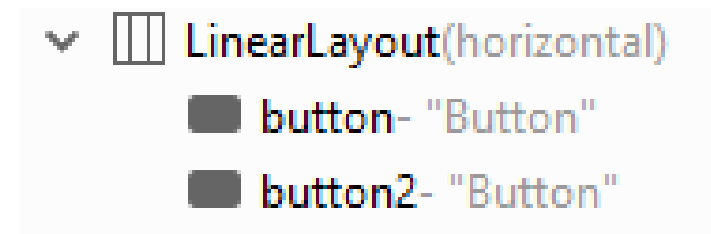
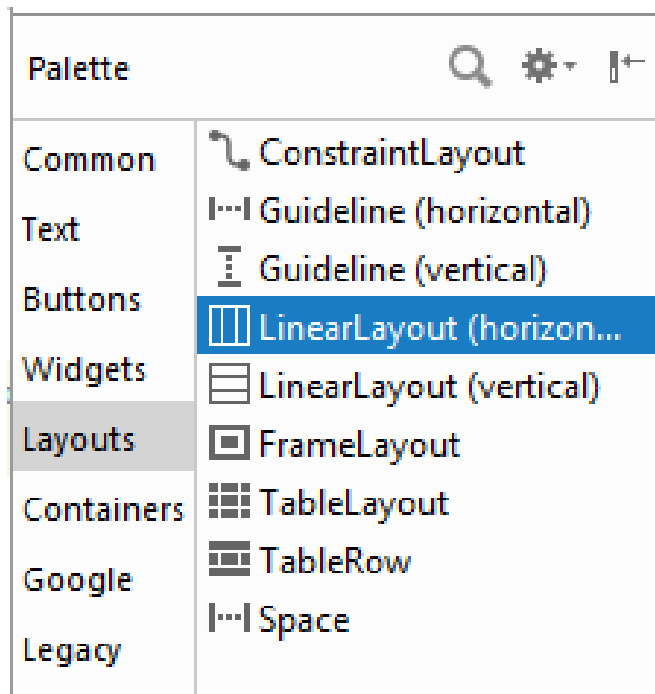
Trabalhando o projeto

- Selecione a pasta “java”, onde se encontra a classe principal “MainActivity”
- Clique com botão direito: **New / Activity / Empty Activity**



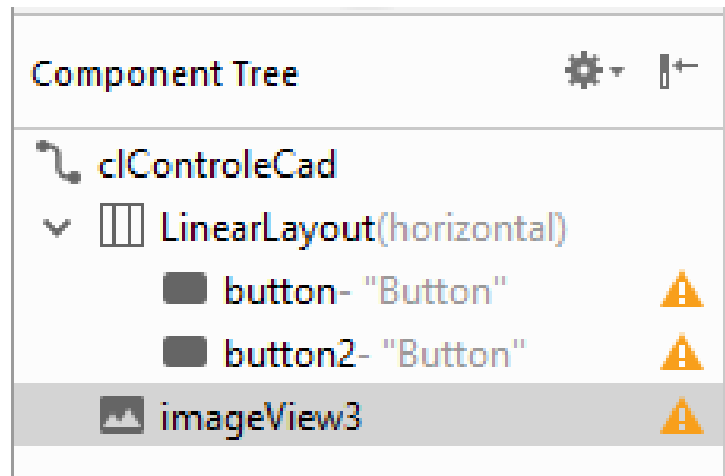
Trabalhando o projeto

- Após a criação da activity, selecione o layout dessa activity “activity_controle”
- Selecione na aba Palette em layout o LinearLayout, para podermos colocar os dois botões (Inserir e Consultar)



Trabalhando o projeto

→ Selecione na aba Palette Common / ImageView e insira o componente fora do linearLayout e muda par o logotipo do projeto.



Trabalhando o projeto

Agora, vamos alterar a parte “logica” do menu e chamar a tela “activity” criada.

→ Primeiro vamos mudar um pouco o código do método
“onNavigationItemSelectedListener(Menuitem item) { ... }, modificando os if ... else if ...,
para switch ... case.

Essa alteração faz com que o aplicativo ganhe em performance.

Trabalhando o projeto

```
@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    if (id == R.id.nav_cadastro) {

    } else if (id == R.id.nav_impressao) {

    } else if (id == R.id.nav_message) {

    } else if (id == R.id.nav_compartilhar) {

    } else if (id == R.id.nav_sair) {

    }
}
```

```
@SuppressWarnings("StatementWithEmptyBody")
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    // Handle navigation view item clicks here.
    int id = item.getItemId();

    switch ( id ) {

        case R.id.nav_cadastro:
            break;

        case R.id.nav_impressao:
            break;

        case R.id.nav_message:
            break;

        case R.id.nav_compartilhar:
            break;

        case R.id.nav_sair:
            break;

    }
}
```

Trabalhando o projeto

→ Agora, vamos modificar o código para chamar a activity criada.

```
switch ( id ) {  
  
    case R.id.nav_cadastro:  
        Intent intent = new Intent( packageContext: this, ControleActivity.class);  
        startActivity( intent );  
        break;  
}
```

Trabalhando o projeto

- Vamos colocar na activity controle um botão de back – voltar
- Selecione na classe java a activity e no método onCreate fazem a modificação abaixo

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

    // Adicionando um botão "up navigation - voltar"
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
    int id = menuItem.getItemId();

    if (id == android.R.id.home){
        // O metodo finish() vai encerrar essa activity
        finish();
        return true;
    }

    return super.onOptionsItemSelected(menuItem);
}
```

Trabalhando o projeto

- Vamos colocar na activity controle um botão de back – voltar
- Selecione na classe java a activity e no método onCreate fazem a modificação abaixo

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

    // Adicionando um botão "up navigation - voltar"
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
}

@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
    int id = menuItem.getItemId();

    if (id == android.R.id.home){
        // O metodo finish() vai encerrar essa activity
        finish();
        return true;
    }

    return super.onOptionsItemSelected(menuItem);
}
```

Trabalhando o projeto

→ Vamos agora, trabalhar a tela principal nos botões modificados anteriormente

→ Deletar o botão floating

→ Apagar do código MainActivity o trecho do botão floating no método onCreate()

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
        this, drawer, toolbar, R.string.navigation_drawer_open, R.string.navigation_drawer_close) {
        @Override
        public void onDrawerClosed() {
            // ...
        }
        @Override
        public void onDrawerOpened() {
            // ...
        }
    };
    drawer.addDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = (NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}
```

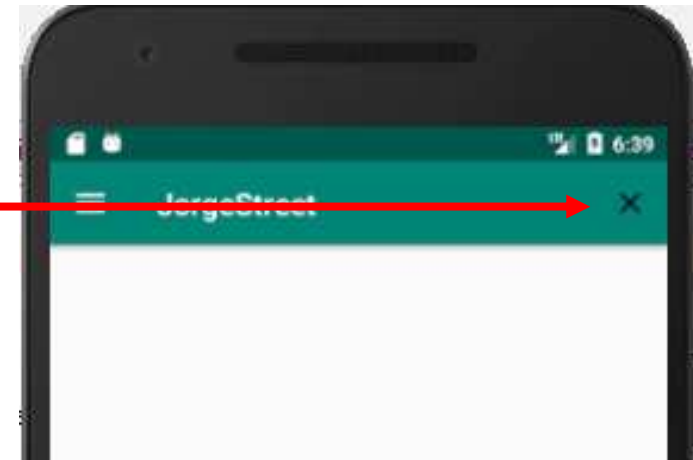

Trabalhando o projeto

→ Modificar o código do botão fechar

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action b
    // automatically handle clicks on the Home/Up butto
    // as you specify a parent activity in AndroidManif
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.ActionSair) {
        finish();
        return true;
    }

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



Trabalhando o projeto

→ Modificar no código da activity MainActivity na opção menu a opção sair.

```
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle navigation view item clicks here
    int id = item.getItemId();

    switch ( id ) {

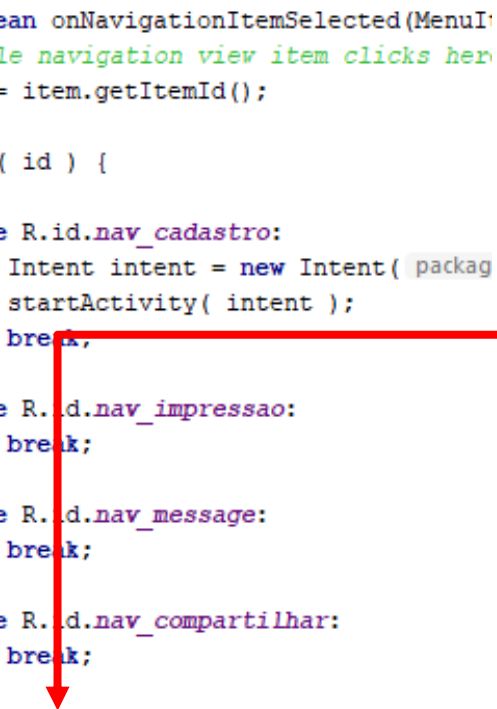
        case R.id.nav_cadastro:
            Intent intent = new Intent( package
            startActivity( intent );
            break;

        case R.id.nav_impressao:
            break;

        case R.id.nav_message:
            break;

        case R.id.nav_compartilhar:
            break;

        case R.id.nav_sair:
            finish();
            break;
    }
}
```



Trabalhando o projeto

Criando a tela de Cadastro

Agora que fizemos algumas modificações no projeto, podemos pensar na tela do cadastro.

- Selecione a pasta java e com botão direito crie uma nova activity vazia
- Dê o nome de CRUDActivity (4 operações do banco de dados)
- Monte o layout conforme abaixo:

Trabalhando o projeto

UF = string-array

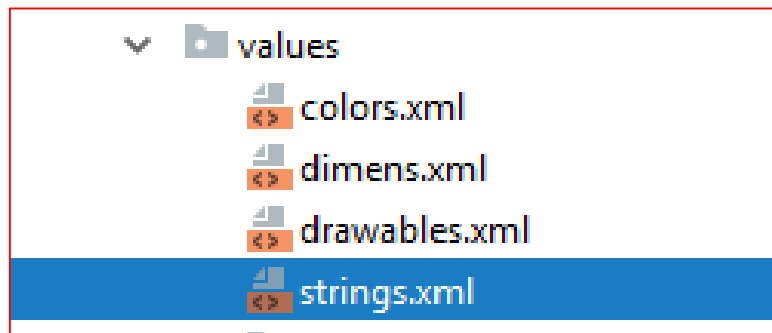
RadioGroup com
RadioButton

LinearLayout Horizontal

The screenshot shows a mobile application interface titled "JorgeStreet". It features a form with the following fields: "Nome", "CEP", "Endereço", "Bairro", "Cidade", "UF", "Telefone", "Celular", "E-mail", "Latitude", and "Longitude". The "UF" field is highlighted with a red arrow pointing to it from the text "UF = string-array". The "Telefone" and "Celular" fields are grouped together, with a red arrow pointing to them from the text "RadioGroup com RadionButton". The "E-mail" field is highlighted with a red arrow pointing to it from the text "RadioGroup com RadionButton". The "Latitude" and "Longitude" fields are grouped together, with a red arrow pointing to them from the text "RadioGroup com RadionButton". At the bottom of the form, there are three buttons: "VOLTAR", "EXCLUIR", and "GRAVAR", which are part of a horizontal LinearLayout, as indicated by the red arrow from the text "LinearLayout Horizontal". The status bar at the top shows the time as 8:31.

Trabalhando o projeto

O uso do componente “Spinner” para guardar os estados, estes por sua vez devem ser armazenados em um estados.xml dentro da pasta res / values:



```
<string name="edtCid" />  
<string name="txvUF">UF</string>
```

```
<string-array name="estados">  
    <item> AC </item>  
    <item> BA </item>  
    <item> CE </item>  
    <item> DF </item>  
    <item> ES </item>  
    <item> RJ </item>  
    <item> SP </item>  
    <item> TO </item>  
</string-array>
```

```
<string name="rbMasc">Masculino</string>  
<string name="rbFemi">Feminino</string>
```

Trabalhando o projeto

Associar o vetor “array” com o componente

→ Selecione o componente no layout



→ Em atributos “entries” / “array”



→ Escolha a variável criada e tecle “ok”

Trabalhando o projeto

Alguns pontos a se considerar aqui são o uso de um `RadioGroup` por fora dos `RadioButtons`, para garantir que apenas um deles seja selecionável.

```
<RadioGroup
    android:id="@+id/rgSexo"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:checkedButton="@+id/rbMasc"
    android:orientation="horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/spUF"
    app:layout_constraintTop_toBottomOf="@+id/edtCid">

    <RadioButton
        android:id="@+id/rbMasc"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Masculino" />

    <RadioButton
        android:id="@+id/rbFemi"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Feminino" />

</RadioGroup>
```

Trabalhando Atividade CRUD

- Selecione a pasta java
- Clique na activity “CRUDActivity”
para que possamos recuperar os
componentes do layout
- Vamos criar os atributos da classe

```
// atributos da classe CRUD
private EditText edtNome;
private EditText edtCEP;
private EditText edtEnde;
private EditText edtBai;
private EditText edtCid;
private Spinner spUF;
private RadioButton rbMasc;
private RadioButton rbFemi;
private EditText edtTel;
private EditText edtCel;
private EditText edtEmail;
private EditText edtLatitude;
private EditText edtLongitude;

private Button btnVoltar;
private Button btnExcluir;
private Button btnGravar;
```


Trabalhando Atividade CRUD

→ No método onCreate dessa activity, vamos criar um método para que possa recuperar os componentes.

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

    // metodo para recuperar os componentes do layout
    recupera( crudActivity: this);
}
```

```
// metodo para recuperar os componentes do layout CRUD
private void recupera(CRUDActivity crudActivity) {

    edtNome = findViewById(R.id.edtNome);
    edtCEP = findViewById(R.id.edtCep);
    edtEnde = findViewById(R.id.edtEnde);
    edtBai = findViewById(R.id.edtBai);
    edtCid = findViewById(R.id.edtCid);

    spUF = findViewById(R.id.spUF);
    rbMasc = findViewById(R.id.rbMasc);
    rbFemi = findViewById(R.id.rbFemi);

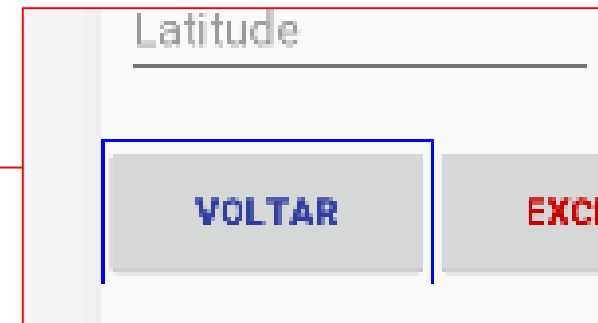
    edtTel = findViewById(R.id.edtTel);
    edtCel = findViewById(R.id.edtCel);
    edtEmail = findViewById(R.id.edtEmail);
    edtLatitude = findViewById(R.id.edtLatitude);
    edtLongitude = findViewById(R.id.edtLongitude);

    btnVoltar = findViewById(R.id.btnVoltar);
    btnExcluir = findViewById(R.id.btnExcluir);
    btnGravar = findViewById(R.id.btnGravar);
}
```

Trabalhando Atividade CRUD

- Vamos agora criar o código para o botão “**Voltar**”
- Criar um método “onClickVoltar” e associar esse método com o click do botão no arquivo layout.

```
// metodo onClickVoltar -> click do botao
public void onClickVoltar(View view) {
    // encerra a activity
    finish();
}
```



Trabalhando Atividade CRUD

- Vamos adicionar um botão “up_navigation”, para que o usuário possa clicar e também fechar a activity.
- Primeiro inserir o código abaixo no método “onCreate” da activity CRUDActivity
- Essa linha de comando dispara um método “onOptionsItemSelected” de menu da activity

```
// Adicionando um botão "up_navigation - voltar"  
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Trabalhando Atividade CRUD

→ Agora, vamos criar o método onOptionsItemSelected(...)

```
@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
    int id = menuItem.getItemId();

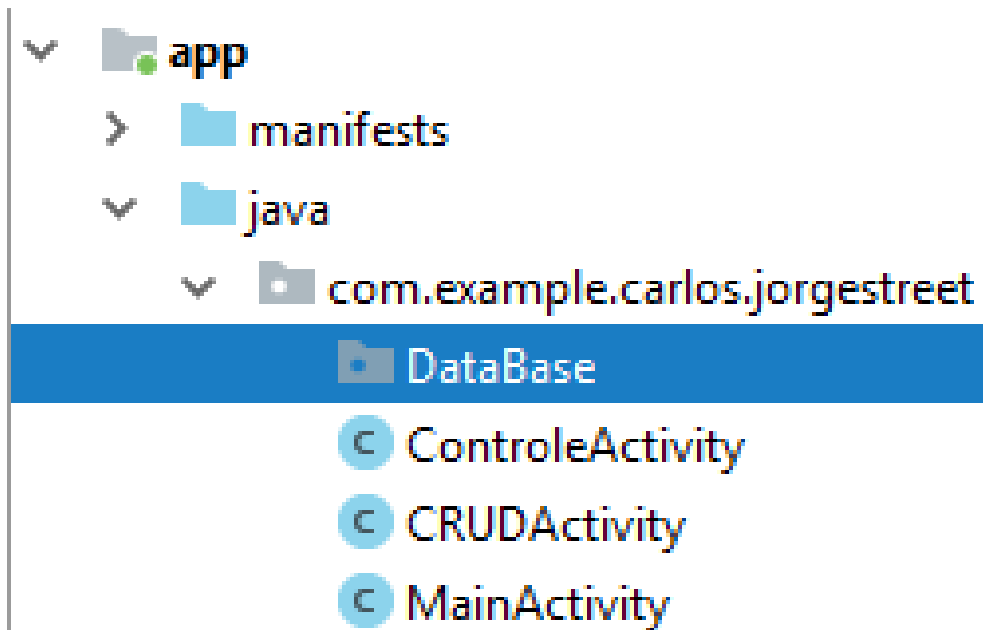
    if (id == android.R.id.home) {
        // O metodo finish() vai encerrar essa activity
        finish();
        return true;
    }

    return super.onOptionsItemSelected(menuItem);
}
```



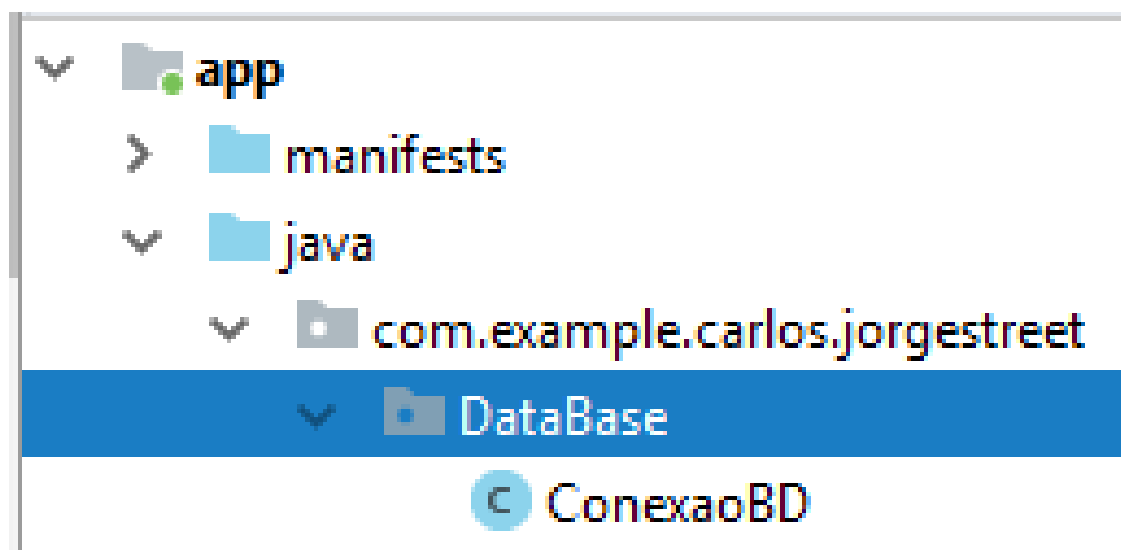
Banco de dados

- Criando a classe de conexão com o banco de dados
- Vamos criar uma pasta “package” em nosso projeto chamando de “DataBase”



Banco de dados

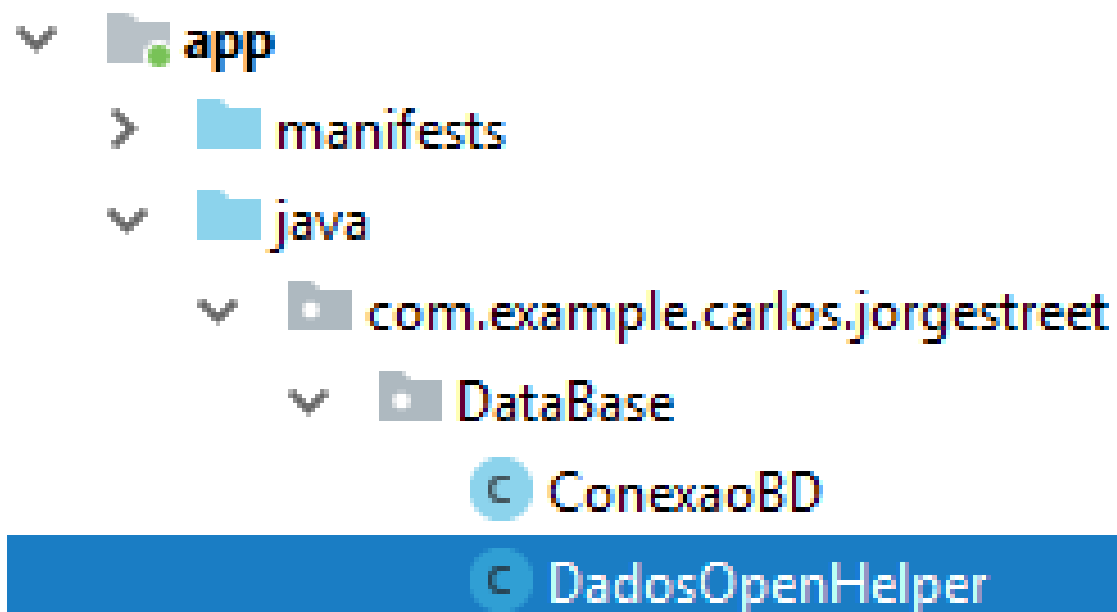
- Vamos criar a classe ConexaoBD() dentro do package DataBase
- Clique no pacote e com o botão direito
- New / Java Class



Banco de dados

→ Vamos criar a classe
DadosOpenHelper() extends
a SQLiteOpenHelper

→ Essa classe é responsável
em criar o banco de dados e
sua versão e o método que
executa as DDL das tabelas

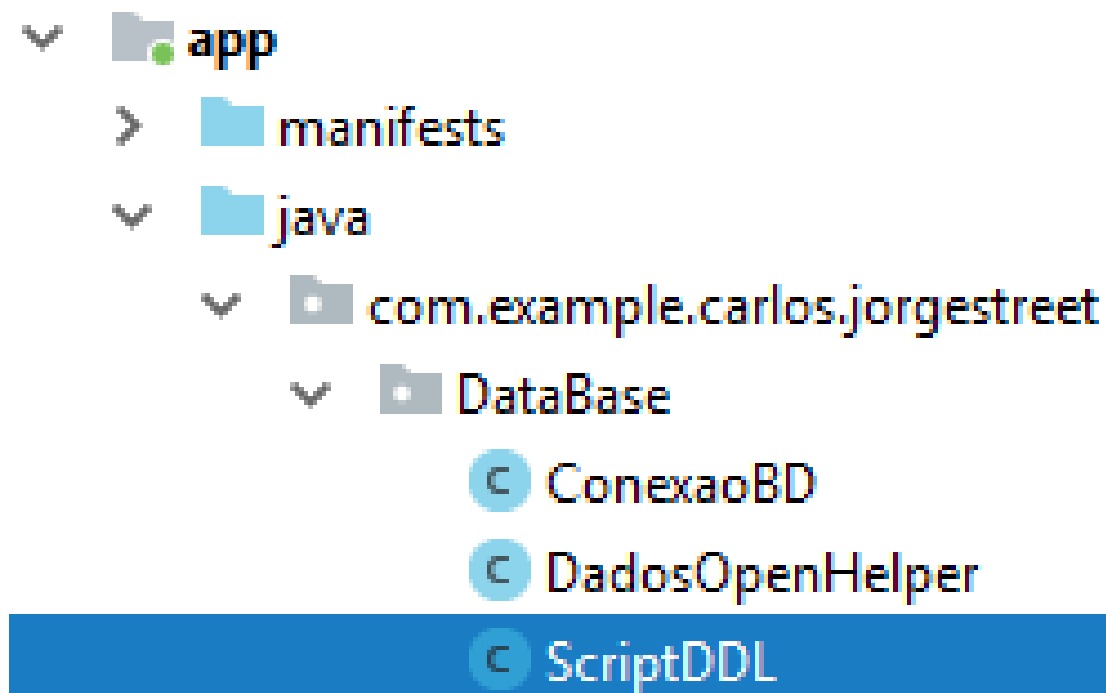


Banco de dados

→ Vamos criar a classe
ScriptDDL()

→ Essa classe é responsável
em criar o script das tabelas

→ Método para Create table



Banco de dados - ConexaoBD

```
public class ConexaoBD {  
  
    // objeto que representa a conexao com o banco de dados (context)  
    private SQLiteDatabase conexao;  
  
    // responsavel com a conexao e criação da tabela se nao existir com o banco de dados  
    private DadosOpenHelper dadosOpenHelper;  
  
    public SQLiteDatabase criarConexao(Context context, ConstraintLayout constraintLayout) {...}  
  
}
```

Banco de dados - ConexaoBD

```
// usar um try - catch por se trabalhar com banco de dados
// pode haver um erro na conexao
try {
    // passa o contexto (Nome do banco de dados e versao atual do banco)
    dadosOpenHelper = new DadosOpenHelper( context );


    // faz a conexao com o banco de dados com parametro de escrita
    conexao = dadosOpenHelper.getWritableDatabase();

    // Mensagem para o usuário, forma parecida do Toast
    Snackbar.make(constraintLayout, text: "Conexão efetuada com sucesso", Snackbar.LENGTH_SHORT)
        .setAction( text: "OK", listener: null ).show();

    return conexao;
} catch (SQLException ex) {
    // Mensagem de erro de conexão
    AlertDialog.Builder dlg = new AlertDialog.Builder( context );
    dlg.setTitle("Aviso de erro");
    dlg.setMessage(ex.getMessage());
    dlg.setNeutralButton( text: "OK", listener: null );
    dlg.show();

    return null;
}
```

Banco de dados - DadosOpenHelper

```
public class DadosOpenHelper extends SQLiteOpenHelper {  
  
     // construtor da classe  
    public DadosOpenHelper(Context context) {...}  
  
    // metodo para criar as tabelas do banco de dados  
    @Override  
    public void onCreate(SQLiteDatabase db) {...}  
  
    // metodo responsavel para atualizar o banco de dados  
    // fazer um alter table  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {}  
  
}
```

Banco de dados - DadosOpenHelper

```
// construtor da classe
public DadosOpenHelper(Context context) {
    // nome do banco dados
    super(context, name: "dbjorgeStreet", factory: null, version: 1);
}

// metodo para criar as tabelas do banco de dados
@Override
public void onCreate(SQLiteDatabase db) {

    /* chamando o metodo da classe ScriptDDL */
    db.execSQL( ScriptDDL.getCreateTableCadastro() );
}
```

Banco de dados - ScriptDDL

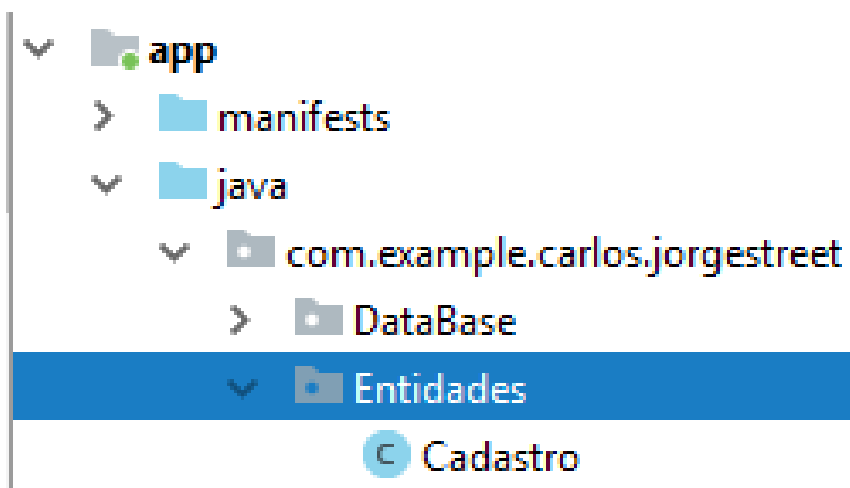
```
public class ScriptDDL {  
  
    public static String getCreateTableCadastro() {...}  
  
}
```

Banco de dados - ScriptDDL

```
public static String getCreateTableCadastro() {  
  
    // criar uma string atraves do objeto para retornar do metodo  
    // aqui você pode criar todas as tabelas do seu projeto  
    StringBuilder sql = new StringBuilder();  
  
    // preparando a string  
    sql.append("CREATE TABLE IF NOT EXISTS CADASTRO (");  
    sql.append("    CODIGO INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,");  
    sql.append("    NOME VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    CEP VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    ENDereco VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    BAIRRO VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    CIDADE VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    ESTADO VARCHAR(200) NOT NULL DEFAULT (''),");  
    sql.append("    SEXO VARCHAR(200) NOT NULL DEFAULT (''),");  
    sql.append("    TELEFONE VARCHAR(200) NOT NULL DEFAULT (''),");  
    sql.append("    CELULAR VARCHAR(200) NOT NULL DEFAULT (''),");  
    sql.append("    EMAIL VARCHAR(250) NOT NULL DEFAULT (''),");  
    sql.append("    LATITUDE VARCHAR(200) NOT NULL DEFAULT (''),");  
    sql.append("    LONGITUDE VARCHAR(200) NOT NULL DEFAULT (''))");  
  
    return sql.toString();  
}
```

Banco de dados - Entidades

- Vamos criar um outro pacote “package” de nome “Entidades” em nosso projeto
- Esse pacote vai conter a classe “Cadastro”
- Essa interface permite passar como parâmetros os atributos como objeto da classe que vai possibilitar a alteração do registro

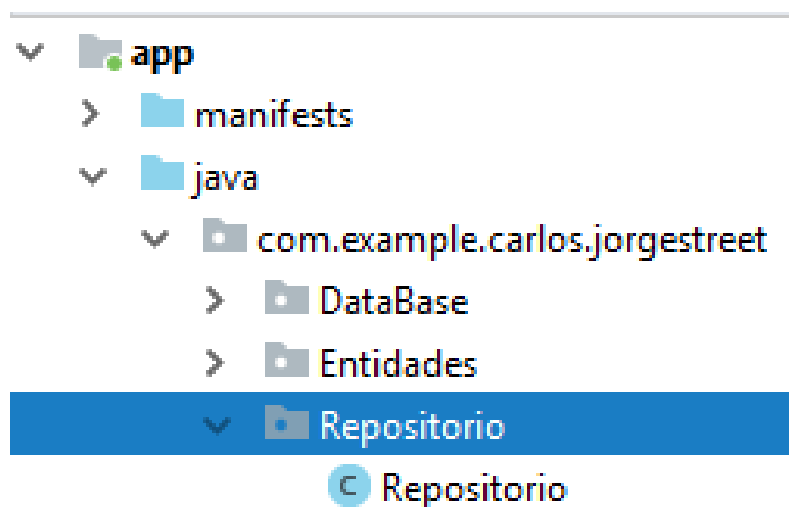


Banco de dados - Entidades

```
public class Cadastro {  
  
    public int CODIGO;  
    public String NOME;  
    public String CEP;  
    public String ENDereco;  
    public String BAIRRO;  
    public String CIDADE;  
    public String ESTADO;  
    public String SEXO;  
    public String TELEFONE;  
    public String CELULAR;  
    public String EMAIL;  
    public double LATITUDE;  
    public double LONGITUDE;  
  
    // iniciando o valor 0 para o atributo codigo  
    public Cadastro() {  
        this.CODIGO = 0;  
    }  
  
}
```


Banco de dados – Repositório

- Vamos criar outro pacote “package” de nome “Repositório”
- Esse pacote vai conter a classe “CadastroRepositório”, por se tratar a entidade Cadastro.
- Essa classe é responsável pelas 4 operações com o banco de dados (insert, update, delete e select), ela vai interagir com a classe criada anteriormente.



Banco de dados – Repositório

```
public class CadastroRepositorio {  
  
    // atributo da classe  
    private SQLiteDatabase conexao;  
  
    // construtor da classe CadastroRepositorio  
    public CadastroRepositorio(SQLiteDatabase conexao) { this.conexao = conexao; }  
  
    // Método de insert na tabela cadastro do banco de dados "jorgeStreet"  
    public void inserir(Cadastro cadastro) {...}  
  
    // Método de excluir registro da tabela cadastro do banco de dados "jorgeStreet"  
    public void excluir(int codigo) {...}  
  
    // Método de alterar registro da tabela cadastro do banco de dados "jorgeStreet"  
    public void alterar(Cadastro cadastro) {...}  
  
    // Método que faz uma busca na tabela de todos os registros  
    public List<Cadastro> buscarTodos() {...}  
  
    // Método que faz uma busca especifica na tabela  
    public Cadastro buscarCliente(int codigo) {...}  
}
```

Banco de dados – Repositório

```
// atributo da classe
private SQLiteDatabase conexao;

// construtor da classe ClienteRepositorio
public CadastroRepositorio(SQLiteDatabase conexao) {
    this.conexao = conexao;
}
```

Banco de dados – Repositório

```
// Método de insert na tabela cadastro do banco de dados "jorgeStreet"
public void inserir(Cadastro cadastro) {

    ContentValues contentValues = new ContentValues();

    contentValues.put("NOME", cadastro.NOME);
    contentValues.put("CEP", cadastro.CEP);
    contentValues.put("ENDERECO", cadastro.ENDERECO);
    contentValues.put("BAIRRO", cadastro.BAIRRO);
    contentValues.put("CIDADE", cadastro.CIDADE);
    contentValues.put("ESTADO", cadastro.ESTADO);
    contentValues.put("SEXO", cadastro.SEXO);
    contentValues.put("TELEFONE", cadastro.TELEFONE);
    contentValues.put("CELULAR", cadastro.CELULAR);
    contentValues.put("EMAIL", cadastro.EMAIL);
    contentValues.put("LATITUDE", cadastro.LATITUDE);
    contentValues.put("LONGITUDE", cadastro.LONGITUDE);
    conexao.insertOrThrow( table: "CADASTRO", nullColumnHack: null, contentValues );

}
```

Trabalhando o projeto

- Vamos trabalhar a “lógica” da ação de incluir registros em nosso banco de dados
- Para auxiliar no tratamento dos dados digitados pelo usuário, vamos criar dois métodos, um para validar campo a campo e o outro para verificar se está vazio determinado campo, ou seja, em nosso banco de dados não vai conter registro com “colunas” sem informação.

Trabalhando o projeto

- Vamos criar um objeto “ConstraintLayout” para poder utilizar o “Snacker” na classe “ConexaoBD” como uma mensagem para usuário, que a conexão foi efetuada com sucesso
- Também, vamos criar o objeto do “CadastroRepositorio” para criar a conexão com o banco de dados.

```
// Entidade, criando um objeto para manipular o registro  
private Cadastro cadastro = new Cadastro();  
private ConexaoBD conexaoBD = new ConexaoBD();  
private CadastroRepositorio cadastroRepositorio;
```

```
// ConstraintLayout objeto criado para passar como parametro  
private ConstraintLayout ctlCRUD;
```

```
// variaveis primitivas  
private String dadosCad[] = new String[12];
```

Trabalhando o projeto

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

    // metodo para recuperar os componentes do layout
    recupera( crudActivity: this);

    // Adicionando um botão "up_navigation - voltar"
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    // recuperado o layout para que possa utilizar a classe Snacker
    ctlCRUD = (ConstraintLayout) findViewById(R.id.ctlCRUD);

    // cria conexao com o banco de dados
    cadastroRepositorio = new CadastroRepositorio(
        conexaoBD.criarConexao( context: CRUDActivity.this, ctlCRUD) );
}
```

Trabalhando o projeto – CRUDActivity

```
// Método onClickSalvar -> click do botão  
public void onClickSalvar(View view) {}
```

```
// Método de validação dos campos digitados pelo usuário  
private boolean ValidaCampos() {...}
```

```
// Método que verifica se o campo digitado está vazio  
private boolean isCampoVazio(String valor) {...}
```


Trabalhando o projeto – CRUDActivity

```
// Método de validação dos campos digitados pelo usuario
private boolean ValidaCampos() {

    // preenchendo o vetor com as informações digitadas pelo usuario
    dadosCad[0] = edtNome.getText().toString();
    dadosCad[1] = edtCEP.getText().toString();
    dadosCad[2] = edtEnde.getText().toString();
    dadosCad[3] = edtBai.getText().toString();
    dadosCad[4] = edtCid.getText().toString();
    dadosCad[5] = spUF.getSelectedItem().toString();
    if ( rbMasc.isChecked() ) {
        dadosCad[6] = "M";
    } else {
        dadosCad[6] = "F";
    }
    dadosCad[7] = edtEmail.getText().toString();
    dadosCad[8] = edtTel.getText().toString();
    dadosCad[9] = edtCel.getText().toString();
    dadosCad[10] = edtLatitude.getText().toString();
    dadosCad[11] = edtLongitude.getText().toString();

    // verifica campo a campo se estão todos preenchidos
    // caso encontre vazio, manda uma msg para o usuario e não deixa incluir
    for (int i = 0; i <= 11; i++){
        if ( isCampoVazio( dadosCad[i] ) ) {
            return true; // o campo está vazio
        }
    }

    return false;
}
```

Trabalhando o projeto – CRUDActivity

```
// Método que verifica se o campo digitado está vazio
private boolean isCampoVazio(String valor) {
    boolean result = (TextUtils.isEmpty(valor) || valor.trim().isEmpty());
    return result;
}
```

CRUDActivity – botão salvar – Parte 1

```
// Método onClickSalvar -> click do botao
public void onClickSalvar(View view) {

    // o método retorna true ou false, se voltar true, existem campos vazios,
    // caso contrario, está ok com os dados digitados
    if ( !ValidaCampos() ) {

        // atribuindo os dados nos atributos da classe cadastro
        cadastro.NOME      = dadosCad[0];
        cadastro.CEP       = dadosCad[1];
        cadastro.ENDERECO  = dadosCad[2];
        cadastro.BAIRRO    = dadosCad[3];
        cadastro.CIDADE    = dadosCad[4];
        cadastro.ESTADO    = dadosCad[5];
        cadastro.SEXO      = dadosCad[6];
        cadastro.TELEFONE   = dadosCad[7];
        cadastro.CELULAR   = dadosCad[8];
        cadastro.EMAIL      = dadosCad[9];
        cadastro.LATITUDE  = Double.parseDouble( dadosCad[10] );
        cadastro.LONGITUDE = Double.parseDouble( dadosCad[11] );
```

CRUDActivity – botão salvar – Parte 2

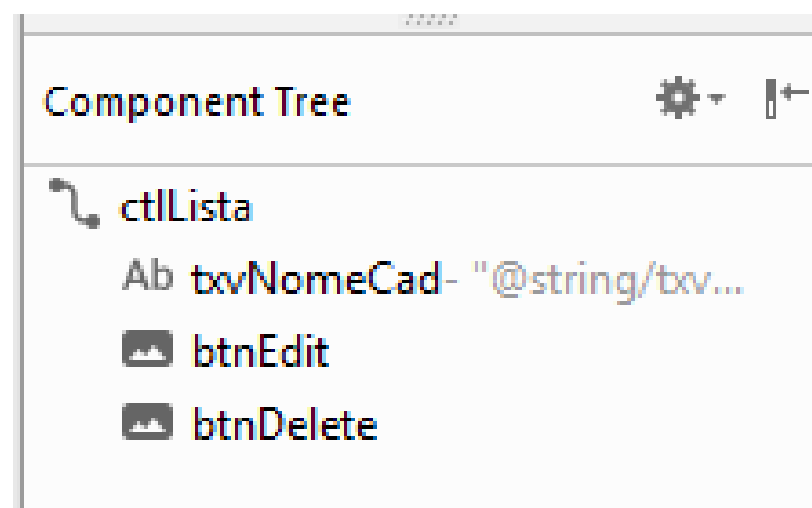
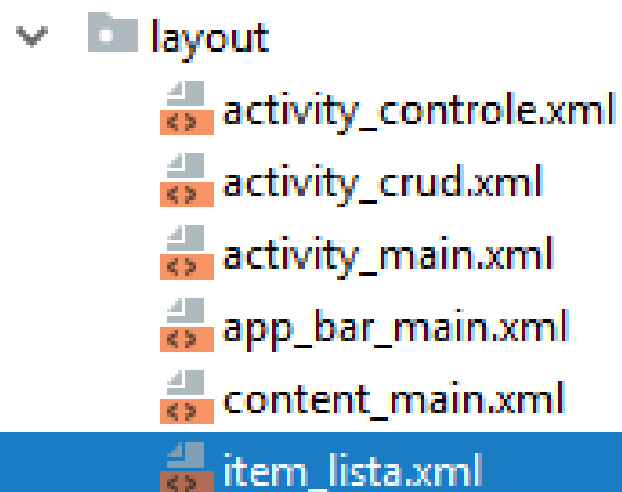
```
try {  
    if ( cadastro.CODIGO == 0 ) {  
        // chamando o metodo inserir dados no banco  
        cadastroRepositorio.inserir( cadastro );  
        // mensagem de inserção concluído com sucesso para o usuario  
        Toast.makeText( context: CRUDActivity.this, text: "Inclusão efetuada com sucesso!",  
                        Toast.LENGTH_LONG).show();  
    }  
  
} catch (SQLException ex) {  
    AlertDialog.Builder dlg = new AlertDialog.Builder( context: CRUDActivity.this );  
    dlg.setTitle("Aviso de Erro");  
    dlg.setMessage(ex.getMessage());  
    dlg.setNeutralButton( text: "OK", listener: null);  
    dlg.show();  
}  
}
```

→ Não se esqueça de associar o método criado com o click do botão no layout

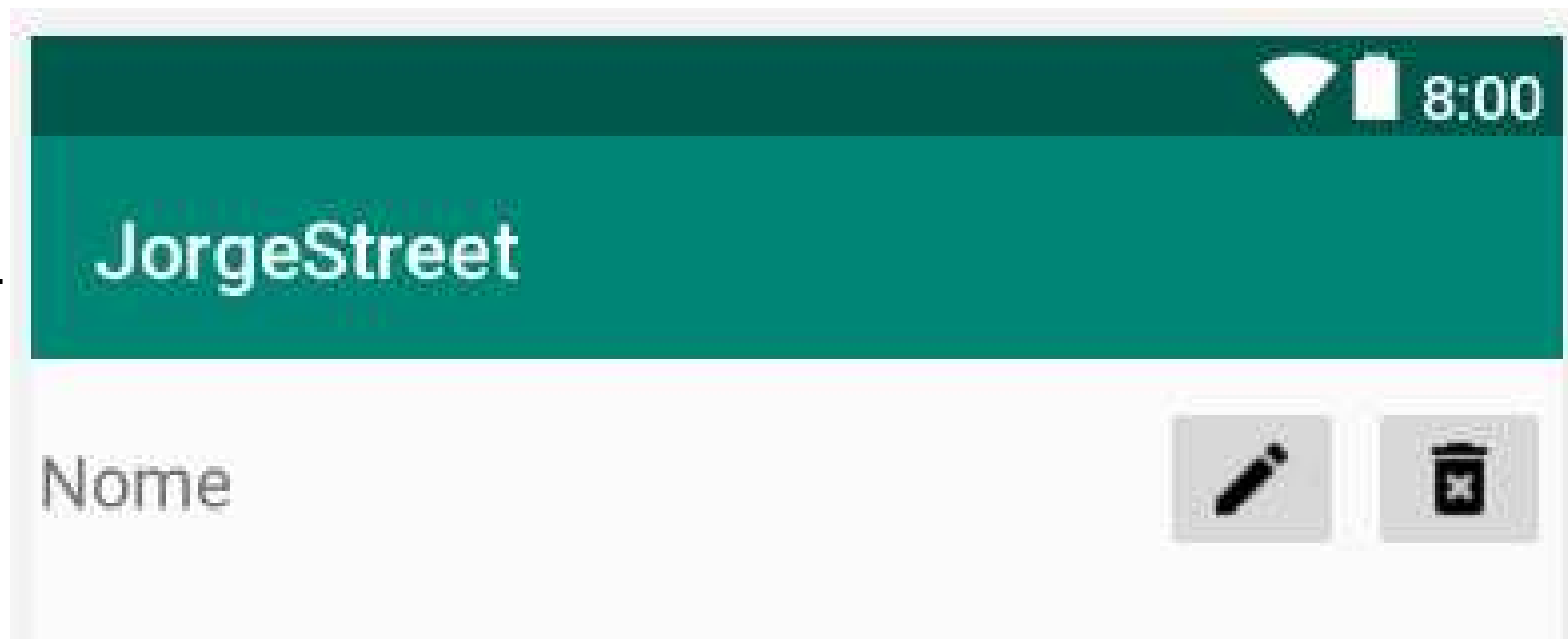
Trabalhando o projeto

- Agora, que a inclusão está funcionando adequadamente, precisamos trabalhar o botão “Consultar”, fazer uma busca de todos os registros na tabela e mostrar adequadamente.
- Primeiro vamos criar um novo layout “item_lista” contendo **TextView** e dois **ImageButton**
 - **TextView** = Nome
 - **ImageButton** = Edit (para efetuar a alteração de registro na tabela)
 - **ImageButton** = Delete (para excluir o registro da tabela)

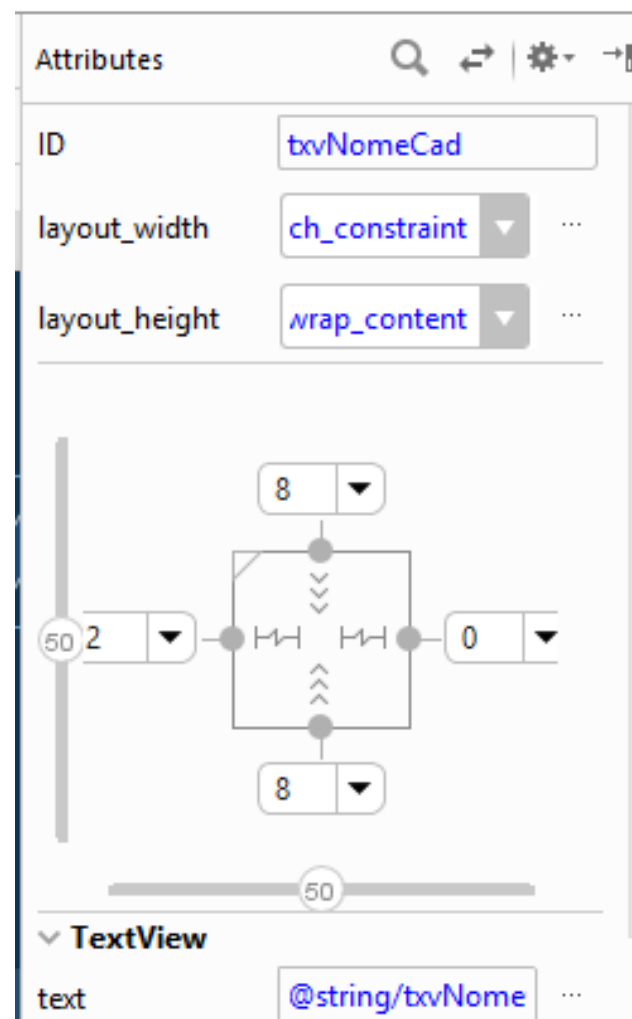
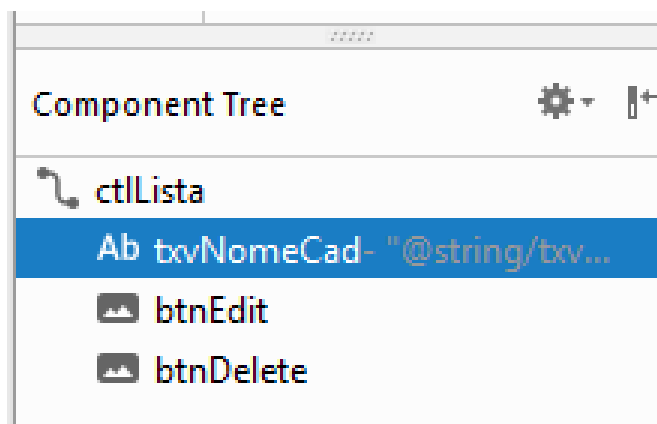
Trabalhando o projeto



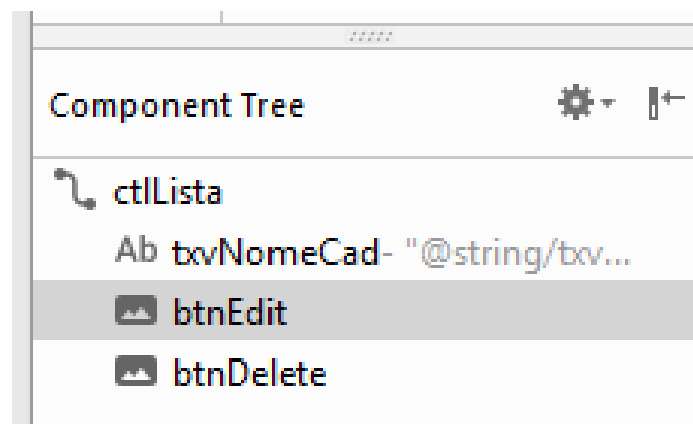
Trabalhando o projeto



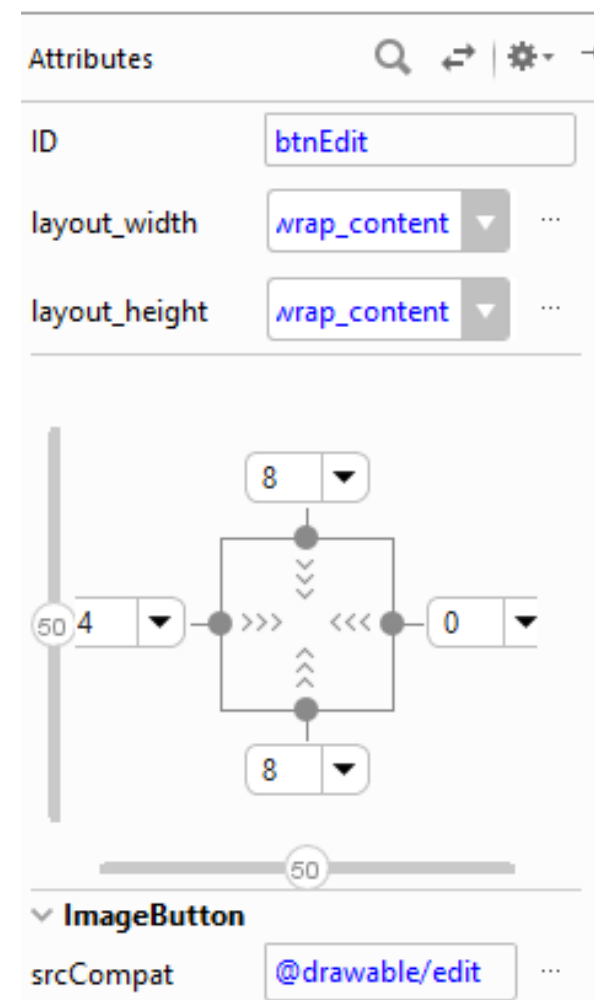
Trabalhando o projeto



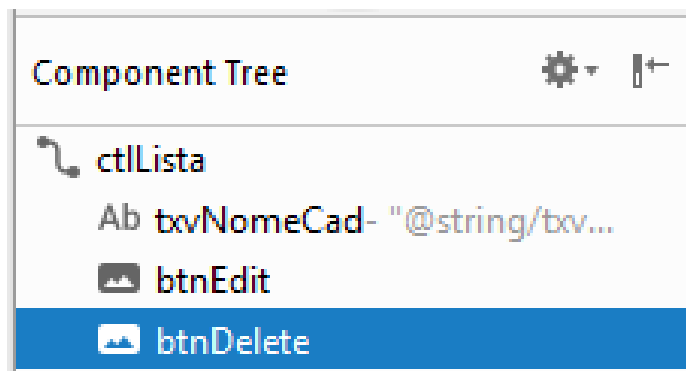
Trabalhando o projeto



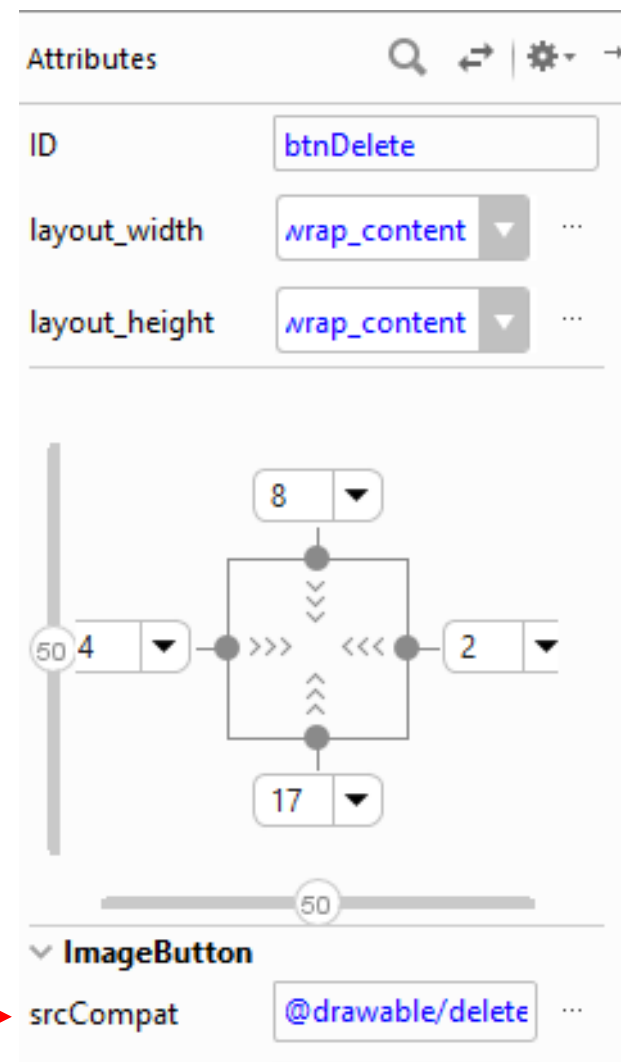
Ícone do símbolo



Trabalhando o projeto



Ícone do símbolo



Trabalhando o projeto

→ Após fazer as modificações, precisamos fazer com que para cada registro mantenha essa formação (TextView, Button, Button), para isso é necessário modificar a propriedade do ConstraintLayout.

The screenshot displays the Android Studio interface with two panels: Component Tree and Attributes.

Component Tree:

- ctLista (selected)
- Ab txvNomeCad- "@string/txv..."
- btnEdit
- btnDelete

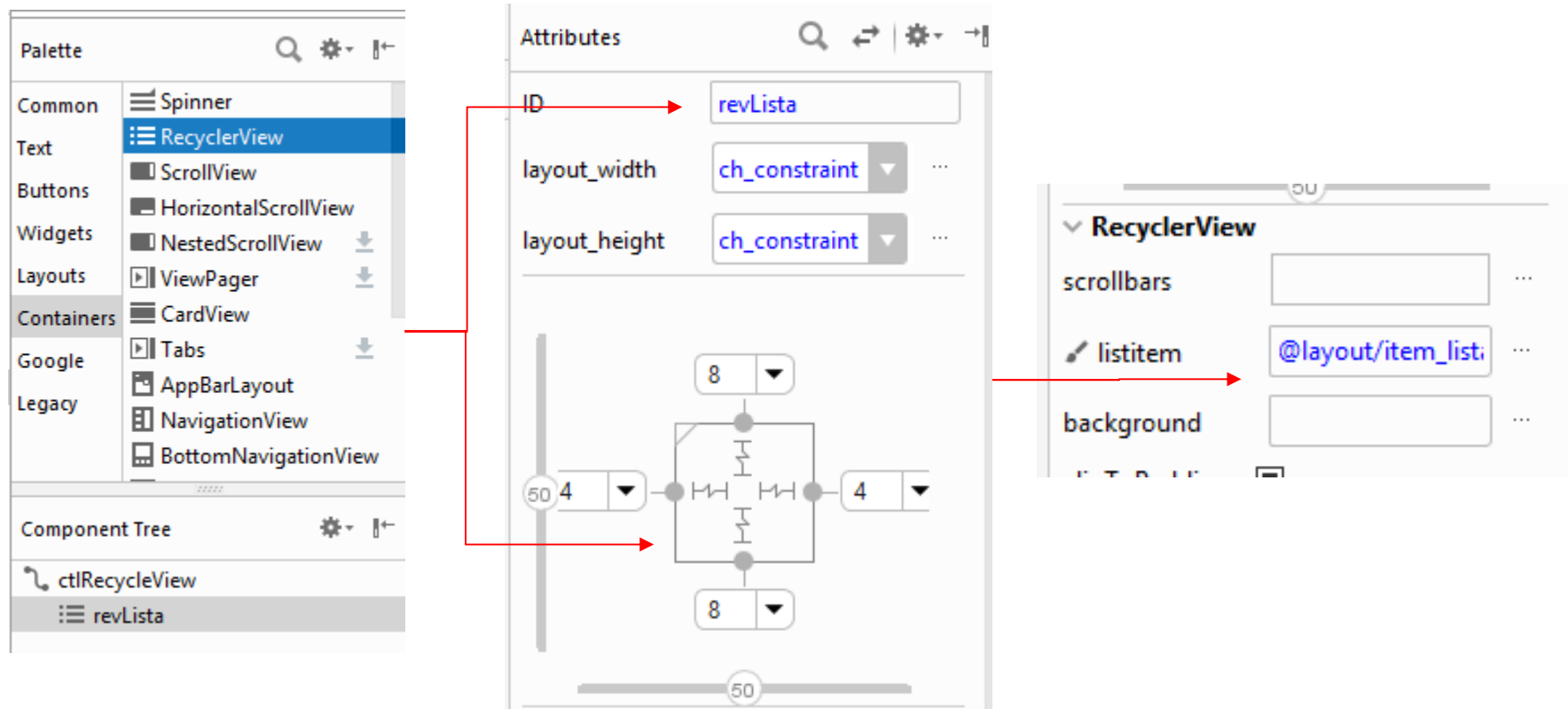
Attributes:

Attribute	Value
ID	ctLista
layout_width	match_parent
layout_height	wrap_content

A red arrow points from the **ctLista** view in the Component Tree to the **layout_height** attribute in the Attributes panel.

Trabalhando o projeto

- Vamos criar agora um nova Activity de nome “RecyclerViewActivity”
- Vamos adicionar nesse novo layout um componente RecyclerView



Trabalhando o projeto

É muito comum aplicativos terem listas para apresentarem seu conteúdo de forma eficiente. E se mal implementada pelo desenvolvedor pode trazer descontentamento para o usuário, o seu cliente final. Pensando nisso o Android nos deu um componente poderoso chamado **RecyclerView**. (Listas com RecyclerView – Android Dev BR – Medium, acesso em 07/07/2008)

RecyclerView → É o substituto do antigo ListView. Ele nada mais é que um contêiner para receber um grande conjunto de dados, mostrado ao usuário em forma de lista de itens.

Trabalhando o projeto

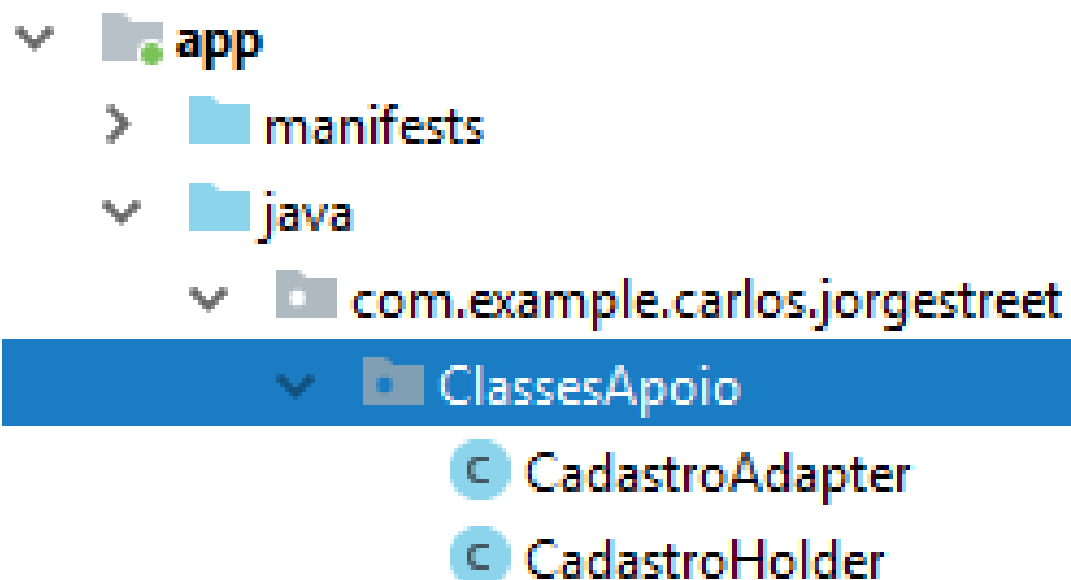
Ele possui mais eficiência que seu antecessor e segundo o próprio Google, ele simplifica a exibição e o tratamento de grande conjunto de dados.

O grande truque é que agora existe gerenciadores de layout (layout manager) que possuem a função de adicionar, excluir e atualizar sua lista de itens.

Trabalhando o projeto

- Selecione a pasta java e vamos criar um novo pacote “package” de nome “ClassesApoio”, responsável por armazenar as classes:
- CadastroHolder = faz o tratamento dos componentes (TextView, Button, Button) do RecyclerView
- CadastroAdapter = basicamente, para conseguir ligar a view dos itens da lista com as informações dos objetos que serão populosos, utilizamos o adapter para fazer essa tarefa. O RecyclerView possui um ViewHolder que traz mais velocidade e melhora a performance das listas, uma vez que ele cria uma cache das listas em seu celular.

Trabalhando o projeto



Trabalhando o projeto

```
public class CadastroHolder extends RecyclerView.ViewHolder {  
  
    public TextView txvNomeCad;  
    public ImageButton btnEdit;  
    public ImageButton btnDelete;  
  
    public CadastroHolder(@NonNull final View itemView, final Context context) {  
        super(itemView);  
  
        txvNomeCad = (TextView) itemView.findViewById(R.id.txvNomeCad);  
        btnEdit     = (ImageButton) itemView.findViewById(R.id.btnEdit);  
        btnDelete   = (ImageButton) itemView.findViewById(R.id.btnDelete);  
    }  
}
```

Trabalhando o projeto

```
// classe que vai trabalhar os botões de <Edit> e <Delete>
public class CadastroAdapter extends RecyclerView.Adapter<CadastroHolder> {

    @NonNull
    @Override
    public CadastroHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int i) { return null; }

    @Override
    public void onBindViewHolder(@NonNull CadastroHolder cadastroHolder, int i) {}

    @Override
    public int getItemCount() { return 0; }
}
```

Trabalhando o projeto

```
private List<Cadastro> dados;

public CadastroAdapter(List<Cadastro> dados) {
    this.dados = dados;
}

@NonNull
@Override
public CadastroHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.item_lista, parent, attachToRoot: false);
    CadastroHolder cadastroHolder = new CadastroHolder( view, parent.getContext() );

    return cadastroHolder;
}
```

Trabalhando o projeto

```
@Override  
public void onBindViewHolder(@NonNull CadastroHolder cadastroHolder, final int position) {  
  
    if ((dados != null) && (dados.size() > 0)) {  
        // posição do registro na tabela (RECNO())  
        Cadastro cadastro = dados.get(position);  
        cadastroHolder.txvNomeCad.setText(cadastro.NOME);  
    }  
}
```

Trabalhando o projeto

```
cadastroHolder.btnEdit.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        if ( getItemCount() > 0 ) {  
            Cadastro cadastro = dados.get(position);  
            Context context = v.getContext();  
            Intent intent = new Intent(context, CRUDActivity.class);  
            intent.putExtra( name: "CADASTRO", cadastro);  
            ((AppCompatActivity) context).startActivityForResult(intent, requestCode: 0);  
        }  
    }  
});
```

Trabalhando o projeto

```
cadastroHolder.btnDelete.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
  
        if ( getItemCount() > 0 ) {  
            Cadastro cadastro = dados.get(position);  
            Context context = v.getContext();  
            Intent intent = new Intent(context, CRUDActivity.class);  
            intent.putExtra( name: "EXCLUIR", cadastro);  
            ((AppCompatActivity) context).startActivityForResult(intent, requestCode: 0);  
        }  
    }  
});
```

Trabalhando o projeto

```
@Override  
public int getItemCount() {  
  
    // retorna a qtde de registros na tabela  
    return dados.size();  
}
```

Trabalhando o projeto - RecyclerViewActivity

```
public class RecyclerViewActivity extends AppCompatActivity {

    private RecyclerView revLista;
    private CadastroAdapter adapter;
    private CadastroAdapter cadastroAdapter;
    private ConexaoBD conexaoDB;
    private CadastroRepositorio cadastroRepositorio;

    // layout da tela principal
    private ConstraintLayout ctlRecyclerView;

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

    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {...}

    @Override
    public boolean onOptionsItemSelected(MenuItem menuItem) {...}

}
```


RecyclerViewActivity – Parte 1

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

    // referencia do RecyclerView
    revLista = (RecyclerView) findViewById(R.id.revLista);

    // recuperado o layout
    ctlRecyclerView = (ConstraintLayout) findViewById(R.id.ctlRecyclerView);

    // instanciando a classe ConexaoDB
    conexaoDB = new ConexaoBD();

    // cria conexao com o banco de dados
    cadastroRepositorio = new CadastroRepositorio(
        conexaoDB.criarConexao( context: RecyclerViewActivity.this, ctlRecyclerView) );

    // classe de como os dados serao visualizados
    LinearLayoutManager layoutManager = new LinearLayoutManager( context: this);
    revLista.setLayoutManager( layoutManager );
}
```

RecyclerViewActivity – Parte 2

```
// passando a lista para o cadastroAdapter
List<Cadastro> dados = cadastroRepositorio.buscarTodos();

// instanciando a classe cadastroAdapter
cadastroAdapter = new CadastroAdapter( dados );

// vincular o cadastroAdapter ao RecyclerView
revLista.setAdapter( cadastroAdapter );

// Adicionando um botão "up_navigation - voltar"
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

RecyclerViewActivity

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {

    // passando a lista para o cadastroAdapter
    List<Cadastro> dados = cadastroRepositorio.buscarTodos();
    // instanciando a classe cadastroAdapter
    cadastroAdapter = new CadastroAdapter( dados );
    revLista.setAdapter( cadastroAdapter );

}
```

RecyclerViewActivity

```
@Override
public boolean onOptionsItemSelected(MenuItem menuItem) {
    int id = menuItem.getItemId();

    if (id == android.R.id.home) {
        // O metodo finish() vai encerrar essa activity
        finish();
        return true;
    }

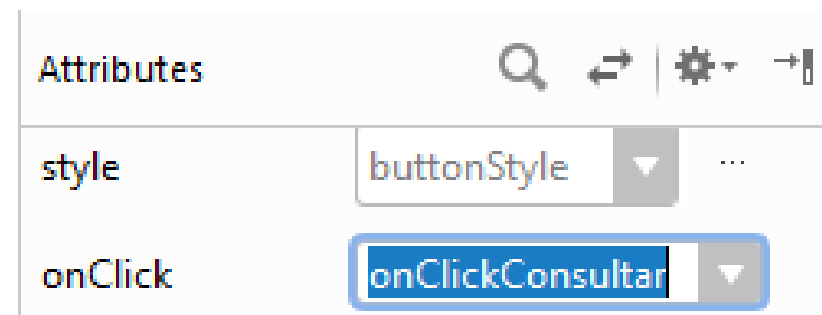
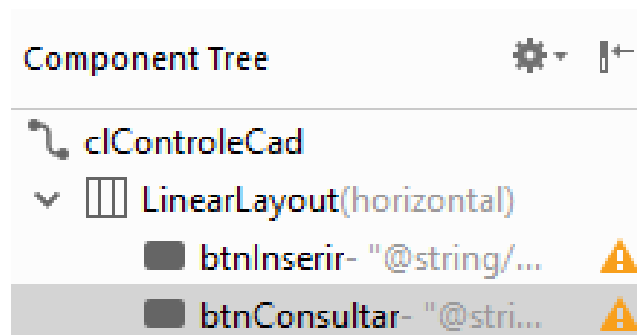
    return super.onOptionsItemSelected(menuItem);
}
```

Trabalhando o projeto

→ Criar o método do botão Consultar na activity ControleActivity

```
public void onClickConsultar(View view) {  
    Intent intent = new Intent( packageContext: this, RecyclerViewActivity.class);  
    startActivity( intent );  
}
```

→ Associar o método acima com o botão do layout “activity_controle”



Trabalhando o projeto - CRUDActivity

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

    // metodo para recuperar os componentes do layout
    recupera( crudActivity: this);

    // Adicionando um botão "up_navigation - voltar"
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    // recuperado o layout para que possa utilizar a classe Snacker
    ctlCRUD = (ConstraintLayout) findViewById(R.id.ctlCRUD);

    // cria conexao com o banco de dados
    cadastroRepositorio = new CadastroRepositorio(
        |      conexaoBD.criarConexao( context: CRUDActivity.this, ctlCRUD) );

    // recupera os dados passados como parametros do botão Edit
    recuperaParametros();
}
```

CRUDActivity – Parte 1

```
// recuperando os parametros para a alteração
private void recuperaParametros() {

    int pos;
    Bundle bundle = getIntent().getExtras();

    // busca os dados atraves da variavel "CADASTRO" gerada no click do botao Edit
    if ( (bundle != null) ) {
        if ( bundle.containsKey("CADASTRO") ) {
            cadastro = (Cadastro) bundle.getSerializable( key: "CADASTRO");
        } else if ( bundle.containsKey("EXCLUIR") ) {
            cadastro = (Cadastro) bundle.getSerializable( key: "EXCLUIR");
        }

        edtNome.setText( cadastro.NOME );
        edtEnde.setText( cadastro.ENDERECO );
        edtCEP.setText( cadastro.CEP );
        edtBai.setText( cadastro.BAIRRO );
        edtCid.setText( cadastro.CIDADE );
    }
}
```

CRUDActivity – Parte 2

```
// selecionando o estado dentro do componente
for (pos = 0; pos < spUF.getCount(); pos++) {
    if ( spUF.getItemAtPosition(pos).toString().equals( cadastro.ESTADO ) ) {
        break;
    }
}
spUF.setSelection( pos ); // atribuindo a posição do estado no componente

// atribuindo o sexo no componente RadionGoup / RadioButton correto de acordo com o sexo
String sexo = cadastro.SEXO;
if ( sexo != null ) {
    RadioButton rb;
    if ( sexo == "M" ) {
        rb = (RadioButton) findViewById(R.id.rbMasc);
    } else {
        rb = (RadioButton) findViewById(R.id.rbFemi);
    }
    rb.setChecked( true );
}
```


CRUDActivity – Parte 3

```
edtEmail.setText( cadastro.EMAIL );  
edtTel.setText( cadastro.TELEFONE );  
edtCel.setText( cadastro.CELULAR );  
edtLatitude.setText( String.valueOf(cadastro.LATITUDE) );  
edtLongitude.setText( String.valueOf(cadastro.LONGITUDE) );
```

```
}
```

```
}
```

CRUDActivity – Parte 4

→ Alterar o método do botão “salvar”, considerando o update

```
try {  
    if ( cadastro.CODIGO == 0 ) {  
        // chamando o metodo inserir dados no banco  
        cadastroRepositorio.inserir( cadastro );  
        // mensagem de inserção concluido com sucesso para o usuario  
        Toast.makeText( context: CRUDActivity.this, text: "Inclusão efetuada com sucesso!",  
                        Toast.LENGTH_LONG).show();  
    } else {  
        // chamando o metodo alterar dados no banco  
        cadastroRepositorio.alterar(cadastro);  
        // mensagem de alteração concluido com sucesso  
        Toast.makeText( context: CRUDActivity.this, text: "Alteração efetuada com sucesso!",  
                        Toast.LENGTH_LONG).show();  
    }  
}
```

CRUDActivity – Parte 5

→ Criar o método do botão “Excluir”

```
// Método onClickDelete -> click do botão
public void onClickDelete(View view) {

    Bundle bundle = getIntent().getExtras();

    if ( (bundle != null) && (bundle.containsKey("EXCLUIR")) ) {
        btnExcluir.setClickable(true);
        cadastro = (Cadastro) bundle.getSerializable( key: "EXCLUIR");
        cadastroRepositorio.excluir( cadastro.CODIGO );
        Toast.makeText( context: this, text: "Exclusão efetuada com sucesso!",
            Toast.LENGTH_LONG).show();
        btnExcluir.setClickable(false);
        finish();
    }
}
```