



Unidade 2

Seção 1



Desenvolvimento para Dispositivos Móveis

Webaula 1

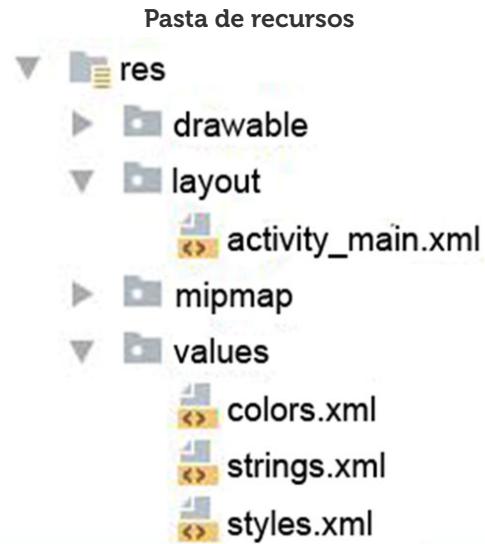
Desenvolvendo UI com *ConstraintLayout*

Nesta webaula vamos apresentar caminhos alternativos para criar um recurso de layout e detectar uma interação entre o usuário e o aplicativo.

Serão apresentados os conceitos de *Listeners*, que são Interfaces capazes de detectar uma interação entre o usuário e o layout. E para que seja possível construir um layout responsivo, também será apresentado o elemento *ConstraintLayout*.

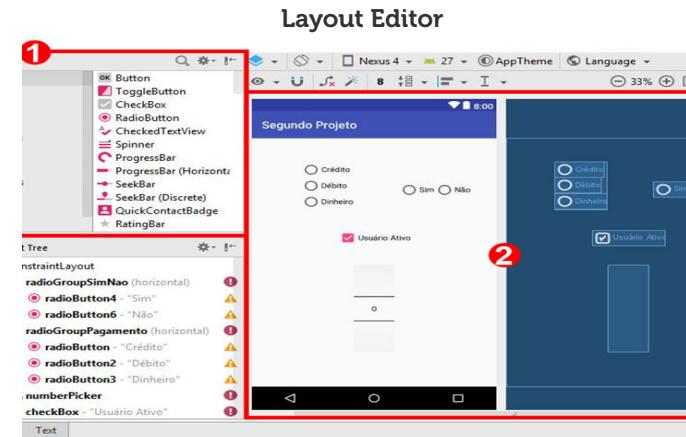
O Android organiza os recursos que não são código-fonte java na pasta “res”, abreviatura para “resources” (recursos). Os layouts utilizados pelos usuários para que possam interagir com o aplicativo são conhecidos como “recursos de layouts” e estão armazenados na subpasta layout.

As demais **subpastas** apresentam outros tipos de recursos.

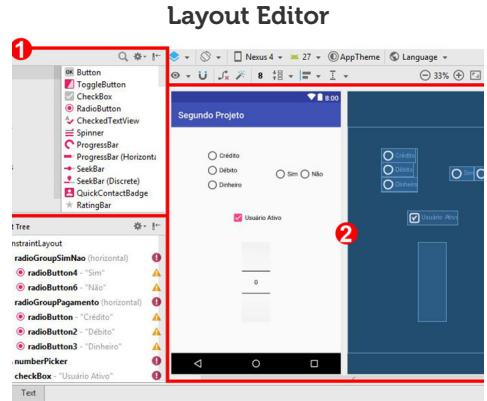


Fonte: captura de tela do Android Studio, elaborada pelo autor.

Além dos elementos *ScrollView* e *LinearLayout*, o Android oferece outros elementos para construirmos os layouts, como o *ConstraintLayout*. Através deste elemento é possível criar um recurso de layout com uma interface complexa, sem a necessidade de inserir os elementos de forma hierárquica, conforme utilizado no *LinearLayout*. Toda a construção da interface poderá ser modelada através do *Layout Editor*. Ao abrirmos um arquivo disponível na pasta de recursos de layout, o *Layout Editor* será carregado.



Fonte: Captura de tela do Android Studio, elaborada pelo autor.



Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Explore a galeria e conheça o modo de criação de layout utilizando o elemento *ConstraintLayout*.

1: A janela *Palette* dispõe os elementos para a construção do layout. É possível arrastar os elementos disponíveis nesta janela para a área de *Design Editor*.

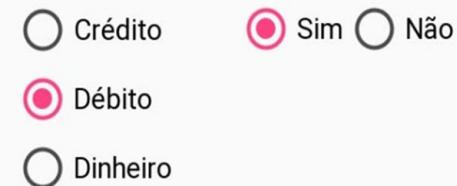


Elementos *RadioGroup* e *RadioButton*

No desenvolvimento de um layout que exibirá para o usuário uma lista com diversas opções, das quais ele poderá selecionar apenas uma, pode-se utilizar o elemento *RadioButton*.

Há dois elementos *RadioGroup*, e cada elemento *RadioGroup* possui os seus elementos-filhos *RadioButton*. Durante a execução do aplicativo, em cada *RadioGroup* criado, será permitido selecionar apenas um *RadioButton*.

RadioGroup e *RadioButton*

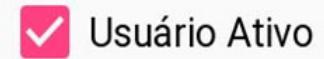


Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Elemento *CheckBox*

Outro elemento que permite selecionar várias opções é o *CheckBox*, em que é possível selecionar vários *CheckBox* disponíveis em um layout, sem a necessidade de criar grupos. O atributo `android:checked="true"` permite inicializar o elemento já selecionado. Se alterar o valor do atributo para “`false`” inicializará o elemento desmarcado.

Elemento *Checkbox*

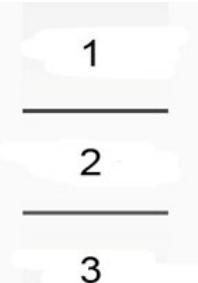


Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Elemento NumberPick

Ainda há o elemento *NumberPick*, que permite selecionar um número dentro de um intervalo predefinido. Para que o *NumberPicker* funcione de maneira adequada, será necessário instanciar este objeto na `MainActivity.java` e definir os métodos `setMinValue(valor)` e `setMaxValue(valor)` para este objeto.

Clique na imagem para ver o código.



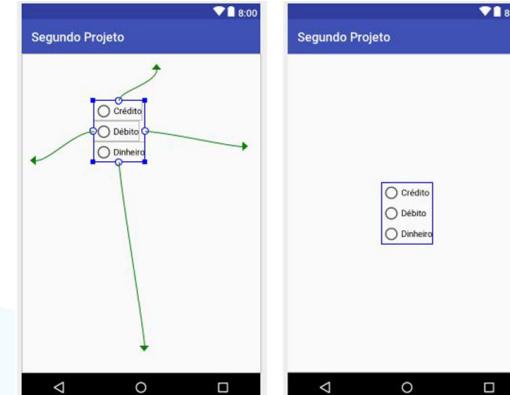
Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Como o *ConstraintLayout* organiza os elementos Views

Diferentemente do *LinearLayout*, não há a necessidade de inserir os elementos de forma hierárquica. O *ConstraintLayout* realiza uma conexão entre todos os elementos disponíveis na interface.

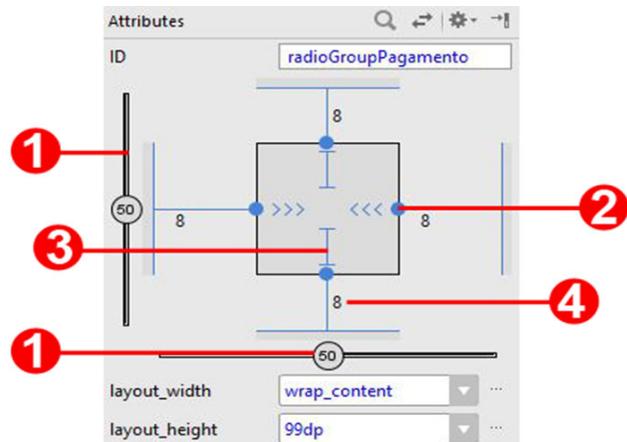
[Explicação do exemplo.](#)

Posicionamento do *RadioGroup*



Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Janela *Attributes* – ajustes para posicionamento de um elemento



Fonte: Captura de tela do Android Studio, elaborada pelo autor.

Explore a galeria e veja as opções disponíveis na janela *Attributes* para posicionamento de um elemento.

1: *Constraint Bias* são barras que permitem realizar ajustes nos elementos verticalmente ou horizontalmente.

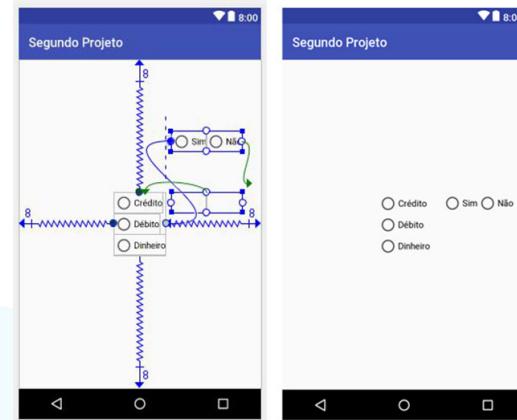


Conexão entre dois elementos do layout

Utilize na janela *Attributes* a barra *Constraint Bias* para posicionar o *RadioGroup* com as opções de pagamento na parte superior do *layout*. Observe que o outro *RadioGroup* também será repositionado automaticamente pois estão conectados.

Explicação do exemplo.

Posicionamento entre elementos



Fonte: Captura de tela do Android Studio, elaborada pelo autor.

No Android também há diversos caminhos para detectarmos a interação entre o usuário e os elementos *Views* de um layout. Todos os elementos *Views* possuem “ouvintes” que são capazes de detectar a interação entre o usuário e o elemento.

[Clique aqui para saber mais](#)

O *Listener* é uma interface que age como uma camada que detecta a ação do usuário e executa um comportamento dentro do aplicativo. Por exemplo, é possível “ouvir” quando um usuário clica em um botão através da interface *OnClickListener*. Também é possível ouvir quando um valor de um *NumberPicker* é alterado através da Interface *OnValueChangeListener*.



Clique nos botões e conheça algumas Interfaces de Listener e os métodos correspondentes que deverão ser sobrepostos.

OnClickListener

OnLongClickListener

OnValueChangeListener

OnCheckedChangeListener

Para “ouvir” uma interação, deve-se criar uma instância do *Listener* que se deseja ouvir na classe *MainActivity.java*.

Com o código apresentado, é possível desenvolver um comportamento específico para o aplicativo sempre que um valor for alterado no elemento *NumberPicker*.

Explicação do exemplo.

Posicionamento entre elementos

```
1. public class MainActivity extends AppCompatActivity {  
2.     // Criando uma classe anônima para o Listener  
3.     private NumberPicker.OnValueChangeListener  
4.         valorAlteradoListener = new NumberPicker.OnValueChangeListener() {  
5.             @Override  
6.             public void onValueChange(NumberPicker numberPicker, int  
7.                 oldValue, int newValue) {  
8.                 // Adicione a lógica aqui  
9.             }  
10.            @Override  
11.            protected void onCreate(Bundle savedInstanceState) {  
12.                super.onCreate(savedInstanceState);  
13.                setContentView(R.layout.activity_main);  
14.                NumberPicker mNumberPicker =  
15.                    findViewById(R.id.numberPicker);  
16.                    // Definimos o intervalo de números entre 0 e 10  
17.                    mNumberPicker.setMinValue(0);  
18.                    mNumberPicker.setMaxValue(10);  
19.                    // Associamos o "ouvinte" com o NumberPicker  
20.                    mNumberPicker.setOnValueChangedListener(valorAlteradoListener);  
21.            }  
}
```

Fonte: Captura de tela do Android Studio, elaborada pelo autor.



Link

Ferramentas disponíveis no *Layout Editor* - *site* oficial de desenvolvimento Android, disponível no *link*

Disponível em: <https://bit.ly/2LwH6RI> . Acesso em: 20 abr. 2018.



Link

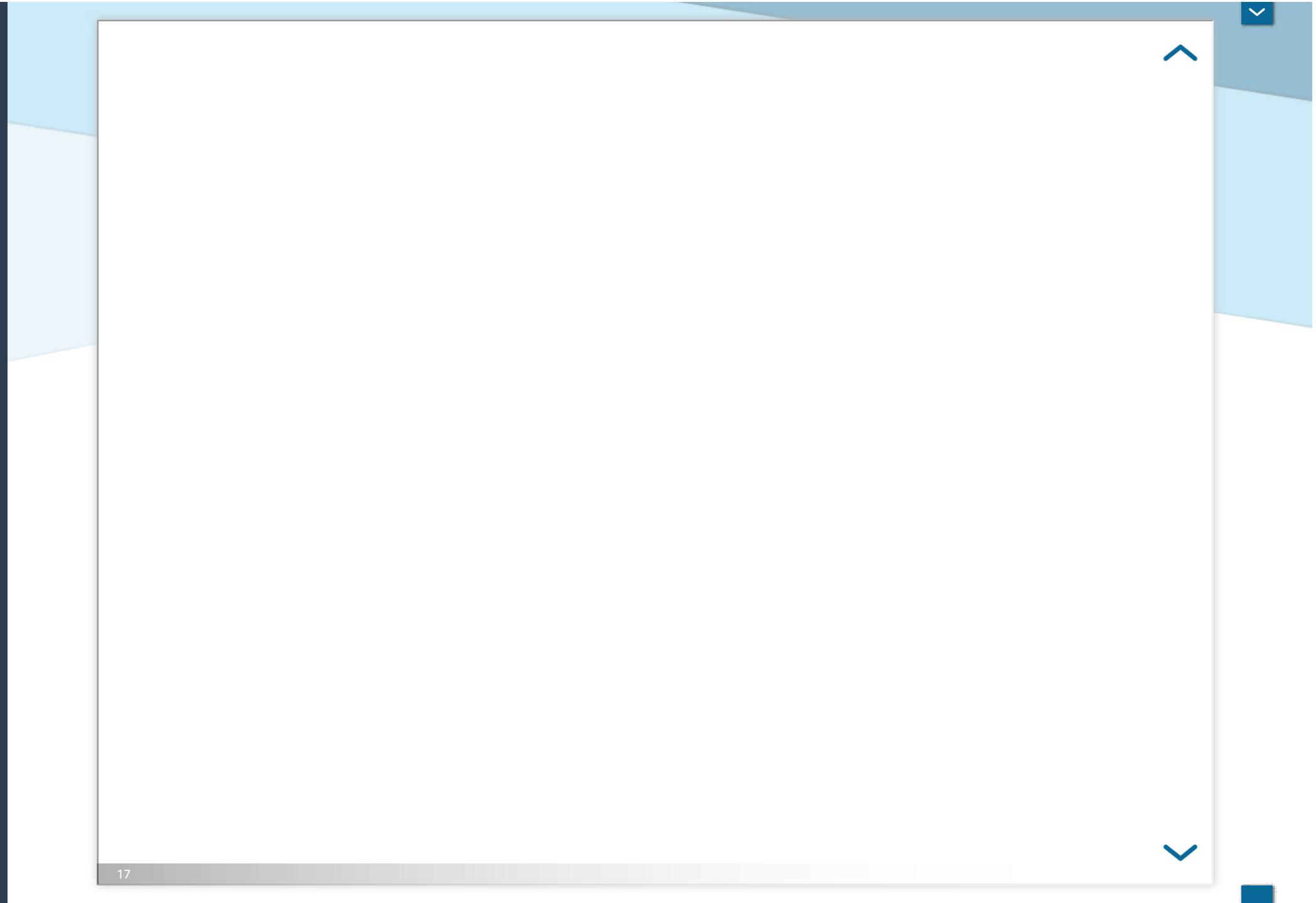
Escutas de Evento - *site* oficial de desenvolvimento do Android, disponível no *link*

Disponível em: <https://bit.ly/2NKILQE> . Acesso em: 20 abr. 2018.



Vimos como trabalhar com o *ConstraintLayout*, um elemento do tipo layout que o permite criar um recurso de layout através do modo *Design* no *Layout Editor*. Também aplicamos uma nova técnica para detectarmos uma interação entre o usuário e o aplicativo através das *Interfaces Listeners*.







Bons estudos!