

Projetos Android Básicos



Prof. Dr. João Paulo Lemos
Escola
Copyright © 2022

Conteúdo

- Primeiros projetos com Android;
- Trocando a UI;
- Eventos;
- String resource;
- Internacionalização;
- Toast;
- Hint;
- Intent;
- Método putExtra;
- Shared Preferences;
- Log;
- Github;
- Menus;
- Listeners;
- AlertDialog;
- Spinner;
- String-array;
- Radio;
- Jogo da memória;
- Resources de Imagens.

Activity

- Classe que representa cada tela de um App;
- Significa uma ‘atividade’ realizada no App;
- Precisa conter pelo menos o método **onCreate**:
 - Linha 11: executa o método onCreate da classe Pai (AppCompatActivity);
 - Linha 12: exibe o layout ‘activity_main.xml’;
- **Package** é o pacote atual, onde o arquivo atual está armazenado;
- **Import** são as classes externas (bibliotecas) importadas para uso na classe atual;
- **Bundle** é um pacote que permite transmitir parâmetros de uma activity para outra. Utilizaremos ele no futuro.

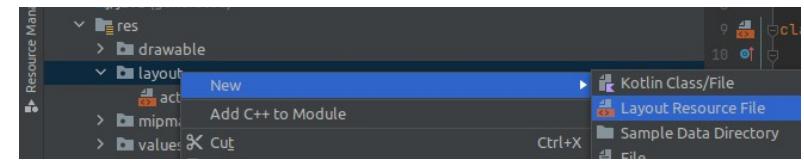
```
1 package br.com.jpescola.myapplication
2
3 import ...
4
5
6
7
8
9 class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.activity_main)
13     }
14 }
```

Activity x Layout

- Vimos anteriormente como manipular layouts na IDE e como criar telas com componentes visuais;
- Vimos também que as Activities são as classes que contém os códigos fonte e que manipulam as telas;
- Agora vamos substituir a tela (UI) exibida por uma Activity.

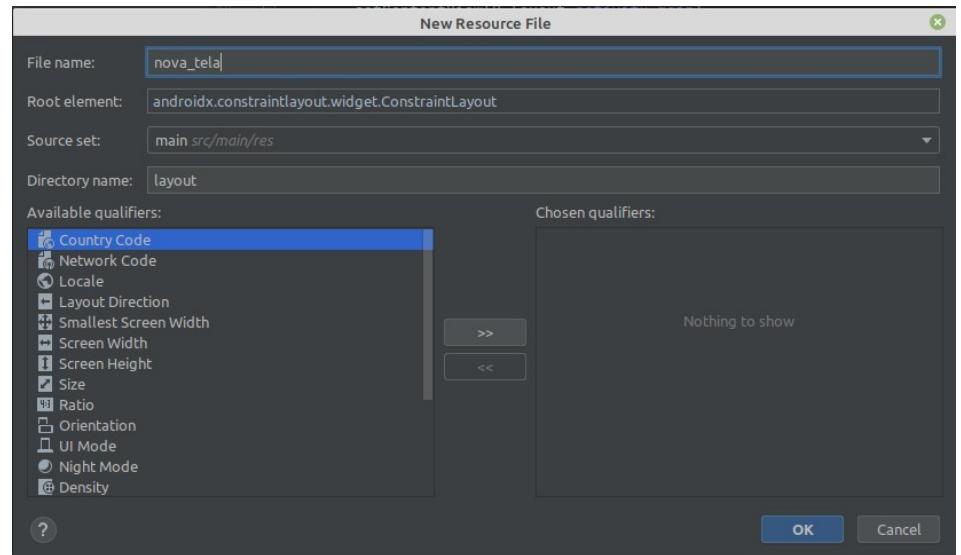
Trocando a UI

- Dê um duplo clique na pasta ‘res’ e depois na pasta ‘layout’;
- Veja que só temos o arquivo `activity_main.xml`;
- Vamos adicionar um layout ao nosso projeto:
 - Clique com o botão direito na pasta ‘layout’ e escolha New / Layout Resource File:



nova_tela.xml

- Defina o nome da nova para nova_tela e clique em OK:



Trocando a UI

- Agora que já temos uma segunda tela, ou seja, um segundo arquivo xml, vamos trocar a UI padrão do nosso App;
- Para isso, altere o id que está sendo passado por parâmetro na função **setContentView**:



The screenshot shows the Android Studio interface. On the left, the code editor displays the `MainActivity.kt` file:

```
9 class MainActivity : AppCompatActivity() {
10     override fun onCreate(savedInstanceState: Bundle?) {
11         super.onCreate(savedInstanceState)
12         setContentView(R.layout.)
13     }
14 }
```

On the right, the `Resource Manager` tool window is open, showing a list of resources:

Resource	Type
activity_main	Int
nova_tela	Int
support_simple_spinner_dropdown_item	Int
arrayOf	arrayOf(expr)

Execute o projeto e veja o resultado.

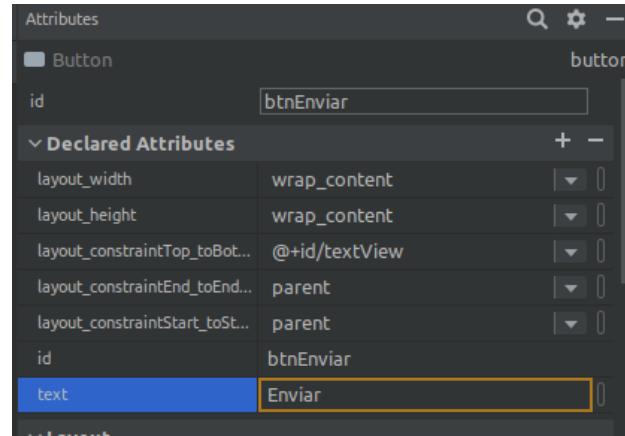
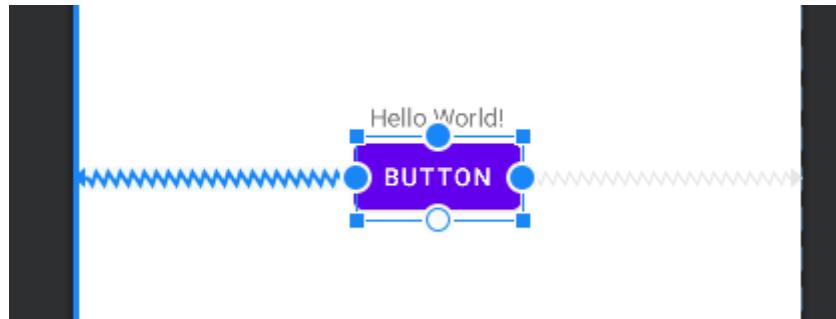
A4ex1

- Vamos criar um app que contém um botão e altera o rótulo ao clicar no botão;



A4ex1: UI

- Abra o design e adicione um botão;
- Defina as constraints acima, esquerda e direita;
- Defina o texto e o ID do botão.



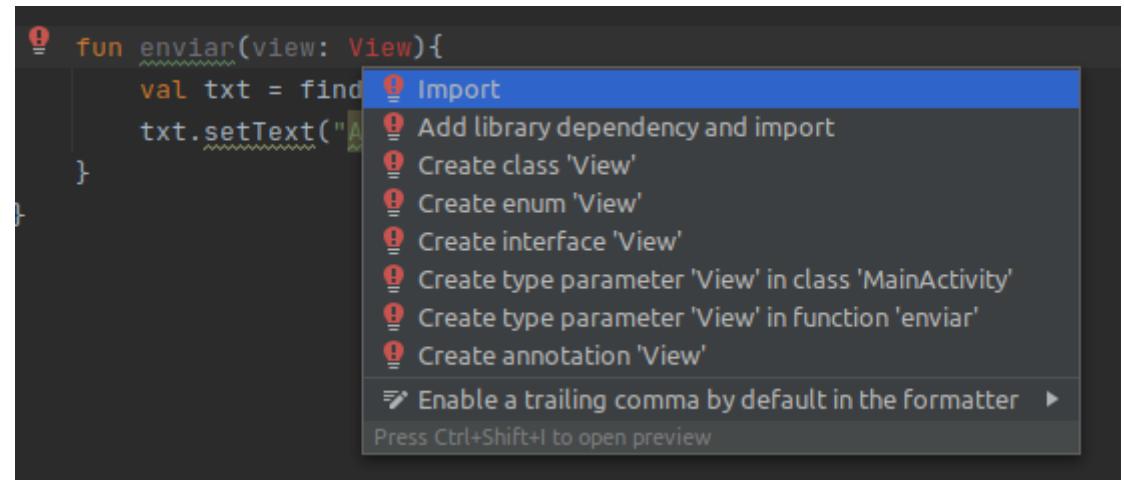
A4ex1: código fonte

- Crie a função “enviar”, responsável por alterar o código do rótulo:

```
1 package br.com.jpescola.myapplication  
2  
3 import ...  
4  
5 class MainActivity : AppCompatActivity() {  
6     override fun onCreate(savedInstanceState: Bundle?) {  
7         super.onCreate(savedInstanceState)  
8         setContentView(R.layout.activity_main)  
9     }  
10    fun enviar(view: View){  
11        val txt = findViewById<TextView>(R.id.textView)  
12        txt.text = "Ativado!"  
13    }  
14}  
15
```

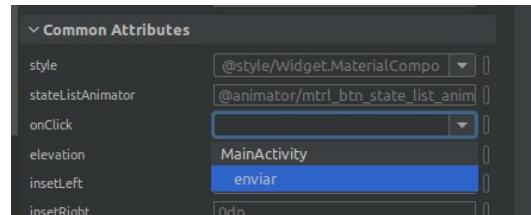
Importando classes

- Veja que a IDE não reconhece a classe View;
- Para importá-la, clique na lâmpada e escolha “Import” ou pressione Alt + Enter.



Definindo a função do botão

- Quando clicamos em um botão, queremos que algo aconteça;
- Para isso, primeiro devemos criar a função que será executada ao clicar no botão;
- Em seguida, devemos vincular o botão à função:



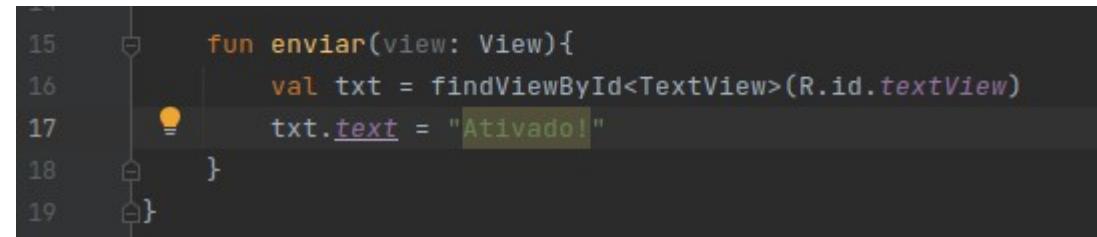
- Para uma função aparecer nessa lista, ela deve receber um objeto View como parâmetro:

```
15     fun enviar(view: View){  
16         val txt = findViewById<TextView>(R.id.textView)  
17         txt.text = "Ativado!"  
18     }  
19 }
```

A screenshot of the Java code editor showing a single-line function definition. The function is named 'enviar' and it takes a 'View' parameter. Inside the function, it finds a 'TextView' by its ID and changes its 'text' property to the string 'Ativado!'. The code is written in a standard Java syntax with some IDE-specific annotations like '@Style' and '@Animator'.

Entendendo o código “enviar”

- **fun**: define uma nova função;
- **enviar**: nome que escolhemos para a função;
- Parênteses: definem os argumentos obrigatórios para uso da função;
- **'view: View'**: argumento único definido = significa que será necessário passar um objeto View ao chamar a função;
- **val txt**: declara um objeto chamado txt;
- **findViewById**: método que recebe o ID como parâmetro e retorna um objeto do tipo definido no operador ‘diamond’;
- **R.id.textView**: busca na classe R o número inteiro armazenado que representa o ID do componente visual TextView incluído na UI anteriormente;
- **txt.text**: definindo o valor do atributo ‘text’ pertencente ao objeto ‘txt’;



```
15     fun enviar(view: View){  
16         val txt = findViewById<TextView>(R.id.textView)  
17         txt.text = "Ativado!"  
18     }  
19 }
```

A screenshot of the Android Studio code editor. The code is written in Java. It defines a function named 'enviar' that takes a 'View' parameter. Inside the function, it uses the 'findViewById' method to find a 'TextView' by its ID 'R.id.textView' and then sets its 'text' attribute to the string 'Ativado!'. The code is numbered from 15 to 19 on the left. A yellow lightbulb icon is positioned above the line 'txt.text = "Ativado!"', indicating a potential issue or suggestion.

Melhorando o projeto

- Agora vamos alterar o código fonte do botão para alternar o texto ao clicar:

```
9 class MainActivity : AppCompatActivity() {  
10  
11     lateinit var txt: TextView  
12  
13     override fun onCreate(savedInstanceState: Bundle?) {  
14         super.onCreate(savedInstanceState)  
15         setContentView(R.layout.activity_main)  
16         txt = findViewById(R.id.textView)  
17     }  
18  
19     fun enviar(view: View){  
20         if (txt.text == "Ativado!")  
21             txt.text = "desativado"  
22         else  
23             txt.text = "Ativado!"  
24     }  
25 }
```

Lateinit

- Late=atrasado, init=inicialização
- Sempre que precisarmos declarar um componente visual (view) em Kotlin, devemos declará-lo com a propriedade ‘lateinit’:
 - Permite que o componente seja declarado sem, obrigatoriamente, ser inicializado;
 - Possibilita que o Android initialize os componentes posteriormente, isso é necessário no caso das Views.

A4ex2

- Agora vamos alternar também o texto da barra superior (Title bar):

```
9  class MainActivity : AppCompatActivity() {  
10  
11     lateinit var txt: TextView  
12  
13    override fun onCreate(savedInstanceState: Bundle?) {  
14        super.onCreate(savedInstanceState)  
15        setContentView(R.layout.activity_main)  
16        txt = findViewById(R.id.textView)  
17    }  
18  
19    fun enviar(view: View){  
20        title = "desativado"  
21        if (txt.text == "Ativado!") {  
22            txt.text = "desativado"  
23            title = "desativado"  
24        }  
25        else {  
26            txt.text = "Ativado!"  
27            title = "Ativado!"  
28        }  
29    }  
30 }
```

Boas práticas: uso de constantes

- Caso necessitemos alterar o texto, teríamos vários lugares para alterar, podendo esquecer algum;
- Vamos definir constantes para organizar melhor o nosso código:

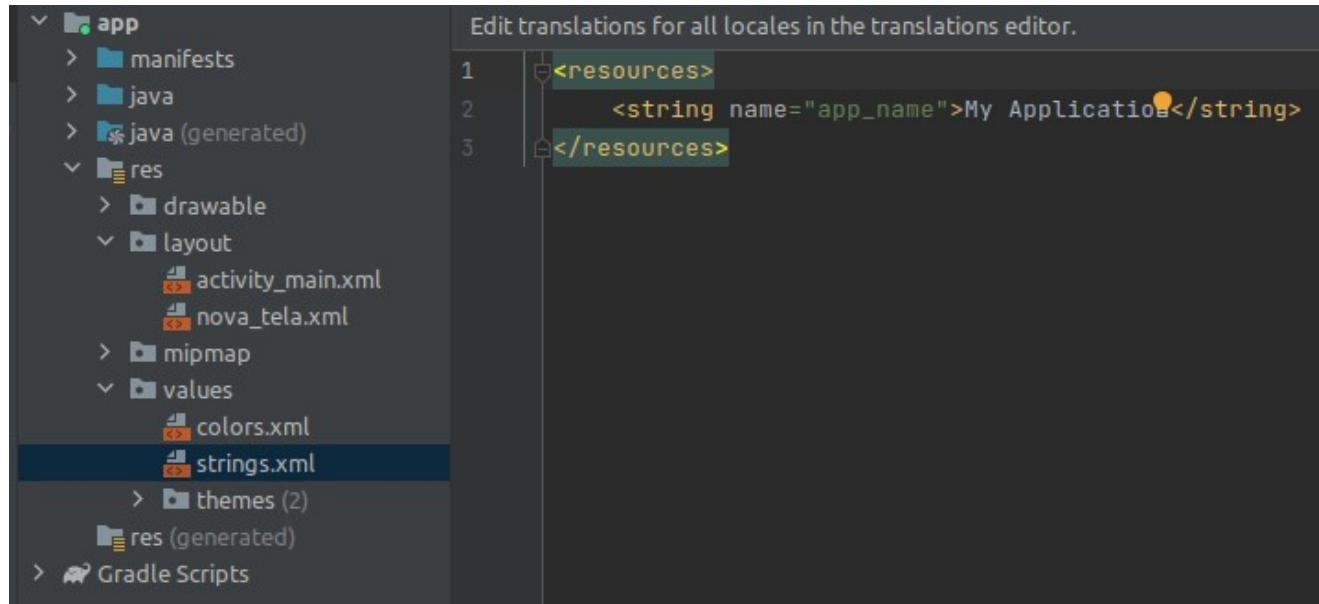
```
9  class MainActivity : AppCompatActivity() {  
10  
11     lateinit var txt: TextView  
12     val A = "ATIVADO!"  
13     val B = "DESATIVADO"  
14  
15     override fun onCreate(savedInstanceState: Bundle?) {  
16         super.onCreate(savedInstanceState)  
17         setContentView(R.layout.activity_main)  
18         txt = findViewById(R.id.textView)  
19     }  
20  
21     fun enviar(view: View){  
22         if (txt.text == A) {  
23             txt.text = B  
24             title = B  
25         }  
26         else {  
27             txt.text = A  
28             title = A  
29         }  
30     }  
31 }
```

Boas práticas: String Resource

- Para seguir o padrão de desenvolvimento Android, vamos utilizar String resources;
- Trata-se de um arquivo onde todas as Strings devem ser definidas;
- Assim, **não devemos** mais utilizar Strings dentro do código fonte;
- Esse recurso facilitará posteriormente o processo de internacionalização.

strings.xml

- Situado dentro da pasta ‘values’;
- Utilizado para definição das Strings que serão empregadas no App:



The screenshot shows the Android Studio project structure on the left and the strings.xml file content on the right.

Project Structure (Left):

- app

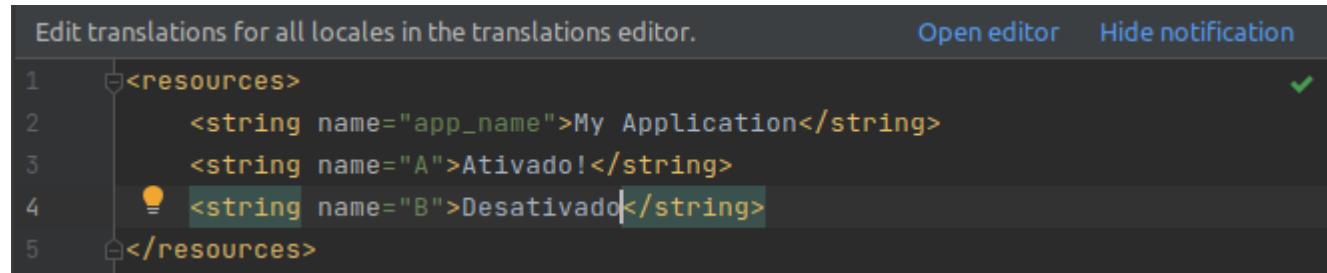
 - manifests
 - java
 - java (generated)
 - res
 - drawable
 - layout
 - activity_main.xml
 - nova_tela.xml
 - mipmap
 - values
 - colors.xml
 - strings.xml
 - themes (2)
 - res (generated)
 - Gradle Scripts

strings.xml Content (Right):

```
1 <resources>
2     <string name="app_name">My Application</string>
3 </resources>
```

Incluindo novas Strings

- Vamos incluir as Strings “Ativado!” e “Desativado” para uso no código fonte.



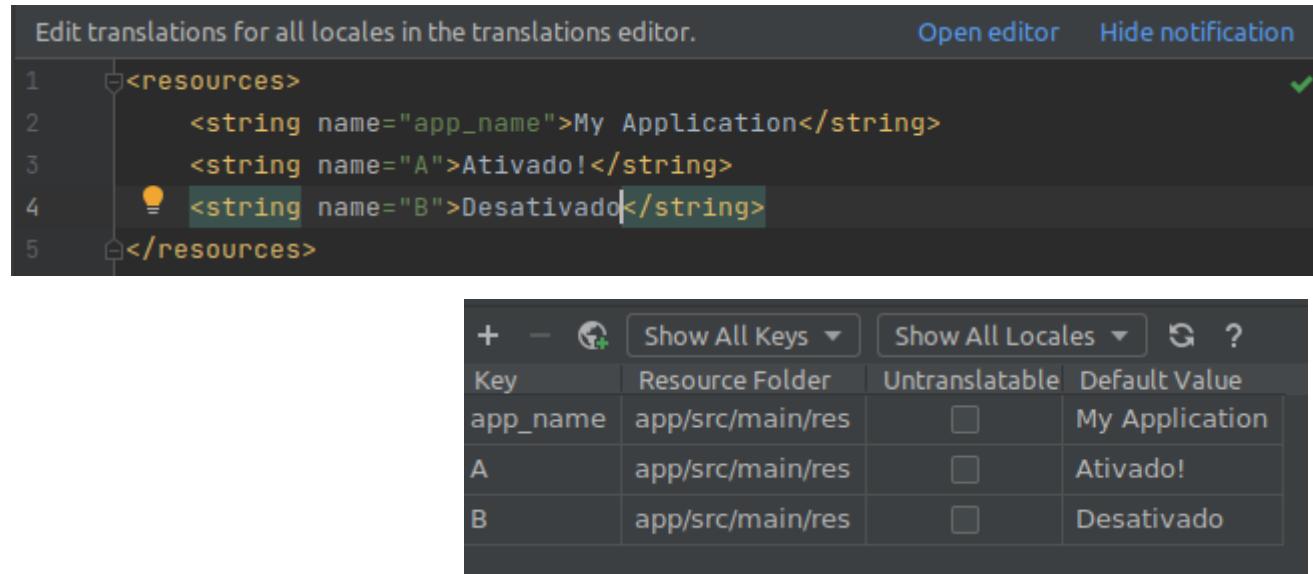
The screenshot shows the "Edit translations for all locales in the translations editor" interface. At the top right are "Open editor" and "Hide notification" buttons. The main area displays the following XML code:

```
1 <resources>
2     <string name="app_name">My Application</string>
3     <string name="A">Ativado!</string>
4     <string name="B">Desativado</string>
5 </resources>
```

Line 4, which contains the string "Desativado", has a yellow lightbulb icon next to it, indicating a suggestion or warning. A green checkmark is visible in the top right corner of the code editor window.

Editor de Strings

- Acessível através do link que aparece acima à direita:



The screenshot shows the Android Studio String Editor interface. At the top, it says "Edit translations for all locales in the translations editor." with "Open editor" and "Hide notification" buttons. Below is an XML code editor with the following content:

```
<resources>
    <string name="app_name">My Application</string>
    <string name="A">Ativado!</string>
    <string name="B">Desativado</string>
</resources>
```

The string with name "B" is highlighted with a yellow dot icon. Below the code editor is a table of translations:

Key	Resource Folder	Untranslatable	Default Value
app_name	app/src/main/res	<input type="checkbox"/>	My Application
A	app/src/main/res	<input type="checkbox"/>	Ativado!
B	app/src/main/res	<input type="checkbox"/>	Desativado

A4ex2: usando String resource

- Veja que foi necessário incluir a busca das Strings no método ‘onCreate’ para ser possível buscar as Strings após a inicialização do App.

```
9  class MainActivity : AppCompatActivity() {  
10  
11     lateinit var txt: TextView  
12     lateinit var A: String  
13     lateinit var B: String  
14  
15     override fun onCreate(savedInstanceState: Bundle?) {  
16         super.onCreate(savedInstanceState)  
17         setContentView(R.layout.activity_main)  
18         txt = findViewById(R.id.textView)  
19         A = getString(R.string.A)  
20         B = getString(R.string.B)  
21     }  
22  
23     fun enviar(view: View){  
24         if (txt.text == A) {  
25             txt.text = B  
26             title = B  
27         }  
28         else {  
29             txt.text = A  
30             title = A  
31         }  
32     }  
33 }
```

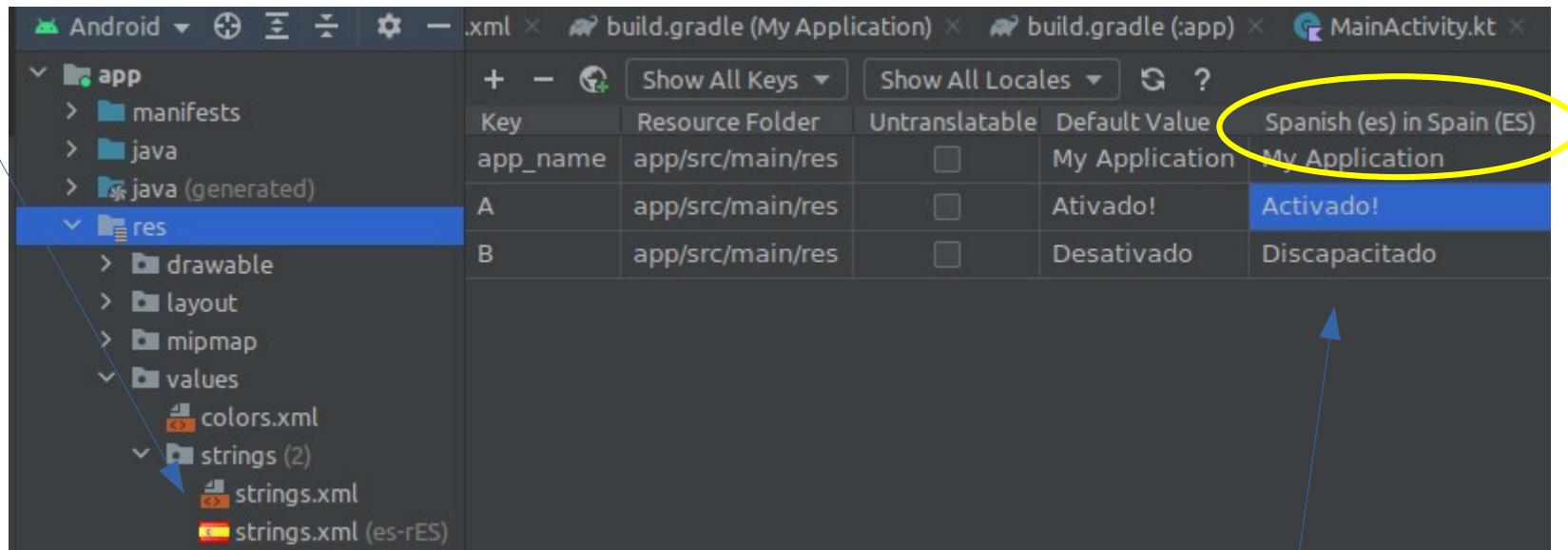
E como seria a internacionalização?

- Nesse caso, ao final do projeto, podemos fazer uma cópia do arquivo XML de Strings para cada língua que queremos prover suporte;
- Também podemos incluir novas línguas pelo botão ‘**add Locale**’ no editor:



Incluindo língua Espanhola

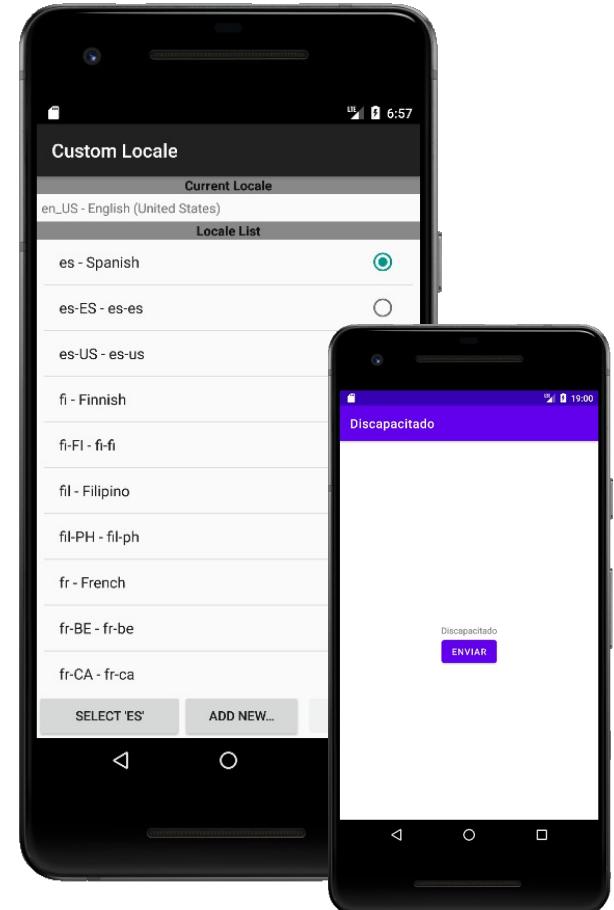
- Veja que agora temos dois arquivos strings.xml:



E incluímos as traduções

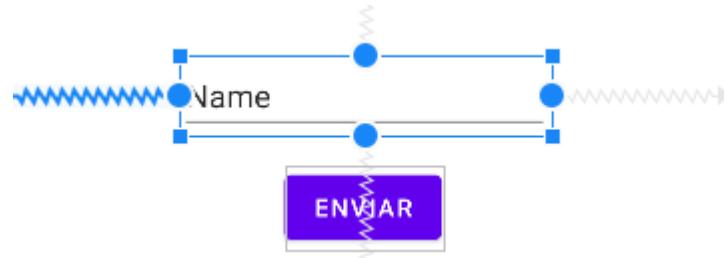
Teste em Espanhol

- Abra o App “Custom Locale” no seu emulador e escolha “es-Spanish”;
- Execute novamente o projeto para ver o resultado.



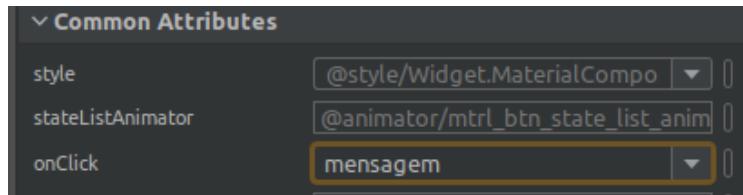
A4ex3

- Agora vamos aprender a utilizar o componente EditText:
 - Crie um novo projeto com o nome A4ex3;
 - Exclua o componente TextView e adicione um EditText no lugar com ID txtNome;
 - Adicione um botão com ID btnEnviar;



A4ex3: código fonte

- Crie a função ‘mensagem’ e vincule-a ao atributo ‘onclick’ do botão btnEnviar;



```
8 class MainActivity : AppCompatActivity() {  
9     override fun onCreate(savedInstanceState: Bundle?) {  
10         super.onCreate(savedInstanceState)  
11         setContentView(R.layout.activity_main)  
12     }  
13  
14     fun mensagem(view: View){  
15         val txtNome = findViewById<EditText>(R.id.txtNome)  
16         title = txtNome.text  
17     }  
18 }
```

The screenshot shows the Java code for the `MainActivity`. It includes the `onCreate` method and a new `mensagem` function. The `mensagem` function retrieves an `EditText` view by ID and changes its `title` to its current `text`. A yellow dot is placed on the first line of the `mensagem` function, indicating it is the target of the click event.

A4ex3: executando

- Vamos executar o projeto no emulador para verificar o resultado:
 - Ao clicar no botão ‘enviar’ o valor digitado no componente EditText é apresentado no título da janela;
 - Se desejar, exclua o valor do atributo ‘text’ do componente EditText para que a caixa de texto apareça vazia por padrão.



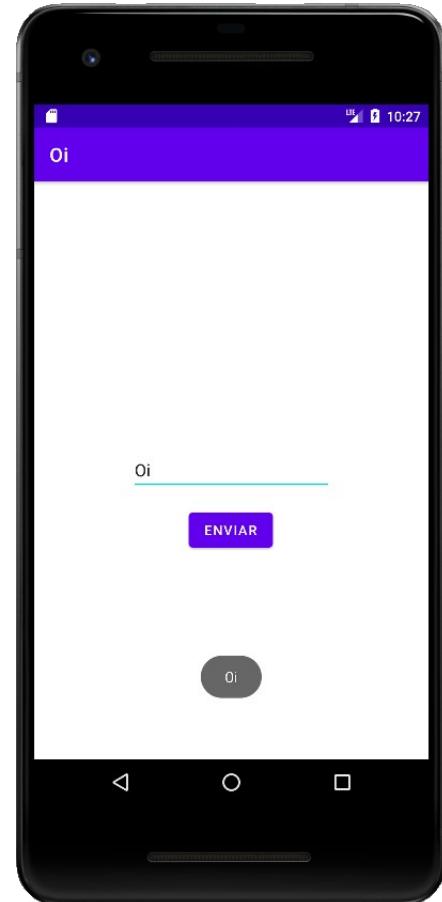
A4ex3: entendendo o código

- A função ‘mensagem’ é executada ao clicarmos no botão ‘enviar’;
- **Linha 15:** cria o objeto que representa o componente txtNome;
- **Linha 16:** define o título da janela de acordo com o valor armazenado no txtNome.

```
8 class MainActivity : AppCompatActivity() {
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12     }
13
14     fun mensagem(view: View){
15         val txtNome = findViewById<EditText>(R.id.txtNome)
16         title = txtNome.text
17     }
18 }
```

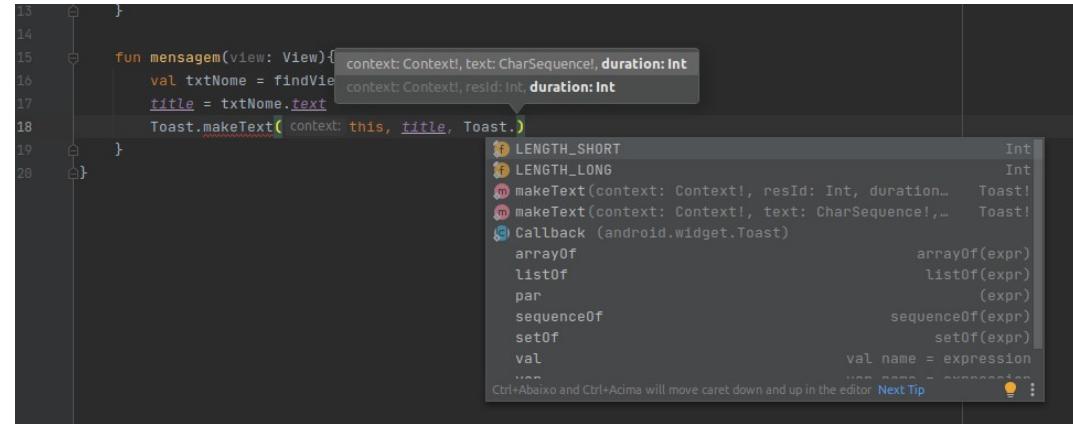
Toast

- A classe Toast permite mostrar uma mensagem rápida na tela;
- Vamos alterar nossa função para mostrar um toast, além de alterar o título da janela ao clicar no botão ‘enviar’:



Toast: detalhando

- Para criar um toast, utilizamos o método ‘makeText’, passando 3 parâmetros:
 - **Contexto**: permite o especificar com qual Activity o toast está vinculado;
 - **Valor**: o texto que será exibido;
 - **Duração**: podemos utilizar duas das duas constantes definidas na classe Toast: LENGTH_SHORT (rápida) e LENGTH_LONG (demorada).



Ao final do `makeText` devemos adicionar a chamada ao método `.show()`

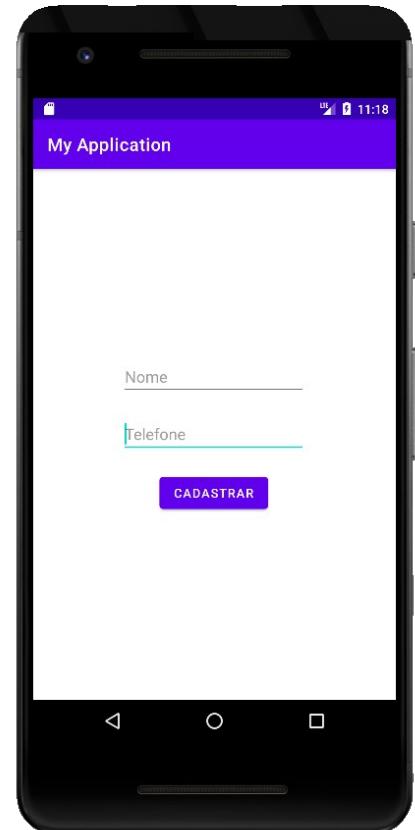
A4ex3: adicionando um toast

- Veja como ficou nossa função ‘mensagem’:
 - Adicionando a linha 18, temos a inclusão do toast sendo exibido ao clicarmos no botão ‘enviar’:

```
15     fun mensagem(view: View){  
16         val txtNome = findViewById<EditText>(R.id.txtNome)  
17         title = txtNome.text  
18         Toast.makeText(context: this, title, Toast.LENGTH_SHORT).show()  
19     }  
20 }
```

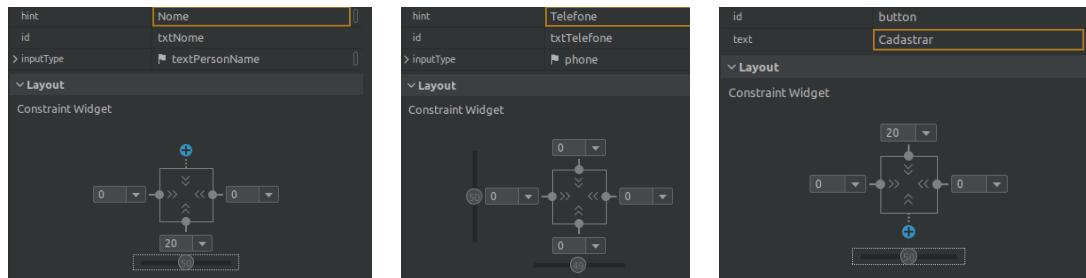
A4ex4

- Nesse projeto teremos duas Activities;
- Vamos passar argumentos de uma Activity para outra;
- Para fazer isso, utilizaremos a classe Bundle.



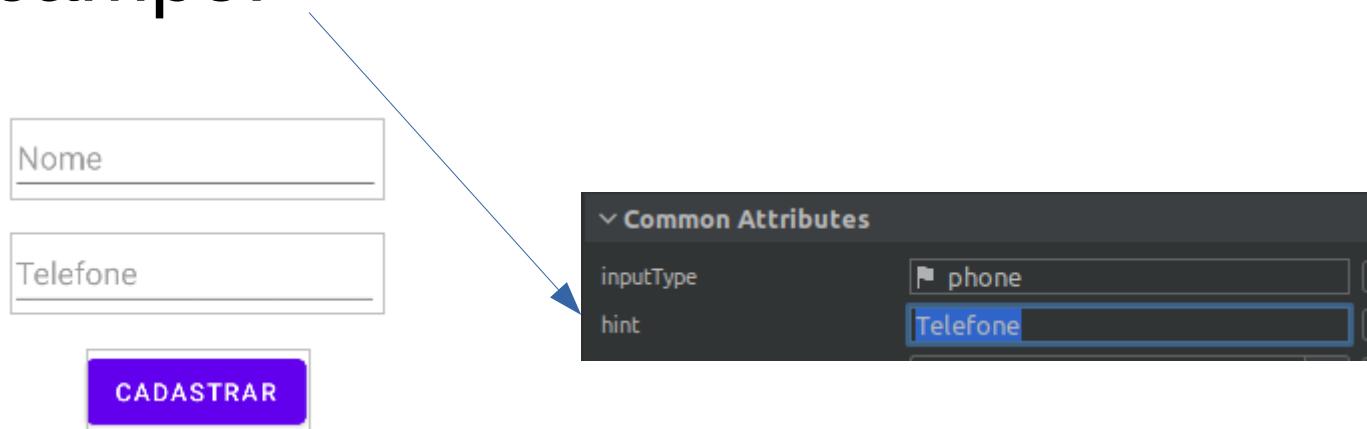
A4ex4: UI

- Adicione:
 - um EditText com ID txtNome;
 - um EditText com ID txtTelefone;
 - um Button com ID btnCadastrar.
- As constraints são:



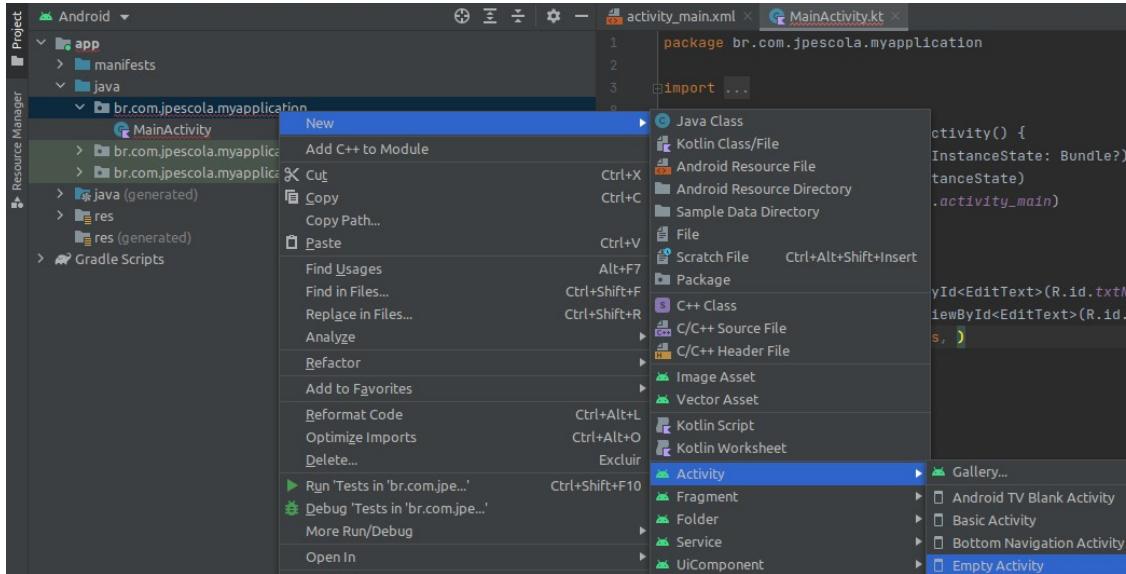
Atributo Hint

- Apresenta um rótulo dentro da caixa de texto, identificando o conteúdo que deve ser digitado no campo:

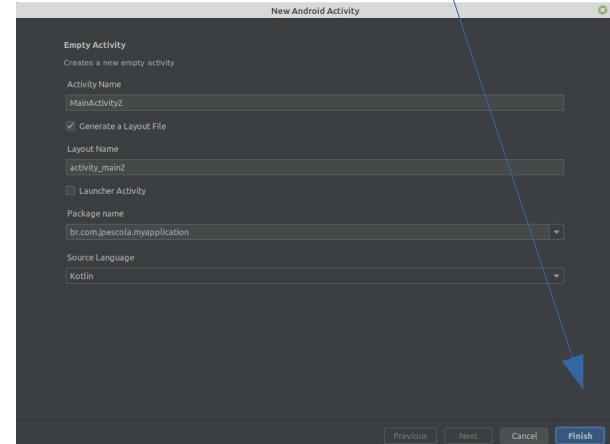


Adicionando uma Activity

- Clique com o botão direito no pacote atual e selecione New / Activity / Empty Activity;



Deixe tudo como está e clique em Finish.



Intent

- Classe que representa uma ‘Intenção’ de executar alguma tarefa no App;
- Vamos criar uma intent para apresentar a nossa nova Activity:

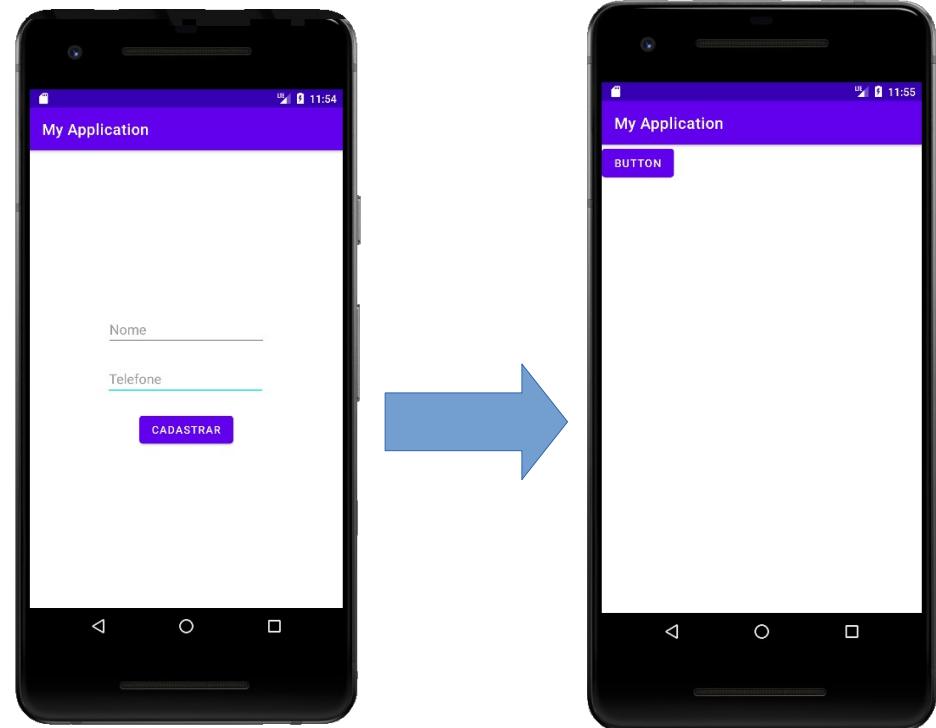
```
15     ↗
16     ↘
17     ↗
18     ↘
      ↗
      ↘
```

```
15     fun cadastrar(view: View){
16         val intent = Intent(packageContext: this, MainActivity2::class.java)
17         startActivity(intent)
18     }
```

- **Linha 16:** cria o objeto ‘intent’, passando como parâmetro o contexto e o nome da Activity que será exibida;
- **Linha 17:** executa o método ‘startActivity’ passando a intent como parâmetro.

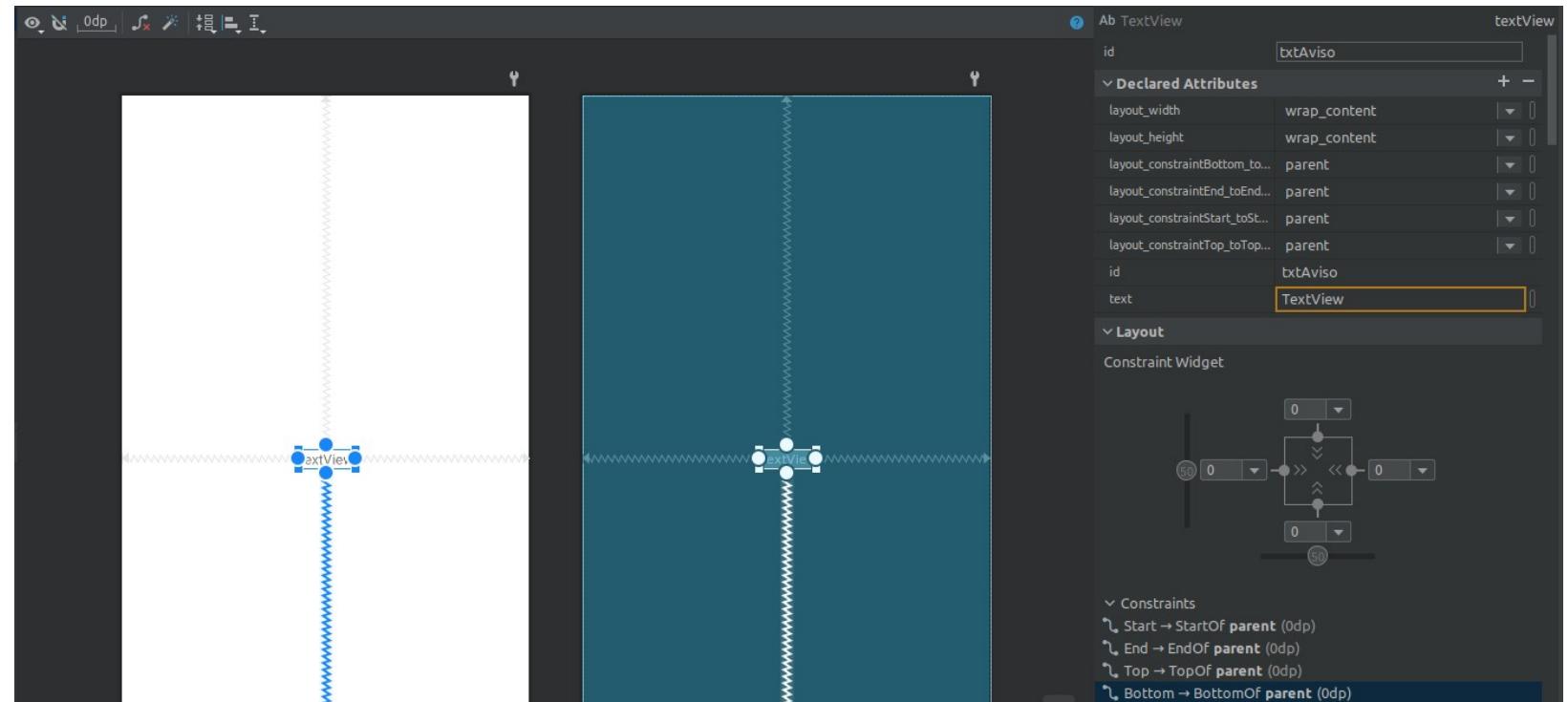
A4ex4: executando

- Vamos testar nosso app, veja que ao clicar no botão ‘cadastrar’ a nova Activity é exibida;
- Ela não tem componentes ainda, então adicione algum componente nela para ver melhor o resultado.



A4ex4: ajustando a 2^a UI

- Vamos alterar o código XML da segunda Activity para receber os argumentos da tela inicial;
 - Para isso, vamos editar o arquivo `activity_main2.xml`:



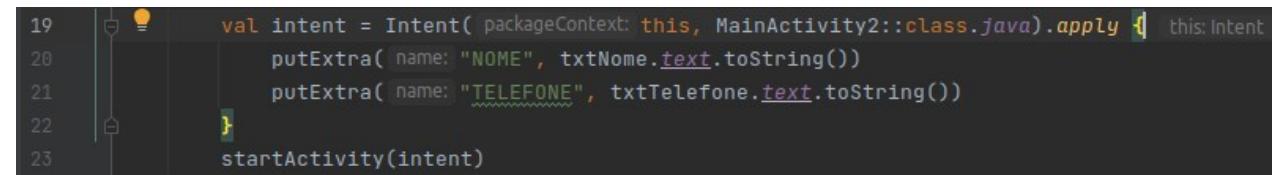
A4ex4: argumentos

- Agora vamos passar argumentos da primeira para a segunda Activity;
- O que queremos é enviar os dados digitados de uma tela para outra;
- Para isso, utilizamos o método ‘putExtra’ da classe Intent:

```
15     fun cadastrar(view: View){  
16         val txtNome = findViewById<EditText>(R.id.txtNome)  
17         val txtTelefone = findViewById<EditText>(R.id.txtTelefone)  
18  
19         val intent = Intent(packageContext: this, MainActivity2::class.java).apply { this: Intent  
20             putExtra( name: "NOME", txtNome.text.toString())  
21             putExtra( name: "TELEFONE", txtTelefone.text.toString())  
22         }  
23         startActivity(intent)  
24     }
```

putExtra

- Vamos entender o método ‘putExtra’ da classe Intent:
 - Veja que agora chamamos a extension function ‘apply’ para incluirmos dois argumentos na Intent;
 - O método ‘putExtra’ permite injetar um argumento na Intent;
 - O primeiro parâmetro é o nome do argumento e em seguida o valor;
 - Ao final, podemos chamar o método ‘startActivity’, pois ao passar a intent como parâmetro estamos também enviando os argumentos embutidos nela.



```
19 val intent = Intent(packageContext: this, MainActivity2::class.java).apply {
20     putExtra(name: "NOME", txtNome.text.toString())
21     putExtra(name: "TELEFONE", txtTelefone.text.toString())
22 }
23 startActivity(intent)
```

The screenshot shows a portion of an Android Studio code editor. Line 19 starts an intent with its target activity. Lines 20 and 21 call the 'apply' extension function on the intent object. This function allows for the addition of key-value pairs using the 'putExtra' method, where the key is a string and the value is the result of calling 'toString()' on a text view ('txtNome' and 'txtTelefone'). Line 23 then calls 'startActivity' with the prepared intent.

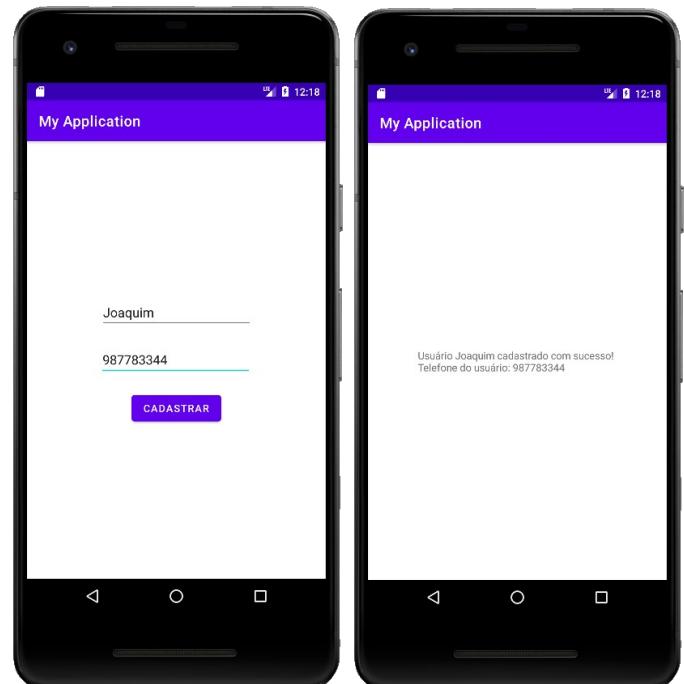
A4ex4: MainActivity2.kt

- Vamos incluir esse código na segunda Activity:
 - Linhas 12 e 13: armazenam os argumentos que vieram embutidos na intent.

```
7 class MainActivity2 : AppCompatActivity() {
8     override fun onCreate(savedInstanceState: Bundle?) {
9         super.onCreate(savedInstanceState)
10        setContentView(R.layout.activity_main2)
11
12        val usuario = intent.getStringExtra("NOME")
13        val telefone = intent.getStringExtra("TELEFONE")
14
15        val txtAviso = findViewById<TextView>(R.id.txtAviso).apply { this: TextView!
16            text = "Usuário $usuario cadastrado com sucesso!\nTelefone do usuário: $telefone"
17        }
18    }
19 }
```

A4ex4: executando

- Agora vamos ver o resultado:



Botão voltar

- Vamos adicionar um botão ‘voltar’ na barra de apps;
- Para isso, edite o arquivo manifest, incluindo o nome da Activity ‘pai’:

```
12 <activity  
13     android:name=".MainActivity2"  
14     android:parentActivityName=".MainActivity"
```



A4ex5

- Jogo: par ou ímpar;
- Crie um novo projeto;
- Na UI: inclua um LinearLayout horizontal e dentro dele dois botões btnPar e btnImpar, chamando a função ‘jogada’;
 - Defina os atributos ‘tag’ dos botões como 0 (par) e 1 (ímpar);
 - Renomeie o TextView para txtResultado;
 - O layout_height do LinearLayout deve ser definido como wrap_content para ser possível movê-lo abaixo do TextView;
 - Mais abaixo inclua um botão btnNovo e deixe como invisível.



A4ex5: campos

- Vamos incluir os campos da Activity:
 - Resultado: armazena o resultado de cada partida;
 - Score: pontuação do usuário;
 - txtOponente e btnNovo: componentes da UI

```
11 class MainActivity : AppCompatActivity() {  
12  
13     var resultado = 0  
14     private var score = 0  
15     lateinit var txtResultado: TextView  
16     lateinit var btnNovo: Button  
17  
18     override fun onCreate(savedInstanceState: Bundle?) {
```

A4ex5: funções

- NovoJogo: função responsável por alterar o texto do rótulo e sortear o número para a próxima partida, ocultando o botão ao final;
- NovoJogo(view: View): essa função é uma sobrecarga da anterior. Permitindo ser disparada pelo botão;
- Jogada: disparada ao clicar em uma das duas respostas possíveis:
 - Verifica o valor da tag do botão clicado;
 - Verifica se o número é par ou ímpar, incrementando a pontuação do usuário;
 - Atualiza o título da janela;
 - Mostra o valor sorteado;
 - Habilita o botão 'Novo jogo'

```
30     private fun novoJogo(){  
31         txtResultado.text = "Par ou Ímpar?"  
32         resultado = Random.nextInt( from: 0, until: 10)  
33         btnNovo.visibility = View.INVISIBLE  
34     }  
35  
36     fun novoJogo(view: View){  
37         novoJogo()  
38     }  
39  
40     fun jogada(view: View){  
41         if (resultado % 2 == view.tag.toString().toInt())  
42             if (btnNovo.visibility == View.INVISIBLE)  
43                 score++  
44             title = "Score: $score"  
45             txtResultado.text = "$resultado"  
46  
47             btnNovo.visibility = View.VISIBLE  
48     }  
49 }
```

A4ex5: onCreate

- No método de inicialização, vamos instanciar os componentes visuais e chamar a função para começar o jogo:

```
18  override fun onCreate(savedInstanceState: Bundle?) {  
19      super.onCreate(savedInstanceState)  
20      setContentView(R.layout.activity_main)  
21  
22      // inicialização dos componentes da UI  
23      txtResultado = findViewById(R.id.txtResultado)  
24      btnNovo = findViewById(R.id.btnNovo)  
25  
26      // dispara a função na inicialização  
27      novoJogo()  
28  }
```

A4ex5: executando

- Execute o jogo e verifique se está tudo funcionando como deveria;
- Veja que o score não é salvo se você fechar e abrir novamente o game;



Shared Preferences

```
getPreferences(MODE_PRIVATE).edit().putInt("SCORE", score).commit() // salvar
```

- Classe que permite armazenar dados no App;
- Utiliza o formato chave-valor:
 - **getPreferences**: método da classe Activity para buscar as preferencias compartilhadas (Shared Preferences);
 - **MODE_PRIVATE**: modo de compartilhamento dos dados (padrão);
 - **edit**: significa que estaremos fazendo uma alteração (edição);
 - **.putInt**: colocar um inteiro (salvar em formato inteiro), passando como parâmetros um par de chave (nome do valor que será salvo) e valor (dado que será salvo);
 - **commit**: salva a alteração.

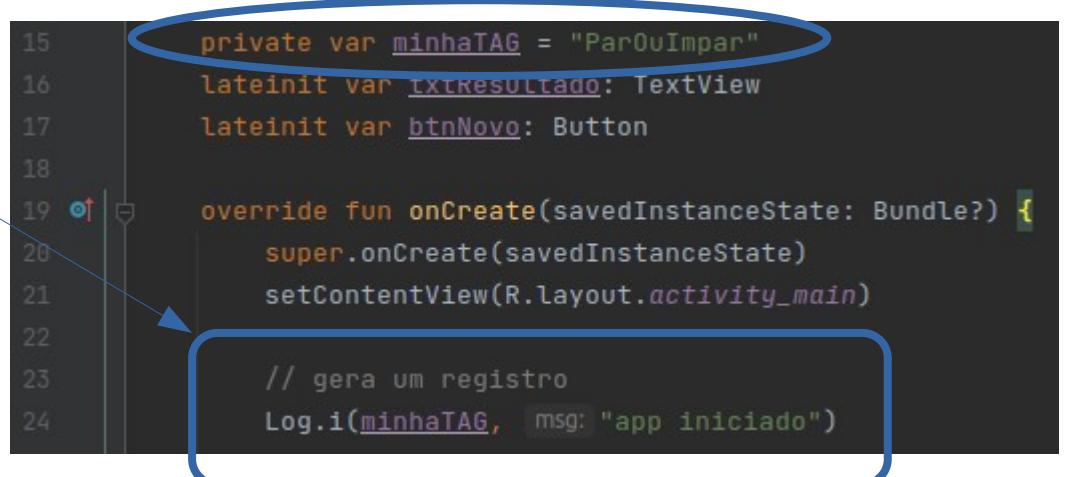
A4ex5: salvando o score

- Inclua a linha 53 para salvar o score a cada vez que o jogador pontuar:

```
49     fun jogada(view: View){  
50         if (resultado % 2 == view.tag.toString().toInt())  
51             if (btnNovo.visibility == View.INVISIBLE) {  
52                 score++  
53                 getPreferences(MODE_PRIVATE).edit().putInt("SCORE", score).commit() // salvar  
54             }  
55             title = "Score: $score"  
56             txtResultado.text = "$resultado"  
57  
58             btnNovo.visibility = View.VISIBLE  
59     }
```

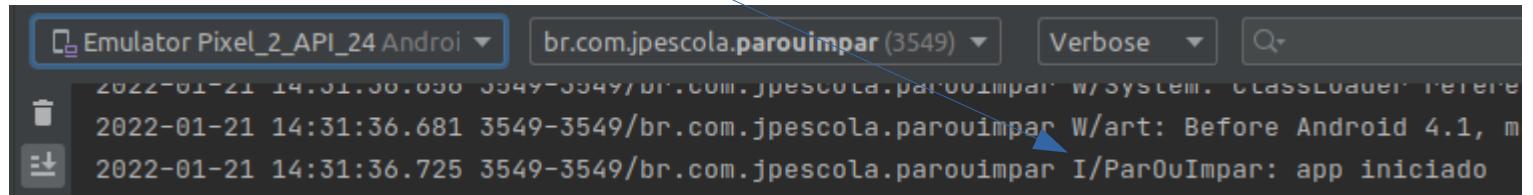
Log para depuração

- Com a classe Log podemos gerar registros do sistema que são exibidos pelo serviço **Logcat**;
- Logs são muito utilizados pelos desenvolvedores para depuração em tempo de execução



A screenshot of the Android Studio code editor. A blue oval highlights the line of code: `private var minhaTAG = "ParOuImpar"`. A blue box highlights the log statement: `// gera um registro Log.i(minhaTAG, msg: "app iniciado")`. A blue arrow points from the highlighted code in the editor to the corresponding log entry in the Logcat window below.

```
15  
16  
17  
18  
19  private var minhaTAG = "ParOuImpar"  
lateinit var txtResultado: TextView  
lateinit var btnNovo: Button  
20  
21  
22  
23  
24  
// gera um registro  
Log.i(minhaTAG, msg: "app iniciado")
```



A screenshot of the Android Logcat window. It shows two log entries:

```
2022-01-21 14:31:36.681 3549-3549/br.com.jpescola.parouimpar W/art: Before Android 4.1, m  
2022-01-21 14:31:36.725 3549-3549/br.com.jpescola.parouimpar I/ParOuImpar: app iniciado
```

Classe Log

- Possui métodos estáticos para informação (i), debug (d), erro (e) e verbose (v);
- O primeiro parâmetro é a TAG, ou seja, o rótulo do registro;
- O segundo parâmetro é o texto que vai ser registrado:

```
22  
23 | // gera um registro  
24 | Log.i(minhaTAG, msg: "app iniciado")
```

Logcat

- Serviço de registro da plataforma Android;
- Na IDE, há um utilitário que permite visualizar, filtrar e manipular os registros de um dispositivo virtual ou outros dispositivos conectados:

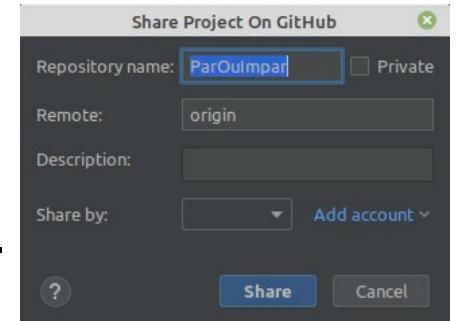
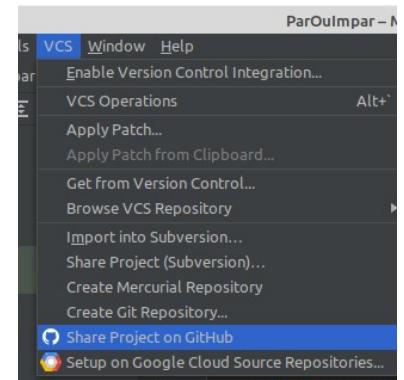
dispositivo	pacote	nível	filtro
Emulator Pixel_2_API_24 Androi	br.com.jpescola.parouimpar (3549)	Verbose	<input type="button" value="Q"/>
<input type="button" value="Delete"/>	2022-01-21 14:31:36.681 3549-3549/br.com.jpescola.parouimpar W/art: Before Android 4.1, m		
<input type="button" value="More"/>	2022-01-21 14:31:36.725 3549-3549/br.com.jpescola.parouimpar I/ParOuImpar: app iniciado		

GitHub

- Serviço gratuito de repositório de projetos de software;
- Muitas empresas analisam o perfil do usuário pelo GitHub antes da contratação;
- Permite compartilhar projetos com outras pessoas no mundo todo;
- Compartilhando você fará novos amigos, aprenderá com eles e pode evoluir como desenvolvedor.

Compartilhar no GitHub

- Vamos salvar nosso projeto no GitHub;
- Clique no menu VCS e escolha “Share Project on GitHub”;
- Clique em Add account e depois em Log in with Token:
 - Clique em ‘Generate’ e faça login no GitHub (ou crie sua conta);
 - No final da página clique em “Generate token”;
 - Copie o código gerado e cole na janela do Android Studio;
 - Clique em “Add Account”
- Escolha o nome do repositório que será compartilhado no GitHub ou mantenha o nome do projeto e clique em “Share”.



Git

- Agora que compartilhamos nosso projeto no Git, todas as linhas alteradas serão destacadas e temos um recurso para desfazer as alterações:



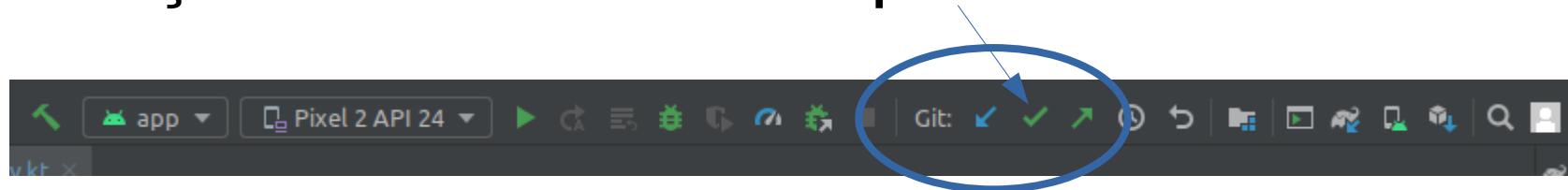
A screenshot of an IDE interface. On the left, there is a code editor with the following Java code:

```
24 // gera um registro
25 Log.i(minhaTAG, msg: "O aplicativo foi iniciado...")
26
27 Log.i(minhaTAG, "app iniciado")
```

The word "msg:" in the first log statement is highlighted in blue, indicating it has been changed. At the bottom of the screen, there is a toolbar with several icons and a dropdown menu labeled "Default Changelist".

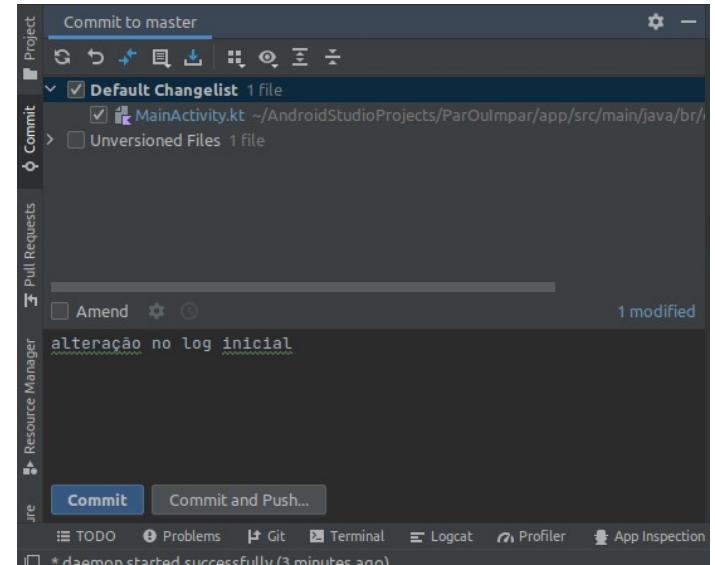
Alterações no Git

- Agora a IDE apresenta alguns botões de manipulação de repositórios;
- Vamos clicar no botão “Commit” para enviar as alterações e atualizar o repositório:



Commit and Push

- Nessa janela, temos a lista dos arquivos que foram alterados;
- Digite um texto que representa as mudanças que foram realizadas e clique em “Commit and Push...”;
- Na janela de confirmação, clique em Push.



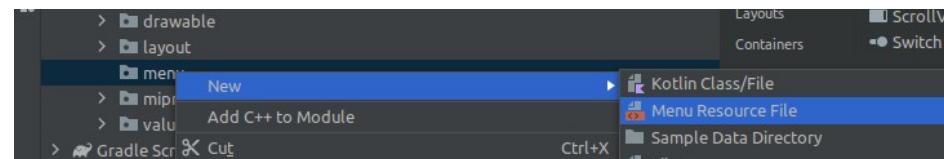
A4ex6

- Neste app vamos aprender a criar um menu;
- O app vai ter um rótulo com o valor zero e vai ser incrementado ou reiniciado através do menu;
- Também teremos uma opção para fechar o app.



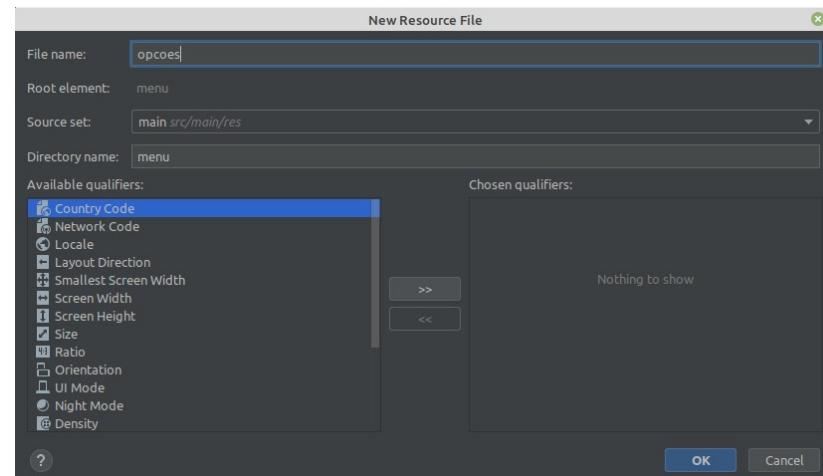
Arquivo XML de menu

- Crie a pasta ‘menu’ dentro da pasta ‘res’: botão direito sobre a pasta ‘res’ / New / Directory;
- Crie o arquivo XML com os valores do menu: botão direito sobre a pasta ‘menu’ / New / Menu Resource File:



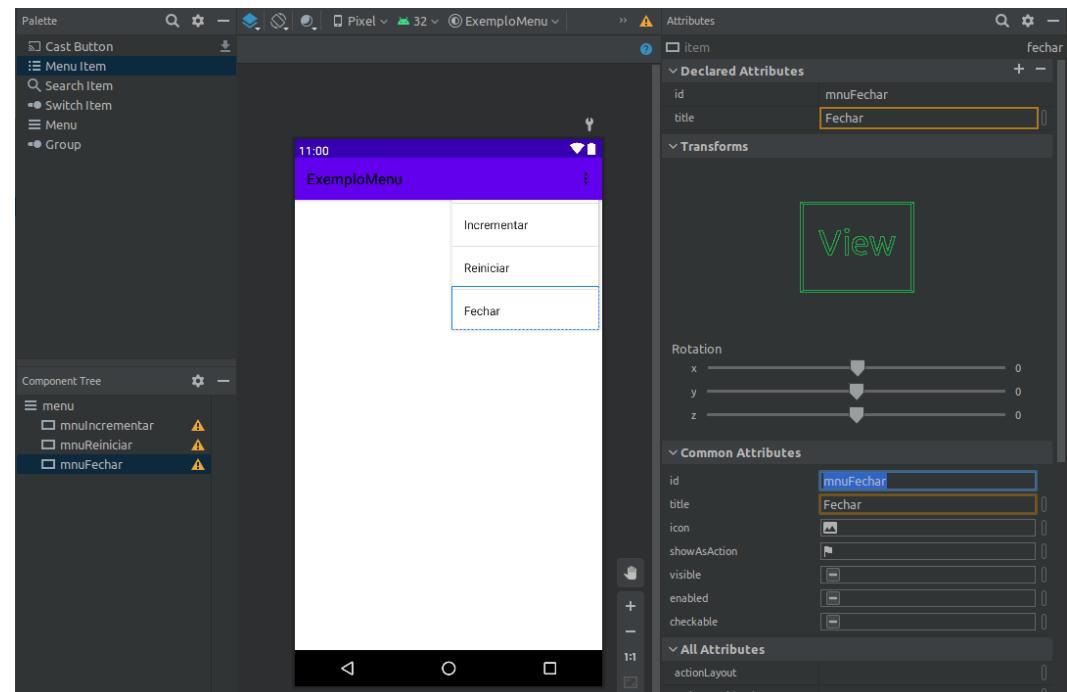
Menu de opções

- Na próxima tela, informe o nome do arquivo que será gerado ‘opcoes’ e clique em OK:



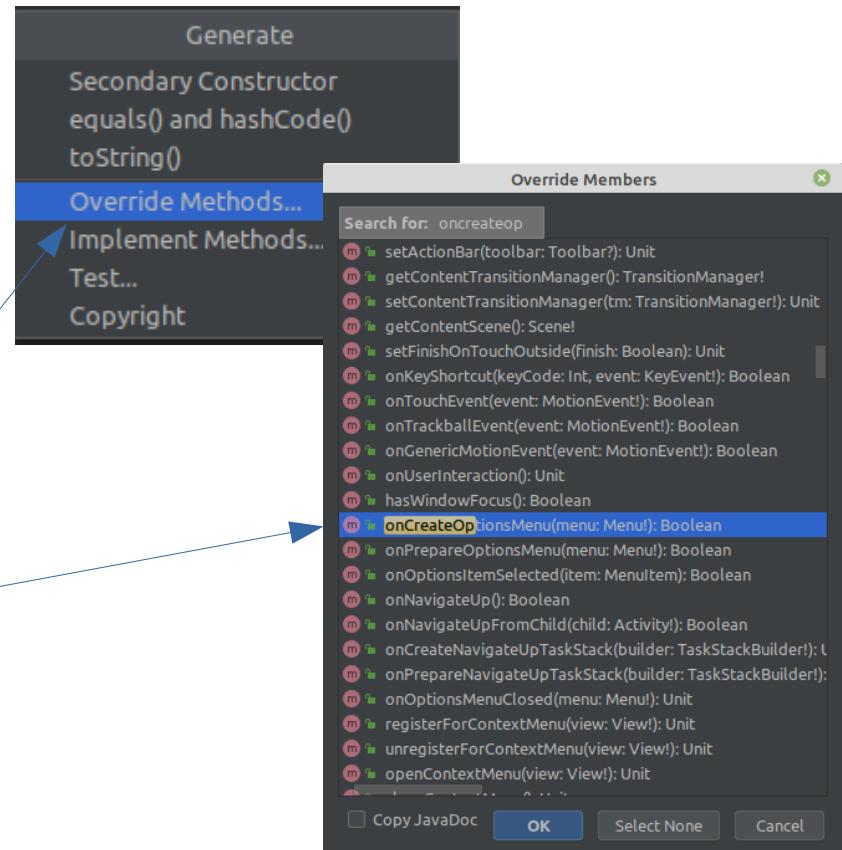
Design do menu

- Será exibida uma tela de design do menu;
- Arraste 3 ‘Menu Item’ para a UI:
 - Incrementar (mnuIncrementar);
 - Reiniciar (mnuReiniciar);
 - e Fechar (mnuFechar).



A4ex6: sobrescrever método

- Se você executar o app agora, vai ver que o menu ainda não aparece;
- Para resolver isso, precisamos sobrescrever o método que o apresenta na tela;
- No arquivo MainActivity.kt, pressione Alt + insert e escolha ‘Override Methods...’;
- Digite onCreateOp para ser direcionado ao método que estamos procurando:



A4ex6: exibindo o menu

- Agora a IDE incluiu sozinha o método ‘onCreateOptionsMenu’, acrescente as linhas 15 e 16:

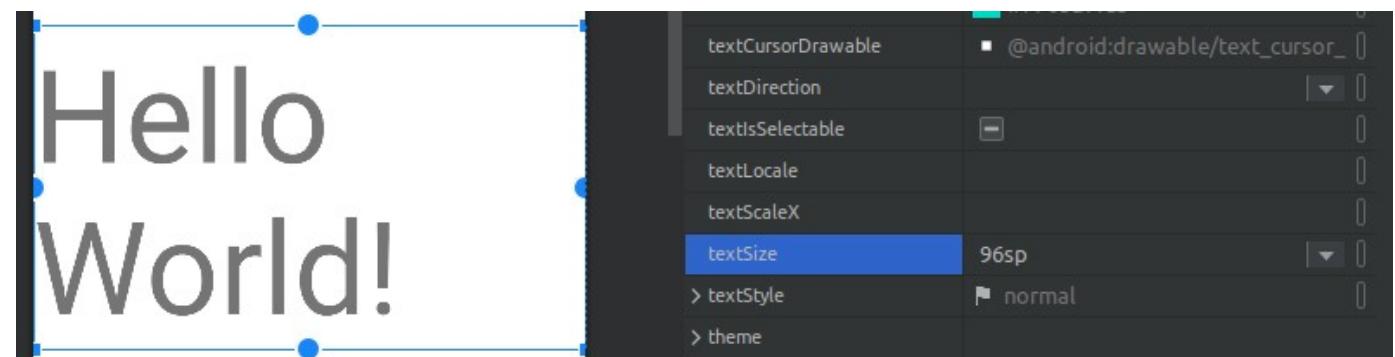
```
14  override fun onCreateOptionsMenu(menu: Menu?): Boolean {  
15      val inflater: MenuInflater = menuInflater  
16      inflater.inflate(R.menu.opcoes, menu)  
17      return super.onCreateOptionsMenu(menu)  
18  }  
19 }
```

Linha 15: cria um objeto ‘inflater’ da classe MenuInflater (trata-se de um inflador de menu)
Linha 16: infla na tela o menu ‘opcoes’ (exibe o menu)

Execute o app e veja que agora o menu já aparece

A4ex6: UI

- Esta UI está bem simples;
- Vamos manter o TextView, renomeando-o para ‘txtResultado’;
- Defina o atributo ‘textSize’ para 96sp.



A4ex6: código fonte

- Neste app teremos dois campos:
 - ‘cont’ é o nosso contador e;
 - ‘txtResultado’ para podermos atualizar a view;
- Inclua também o código do método principal:

```
10 class MainActivity : AppCompatActivity() {  
11  
12     private var cont = 0  
13     private lateinit var txtResultado: TextView  
14  
15     override fun onCreate(savedInstanceState: Bundle?) {  
16         super.onCreate(savedInstanceState)  
17         setContentView(R.layout.activity_main)  
18         txtResultado = findViewById(R.id.txtResultado)  
19         txtResultado.text = "$cont"  
20     }  
}
```

onOptionsItemSelected

- Sobrescreva esse método também;
- Ele é o responsável por validar as ações no menu;
- Inclua o código fonte:
 - O ‘when’ verifica qual item foi selecionado e executa a ação correspondente;
 - Linha 32: ‘finish()’ é o método que fecha o app;
 - Caso nenhum item tenha sido selecionado, ele chama o método da classe pai;
 - Linha 35: atualiza o rótulo com o incremento ou reset, de acordo com o valor resultante da variável ‘cont’:

```
28     override fun onOptionsItemSelected(item: MenuItem): Boolean {  
29         when (item.itemId) {  
30             R.id.mnuReiniciar -> cont = 0  
31             R.id.mnuIncrementar -> cont++  
32             R.id.mnuFechar -> finish() // fecha o aplicativo  
33             else -> return super.onOptionsItemSelected(item)  
34         }  
35         txtResultado.text = "$cont"  
36         return true  
37     }
```

A4ex6: executando

- Finalizamos nosso aplicativo para aprendermos a manipular menus na tela do app:
 - Agora podemos executar o app;
 - Veja que o menu permite incrementar e resetar o contador da tela;
 - E o menu ‘Fechar’ finaliza a aplicação.

<https://github.com/jpescola/ExemploMenu>

Perguntas...

- Será que existem outros tipos de menu?
- Será que é possível executar uma função ao clicar em outro View, como um rótulo?

A4ex7

- Menu pop-up vinculado ao componente visual;
- Ao clicar no TextView, um menu será exibido:
 - Este menu é semelhante ao aprendido no exercício anterior, entretanto, agora vinculado ao View;



A4ex7: criando o menu XML

- Crie um arquivo XML de menu, da mesma forma que fizemos no exercício anterior;
- Você pode, se preferir, copiar o mesmo arquivo e adicionar ao projeto atual.

A4ex7: código fonte

- Vamos criar a função ‘showPopup’;
- Ela mostra um menu vinculado ao View especificado;
- Veja que, novamente, utilizamos o MenuInflater para apresentar o menu;

```
9  class MainActivity : AppCompatActivity() {
10  override fun onCreate(savedInstanceState: Bundle?) {
11      super.onCreate(savedInstanceState)
12      setContentView(R.layout.activity_main)
13  }
14
15  fun showPopup(v: View) {
16      val popup = PopupMenu(context: this, v)
17      val inflater: MenuInflater = popup.menuInflater
18      inflater.inflate(R.menu.opcoes, popup.menu)
19      popup.show()
20  }
21
22 }
```

A4ex7: testando

- Vamos executar o app e ver como está;
- O menu deve ser exibido ao clicarmos no rótulo;
- Entretanto, ainda não executa a ação desejada;
- Para isso, vamos precisar implementar um Listener...

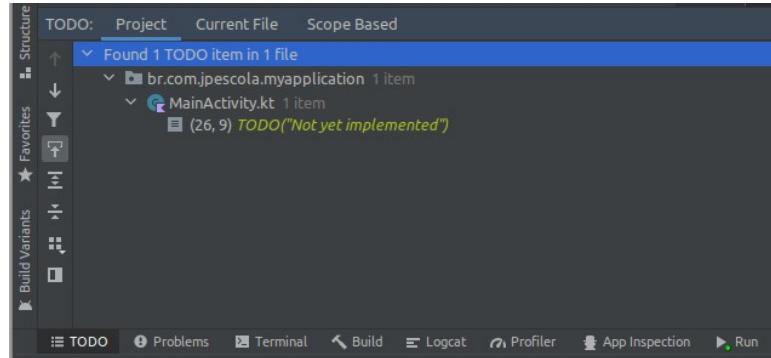
Listener

- Um Listener é uma interface que precisa ser implementada para responder às ações de um componente;
- No caso da classe PopupMenu, é necessário implementar a interface OnMenuItemClickListener, que contém o método onMenuItemClick:

```
25  ⚡  override fun onMenuItemClick(p0: MenuItem?): Boolean {  
26      TODO(reason: "Not yet implemented")  
27  }
```

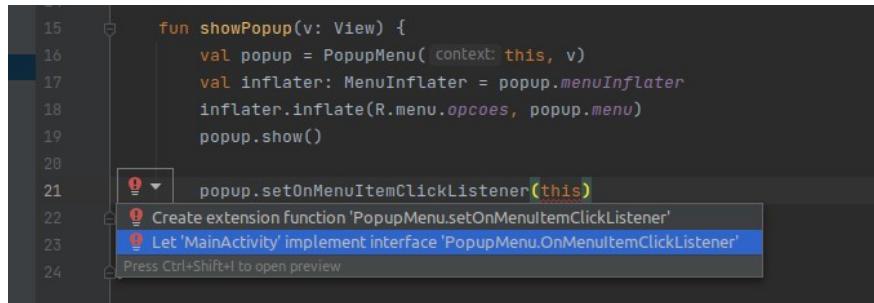
O que é esse TODO?

- Um TODO (to=para; do=fazer) é uma tarefa a fazer;
- São uma boa prática de desenvolvimento de software;
- Permite incluir tarefas a serem executadas posteriormente por você ou como sugestão aos demais membros da equipe de desenvolvimento;
- O primeiro botão no rodapé da IDE exibe todos os TODOs do seu projeto:



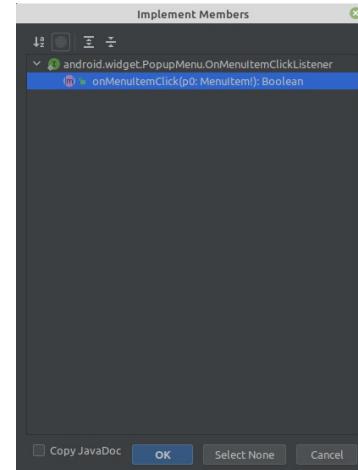
A4ex7: adicionando o Listener

- Ao incluir a linha 22, uma lâmpada vermelha surgirá, escolha a opção “Let ... implement”:



The screenshot shows a portion of an Android Java code file. Line 21 contains the call `popup.setOnMenuItemClickListener(this)`. A code completion tooltip is open over this line, displaying three options: "Create extension function 'PopupMenu.setOnMenuItemClickListener'", "Let 'MainActivity' implement interface 'PopupMenu.OnMenuItemClickListener'", and "Press Ctrl+Shift+I to open preview".

- A janela seguinte vai ser exibida.
Clique em OK para confirmar a
criação da função:



A4ex7: ação do menu

- Agora vamos implementar o fechamento do app ao clicar no menu Fechar:
- A linha 26 só é necessária se você mantiver a interrogação depois da declaração dos Argumentos;

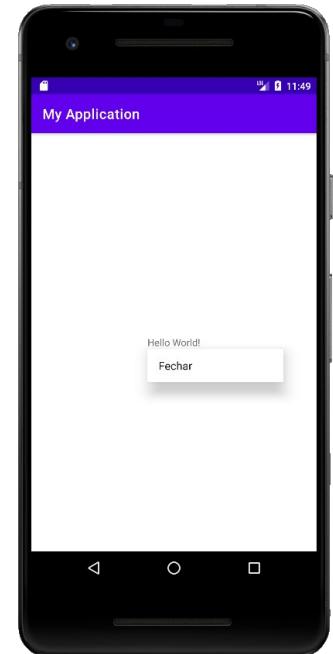
```
25 override fun onOptionsItemSelected(p0: MenuItem?): Boolean {  
26     if (p0 != null)  
27         if (p0.itemId == R.id.mnuFechar)  
28             finish()  
29     return true  
30 }
```

- Versão reduzida:

```
25 override fun onOptionsItemSelected(item: MenuItem): Boolean {  
26     if (item.itemId == R.id.mnuFechar)  
27         finish()  
28     return true  
29 }
```

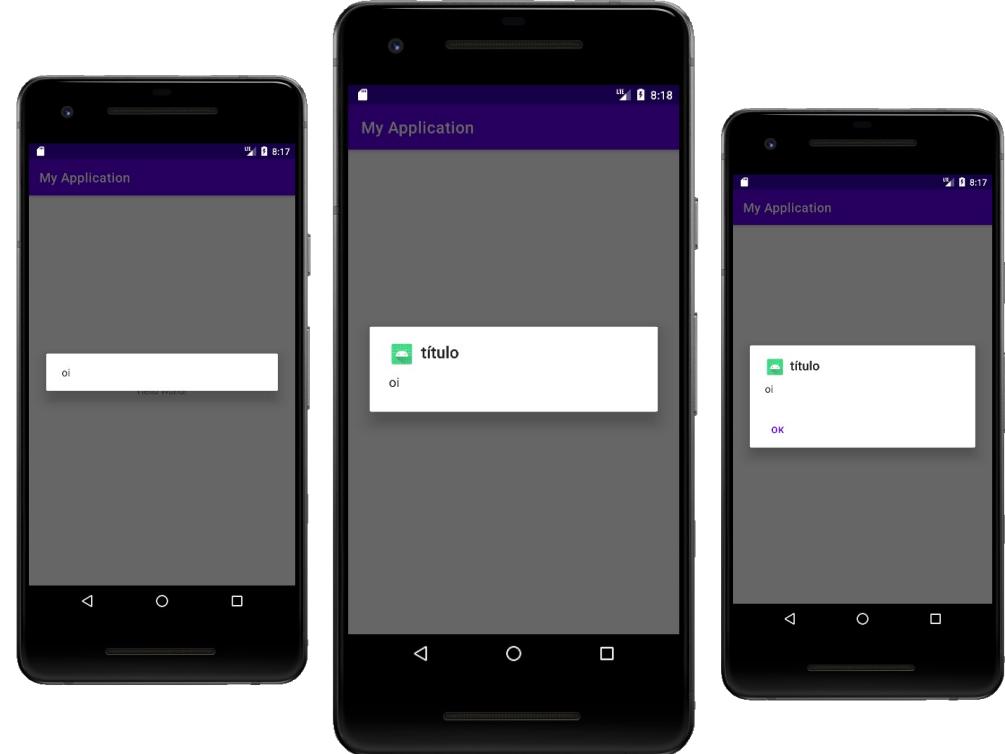
A4ex7: executando

- Agora vamos rodar o app e ver como ficou;
- Ao clicar no rótulo, o menu é exibido, permitindo fechar o app;
- Com esse tipo de menu, podemos criar listas de opções diversas em nossos aplicativos tornando-os mais atrativos e funcionais.



A4ex8

- Neste projeto vamos aprender a criar mensagens na tela;



AlertDialog

- Permite criar janelas de dialogo para interação com o usuário;
- A janela pode ter título, mensagem, ícone e botões;
- O primeiro passo é criar um objeto Builder da classe;
- Em seguida, incluir os métodos correspondentes às configurações desejadas para a janela.

A4ex8: código fonte

```
7 class MainActivity : AppCompatActivity() {
8     override fun onCreate(savedInstanceState: Bundle?) {
9         super.onCreate(savedInstanceState)
10        setContentView(R.layout.activity_main)
11
12        // mensagem simples
13        AlertDialog.Builder( context: this).setMessage("oi").show()
14
15        // mensagem com título
16        //        AlertDialog.Builder(this).setMessage("oi").setTitle("título").show()
17
18        // mensagem com ícone
19        //        AlertDialog.Builder(this).setMessage("oi").setTitle("título").setIcon(R.mipmap.ic_launcher).show()
20
21        // mensagem com botão OK
22        //        AlertDialog.Builder(this).setMessage("oi").setTitle("título")
23        //                .setIcon(R.mipmap.ic_launcher)
24        //                .setNeutralButton("OK", null)
25        //                .show()
26
27
28    }
29 }
```

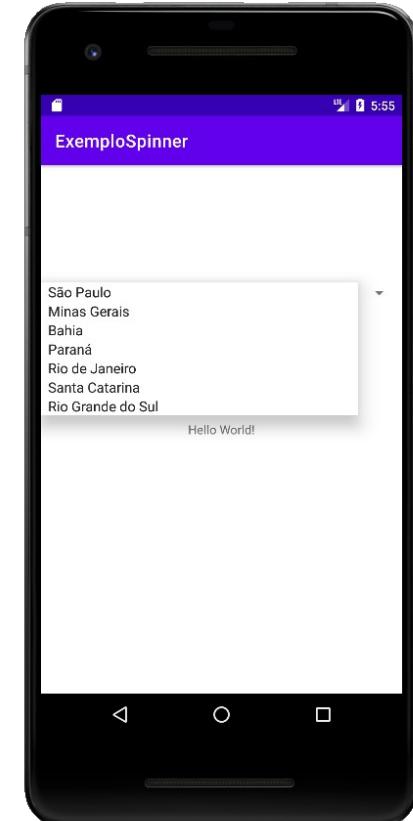
AlertDialog com objeto

- Neste exemplo, criamos um objeto 'b' (builder) para definir os ajustes em linhas seguintes:
 - Linha 29: especifica o título;
 - Linha 30: especifica a mensagem;
 - Linha 31: apresenta a caixa de diálogo.

```
27          // mensagem com objeto builder
28          var b = AlertDialog.Builder( context: this)
29          b.setTitle("titulo")
30          b.setMessage("oi")
31          b.show()
```

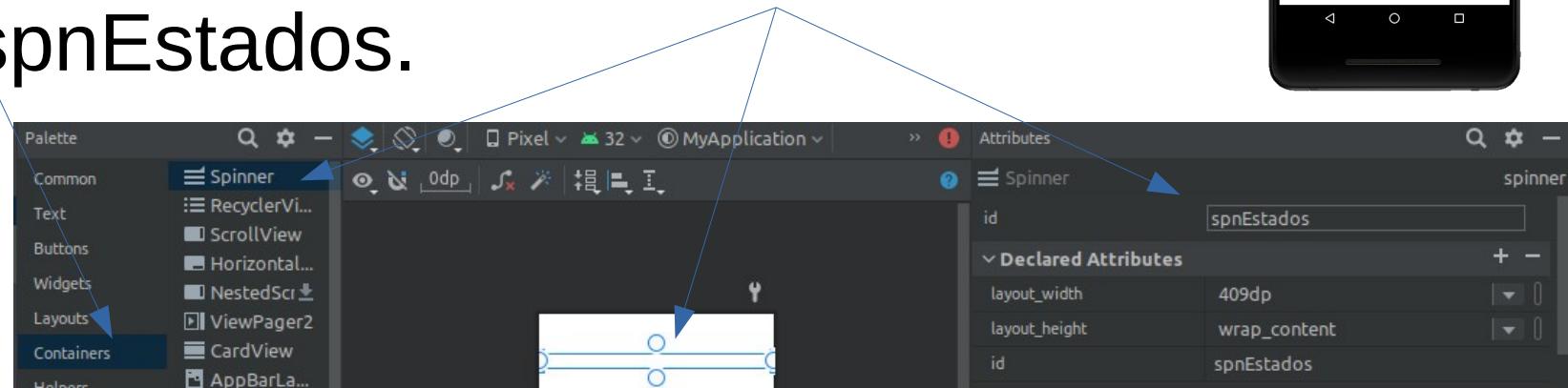
Spinner

- Componente visual que permite exibição de lista de itens;
- Semelhante ao componente <select> do HTML;
- Semelhante ao JComboBox do Java:



A4ex8

- App com um Spinner de estados do Brasil;
- Crie um novo projeto e adicione um Spinner com o nome spnEstados.

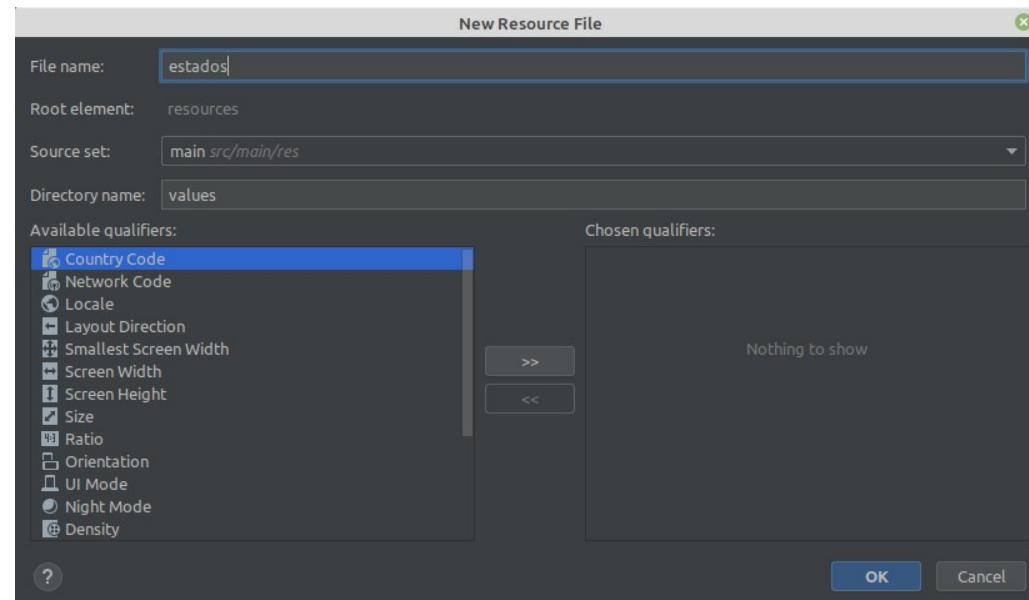


Dados do Spinner

- Os valores a serem exibidos no Spinner podem vir de um arquivo XML de dados;
- Esse arquivo deve conter um conjunto de dados no formato String-array;
- Clique com o botão direito no projeto e escolha a opção New / Android Resource File.

Arquivo XML

- Digite o nome do arquivo que será gerado;
- Deixe tudo como está e clique em OK:



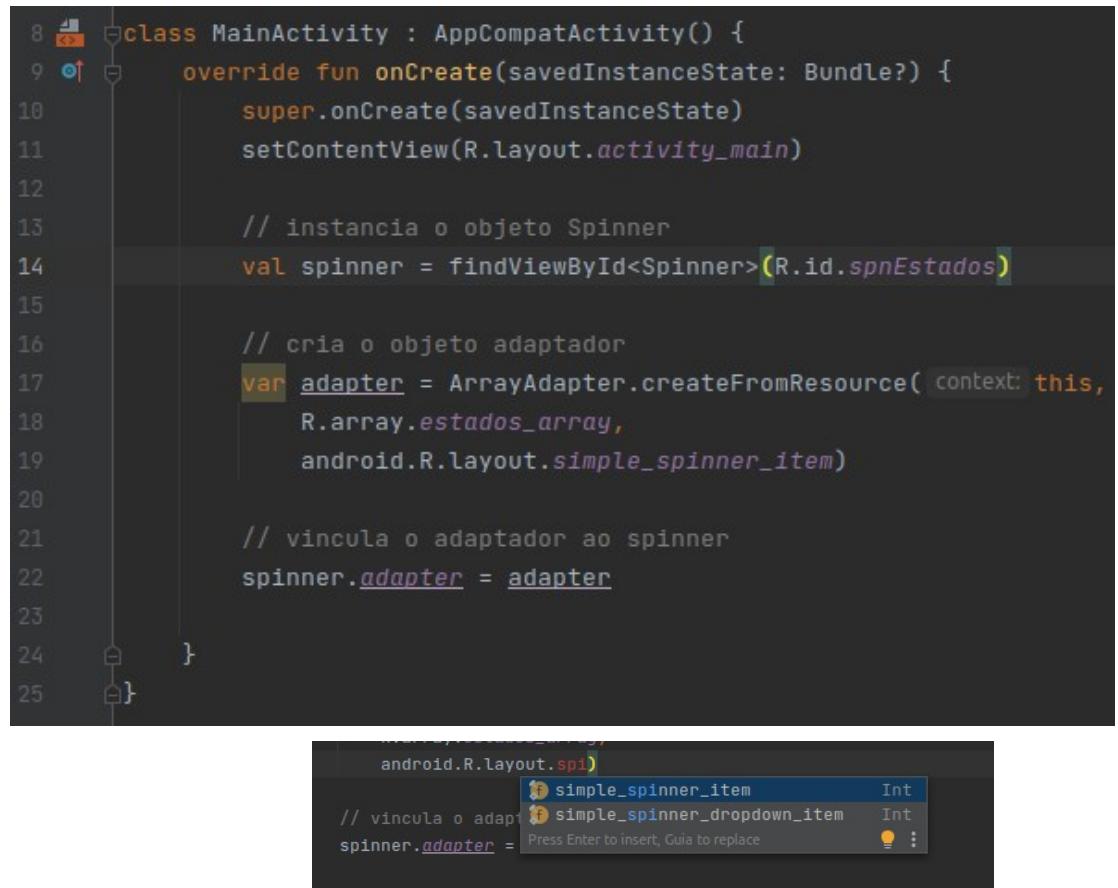
String-array de Estados

- Preencha o conteúdo do arquivo:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string-array name="estados_array">
4          <item>São Paulo</item>
5          <item>Minas Gerais</item>
6          <item>Bahia</item>
7          <item>Paraná</item>
8          <item>Rio de Janeiro</item>
9          <item>Santa Catarina</item>
10         <item>Rio Grande do Sul</item>
11     </string-array>
12 </resources>
```

A4ex8: código fonte

- Linha 14: cria o objeto spinner;
- Linha 17: cria o adaptador;
- Linha 18: especifica os dados do spinner;
- Linha 19: especifica o layout do spinner;
- Linha 22: vincula o adaptador ao spinner

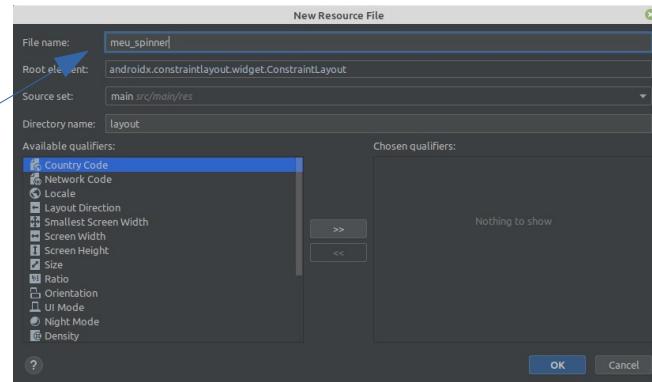


```
8 class MainActivity : AppCompatActivity() {
9     override fun onCreate(savedInstanceState: Bundle?) {
10         super.onCreate(savedInstanceState)
11         setContentView(R.layout.activity_main)
12
13         // instancia o objeto Spinner
14         val spinner = findViewById<Spinner>(R.id.spnEstados)
15
16         // cria o objeto adaptador
17         var adapter = ArrayAdapter.createFromResource(context: this,
18             R.array.estados_array,
19             android.R.layout.simple_spinner_item)
20
21         // vincula o adaptador ao spinner
22         spinner.adapter = adapter
23
24     }
25 }
```

The screenshot shows the Java code for the `MainActivity`. The code creates a `Spinner` object and sets its `Adapter` to an `ArrayAdapter` created from the `simple_spinner_item` layout resource. A dropdown menu at the bottom right shows the available layout resources: `simple_spinner_item` and `simple_spinner_dropdown_item`.

Spinner com layout personalizado

- Podemos criar nosso próprio arquivo de layout XML e vincular ao Spinner;
- Crie um novo Layout Resource File e adicione somente um TextView, removendo o ConstraintLayout padrão do arquivo;
- Exclua o código fonte do arquivo XML e substitua por este:



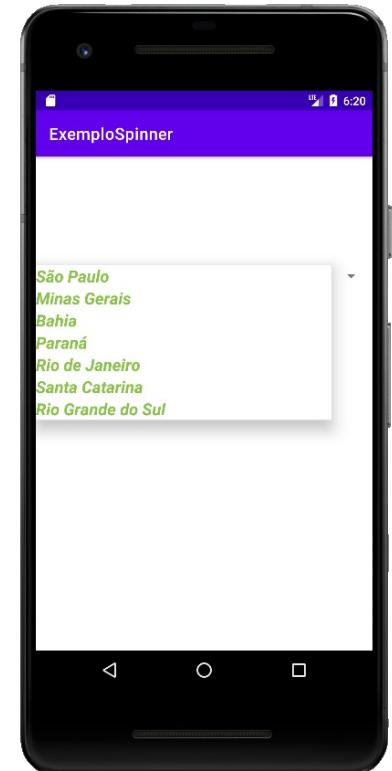
```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <TextView xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:id="@+id/textView2"
6     android:layout_width="wrap_content"
7     android:layout_height="wrap_content"
8     android:layout_gravity="center_horizontal"
9     android:text="TextView"
10    android:textColor="#8BC34A"
11    android:textSize="20sp"
12    android:textStyle="bold|italic"
13    tools:layout_editor_absoluteX="165dp" />
```

A4ex8: novo layout

- Vamos alterar o código fonte para utilizar o layout ‘meu_spinner’;

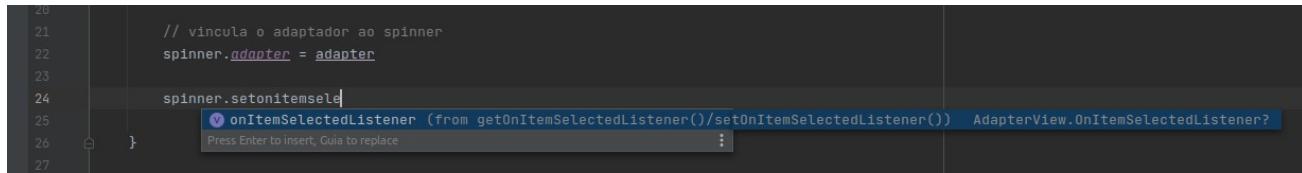
```
16     // cria o objeto adaptador
17     var adapter = ArrayAdapter.createFromResource(context: this,
18             R.array.estados_array,
19             R.layout.meu_spinner)
```

- Execute para ver o resultado.



A4ex8: Listener

- Agora vamos alterar nosso app para executar algum evento caso o usuário escolha um estado;
- Para isso, é necessário definir quem irá ‘responder’ pelo clique que o usuário fizer ao escolher um estado;
- Sempre que queremos incluir eventos em componentes visuais, temos que definir seu Listener.
- Vamos declarar o Listener do Spinner:

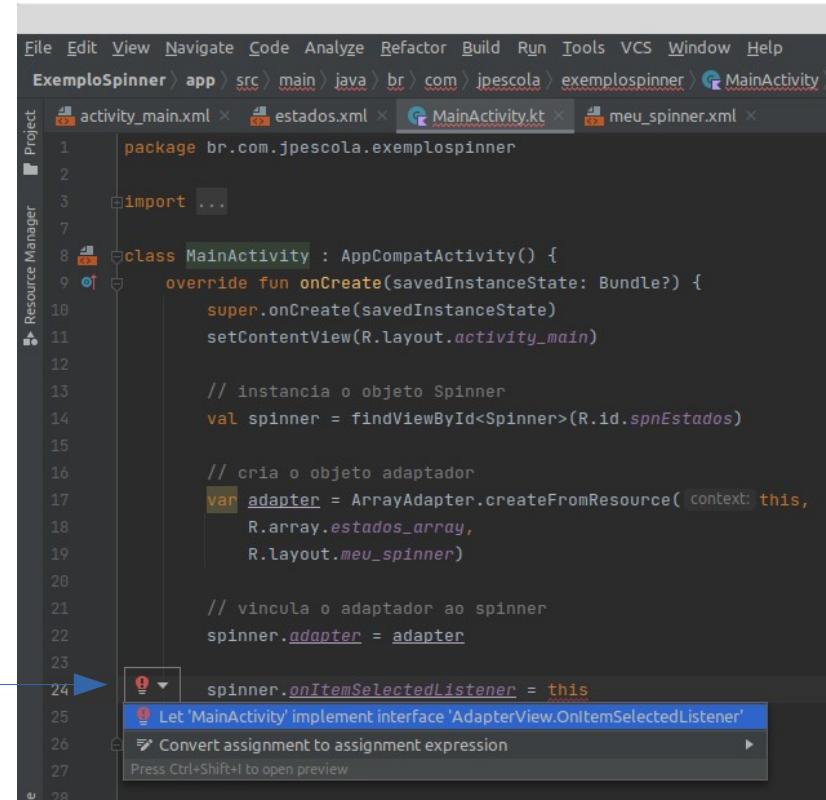


```
20
21     // vincula o adaptador ao spinner
22     spinner.setAdapter(adapter)
23
24     spinner.setOnItemSelectedListener(
25         v -> onItemSelectedListener.onItemSelected(v, spinner.getSelectedItem())
26     )
27 }
```

The screenshot shows a code editor with Java code. Line 24 contains the declaration of a spinner's OnItemSelectedListener. A tooltip is displayed over the 'onItemSelectedListener' part of the code, showing the type 'AdapterView.OnItemSelectedListener'. The tooltip text is: 'onItemSelectedListener (from getOnItemSelectedListener()/setOnItemSelectedListener()) AdapterView.OnItemSelectedListener?'. There is also a note at the bottom of the tooltip: 'Press Enter to insert, Guia to replace'.

A4ex8: implementando os métodos

- Linha 24: estamos definindo a Activity atual como sendo a Listener do Spinner;
 - Isso nos obriga a implementar a interface correspondente;
 - Para isso, clique na lâmpada vermelha e escolha ‘Let...implement interface...’ :

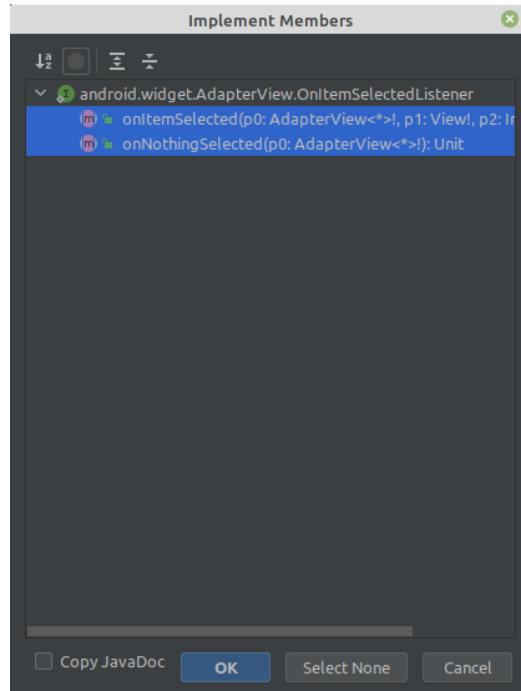


The screenshot shows the Android Studio IDE with the file `MainActivity.kt` open. The code defines a `MainActivity` class that implements `AppCompatActivity`. It creates a `Spinner` and an `ArrayAdapter` to fill it with state names from a resource array. On line 24, there is a red error icon next to the assignment statement `spinner.onItemSelectedListener = this`. A tooltip appears over the error icon with the text "Let 'MainActivity' implement interface 'AdapterView.OnItemSelectedListener'" and a "Convert assignment to assignment expression" option. The code is as follows:

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help  
ExemploSpinner > app > src > main > java > br > com > jpescola > exemplospinner > MainActivity.kt > meu_spinner.xml  
Project Manager  
activity_main.xml estados.xml MainActivity.kt meu_spinner.xml  
1 package br.com.jpescola.exemplospinner  
2  
3 import ...  
4  
5 class MainActivity : AppCompatActivity() {  
6     override fun onCreate(savedInstanceState: Bundle?) {  
7         super.onCreate(savedInstanceState)  
8         setContentView(R.layout.activity_main)  
9     }  
10    // instancia o objeto Spinner  
11    val spinner = findViewById<Spinner>(R.id.spnEstados)  
12  
13    // cria o objeto adaptador  
14    var adapter = ArrayAdapter.createFromResource(context: this,  
15        R.array.estados_array,  
16        R.layout.meu_spinner)  
17  
18    // vincula o adaptador ao spinner  
19    spinner.adapter = adapter  
20  
21    spinner.onItemSelectedListener = this  
22  
23    spinner.onItemSelectedListener = this  
24    spinner.onItemSelectedListener = this  
25  
26    Let 'MainActivity' implement interface 'AdapterView.OnItemSelectedListener'  
27  
28    Convert assignment to assignment expression  
Press Ctrl+Shift+I to open preview
```

A4ex8: confirmando métodos

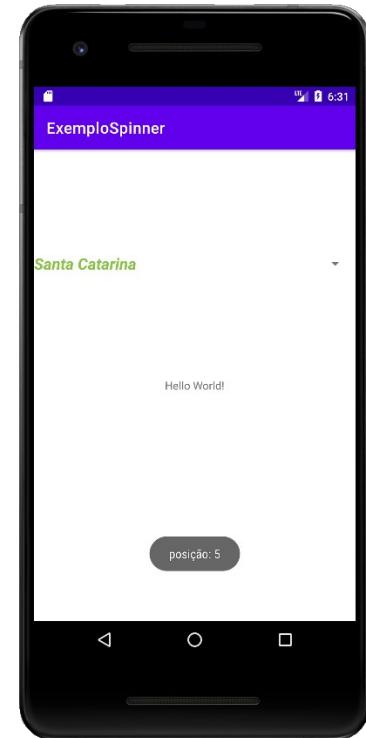
- Escolha os dois métodos obrigatórios e clique em OK:



A4ex8: Toast como evento

- Vamos programar um Toast para ser exibido quando o usuário escolhe um estado;
- Inclua a linha 36 no método ‘onItemSelected’ e faça o teste para ver o resultado.

```
35  override fun onItemSelected(p0: AdapterView<*>?, p1: View?, p2: Int, p3: Long) {  
36      Toast.makeText( context: this, text: "posição: $p2", Toast.LENGTH_SHORT).show()  
37  }  
38  
39  override fun onNothingSelected(p0: AdapterView<*>?) {  
40      TODO( reason: "Not yet implemented")  
41  }
```



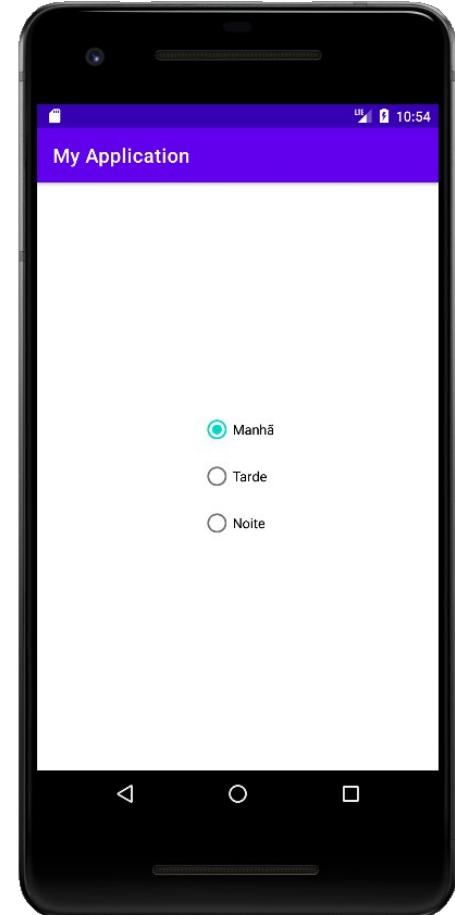
A4ex8: utilizando array

- Também é possível preencher o Spinner com dados provenientes de um array;
- Nesse exemplo, estamos utilizando um array ao invés de buscar os dados do arquivo XML:

```
19 // adaptador de array
20 val dados = arrayOf("São Paulo", "Minas Gerais", "Bahia")
21 var adapter = ArrayAdapter<String>( context: this, android.R.layout.simple_spinner_item, dados)
22
23 // cria o objeto adaptador
24 // var adapter = ArrayAdapter.createFromResource(this,
25 //     R.array.estados_array,
26 //     R.layout.meu_spinner)
```

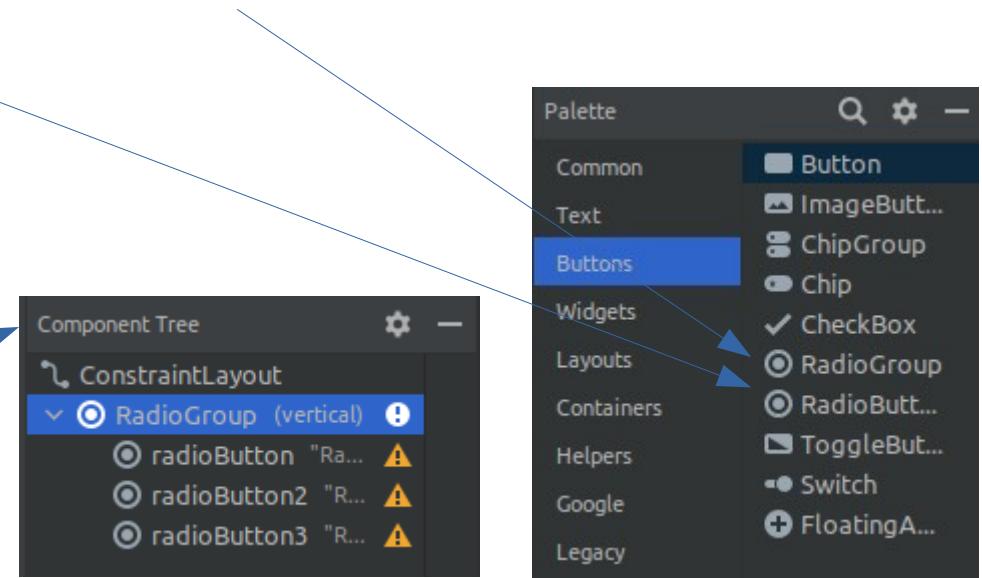
A4ex9

- Neste app vamos aprender a utilizar o componente radio;
- Bastante utilizado em formulários;
- Ideal para quando temos um item (com pouca variação) a ser solicitado ao usuário.



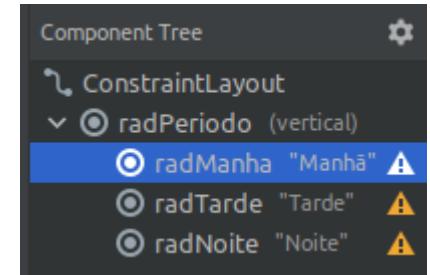
A4ex9: UI

- Crie um novo projeto;
- Na UI, adicione um RadioGroup e dentro dele três RadioButtons;
- Utilize a árvore de componentes para facilitar o processo:



A4ex9: nomeie os radios

- Nosso app vai mostrar um grupo de radios para escolha do período em um cadastro;
 - As opções são manhã, tarde e noite;
- Vamos inserir os IDs dos componentes;
- Defina também seus rótulos.



A4ex9: testando

- Vamos executar o app e ver se os radios estão funcionando;
- Somente uma opção pode ser selecionada, dentre as três possíveis;
- Assim, a escolha do usuário deve alternar entre as definidas pelo desenvolvedor;

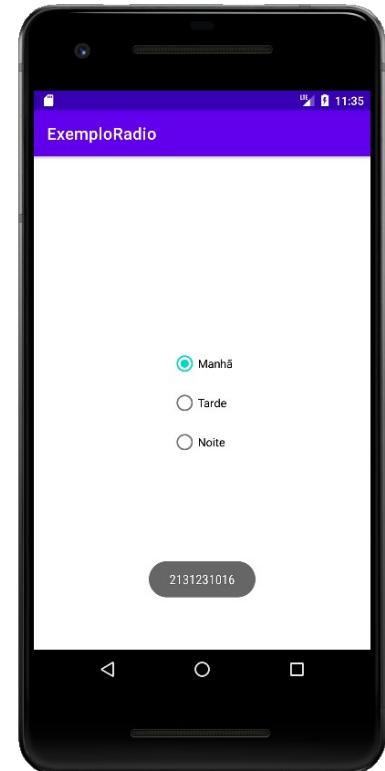
A4ex9: evento radio

- Vamos definir um evento a ser disparado ao escolher o período;
- Para isso, vamos criar o objeto RadioGroup no código fonte;
- Em seguida definir seu listener.

```
8 class MainActivity : AppCompatActivity(), RadioGroup.OnCheckedChangeListener {
9
10    lateinit var radPeriodo: RadioGroup
11
12    override fun onCreate(savedInstanceState: Bundle?) {
13        super.onCreate(savedInstanceState)
14        setContentView(R.layout.activity_main)
15
16        // instancia o objeto RadioGroup
17        radPeriodo = findViewById<RadioGroup>(R.id.radPeriodo)
18
19        // define a Activity atual como listener do objeto
20        radPeriodo.setOnCheckedChangeListener(this)
21    }
22
23
24    // gerencia os eventos do RadioGroup
25    override fun onCheckedChanged(p0: RadioGroup?, p1: Int) {
26        if (p1 == R.id.radManha) // se o periodo da manhã for escolhido
27            Toast.makeText(context: this, text: "$p1", Toast.LENGTH_SHORT).show() // mostra msg
28    }
29}
```

A4ex9: executando

- Vamos rodar o app para ver o resultado;
- Veja que, ao escolher o período da manhã, o ID do componente é exibido pelo Toast.



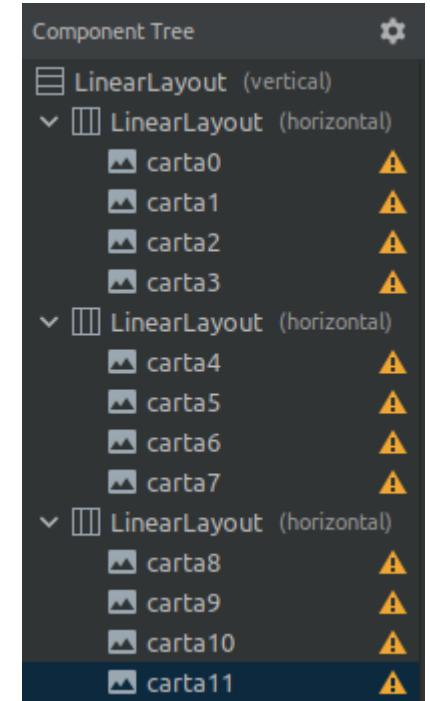
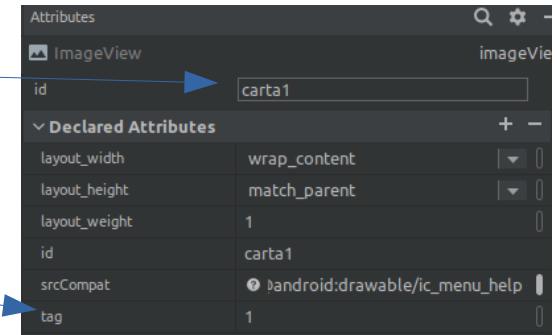
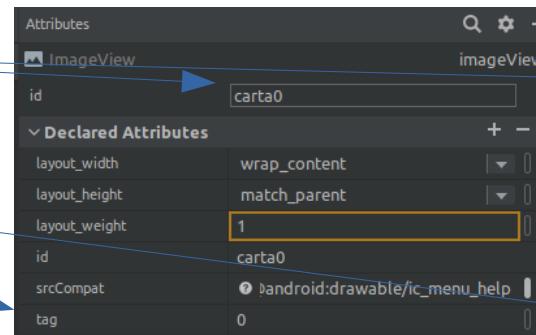
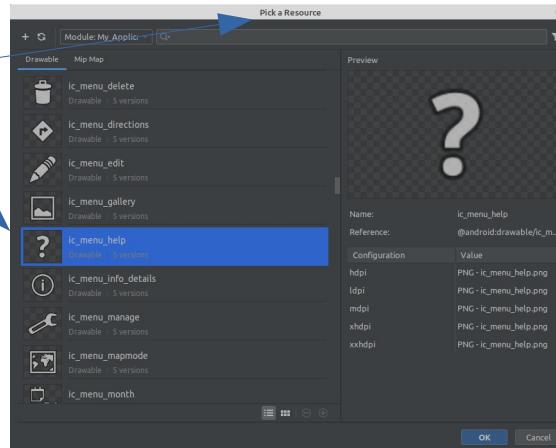
Jogo da Memória

- Vamos criar um jogo da memória em Android utilizando ImageViews;
- Crie um novo projeto e utilize LinearLayout vertical;
- Inclua 3 LinearLayout horizontal e adicione 4 ImageViews em cada.



MemGame: UI

- Quando colocar o ImageView no layout, será exibida a janela “Pick a Resource”;
- Selecione o drawable “ic_menu_help”;
- Os layouts horizontais devem ter layout_height = wrap_content e layout_weight=0.33;
- Para que as imagens fiquem grandes, selecione todos os ImageViews e defina o layout_height = match_parent;
- Defina os IDs das imagens no padrão ‘carta0’, ‘carta1’...;
- Carta0 deve ter tag=0; Carta1 deve ter tag=1...;
- Faça o mesmo para todas as 12 cartas.



MemGame: campos

- Teremos dois objetos ImageView representando as cartas escolhidas;
- As variáveis das linhas 15 e 16 controlam a posição das cartas escolhidas;
- Na linha 17 temos um array com os IDs das imagens das cartas possíveis;
- Na linha 18 um contador de jogadas.

```
11 class MainActivity : AppCompatActivity() {  
12  
13     private lateinit var primeiraImg: ImageView  
14     private lateinit var segundaImg: ImageView  
15     private var primeiraEscolha = -1  
16     private var segundaEscolha = -1  
17     private var cartas = IntArray( size: 12)  
18     private var jogadas = 0  
19  
20 }
```

MemGame: onCreate

- No método principal vamos definir os drawables possíveis do tabuleiro;
- A linha 39 embaralha o tabuleiro.

```
20 override fun onCreate(savedInstanceState: Bundle?) {  
21     super.onCreate(savedInstanceState)  
22     setContentView(R.layout.activity_main)  
23  
24     // definindo as cartas  
25     cartas[0] = android.R.drawable.btn_star  
26     cartas[1] = android.R.drawable.btn_star  
27     cartas[2] = android.R.drawable.ic_menu_edit  
28     cartas[3] = android.R.drawable.ic_menu_edit  
29     cartas[4] = android.R.drawable.ic_menu_camera  
30     cartas[5] = android.R.drawable.ic_menu_camera  
31     cartas[6] = android.R.drawable.ic_menu_call  
32     cartas[7] = android.R.drawable.ic_menu_call  
33     cartas[8] = android.R.drawable.ic_menu_manage  
34     cartas[9] = android.R.drawable.ic_menu_manage  
35     cartas[10] = android.R.drawable.ic_menu_myplaces  
36     cartas[11] = android.R.drawable.ic_menu_myplaces  
37  
38     // embaralha  
39     cartas.shuffle()  
40 }
```

MemGame: onClick

- O método 'onClick' de todas as cartas será o método 'play':
 - Linha 40: armazena o ImageView clicado;
 - Linha 41: verifica se a imagem está habilitada, ignorando o restante do código caso contrário;
 - Linha 45: armazena a tag da carta escolhida;
 - Linha 47: mostra a carta escolhida;
 - Linha 49: inclui o contador de jogadas no título da janela;
 - Linha 52: verifica se as variáveis de posições escolhidas foram definidas;
 - Linha 53: verifica se as cartas são iguais;
 - Linhas 54 e 55: desvira as cartas;
 - Linhas 56 e 57: habilita as cartas novamente;
 - Linhas 59 e 60: reinicia as variáveis de escolhas;
 - Linha 64: verifica se a primeira carta foi escolhida, armazenando a posição, imagem e desabilitando a carta para não ser novamente clicada;
 - Linha 70: executa o procedimento caso seja a segunda carta.

```
38     fun play(view: View){  
39         // armazena a img escolhida  
40         val img = findViewById<ImageView>(view.id)  
41         if (!img.isEnabled) // ignora clicks em cartas viradas  
42             return  
43  
44         // verifica a carta escolhida  
45         val escolha = view.tag.toString().toInt()  
46         // mostra a carta  
47         img.setImageResource(cartas[escolha])  
48         // atualiza o score  
49         title = "${getString(R.string.app_name)} - jogadas: ${++jogadas}"  
50  
51         // na terceira escolha, verifica o resultado  
52         if (primeiraEscolha != -1 && segundaEscolha != -1){ // verifica se acertou  
53             if (cartas[primeiraEscolha] != cartas[segundaEscolha]){ // errou  
54                 primeiraImg.setImageResource(android.R.drawable.ic_menu_help)  
55                 segundaImg.setImageResource(android.R.drawable.ic_menu_help)  
56                 primeiraImg.isEnabled = true  
57                 segundaImg.isEnabled = true  
58             }  
59             primeiraEscolha = -1  
60             segundaEscolha = -1  
61         }  
62  
63         // primeira e segunda escolhas  
64         if (primeiraEscolha == -1) {  
65             primeiraEscolha = escolha  
66             primeiraImg = img  
67             primeiraImg.isEnabled = false  
68         }  
69         else{  
70             segundaEscolha = escolha  
71             segundaImg = img  
72             segundaImg.isEnabled = false  
73         }  
74     }  
75 }
```

MemGame: testando

- Execute o game e veja que está tudo funcionando, entretanto...
 - Como o usuário faz para começar novamente quando termina o jogo?

My Application - jogadas: 2



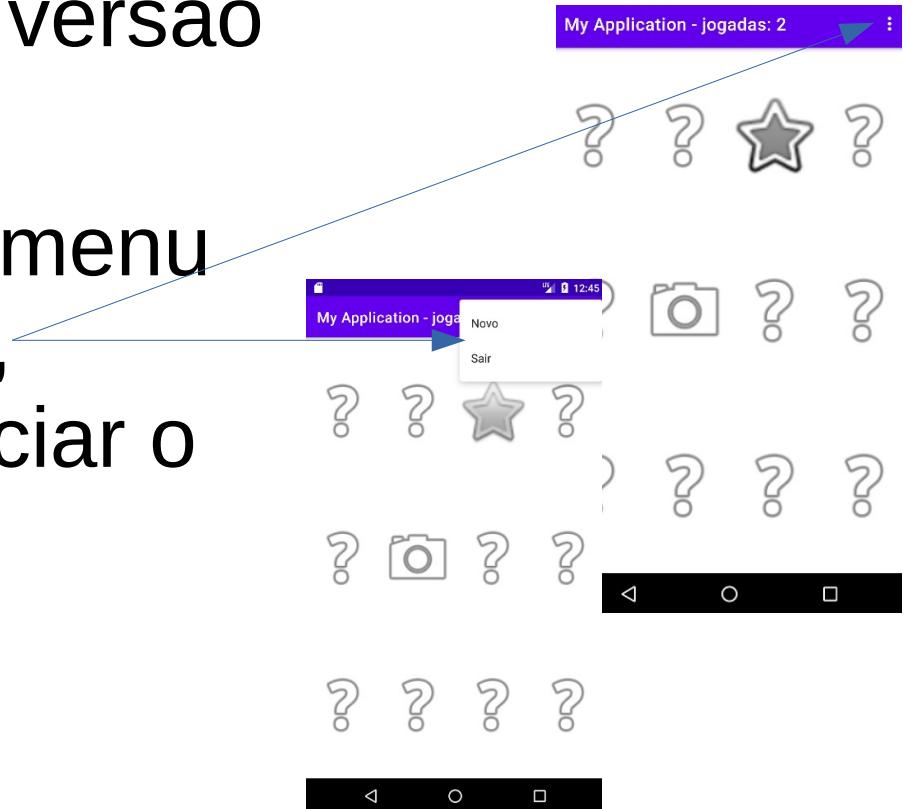
MemGame: plus

- Vamos adicionar um menu com as opções de começar um novo jogo e sair:
 - Crie a pasta ‘menu’ dentro de ‘res’;
 - Crie o arquivo game_menu.xml com as opções ‘Novo’ e ‘Sair’;
 - Inclua os métodos listeners de menu;
- A função ‘novo()’ executa o ‘finish()’ para fechar a aplicação atual e o método ‘startActivity’ para começar novamente o app atual.

```
80    override fun onCreateOptionsMenu(menu: Menu?): Boolean {  
81        val inflater: MenuInflater = menuInflater  
82        inflater.inflate(R.menu.game_menu, menu)  
83        return super.onCreateOptionsMenu(menu)  
84    }  
85  
86    override fun onOptionsItemSelected(item: MenuItem): Boolean {  
87        if (item.itemId == R.id.mnuNovo)  
88            novo()  
89        else if (item.itemId == R.id.mnuSair)  
90            finish()  
91        return super.onOptionsItemSelected(item)  
92    }  
93  
94    fun novo(){  
95        finish(); // fecha o app  
96       .startActivity(intent); // inicia o app atual  
97    }  
98}
```

MemGame: executando

- Agora podemos testar a versão final;
- Veja que agora temos o menu no canto superior direito, permitindo fechar e reiniciar o game.

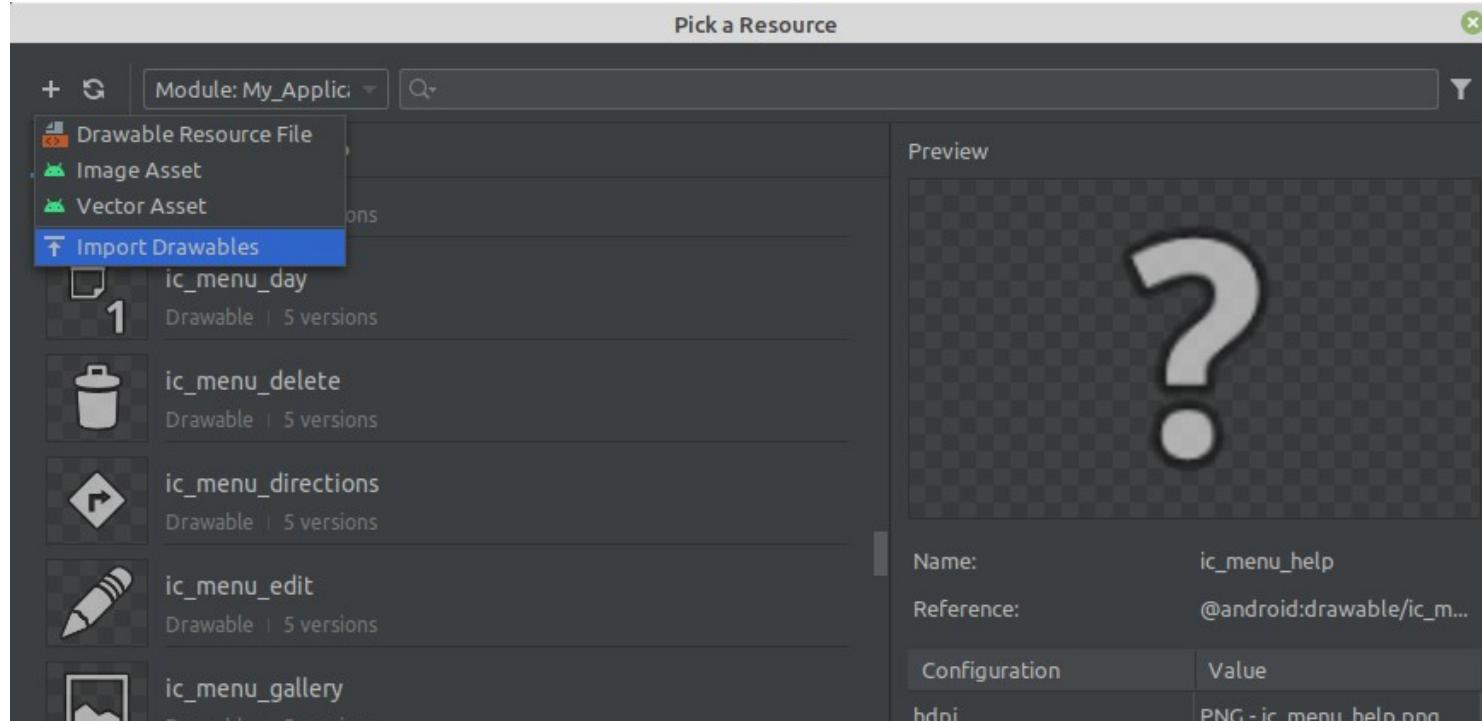


Imagens personalizadas

- Agora vamos adicionar nossas próprias imagens no jogo da memória;
- No atributo srcCompat, podemos definir imagens importadas de outro diretório;
- Selecione um ImageView, busque pelo atributo srcCompat e clique no botão “...”.

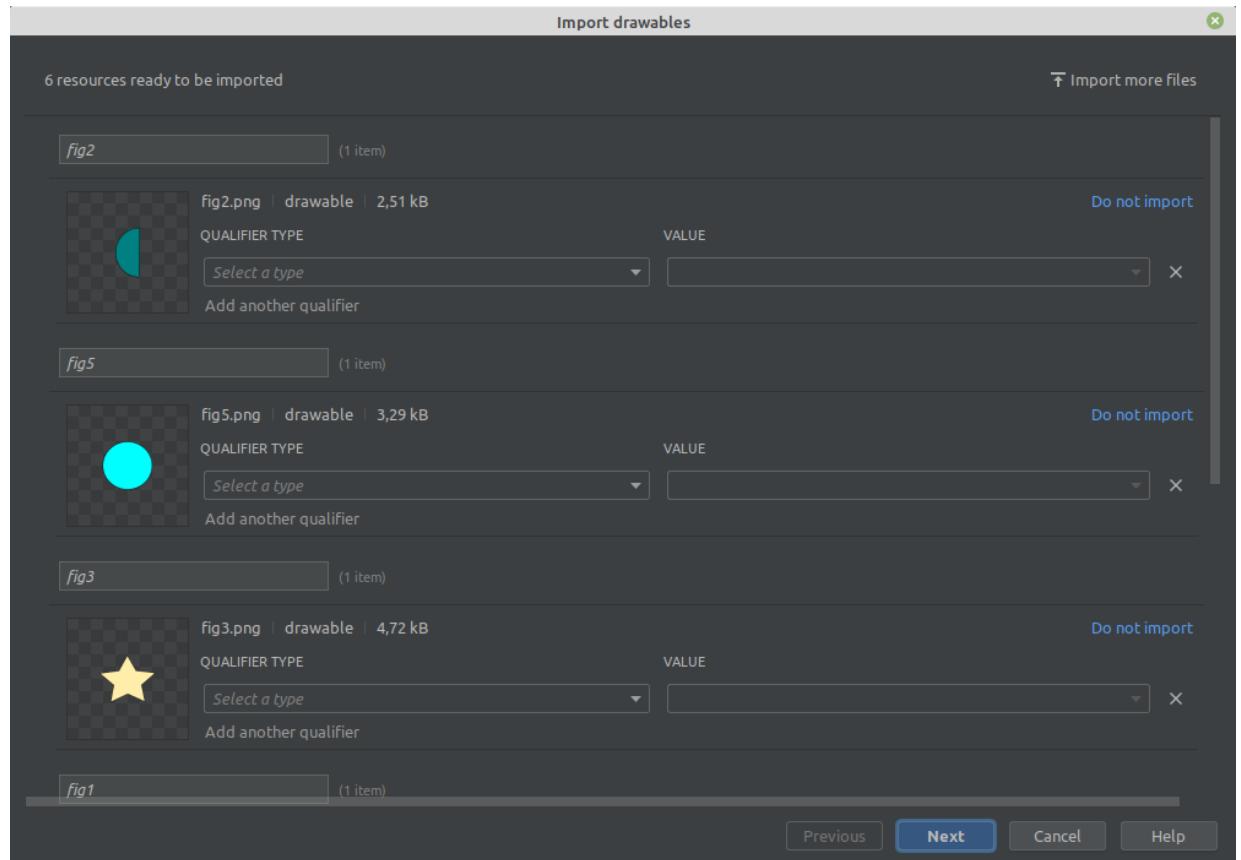
Importando imagens

- Na janela ‘Pick a Resource’, clique em ‘+’ e selecione ‘Import Drawables’:



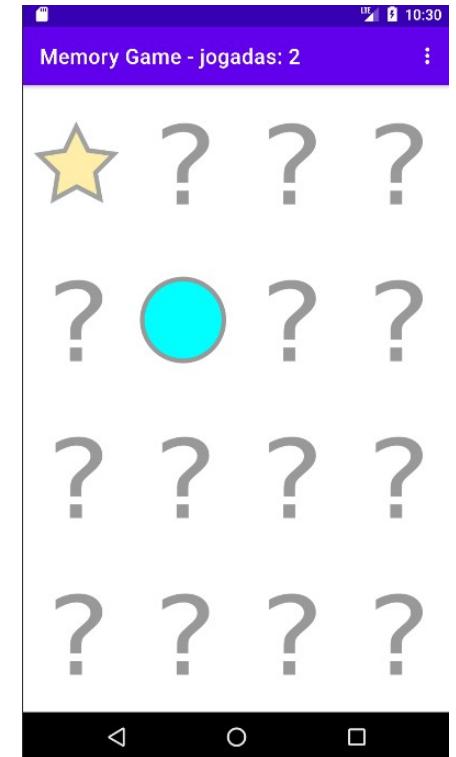
Finalizando importação

- As imagens serão exibidas na janela ‘Import drawables’;
- Clique em ‘Next’.



Jogo Memória Novo

- Agora criar outro projeto para ser um novo jogo da memória, agora utilizando imagens personalizadas;
- As imagens estão em anexo a esta aula.



O que aprendemos?

- Primeiros projetos com Android;
- Trocando a UI;
- Eventos;
- String resource;
- Internacionalização;
- Toast;
- Hint;
- Intent;
- Método putExtra;
- Shared Preferences;
- Log;
- Github;
- Menus;
- Listeners;
- AlertDialog;
- Spinner;
- String-array;
- Radio;
- Jogo da memória;
- Resources de Imagens.

Na próxima aula...

- Web services e APIs;

Referências

- <https://github.com/jpescola/ExemploRadio>
- <https://github.com/jpescola/ExemploSpinner>
- <https://github.com/jpescola/ExemploMenu>
- <https://github.com/jpescola/ParOuImpar>
- <https://github.com/jpescola/MemoryGame>
- <https://github.com/jpescola/MemoryGameNew>