Desenvolvimento Mobile

Aula 5

Evoluindo Projeto Listagem de Pessoas

Vamos adicionar um botão *Novo* no topo da listagem de pessoas

Listagem de Pessoas

Vamos criar uma nova *Activity* para incluir uma nova pessoa

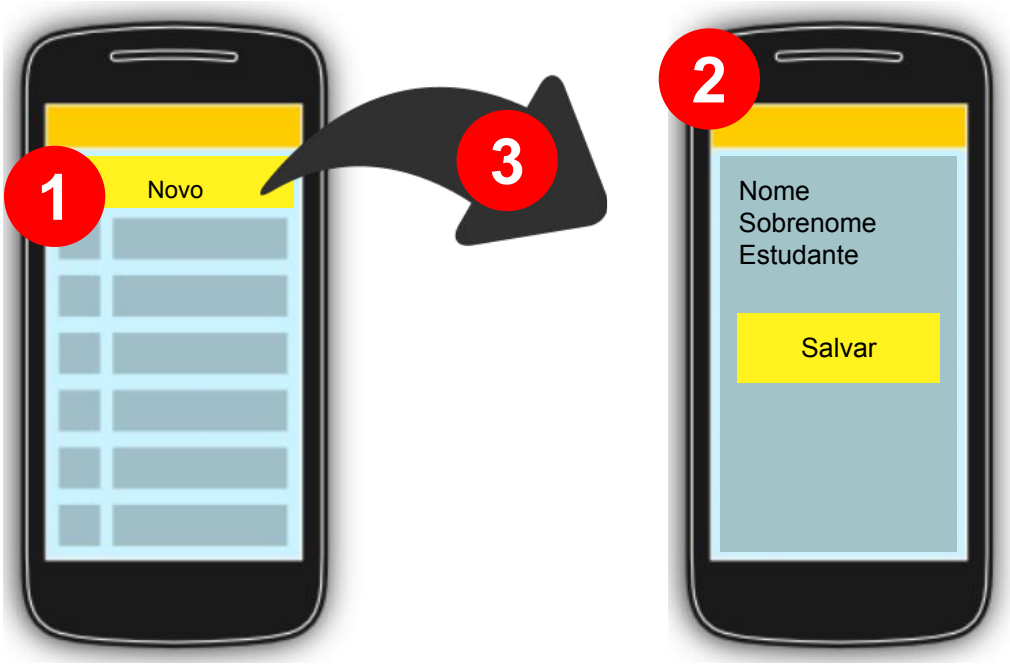
Listagem de Pessoas

Incluir Pessoa

O botão *Novo* deve direcionar para a tela de incluir pessoas

Listagem de Pessoas

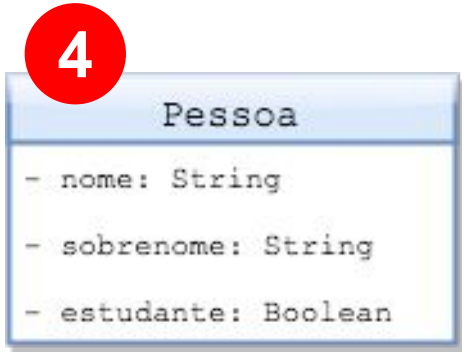
Incluir Pessoa



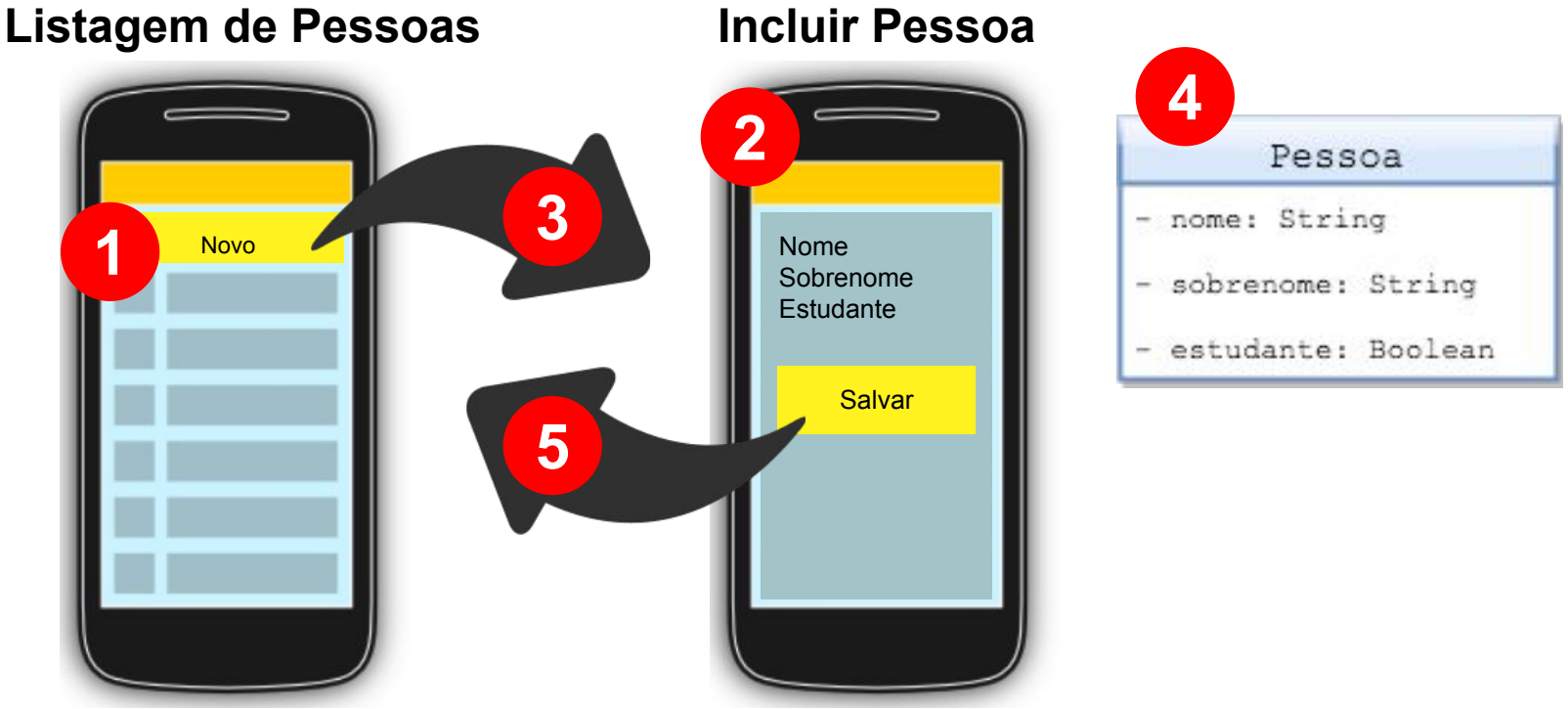
A tela de incluir pessoas deve criar uma instância da classe *Pessoa*

Listagem de Pessoas

Incluir Pessoa

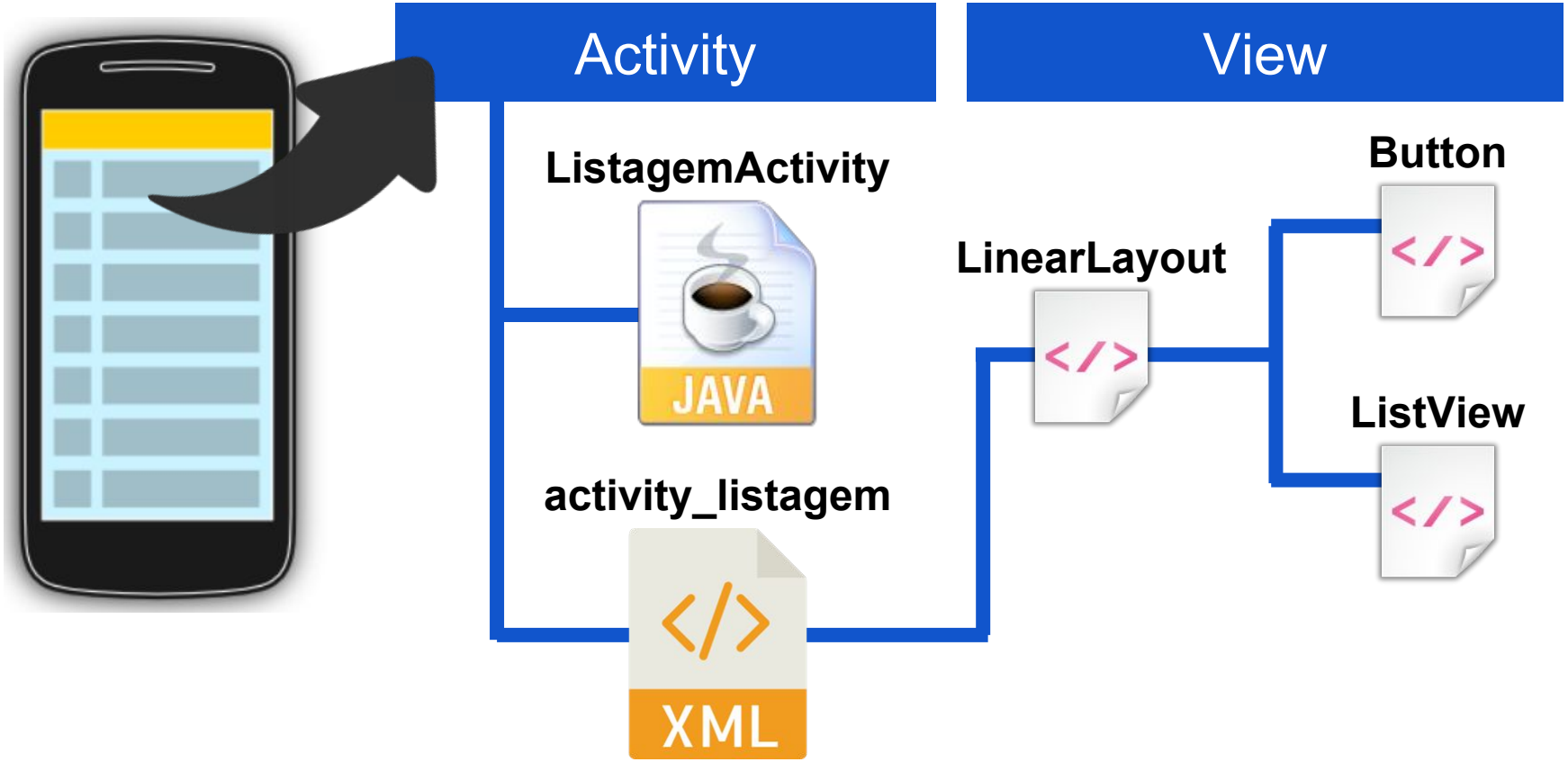


O botão *Salvar* deve voltar à tela de listagem de pessoas



Incluindo um botão na listagem de pessoas

Incluindo o botão na listagem de pessoas



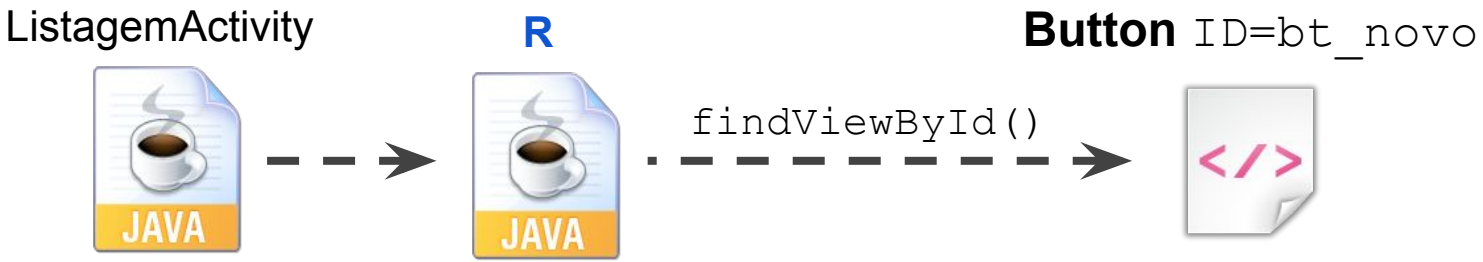
Incluindo o botão na listagem de pessoas


```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Novo"
        android:id="@+id/bt_novo" />

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lista_pessoas" />

</LinearLayout>
```

Incluindo o botão na listagem de pessoas

Buscando o ID do botão e exibindo um Toast

Através do método *findViewById()* nós podemos obter o botão, adicionar um *listener* para o evento de click e, quando o evento for realizado, exibir um *Toast* na tela.

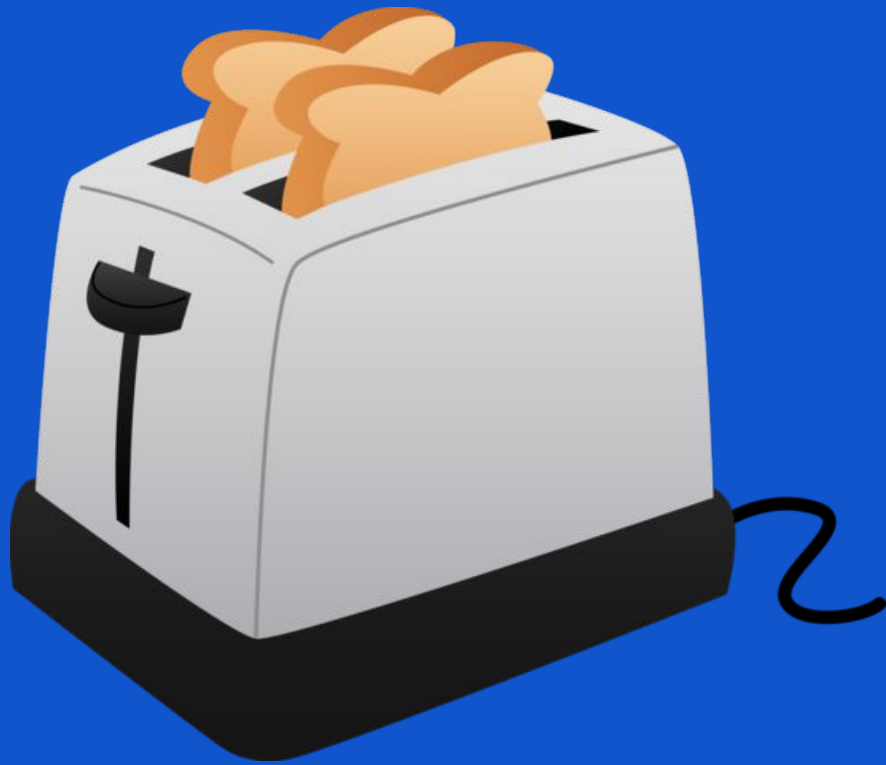


Botão clicado!

Curiosidade

Reza a lenda que o componente *Toast,* muito utilizado em smartphones e sites responsivos, recebeu este nome inspirado nas máquinas de torrar pão.

O *Toast* faz um movimento semelhante ao pão que sai dessas máquinas.



Incluindo o botão na listagem de pessoas

PASSO 1: crie um atributo *botaoNovo* na classe *ListagemActivity*

```
public class ListagemActivity extends AppCompatActivity {

    private Button botaoNovo;

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

        List<Pessoa> pessoas = criarPessoas();
```

Incluindo o botão na listagem de pessoas

PASSO 2: dentro do método *onCreate()*, instancie o botão buscando-o pelo ID. Em seguida, adicione um *OnClickListener*.

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

    this.botaoNovo = findViewById(R.id.bt_novo);
    this.botaoNovo.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

        }
    });

    // Código de listar as pessoas...
}
```

Observação: *OnClickListener* precisa implementar o método *onClick()*.

Incluindo o botão na listagem de pessoas

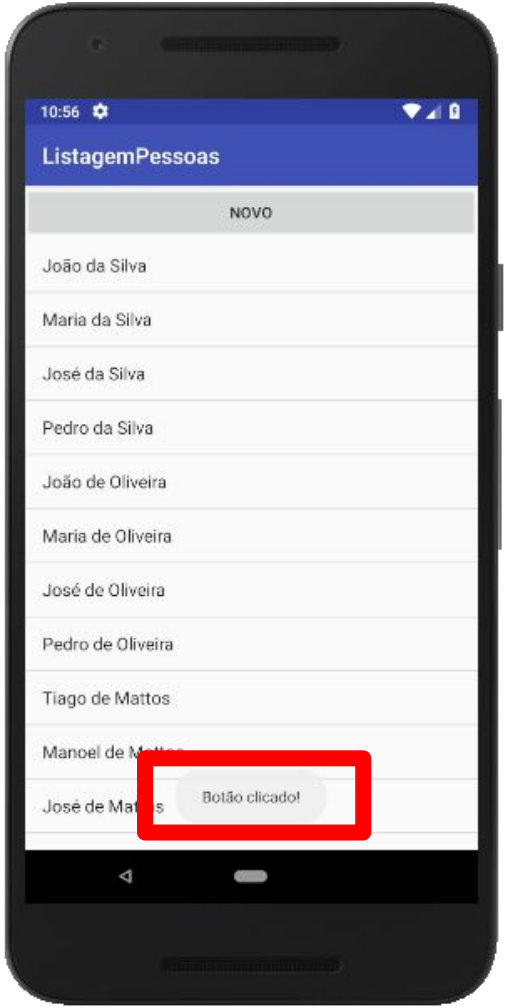
PASSO 3: faça uma chamada ao Toast

```
this.botaoNovo = findViewById(R.id.bt_novo);
this.botaoNovo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Toast.makeText( context: ListagemActivity.this, text: "Botão clicado!",Toast.LENGTH_LONG).show();
    }
});
```

O componente *Toast* possui um método estático chamado *makeText()* que recebe: o contexto, a mensagem que será exibida, e o tempo de exibição.

Para efetivamente exibir a mensagem, deve-se chamar o método *show()* de forma encadeada à chamada de *makeText()*.

Após clicar no botão o *Toast* é exibido.



Outra opção de implementar o click do botão

Existe uma alternativa de implementação do evento *onClick* que consiste em declarar um método na **Activity** e apontar este método diretamente a partir do layout XML.

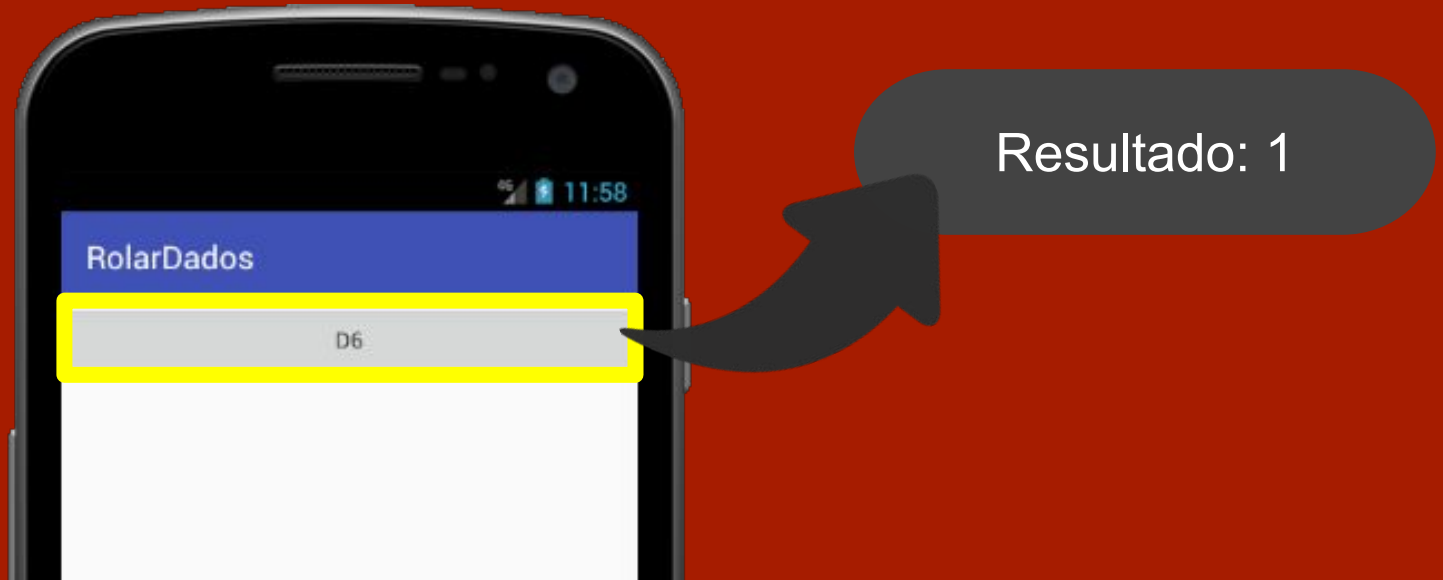
```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Novo"
    android:id="@+id/bt_novo"
    android:onClick="clickBotaoNovo"
    />
```

```
public void clickBotaoNovo(View view) {
    Toast.makeText( context: ListagemActivity.this, text: "Botão clicado!",Toast.LENGTH_LONG).show();
}
```

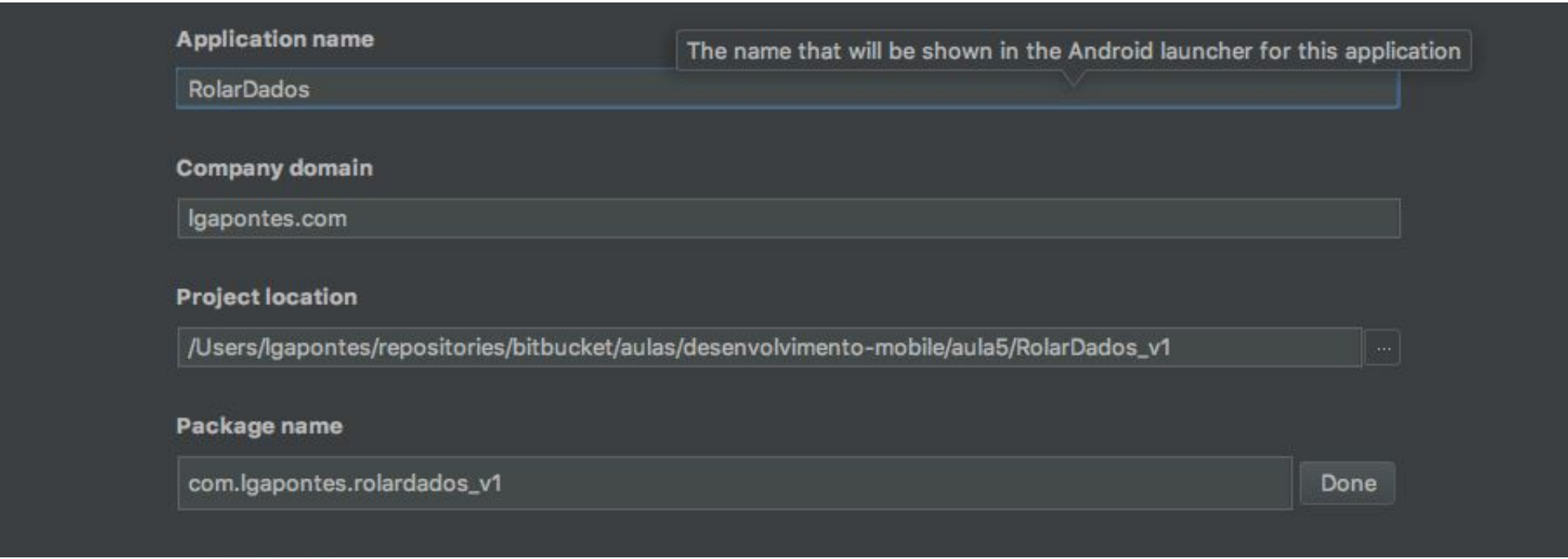
Fica a critério do desenvolvedor escolher a opção mais adequada ao app. Particularmente eu prefiro esta segunda, por exigir menos código.

Exercício em Sala

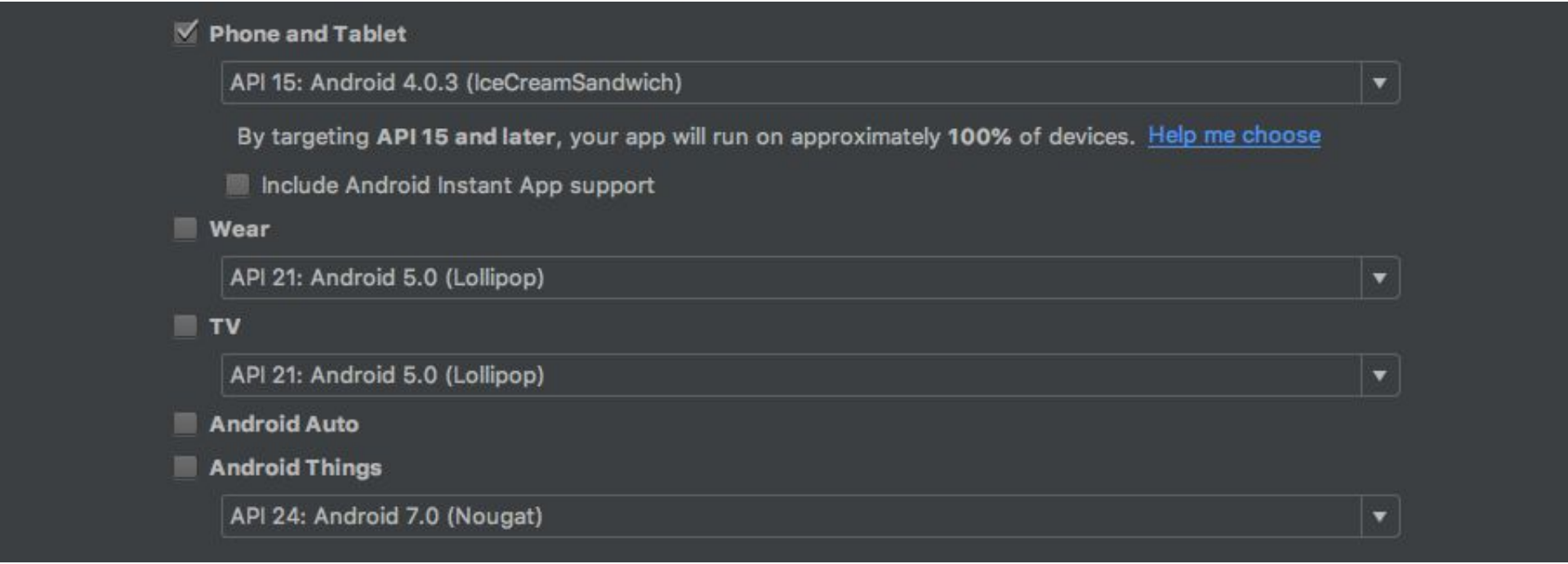
Crie um projeto chamado *RolarDados*. Este projeto deve possuir um botão com a label "d6" que, quando clicado, deve exibir um toast com um número aleatório entre 1 e 6.



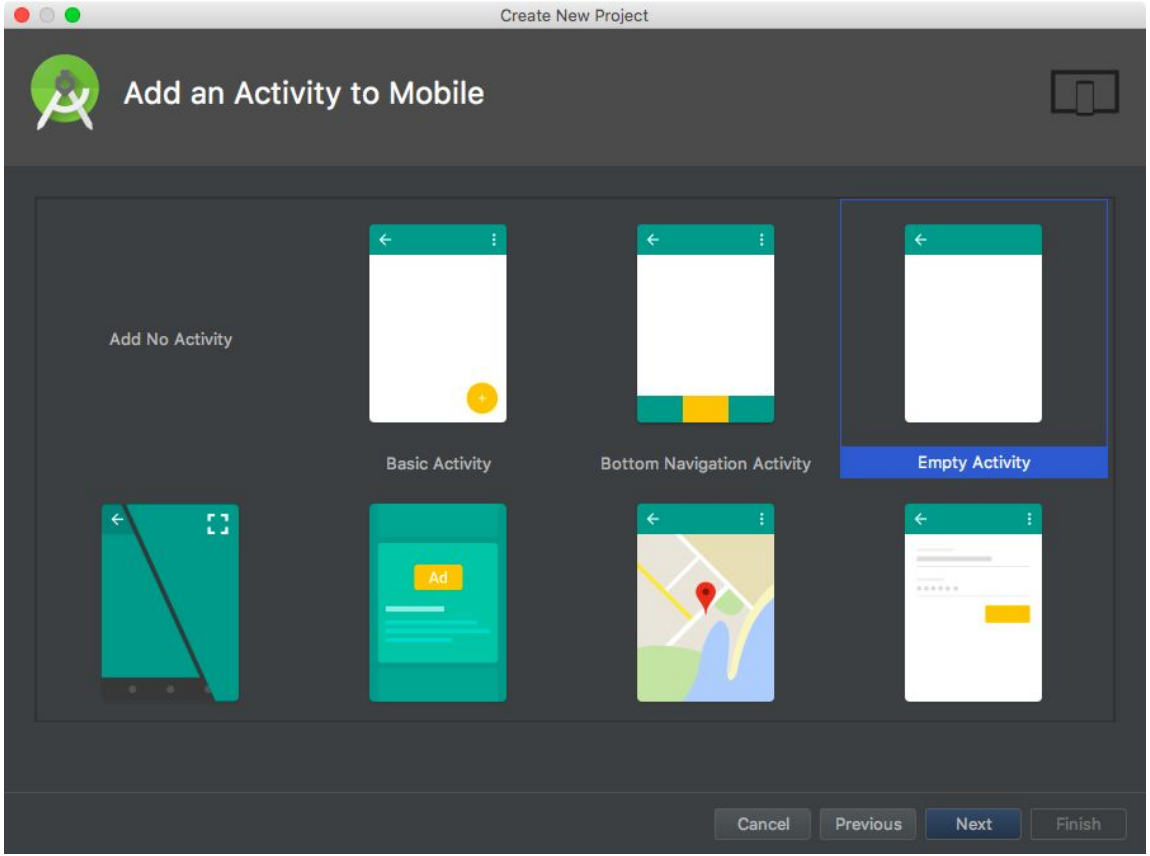
Resolvendo o Exercício



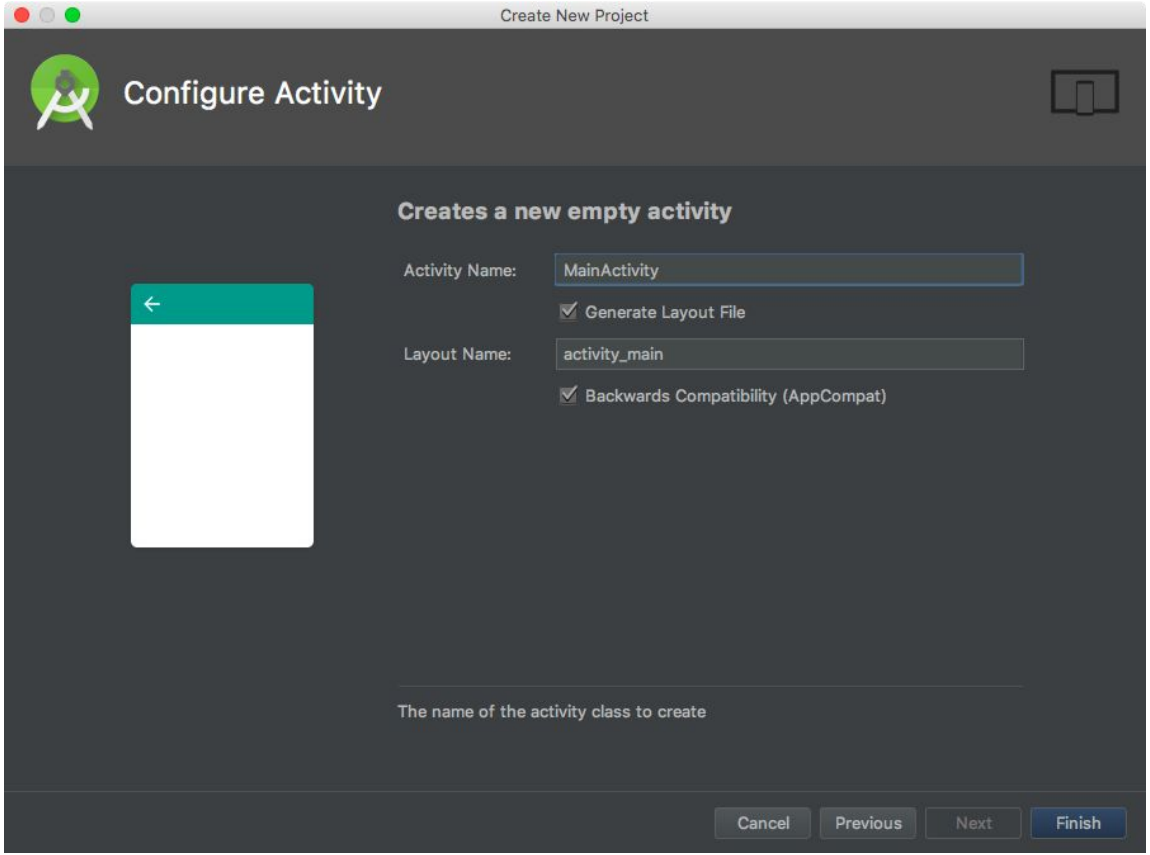
Resolvendo o Exercício



Resolvendo o Exercício



Resolvendo o Exercício



Resolvendo o Exercício

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="d6"
        android:id="@+id/rolar" />

</LinearLayout>
```

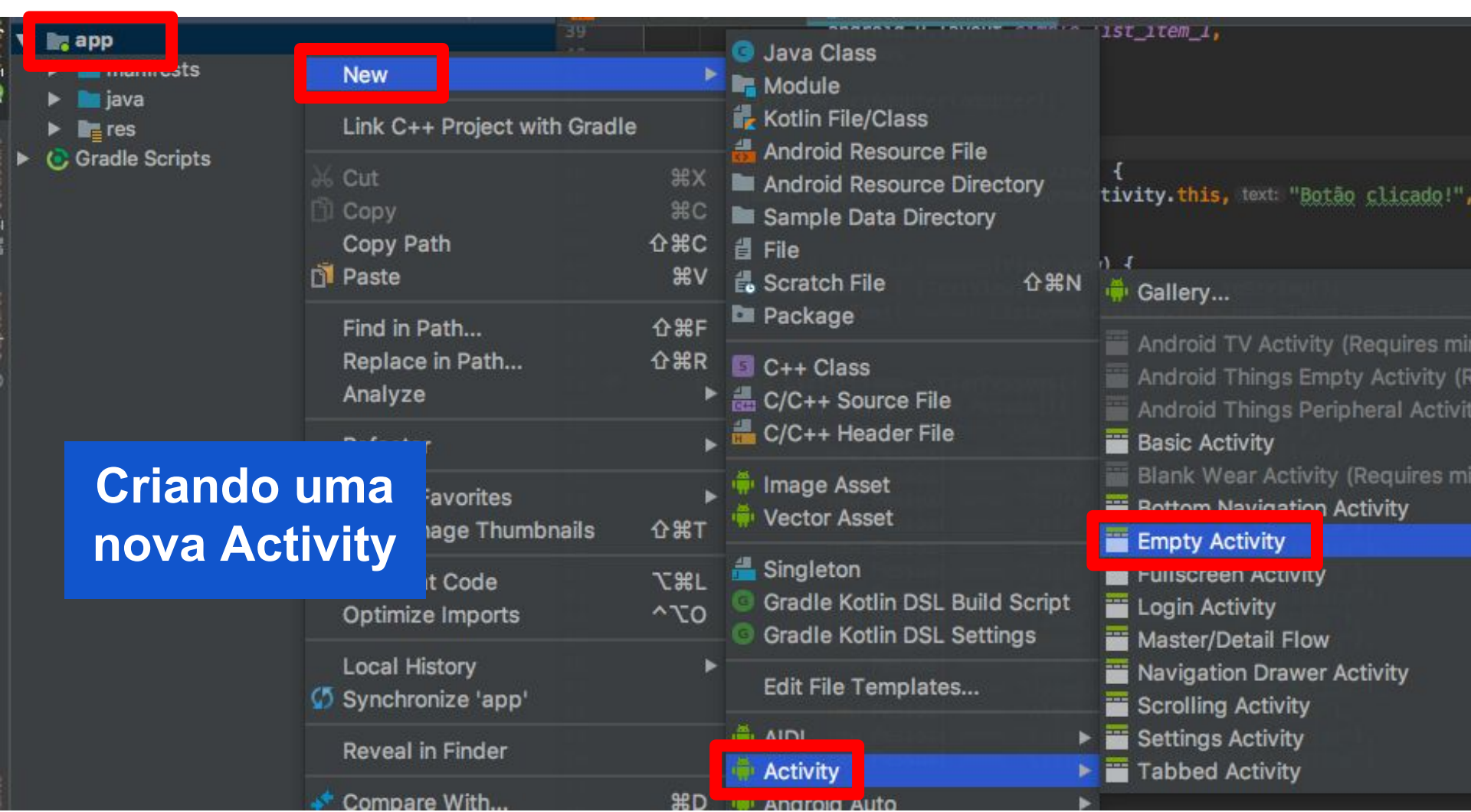
Resolvendo o Exercício

```
Button button = findViewById(R.id.rolar);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Random r = new Random();
        int numero = r.nextInt( bound: 6) + 1;
        Toast.makeText( context: MainActivity.this, text: "Resultado: " + numero, Toast.LENGTH_SHORT).show();
    }
});
```

Para criar números pseudo-aleatórios no Java, podemos utilizar o método *nextInt()* da classe *Random*.
Este método retorna um número inteiro entre 0 (inclusive) e o número passado por parâmetro (exclusive).


11:58
RolarDados
D6
Resultado: 1

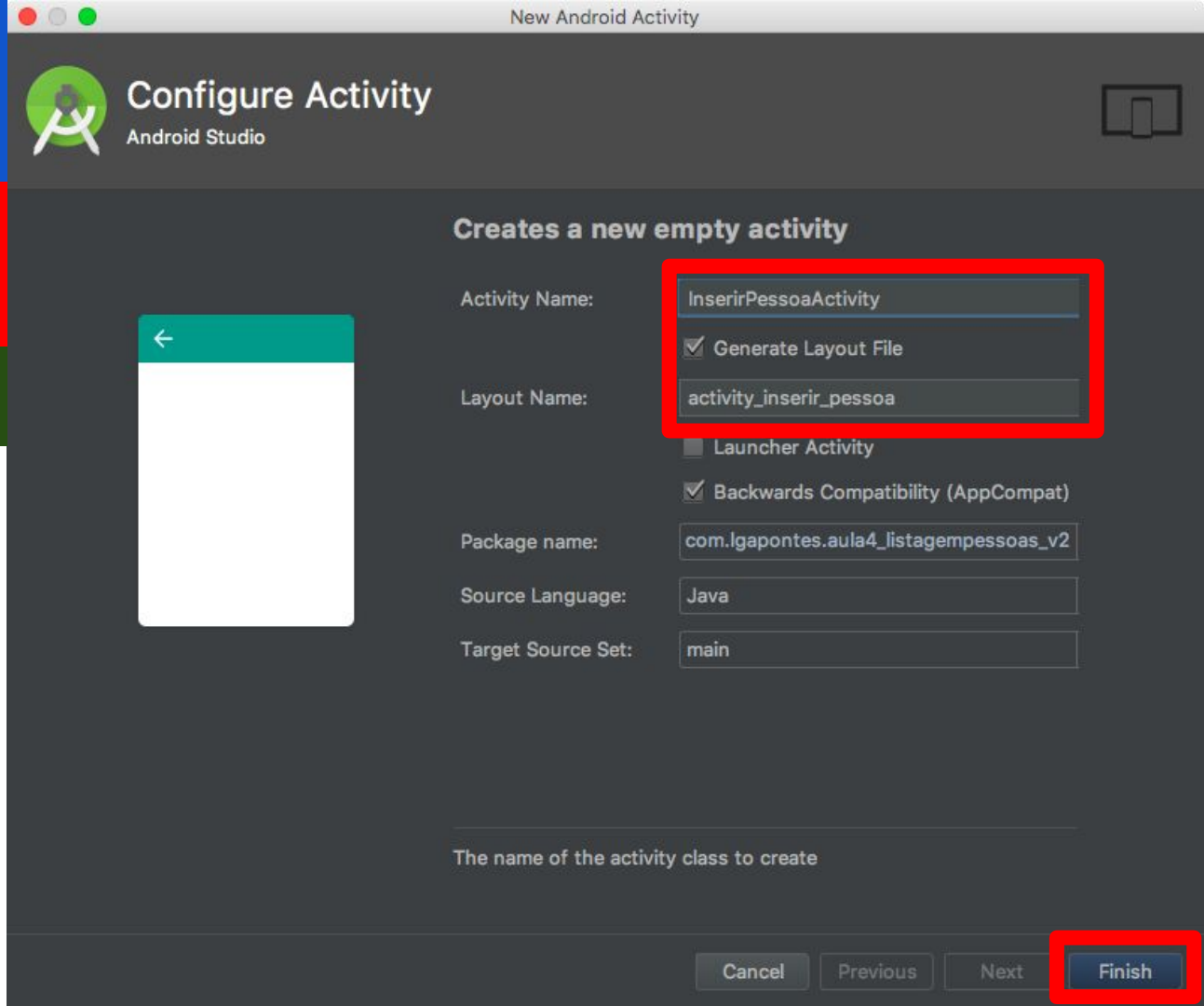
Criando a Activity Incluir Pessoa


app
java
res
Gradle Scripts
New
Link C++ Project with Gradle
Cut ⌘X
Copy ⌘C
Copy Path ⇧⌘C
Paste ⌘V
Find in Path... ⇧⌘F
Replace in Path... ⇧⌘R
Analyze
Optimize Imports ^⌥O
Local History
Synchronize 'app'
Reveal in Finder
Java Class
Module
Kotlin File/Class
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File ⇧⌘N
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
Edit File Templates...
Activity
Gallery...
Basic Activity
Empty Activity
Login Activity
Master/Detail Flow
Navigation Drawer Activity
Scrolling Activity
Settings Activity
Tabbed Activity
Criando uma nova Activity

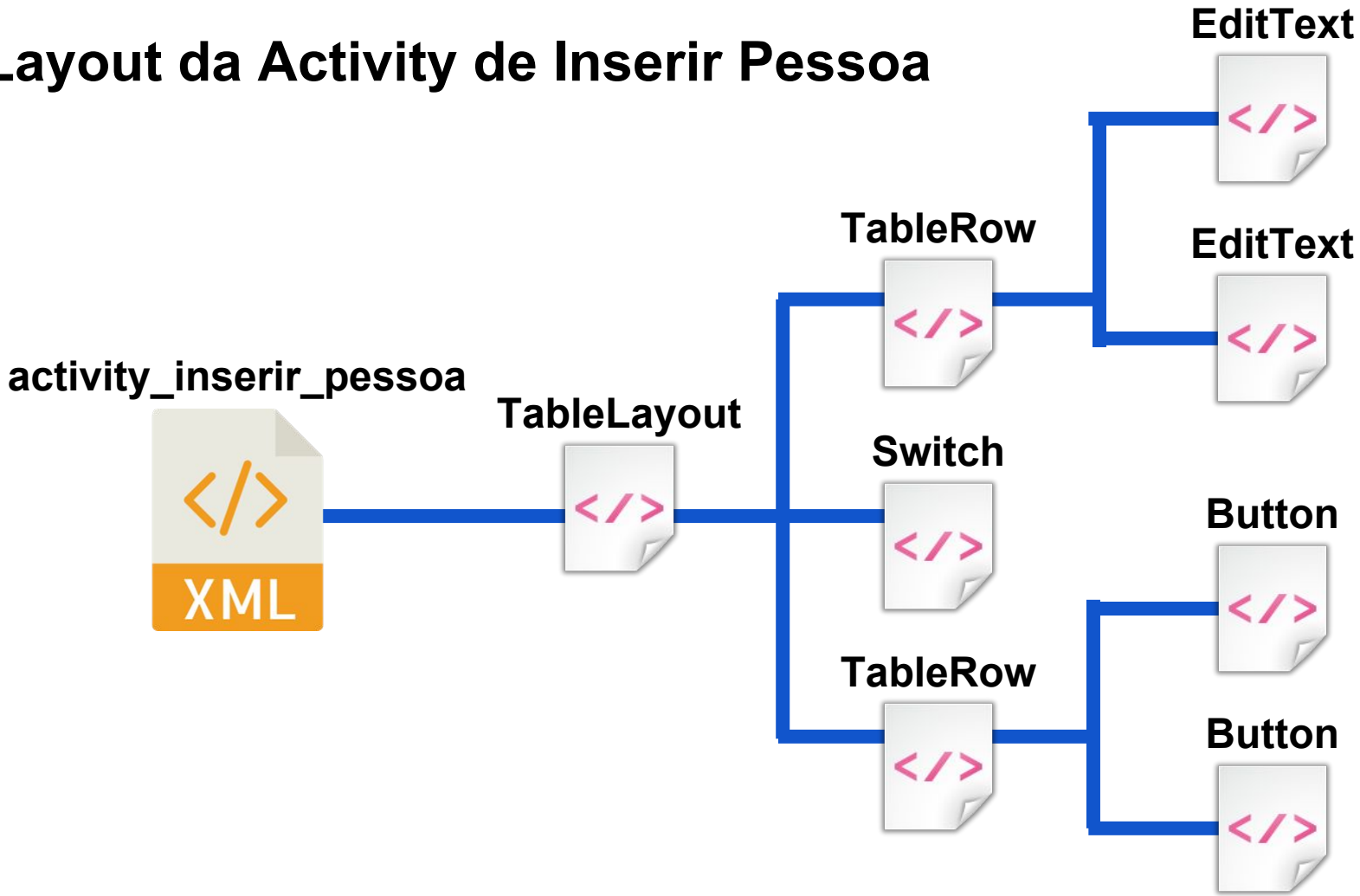
Nome:
InserirPessoaActivity

Crie o layout automaticamente.

Clique em *Finish*



Layout da Activity de Inserir Pessoa



Layout da Activity de Inserir Pessoa

TableLayout

O *TableLayout* organiza os elementos filhos em linhas e colunas. Qualquer elemento inserido será considerado como uma linha de uma única coluna.

Dica 1: seus elementos filhos não podem definir o atributo *layout_width*. Ele sempre será *match_parent*.

Dica 2: use o atributo *stretchColumns* especificando uma relação de colunas (ex. "0,1,2") que devem ter seus tamanhos equivalentes em relação ao pai.

TableRow

O *TableRow* é um agregador de elementos que quando utilizado em conjunto com o *TableLayout*, permite configurar mais de uma coluna por linha.

Exemplo de *TableLayout*

TableLayout, com stretchColumns="0,1"		
TableRow	`TextView`	`TextView`
	`Switch`	
TableRow	`Button`	`Button`

Layout da Activity de Inserir Pessoa

Dica 1: No elemento *Switch*, é possível alterar o conteúdo de "on/off" para um texto através dos atributos *textOn* e *textOff*.

Dica 2: É comum utilizar o atributo *hint* em *EditText* ao invés de criar *TextView* como labels dos campos.

Dica 3: coloque *focusable* igual a *true* no primeiro campo do formulário.

activity_inserir_pessoa.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:stretchColumns="0,1" >
    <TableRow>
        <EditText
            android:hint="Nome"
            android:id="@+id/incluir_pessoa_nome"
            android:focusable="true" />
        <EditText
            android:hint="Sobrenome"
            android:id="@+id/incluir_pessoa_sobrenome" />
    </TableRow>
    <Switch
        android:id="@+id/incluir_pessoa_estudante"
        android:text="Estudante?"
        android:textOn="Sim"
        android:textOff="Não"
        android:textSize="18sp"
        android:padding="4dp" />
    <TableRow>
        <Button
            android:id="@+id/incluir_pessoa_cancelar"
            android:text="Cancelar" />
        <Button
            android:id="@+id/incluir_pessoa_salvar"
            android:text="Salvar" />
    </TableRow>
</TableLayout>
```

Layout da A
Inserir P

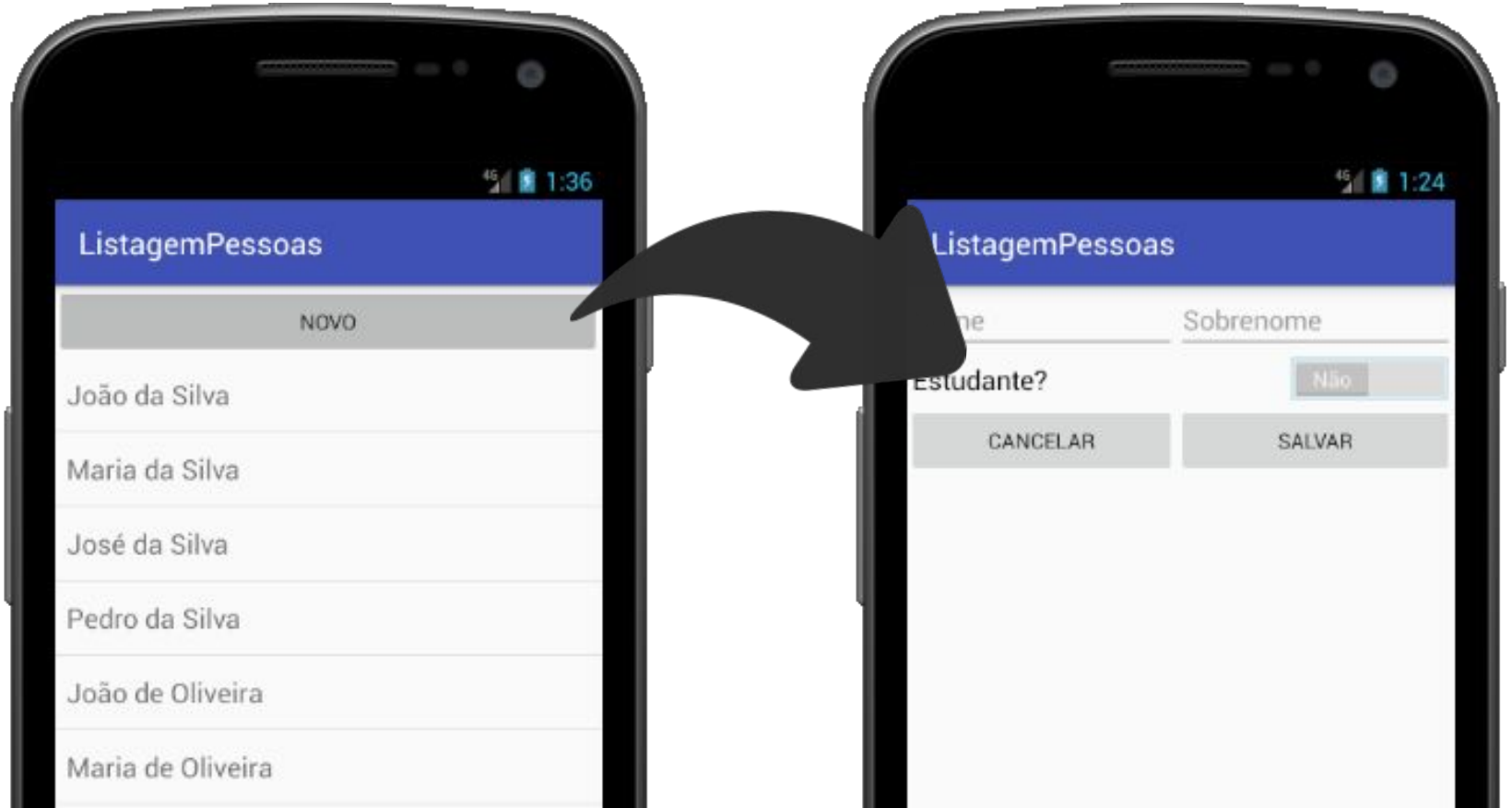
Dica 1: N
possível al
"on/off" para
dos atributos *tex*

Dica 2: É comum utilizar o atributo *hint* em *EditText* ao invés de criar *TextView* como labels dos campos.

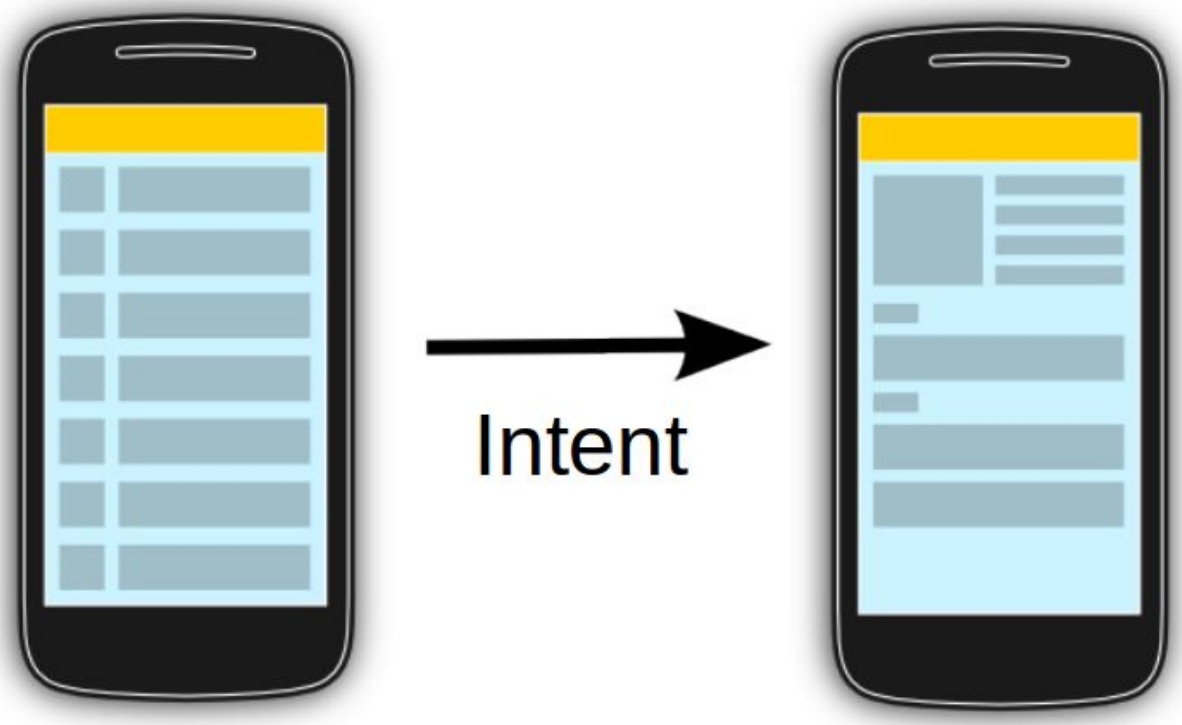
Dica 3: coloque *focusable* igual a *true* no primeiro campo do formulário.

Veremos outros elementos de formulário no futuro!

Como vamos navegar entre as Activities?

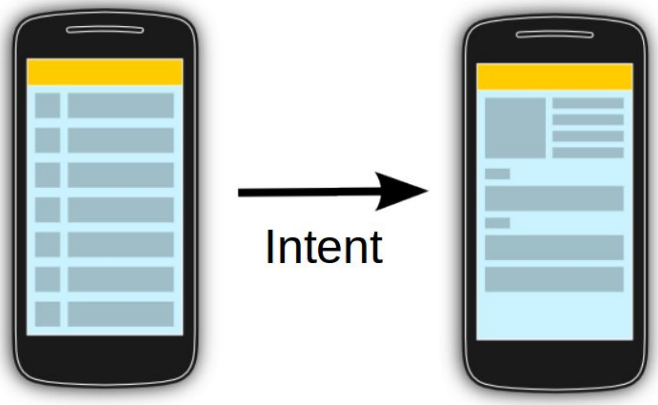


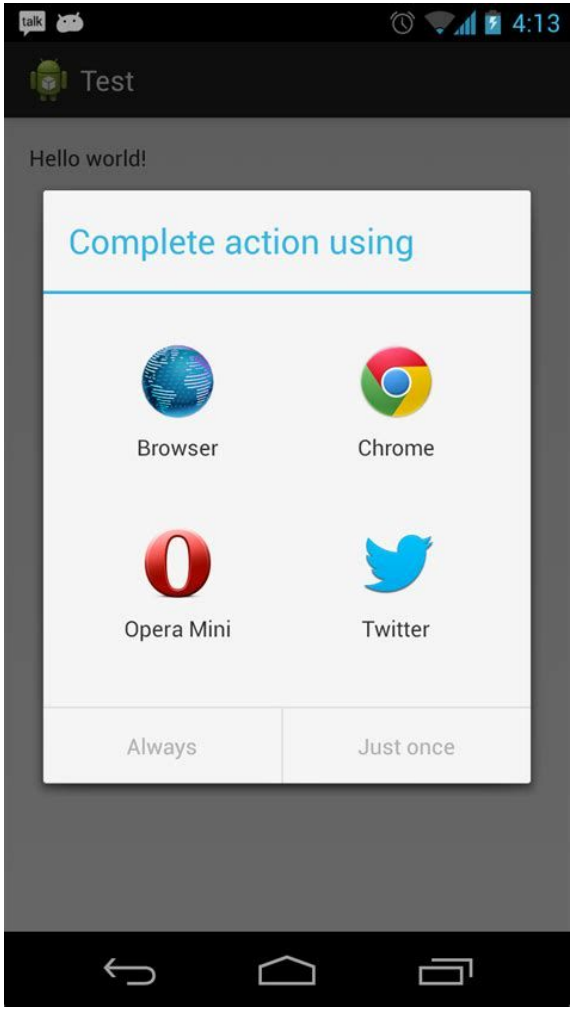
Trabalhando com Intents



Trabalhando com Intents

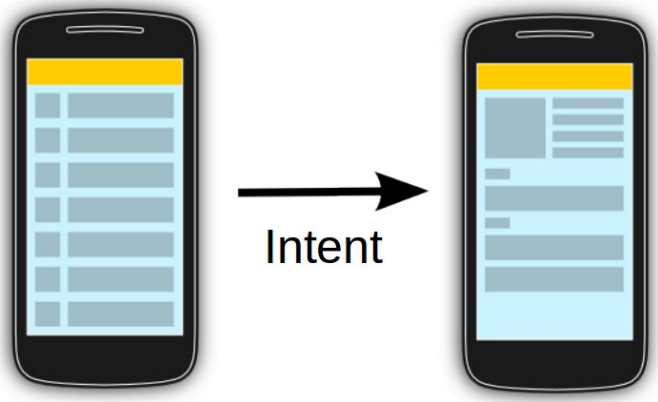
Intent é um objeto da especificação Android que deve ser utilizado para criar uma intenção de realizar alguma coisa. Ex. abrir uma nova Activity.





Trabalhando com Intents

Intent é um objeto da especificação Android que deve ser utilizado para criar uma intenção de realizar alguma coisa. Ex. abrir uma nova Activity.



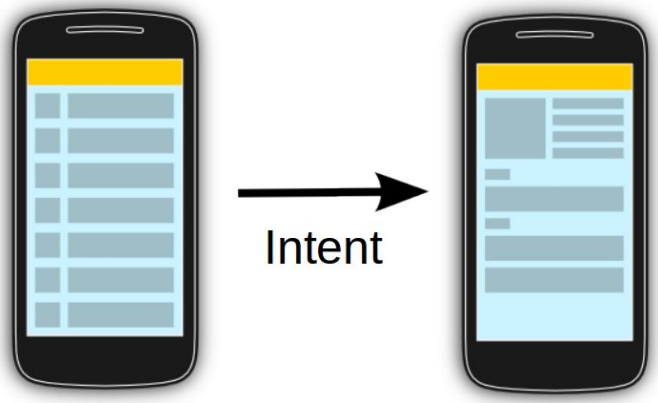
Existem dois tipos de Intents:

Implícita: utilizada para realizar uma **ação** a partir de dados informados na criação da *Intent*, porém, sem especificar exatamente qual aplicação será aberta. Neste caso o Android vai buscar as aplicações relacionadas ao tipo da **ação**.

Explícita: utilizada para informar a aplicação ou a parte que deve ser executada. Geralmente é utilizada para iniciar telas dentro da mesma aplicação.

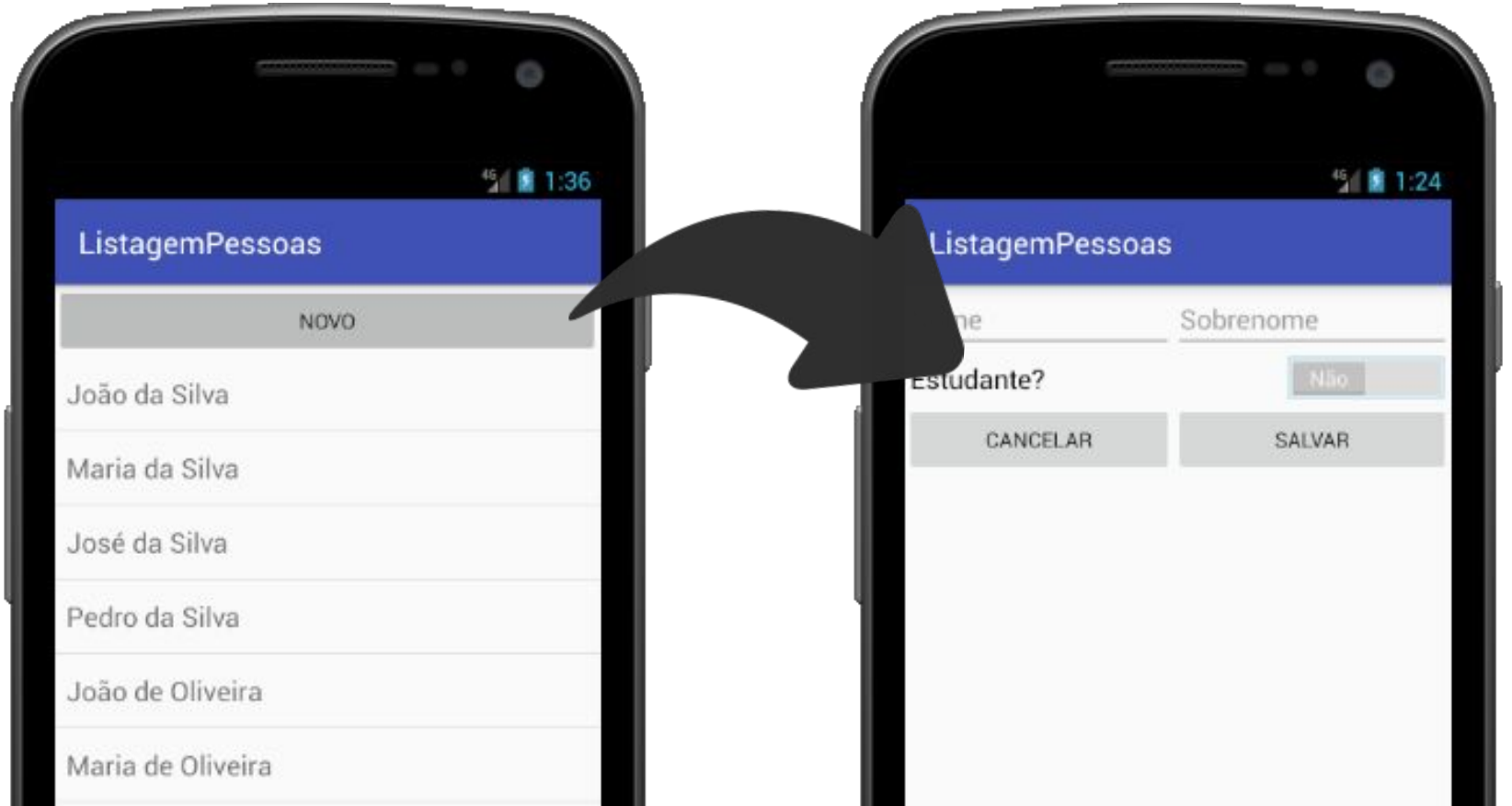
Trabalhando com Intents

No nosso caso, como queremos navegar entre *Activities* da mesma aplicação e já sabemos qual delas deve ser aberta, vamos trabalhar com uma *Intent* explícita, cujo código está disponível a seguir.




```
public void clickBotaoNovo(View view) {
    Intent abrirInserirPessoa = new Intent( packageContext: this,InserirPessoaActivity.class);
    startActivity(abrirInserirPessoa);
}
```

Pronto, está funcionando!

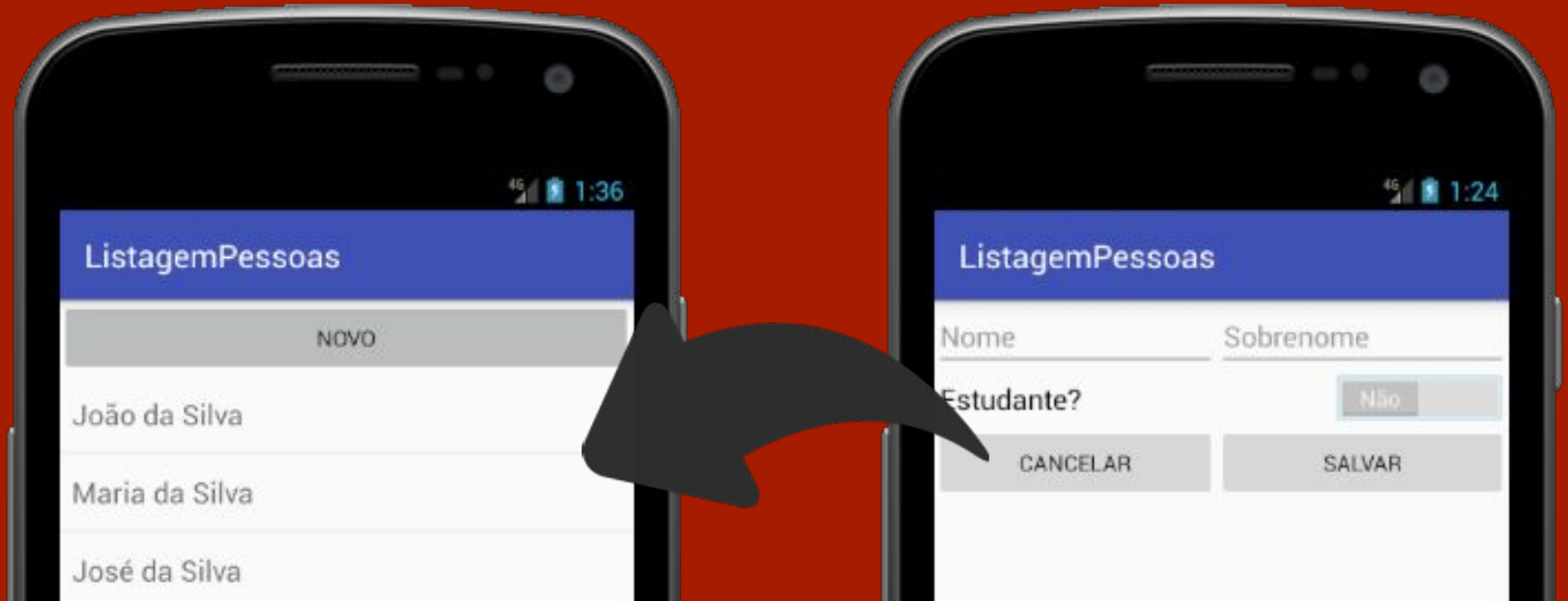


Pronto, est


No futuro veremos outros exemplos de Intents!
ListagemPess
João da Silv
Maria da S
José da
Pedro
1:24
Sobrenome
Estudante?
Não
CANCELAR
SALVAR

Exercício em Sala

Altere a *InserirPessoaActivity* para que os botões *Cancelar* e *Salvar* abram a *ListagemActivity*.



Resolvendo o Exercício

Primeiramente vamos criar um método *voltarParaListagem()* dentro da *InserirPessoaActivity.class*. Este método vai abrir uma nova *Intent* apontando para a *ListagemPessoa.class*.

```
private void voltarParaListagem() {
    Intent voltar = new Intent( packageContext: this,ListagemActivity.class);
    startActivity(voltar);
}
```

Resolvendo o Exercício

Em seguida vamos definir *listeners* para o evento de click nos botões de cancelar e salvar. Ambos devem invocar o método *voltarParaListagem().* Isso deve ser definido no método *onCreate()*.

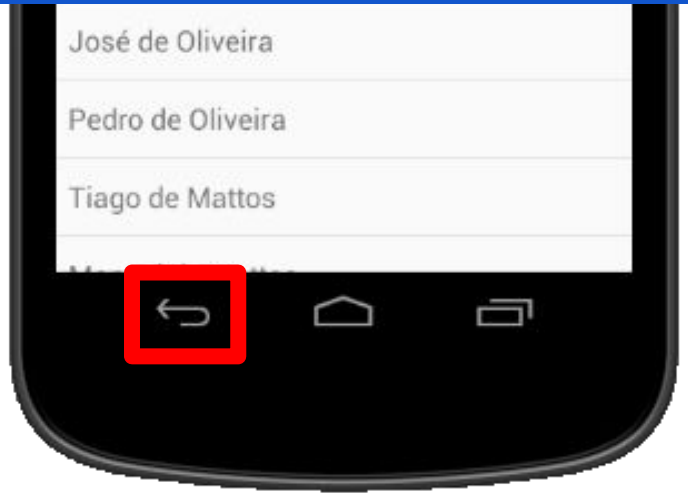
```
Button cancelar = findViewById(R.id.incluir_pessoa_cancelar);
cancelar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        voltarParaListagem();
    }
});

Button salvar = findViewById(R.id.incluir_pessoa_salvar);
salvar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        voltarParaListagem();
    }
});
```

Resolvendo o Exercício

Discussão em Sala

Existe algo de errado aqui?

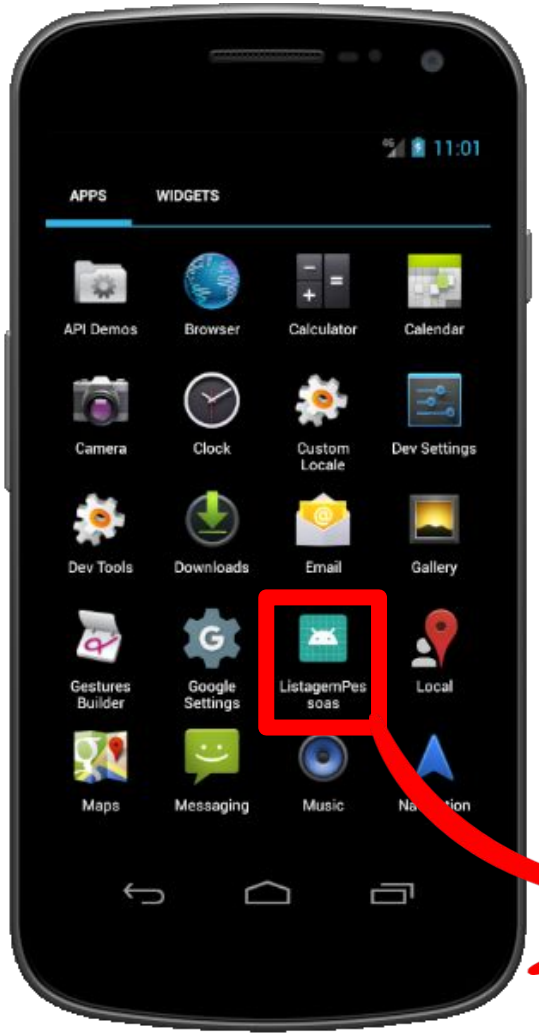
Discussão em Sala

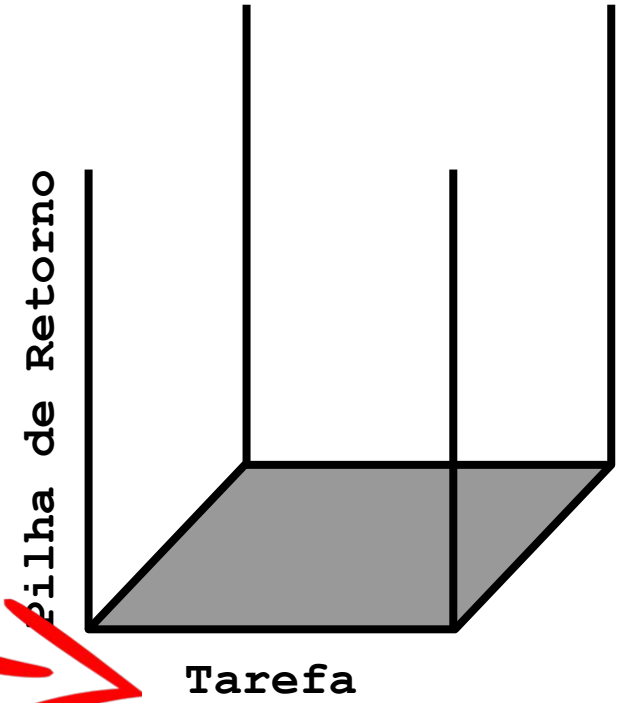


Rode o aplicativo, utilize os botões *Novo* e *Cancelar* e em seguida o *Back Button*.

Por que o Back Button volta para Inserir Pessoa ao invés de retornar à home do Android?

Pilha de Retorno

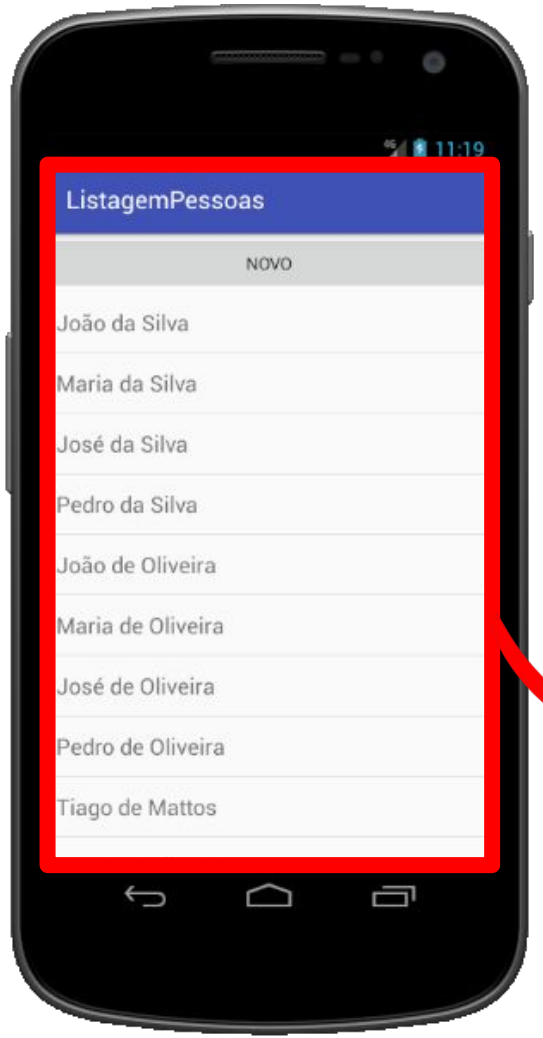


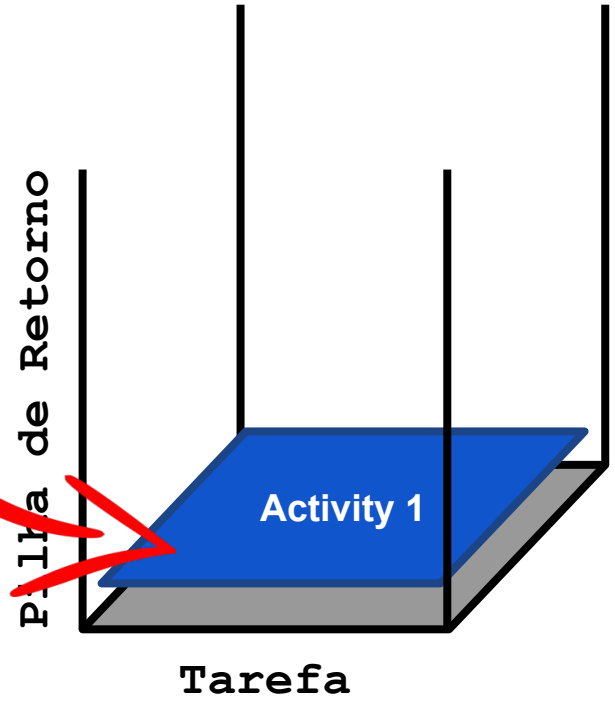


⇒ Quando um ícone é clicado no área do dispositivo, o Android inicializa uma nova **Tarefa** e coloca a **Activity** inicial na raiz de uma pilha.

⇒ **Tarefas** são coleções de atividades com as quais os usuários interagem ao realizar determinado trabalho.

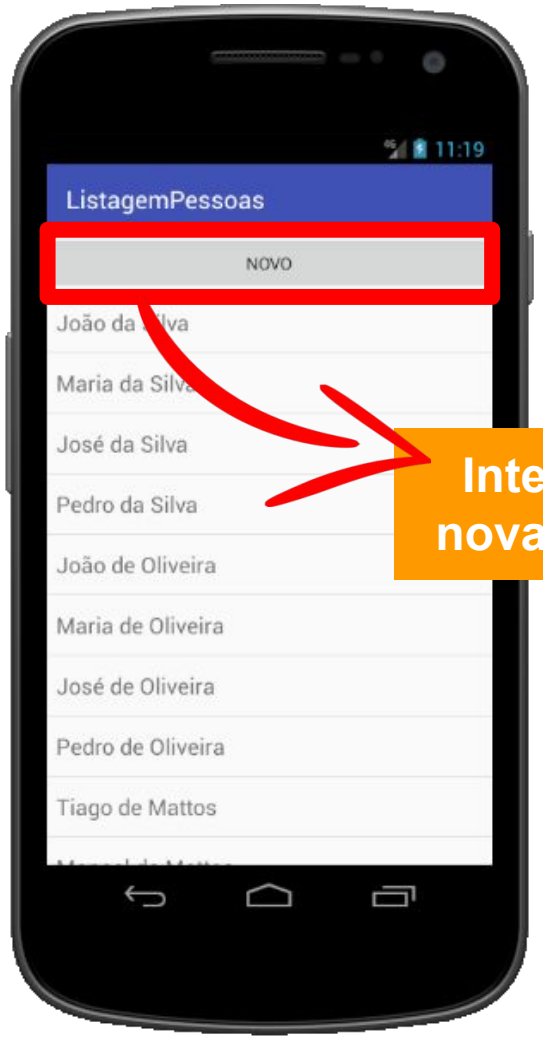
Pilha de Retorno



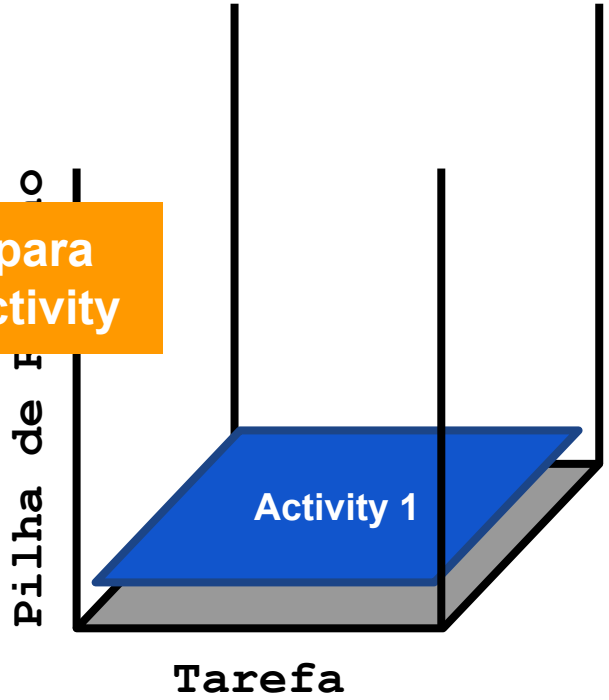


⇒ As **Activities** são colocadas na pilha na ordem em que são abertas.

Pilha de Retorno

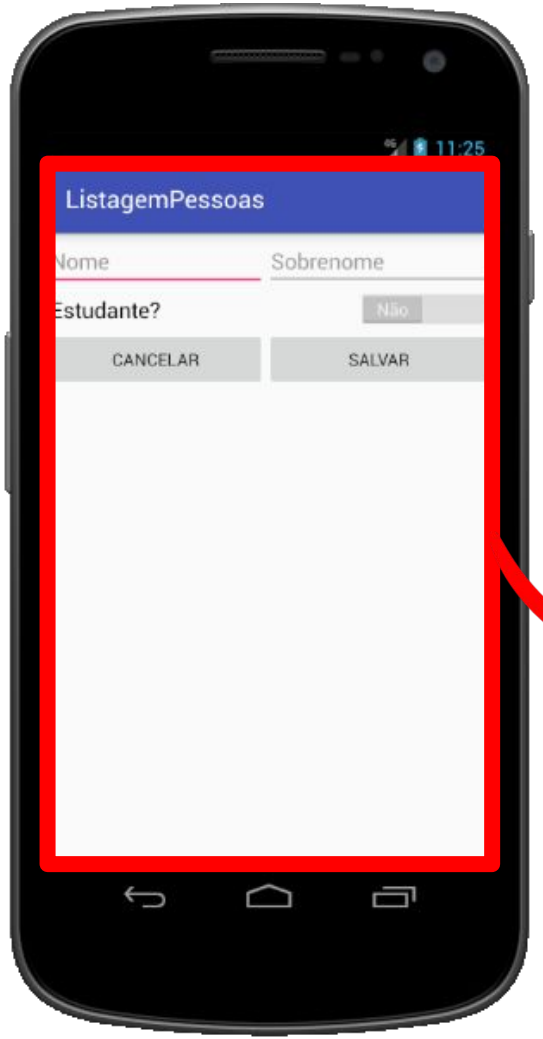


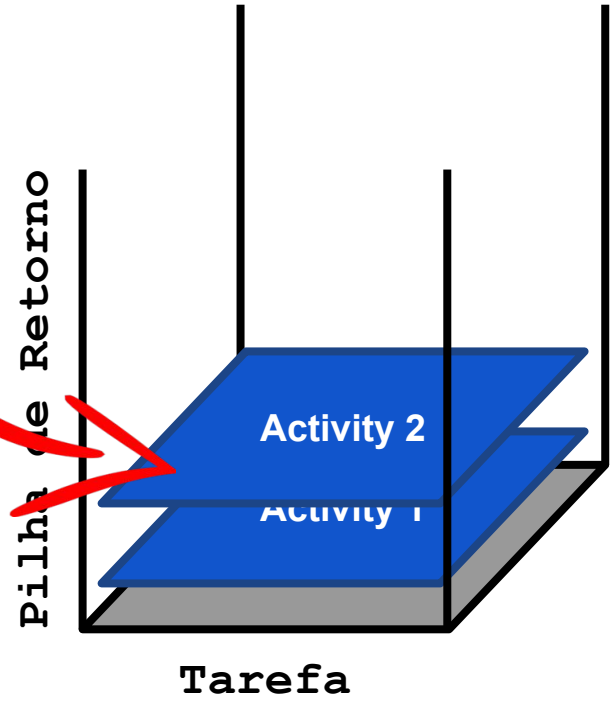
Intent para nova Activity



⇒ Quando há uma tentativa de abertura de uma nova **Activity**, a aplicação abre uma **Intent** e aguarda o retorno do Android pela criação da mesma.

Pilha de Retorno

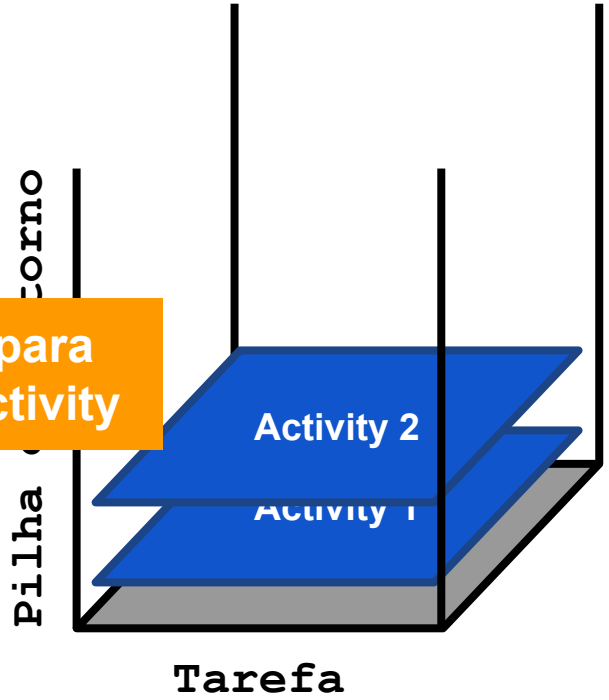




⇒ Quando a **Intent** concluir sua ação, a nova **Activity** será colocada sobre a primeira na pilha de retorno.

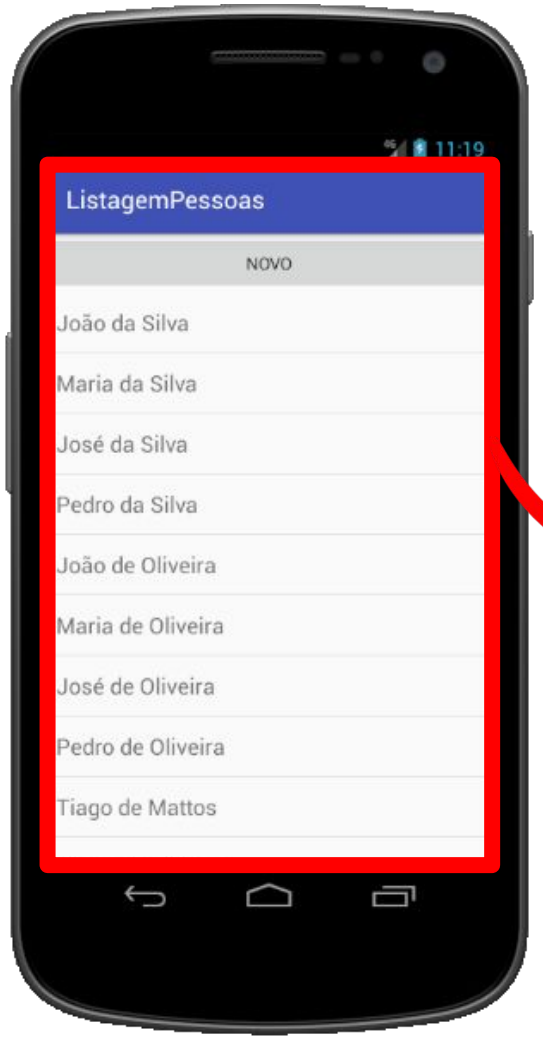
Pilha de Retorno

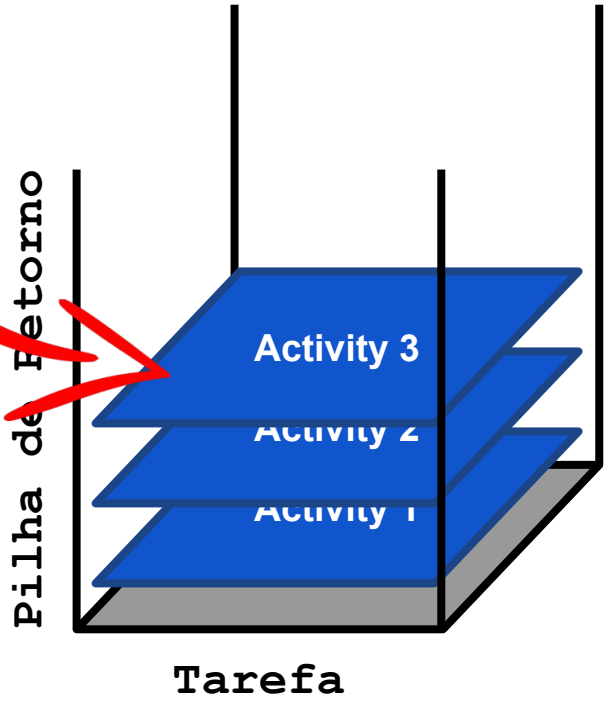
Intent para nova Activity



⇒ Da forma como o aplicativo de listagem de pessoas foi construído, ao clicar no botão *Cancelar*, por exemplo, uma nova **Intent** é criada para solicitar ao Android a abertura de uma nova **Activity** (no caso, *ListagemActivity*).

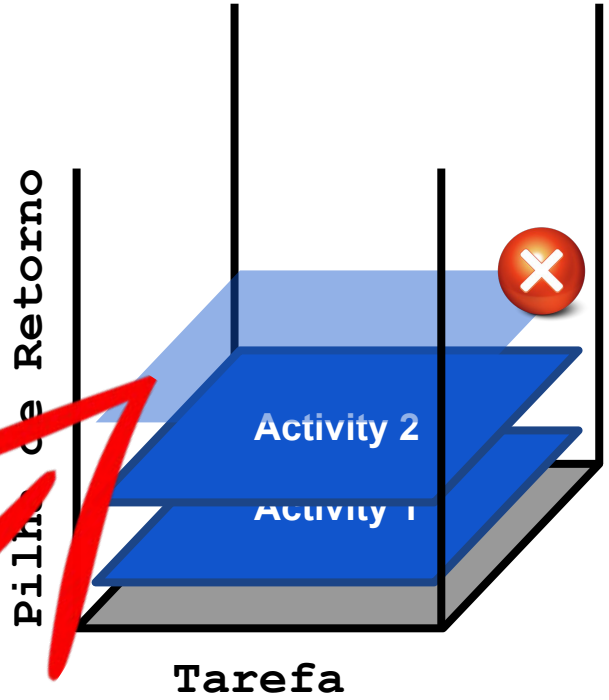
Pilha de Retorno





⇒ Isso fez com que a pilha de retorno esteja carregada com três Activities, sendo que a primeira e a terceira são instâncias da mesma Activity de listagem de pessoas.

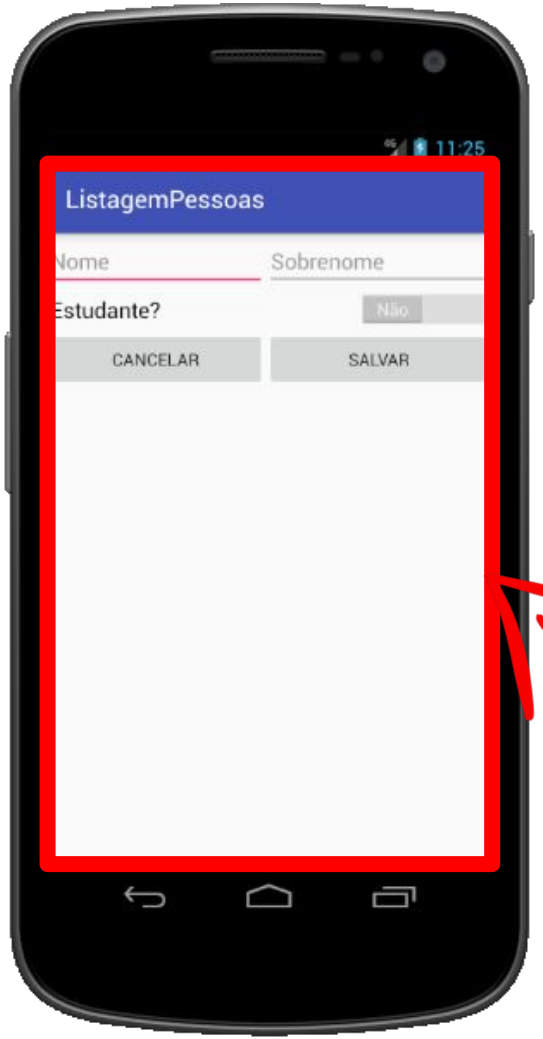
Pilha de Retorno

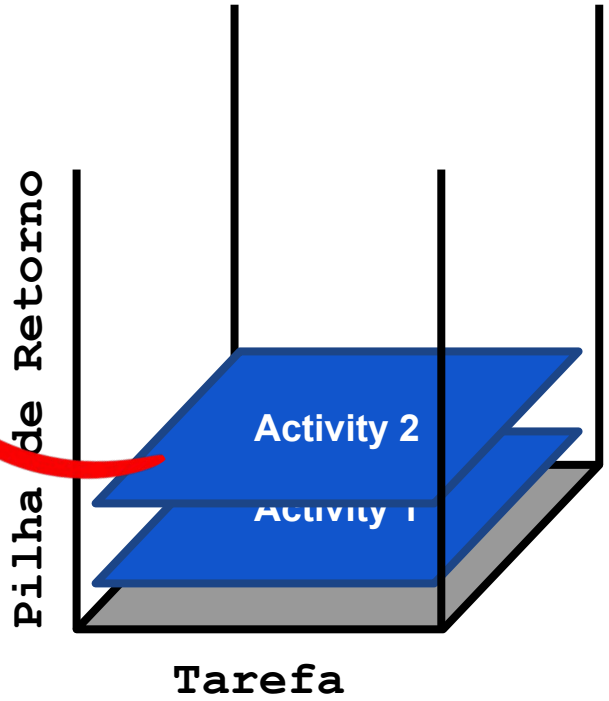


⇒ Por fim, ao clicar no Back Button, o Android destrói a terceira **Activity** e exibe na tela a próxima **Activity** empilhada.

Pilha de Retorno

Qual é o erro?



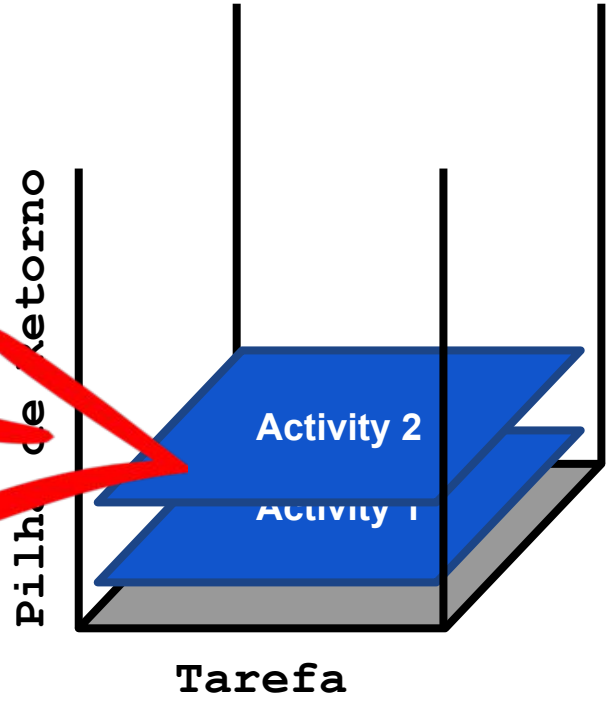


⇒ O erro na lógica ocorreu porque quando clicamos em *Cancelar*, ao invés de fechar a segunda **Activity**, nós solicitamos (via **Intent**) a criação de uma nova **Activity**.

Como resolver?

Trocando a sintaxe de abertura de uma **Activity** pela sintaxe de fechamento, através da chamada ao método *finish()*

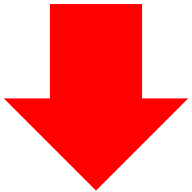
Pilha de Retorno



Fechando uma Activity

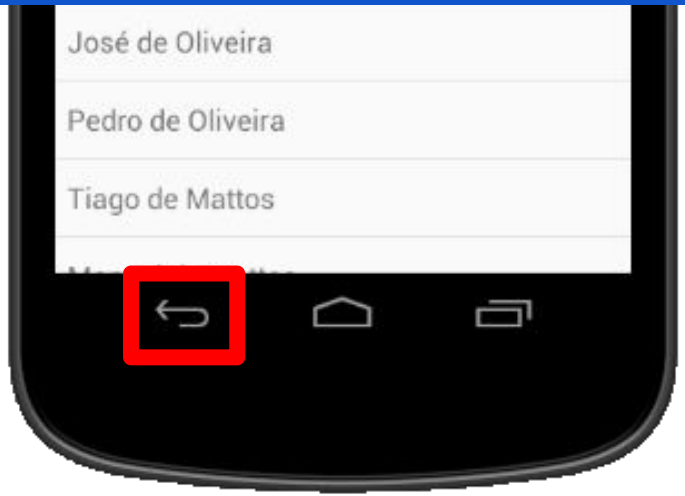
Troque a **Intent** e o método *startActivity()* pelo método *finish()*

```
private void voltarParaListagem() {
    Intent voltar = new Intent( packageContext: this,ListagemActivity.class);
    startActivity(voltar);
}
```



```
private void voltarParaListagem() {
    finish();
}
```

Discussão em Sala

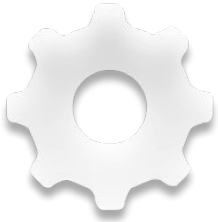


Rode o aplicativo, utilize os botões *Novo* e *Cancelar* e em seguida o *Back Button*.
Agora está tudo funcionando conforme o esperado!

Incluindo pessoas na listagem principal

Guardando as *Pessoas* em memória

Além do **Modelo de Domínio**, **Controller** e **Apresentação**, que já estudamos, é muito comum trabalhar com mais uma camada chamada **Repositório** (ou **DAO**). Esta camada centraliza o acesso ao banco de dados do aplicativo.

	Apresentação	Controller	Modelo	Repositorio
MVC				
Android	activity_listagem.xml 	ListagemActivity.java 	Pessoa.java 	Repositorio.java 

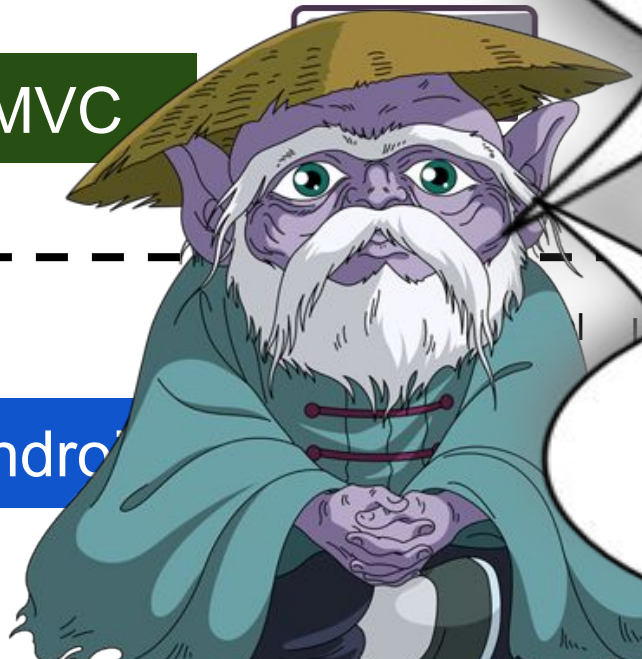
Guardando as *Pessoas e*

Além do **Modelo de Domínio** [illegible] studamos, é muito comum trabalhar com [illegible] (ou **DAO**). Esta camada centraliza o ac[illegible]

Por enquanto **não** vamos utilizar um banco real. No futuro voltaremos a tratar do assunto!

Apresentaçã[illegible] **Repositorio**

MVC



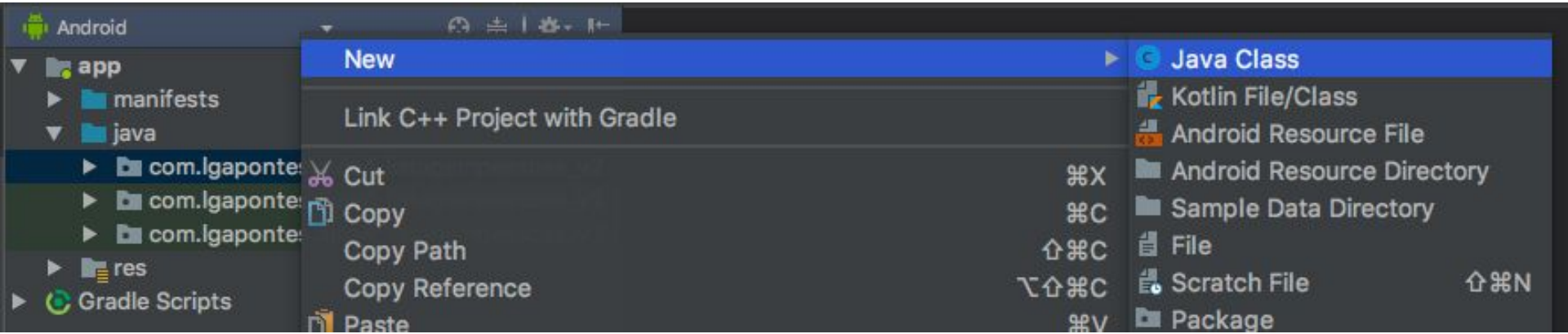
Vamos guardar os dados em memória!

Androi[illegible]

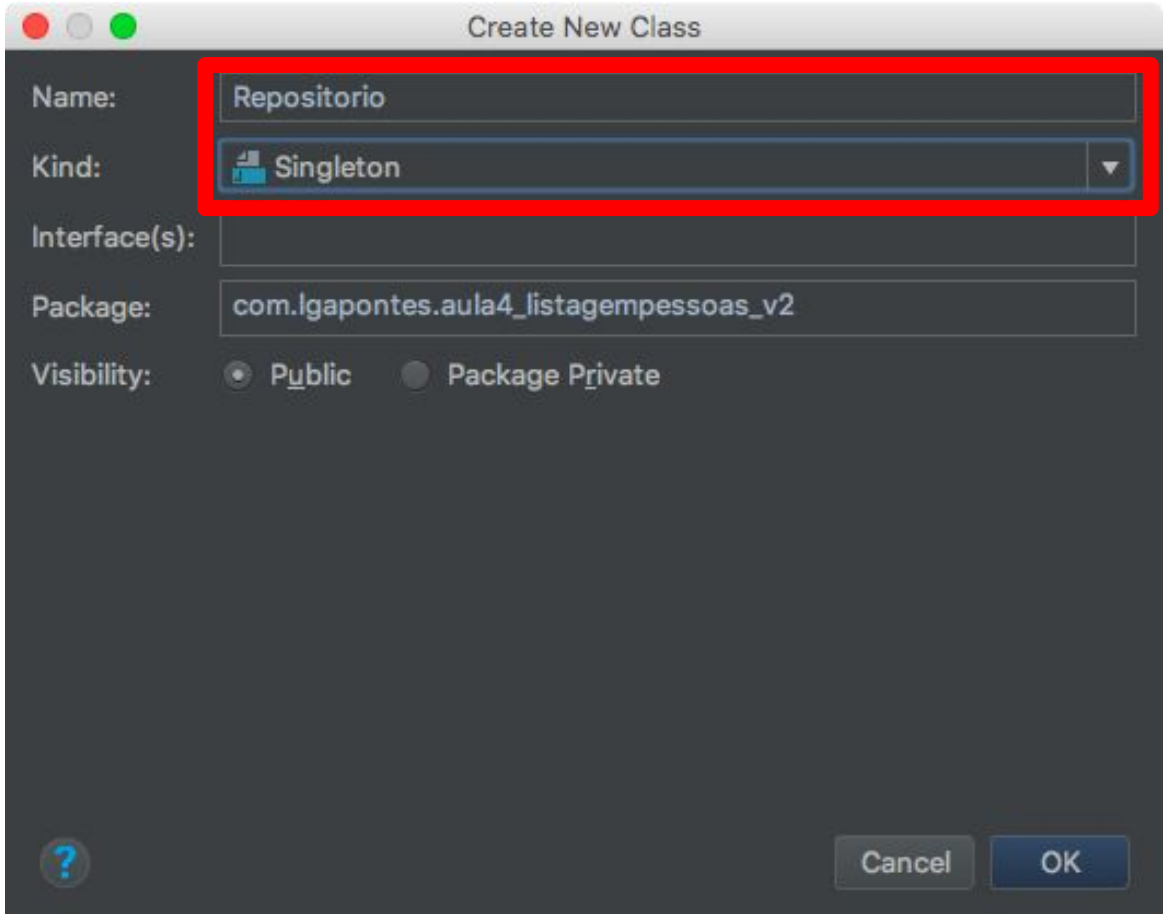
soa.java

Repositorio.java

Criando a classe de repositório



Criando a classe de repositório



Dica Extra

Uma das abordagens utilizadas para criar repositórios é trabalhar com o padrão de projeto (GoF) **Singleton**.

Mais detalhes?

Singleton
- singleton : Singleton
- Singleton() + getInstance() : Singleton

Criando a classe de repositório


```java
public class Repositorio {

    private static final Repositorio singleton = new Repositorio();
    public static Repositorio getInstance() { return singleton; }

    private List<Pessoa> pessoas;

    private Repositorio() {
        this.pessoas = new ArrayList<Pessoa>();
    }

    public List<Pessoa> listar() {
        return this.pessoas;
    }

    public void inserir(Pessoa pessoa) {
        this.pessoas.add(pessoa);
    }
}
```

Alterando a *ListagemActivity*

Apague o método *criarPessoas()* e altere sua invocação no método *onCreate()* pela chamada do método *listar()* do Singleton **Repositorio.**


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

    List<Pessoa> pessoas = Repositorio.getInstance().listar();

    ListView listView = (ListView) findViewById(R.id.lista_pessoas);

    ArrayAdapter<Pessoa> adapter = new ArrayAdapter<>(
            context: this,
            android.R.layout.simple_list_item_1,
            pessoas
    );
    listView.setAdapter(adapter);
}
```

Por enquanto, se você executar o projeto, nenhuma pessoa será exibida.

Isso ocorre porque nosso repositório está vazio.

Alterando a classe Pessoa

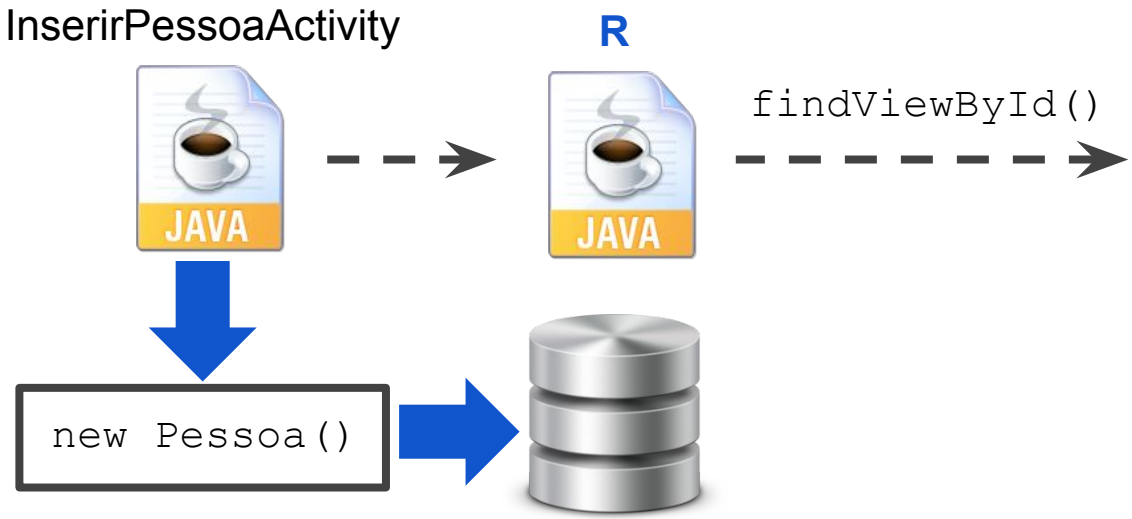
Vamos alterar a classe **Pessoa** para incluir o atributo *estudante*.

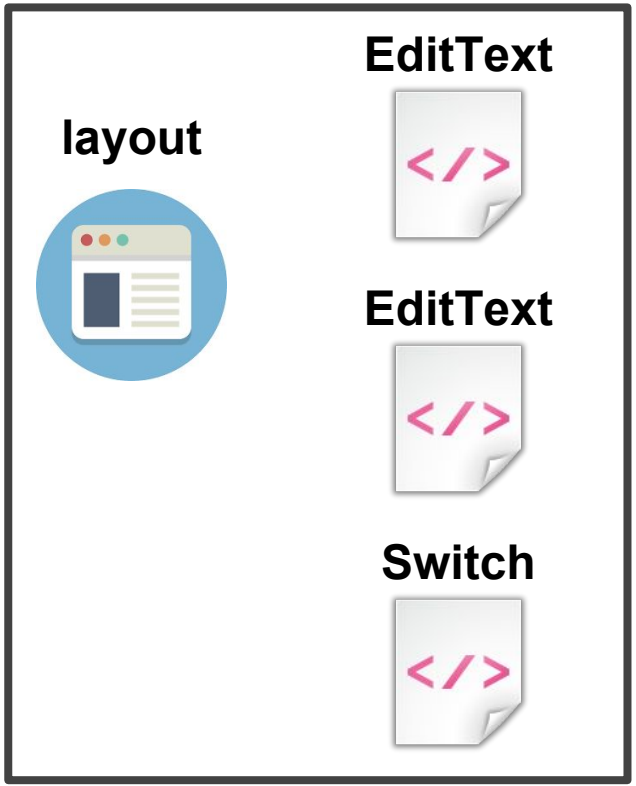

```java
public class Pessoa {

    private String nome;
    private String sobrenome;
    private Boolean estudante;

    public Pessoa(String nome, String sobrenome, Boolean estudante) {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.estudante = (estudante == null) ? false : estudante;
    }

    @Override
    public String toString() {
        return this.nome + " " + sobrenome +
                (this.estudante ? " (estudante)" : "");
    }

}
```

Pessoa
- nome: String
- sobrenome: String
- estudante: Boolean

Alterando a classe InserirPessoaActivity

No clique do botão *Salvar*, vamos obter os dados dos **views** *nome*, *sobrenome* e *estudante*, criar um novo objeto **Pessoa**, e inserir no repositório.



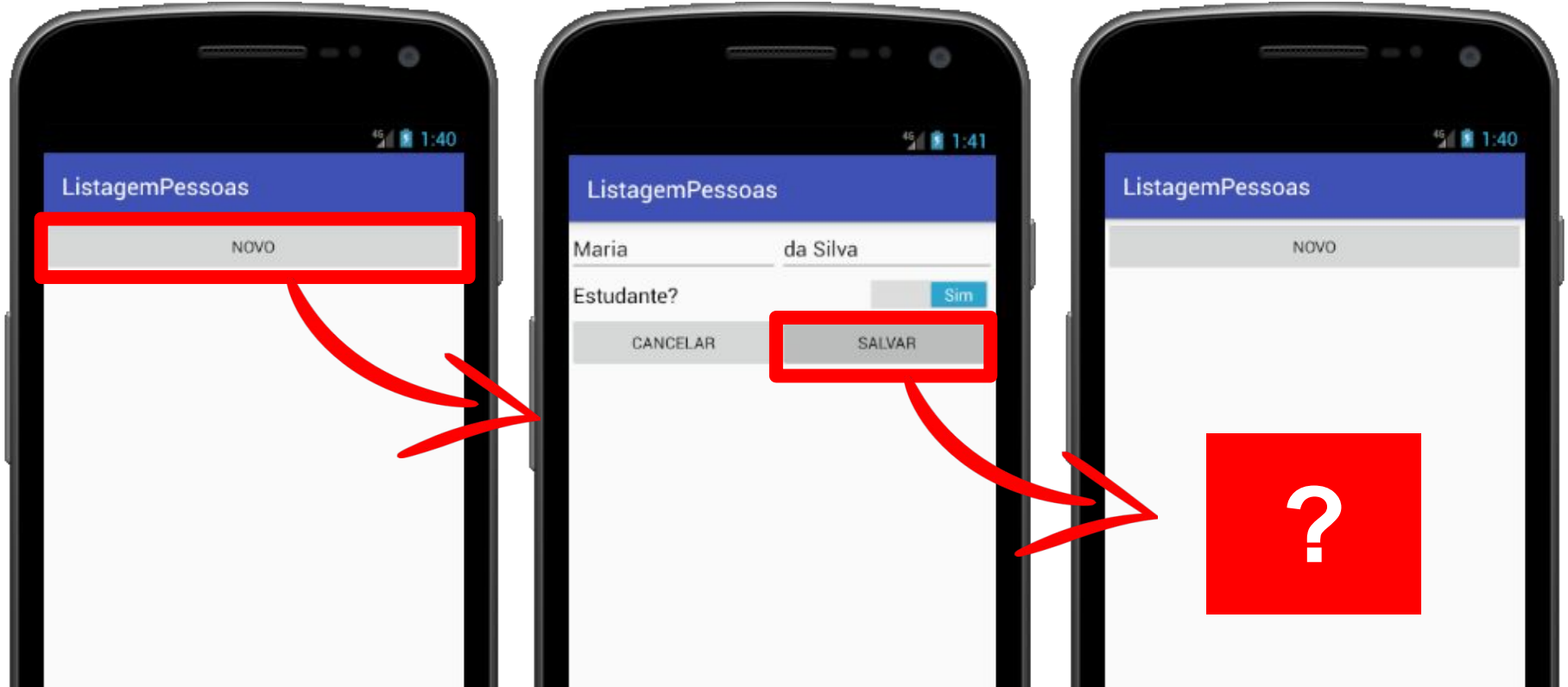


Alterando a classe InserirPessoaActivity


```
Button salvar = findViewById(R.id.incluir_pessoa_salvar);
salvar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        TextView textViewNome = findViewById(R.id.incluir_pessoa_nome);
        String nome = textViewNome.getText().toString();

        TextView textViewSobrenome = findViewById(R.id.incluir_pessoa_sobrenome);
        String sobrenome = textViewSobrenome.getText().toString();

        Switch switchEstudante = findViewById(R.id.incluir_pessoa_estudante);
        Boolean estudante = switchEstudante.isChecked();

        Pessoa pessoa = new Pessoa(nome,sobrenome,estudante);
        Repositorio.getInstance().inserir(pessoa);

        voltarParaListagem();
    }
});
```

Testando o App

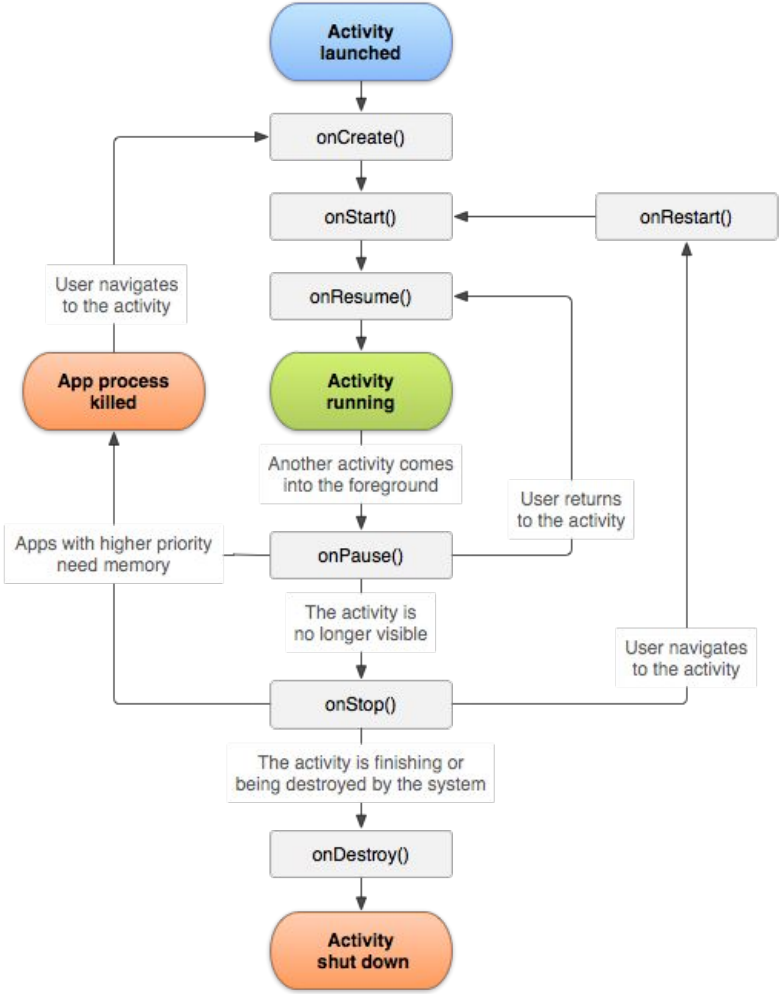
Por que a listagem não exibiu a nova pessoa cadastrada?



Ciclo de Vida das Activities

Ciclo de Vida da Activity

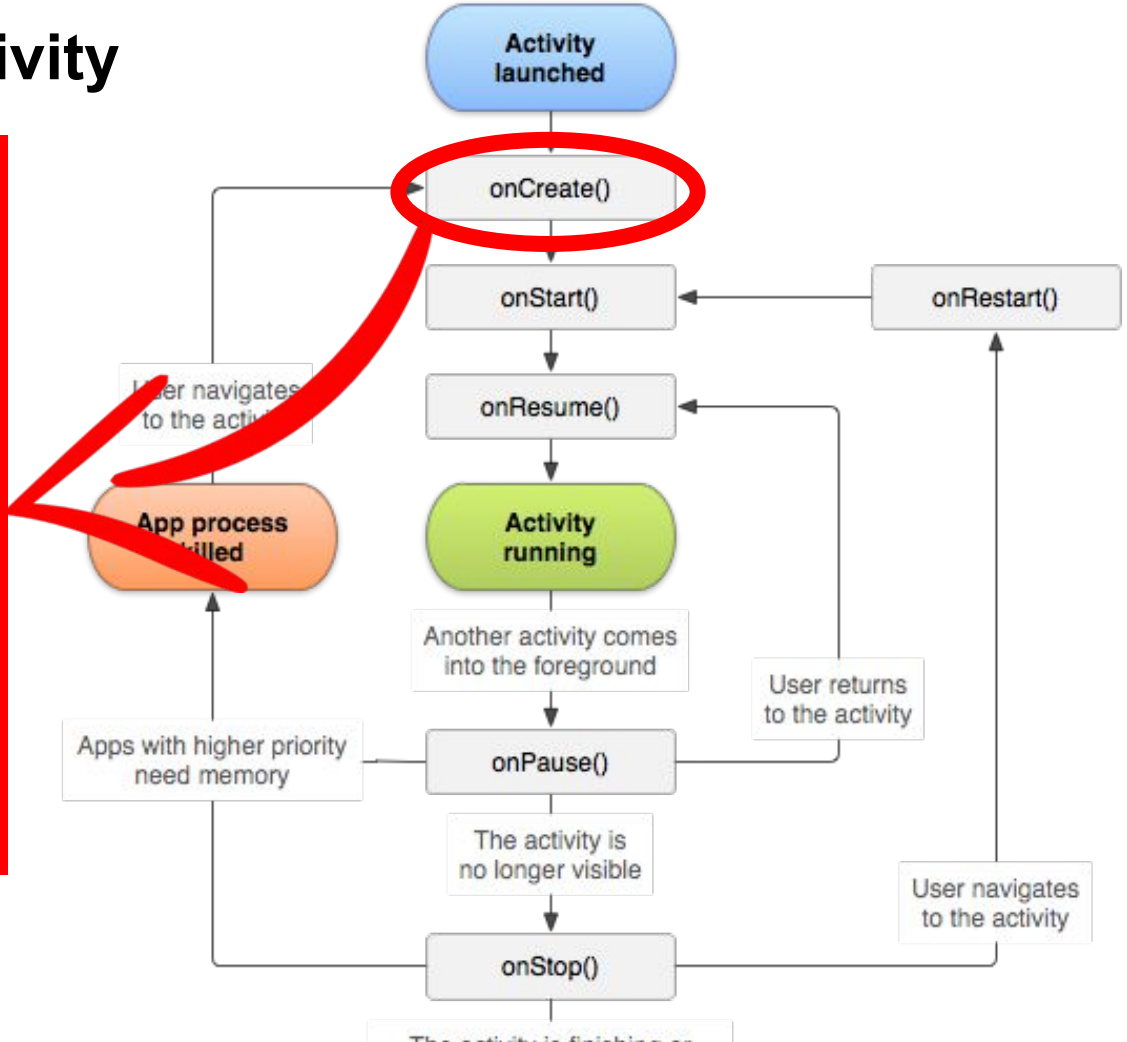
https://developer.android.com/guide/components/activities?hl=pt-br



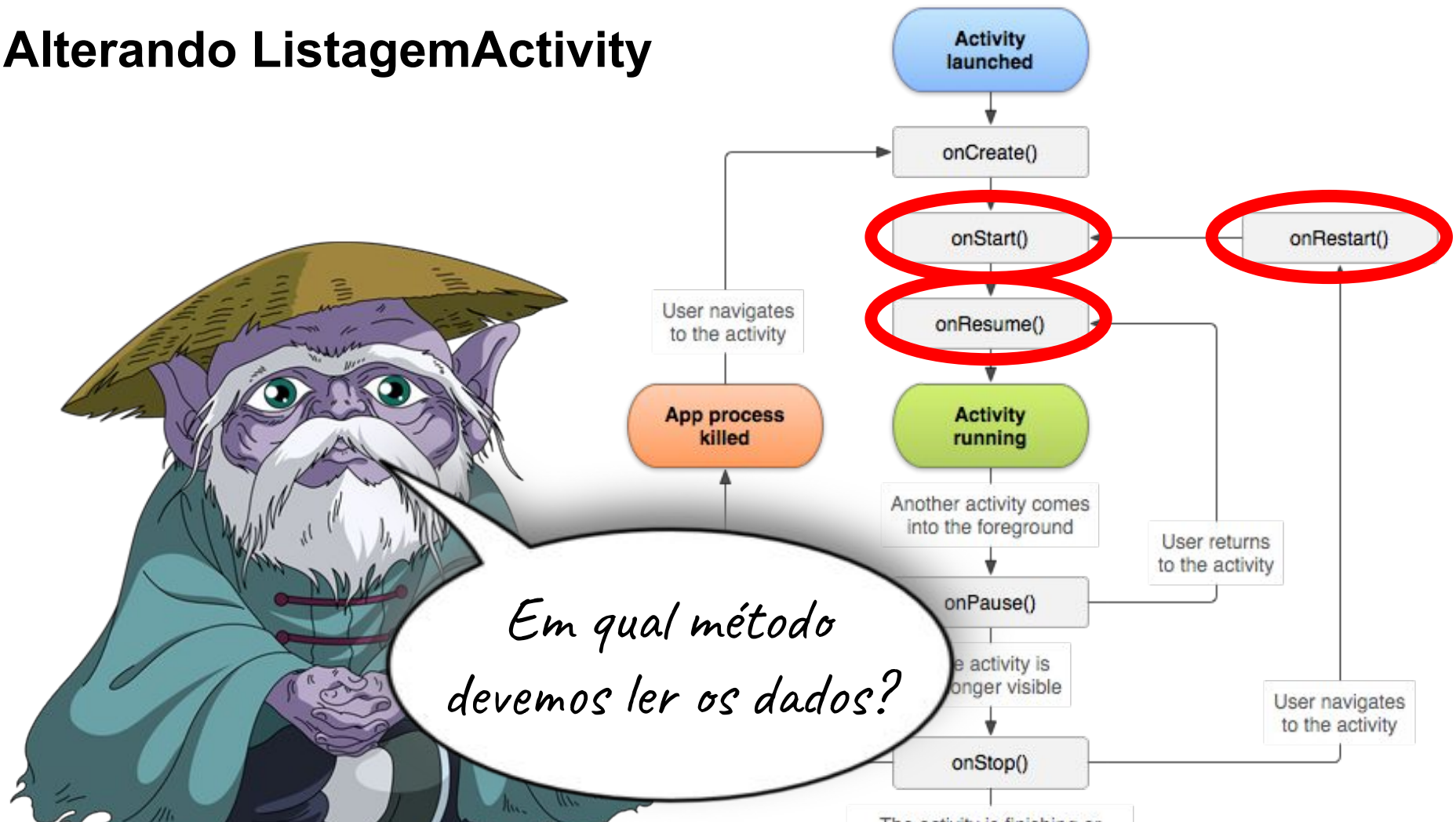
Alterando ListagemActivity

Nosso problema é que a leitura dos dados do Singleton **Repositorio** está codificada no método *onCreate()*.

Quando a **Activity** inserir pessoa é finalizada, a **Activity** listagem passa pelos métodos *onRestart(), onStart()* e *onResume()*, **mas não passa novamente pelo método *onCreate().***

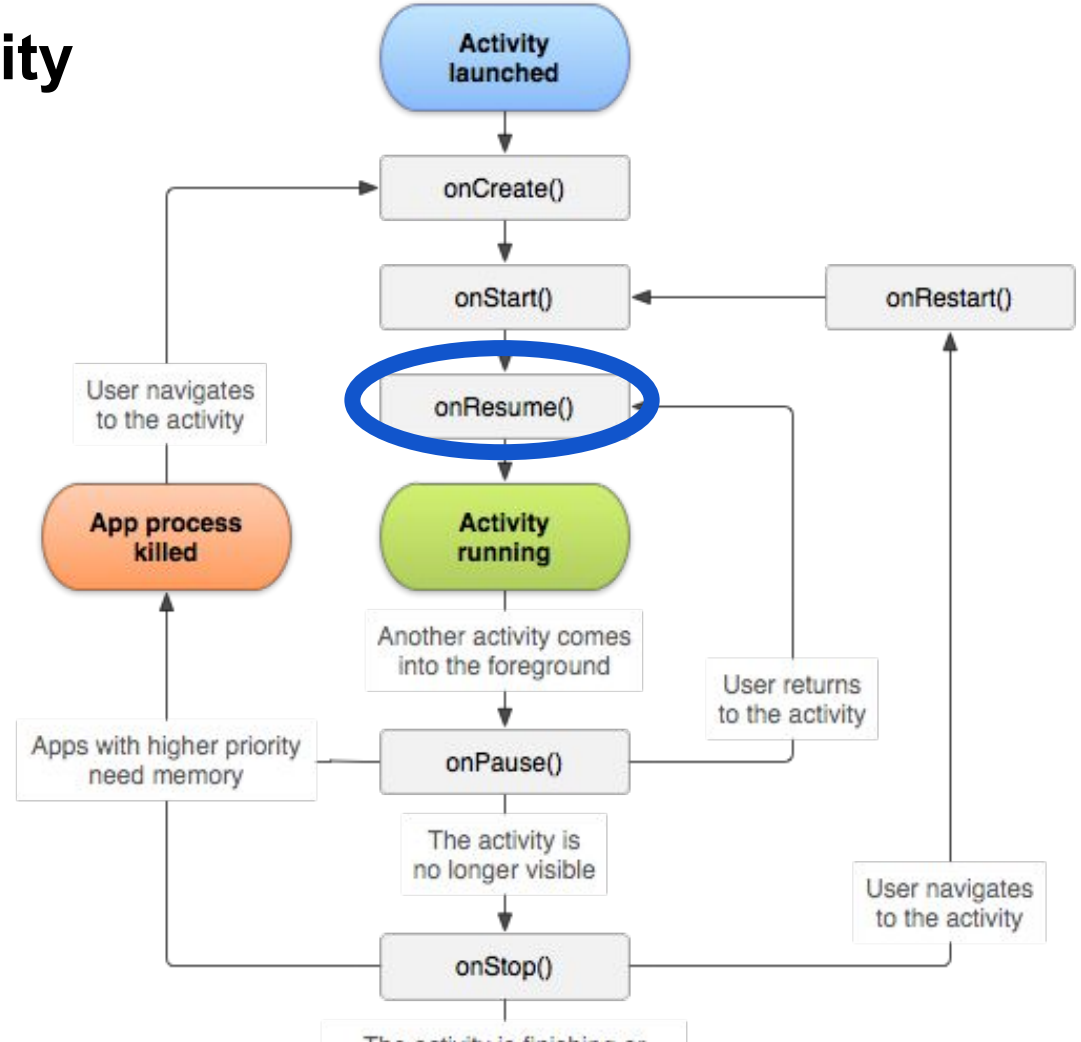


Alterando ListagemActivity



Alterando ListagemActivity

O melhor é ler os dados no método *onResume()*, pois mesmo que algo tenha sido alterado no *onPause()*, ao retornar para o foco do usuário a **Activity** listagem será atualizada.



```java
public class ListagemActivity extends AppCompatActivity {

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

    @Override
    protected void onResume() {
        super.onResume();
        List<Pessoa> pessoas = Repositorio.getInstance().listar();
        ListView listView = (ListView) findViewById(R.id.lista_pessoas);
        ArrayAdapter<Pessoa> adapter = new ArrayAdapter<Pessoa>(
                context: this,
                android.R.layout.simple_list_item_1,
                pessoas
        );
        listView.setAdapter(adapter);
    }

    public void clickBotaoNovo(View view) {
        Intent abrirInserirPessoa = new Intent( packageContext: this,InserirPessoaActivity.class);
        startActivity(abrirInserirPessoa);
    }

}
```

Testando o App

Agora sim, tudo ok!!!

Exercício em Sala

Execute o App no seu celular e cadastre três nomes, conforme abaixo.

Enviando dados para a Activity

Enviando dados para a Activity

E se quiséssemos alterar o dado de uma pessoa?

Poderíamos, por exemplo, ao clicar em uma das pessoas da lista abrir a Activity de inserir pessoas com os dados já preenchidos. Neste caso, ao clicar em salvar, ao invés de criar um novo registro, devemos atualizar o registro clicado.

Enviando dados para a Activity

PASSO 1: Verificar qual item da lista foi clicado

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> lista, View item, int position, long id) {
        ListView listView = (ListView) lista;
        Pessoa pessoa = (Pessoa) listView.getItemAtPosition(position);
        Toast.makeText( context: ListagemActivity.this, pessoa.toString(), Toast.LENGTH_SHORT).show();
    }
});
```

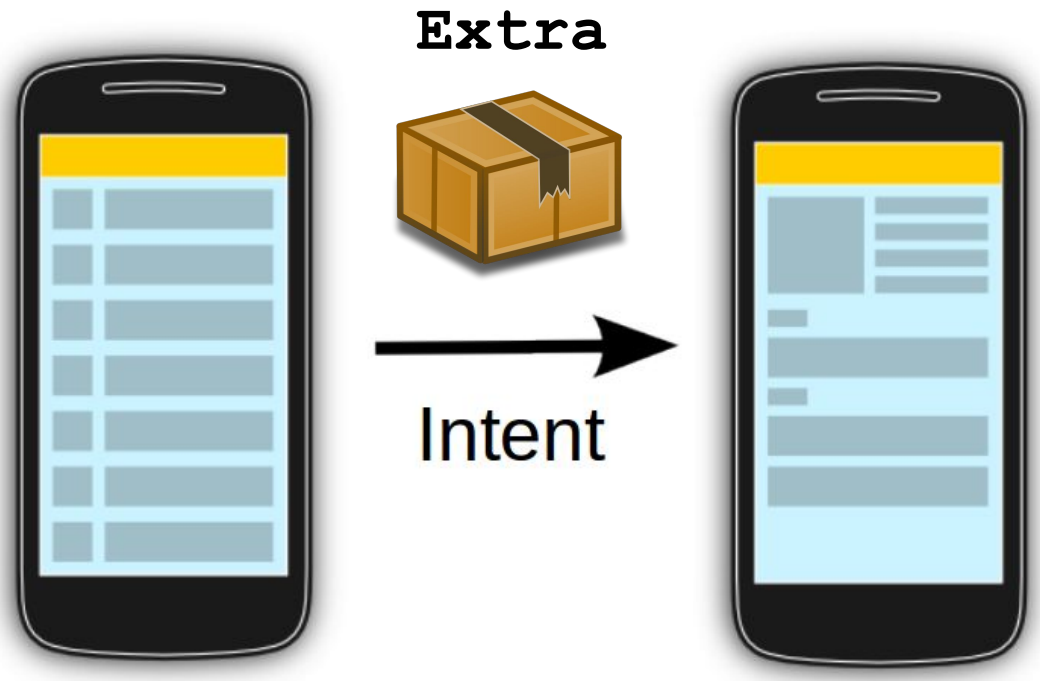
O método *onItemClick()* recebe 4 parâmetros:
`lista:` a própria lista cujos itens dispararam o evento quando clicados
`item:` uma View representando o item.
`position:` o índice do item clicado.
`id:` o ID do item clicado (útil em implementações particulares de um Adapter).
Como "guardamos" objetos Pessoa dentro dos itens, podemos obtê-los através do método *getItemAtPosition()* do *ListView* passando como parâmetro a `position` que foi clicada.

Enviando dados para

PASSO 1: Verificar qual item da lista foi clicado

Teste e veja que já está exibindo um Toast com os dados da pessoa clicada!

Melhorando o uso das Intents



É possível passar dados pela *Intent* através do método *putExtra()*. Este método recebe dois parâmetros:

`name`: identificador que será usado pela nova Activity para recuperar o valor.

`value`: objeto contendo os dados. Deve implementar a interface *Serializable*.

PASSO 2: Altere a classe *Pessoa* para implementar a interface *Serializable*.

```
public class Pessoa implements Serializable {

    private String nome;
    private String sobrenome;
    private Boolean estudante;

    public Pessoa(String nome, String sobrenome, Boolean estudante) {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.estudante = (estudante == null) ? false : estudante;
    }

    @Override
    public String toString() {
        return this.nome + " " + sobrenome +
                (this.estudante ? " (estudante)" : "");
    }

}
```

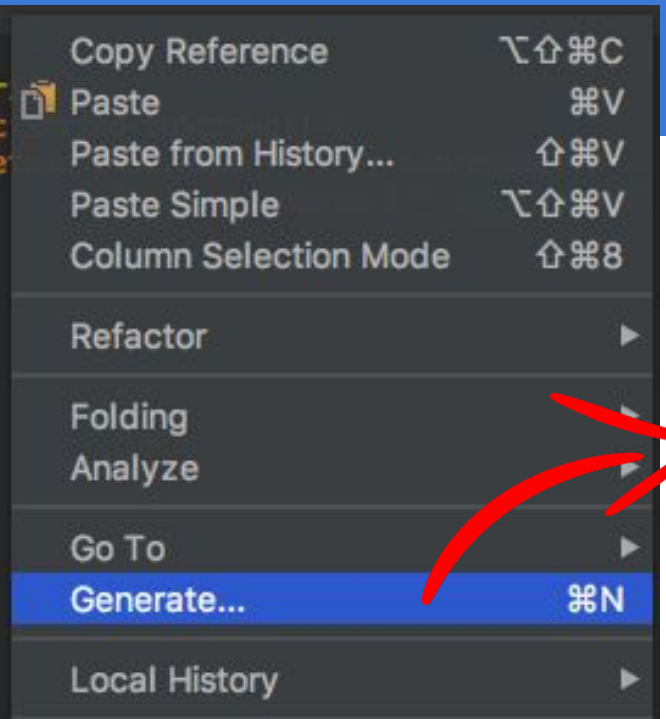
Enviando dados para a Activity

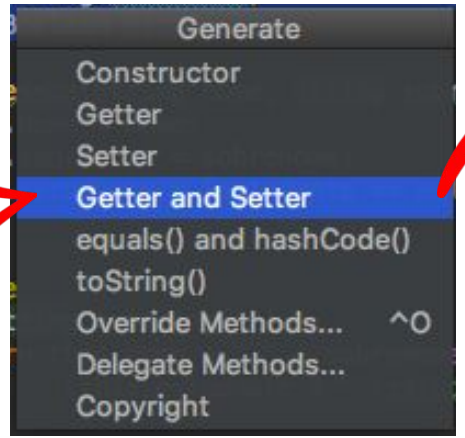
PASSO 3: Vamos acrescentar uma propriedade chamada *index* na classe Pessoa.
Esta propriedade receberá o índice da Pessoa na lista de pessoas do repositório. Esta informação servirá para informarmos qual pessoa deve ser alterada.

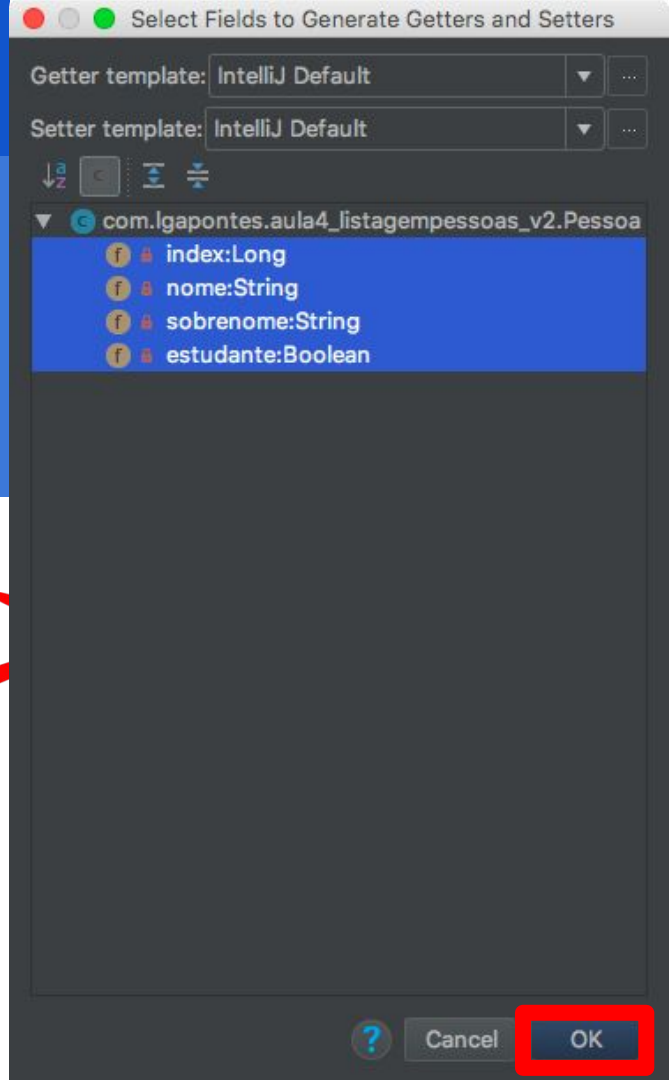
```
public class Pessoa implements Serializable {

    private Long index;
    private String nome;
    private String sobrenome;
    private Boolean estudante;

    public Pessoa(String nome, String sobrenome, Boolean estudante) {
        this.nome = nome;
        this.sobrenome = sobrenome;
        this.estudante = (estudante == null) ? false : estudante;
    }

    @Override
    public String toString() {
        return this.nome + " " + sobrenome +
                (this.estudante ? " (estudante)" : "");
    }

}
```

PASSO 4: Aproveite para criar os métodos *getters* e *setters* (que serão necessários na *InserirPessoaActivity*)

Clique com o botão direito no trecho onde serão inseridos os métodos. Selecione a opção "*Generate...*". Em seguida, selecione "Getter and Setter". Por fim, selecione os atributos e clique em "OK"







Enviando dados para a Activity

PASSO 5: Altere o método *inserir()* da classe repositório para acrescentar o *index* da nova pessoa.

Crie também um método chamado *alterar()* que seja capaz de atualizar a pessoa a partir do índice.

```
public class Repositorio {

    private static final Repositorio singleton = new Repositorio();
    public static Repositorio getInstance() { return singleton; }

    private List<Pessoa> pessoas;

    private Repositorio() { this.pessoas = new ArrayList<Pessoa>(); }

    public List<Pessoa> listar() { return this.pessoas; }

    public void inserir(Pessoa pessoa) {
        int index = this.pessoas.size();
        pessoa.setIndex(Long.valueOf(index));
        this.pessoas.add(pessoa);
    }

    public void alterar(Pessoa pessoa) {
        int index = pessoa.getIndex().intValue();
        pessoas.set(index,pessoa);
    }
}
```

Enviando dados para a Activity

PASSO 6: Altere a implementação do método *setOnItemClickListener()* para criar uma *Intent* apontando para *InserirPessoaActivity.class*. Antes de iniciar a *Activity* pelo método *startActivity()*, informe o objeto pessoa que será enviado através do método *putExtra()*, presente na própria *Intent*.

```
listView.setOnItemClickListener((lista, item, position, id) → {
        ListView listView = (ListView) lista;
        Pessoa pessoa = (Pessoa) listView.getItemAtPosition(position);
        Intent intent = new Intent( packageContext: ListagemActivity.this,InserirPessoaActivity.class);
        intent.putExtra( name: "pessoa",pessoa);
        startActivity(intent);
});
```

Enviando dados para a Activity

PASSO 7: Os últimos ajustes serão aplicados na *InserirPessoaActivity*. Primeiramente vamos criar propriedades para os TextViews e Switch, e para guardar uma instância da classe pessoa.

```
private Pessoa pessoa;
private TextView textViewNome;
private TextView textViewSobrenome;
private Switch switchEstudante;

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

    textViewNome = (TextView) findViewById(R.id.incluir_pessoa_nome);
    textViewSobrenome = (TextView) findViewById(R.id.incluir_pessoa_sobrenome);
    switchEstudante = (Switch) findViewById(R.id.incluir_pessoa_estudante);
```

Enviando dados para a Activity

PASSO 8: Vamos agora verificar se a *Intent* de criação da Activity possui ou não um objeto Pessoa guardado no campo *Extra*. Como este campo é um *Serializable*, o compilador Java nos obriga a fazer o casting para a classe Pessoa.

```
Intent intent = getIntent();
pessoa = (Pessoa) intent.getSerializableExtra( name: "pessoa");
if (pessoa != null) {
    textViewNome.setText(pessoa.getNome());
    textViewSobrenome.setText(pessoa.getSobrenome());
    switchEstudante.setChecked(pessoa.getEstudante());
}
```

A regra é simples: se existir uma pessoa, vamos obter seus dados e popular os respectivos views do layout.

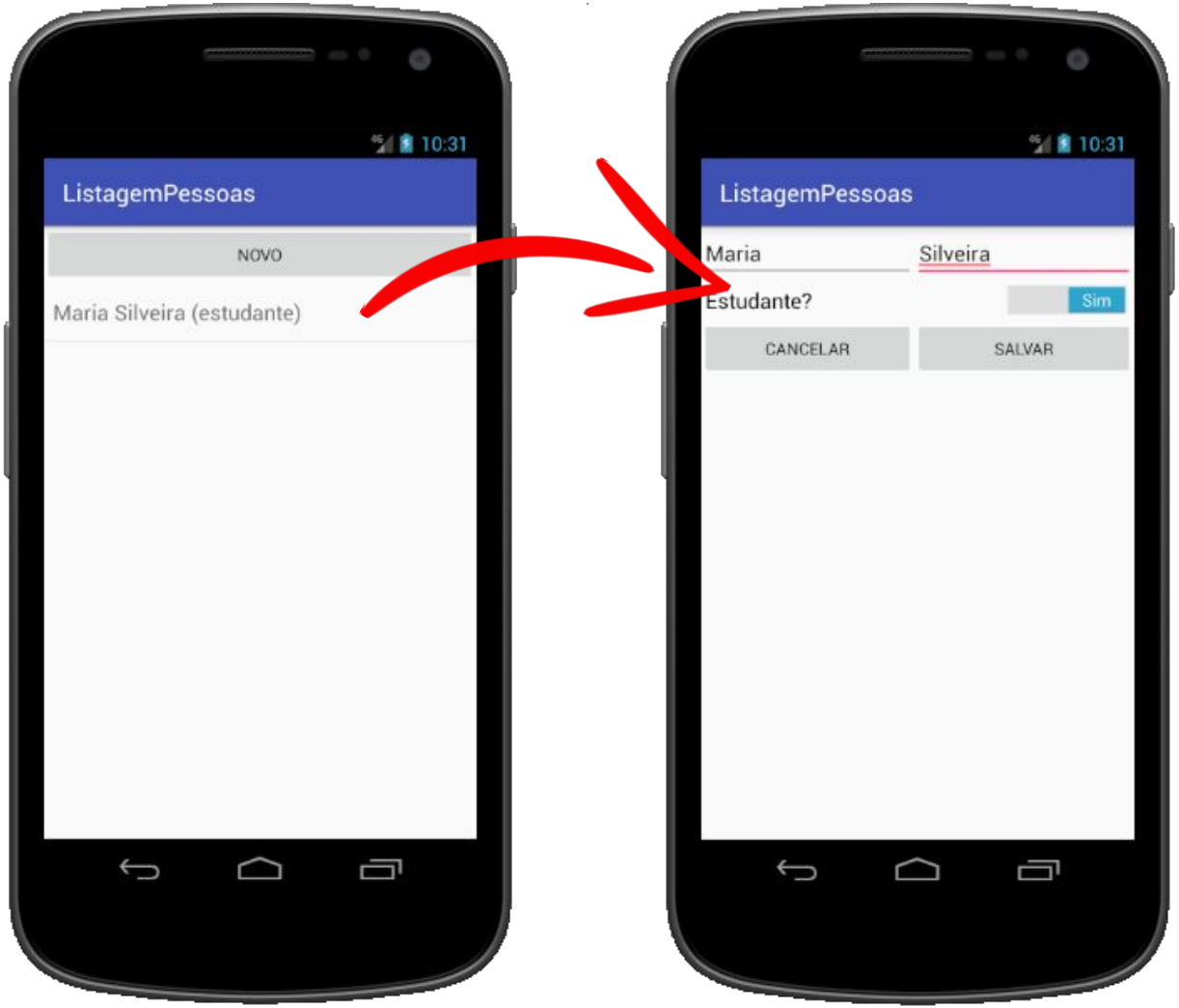
Enviando dados para a Activity

PASSO 9: Por fim, no *Listener* de tratamento do botão salvar, verificamos se o objeto pessoa existe.

Se não existir, vamos invocar o método *inserir()* do repositório.

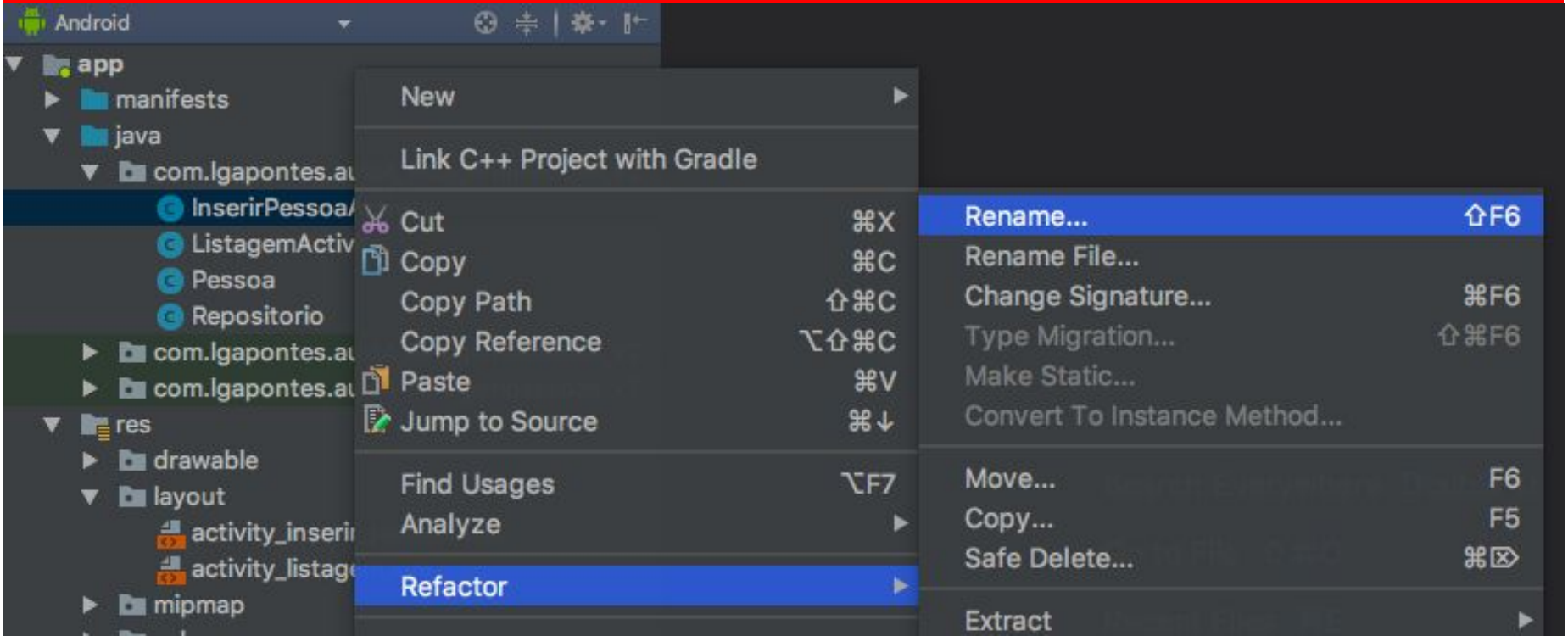
Se existir, devemos populá-lo com os dados dos views e enviá-lo para o repositório pelo método *alterar()*.

```
Button salvar = findViewById(R.id.incluir_pessoa_salvar);
salvar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        String nome = textViewNome.getText().toString();
        String sobrenome = textViewSobrenome.getText().toString();
        Boolean estudante = switchEstudante.isChecked();

        if (pessoa == null) {
            pessoa = new Pessoa(nome,sobrenome,estudante);
            Repositorio.getInstance().inserir(pessoa);
        } else {
            pessoa.setNome(nome);
            pessoa.setSobrenome(sobrenome);
            pessoa.setEstudante(estudante);
            Repositorio.getInstance().alterar(pessoa);
        }

        voltarParaListagem();
    }
});
```


10:31
ListagemPessoas
NOVO
Maria Silveira (estudante)
10:31
ListagemPessoas
Maria
Silveira
Estudante?
Sim
CANCELAR
SALVAR

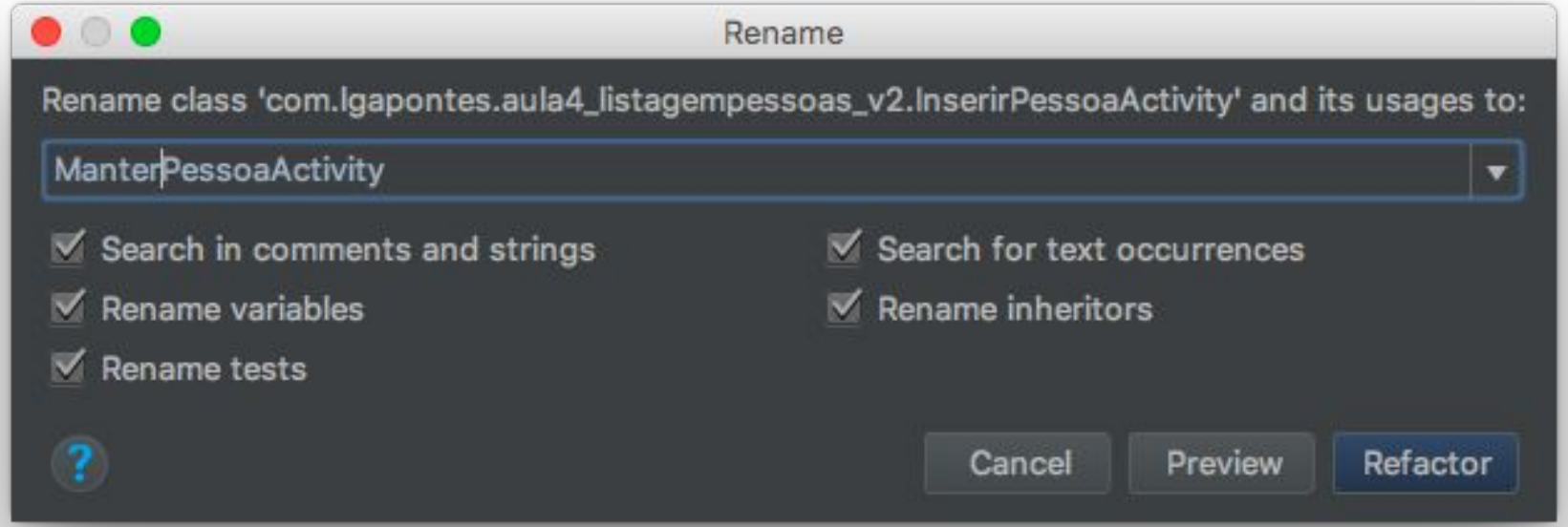
Um último detalhe...

Note que **semanticamente** o nome *InserirPessoaActivity* não é mais apropriado. Podemos refatorá-lo através do menu.

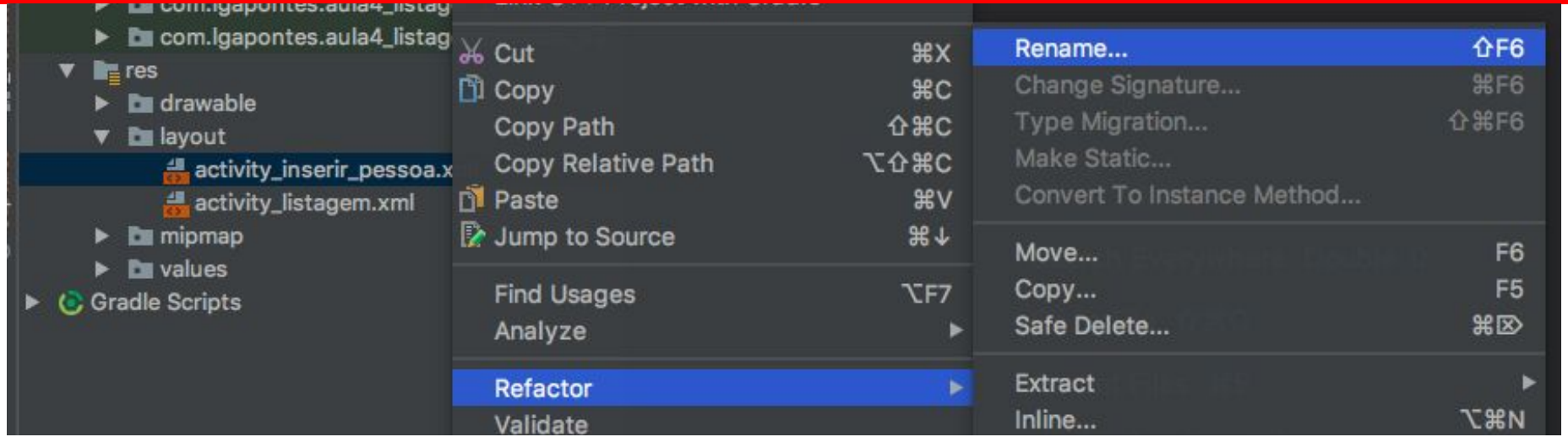


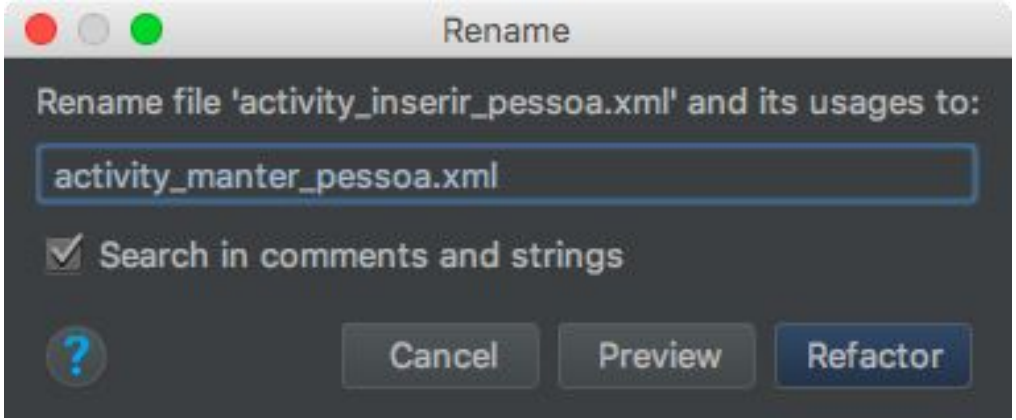
Um último detalhe...

Um nome mais apropriado seria *ManterPessoaActivity*



Faça o mesmo processo para o layout XML





Exercício em Sala

Execute o App no seu celular, cadastre e altere alguns registros...

Obrigado!