

Programação para Dispositivos Móveis

Leopoldo Teixeira

lm@cin.ufpe.br | @leopoldomt

O que se entende por
interface com o usuário?

Neste curso

- Foco nas telas
- Interface com o usuário é algo mais amplo
- Activities exibem telas que interagem com o usuário
- Android fornece diferentes classes para estruturar interação

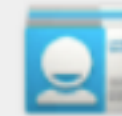
O que são widgets?

In computer programming, a widget (or control) is an element of a graphical user interface (GUI) that displays an information arrangement changeable by the user, such as a window or a text box.

*The **defining characteristic** of a widget is to provide a **single interaction point** for the direct manipulation of a given kind of data.*

In other words, widgets are basic visual building blocks which, combined in an application, hold all the data processed by the application and the available interactions on this data.

Phone-only, unsynced co...



Name



Company

Title

PHONE

Phone

MOBILE

EMAIL

Email

HOME

ADDRESS

Address

HOME

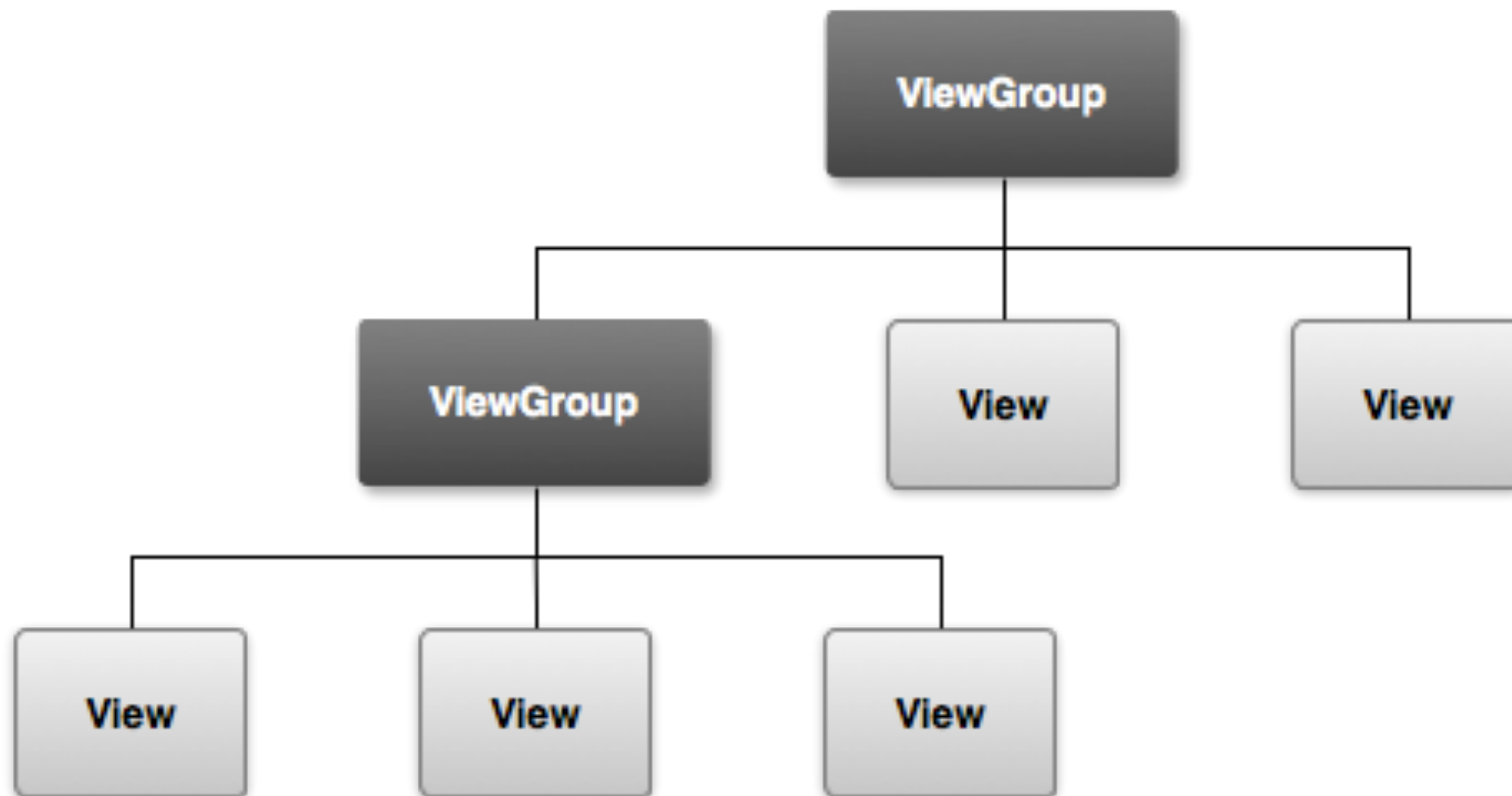
Size, Margin, Padding

- Widgets tem tamanho, pode ser definido pelo conteúdo, ou pelo elemento pai (container)
- Widgets tem margens, assim como em CSS, permitindo separação entre widgets
- Widgets tem padding, assim como em CSS, separando o espaço entre conteúdo e limite do widget, para visual mais agradável

Containers

- Forma de organizar múltiplos widgets em um tipo de estrutura
- Definimos um padrão de organização para os widgets
- Na maioria dos frameworks de GUI, um container tem filhos, que podem ser widgets ou outros containers

User Interface Layout



<http://developer.android.com/guide/topics/ui/overview.html>

Padrões para Containers

- Coloque todos os elementos em linha, um após o outro
- Coloque todos os elementos em uma coluna, um abaixo do outro
- Arranje os elementos em uma tabela ou grid, com quantidade específica de linhas e colunas
- Ancore os elementos nas laterais do container
- Empilhe todos os elementos...

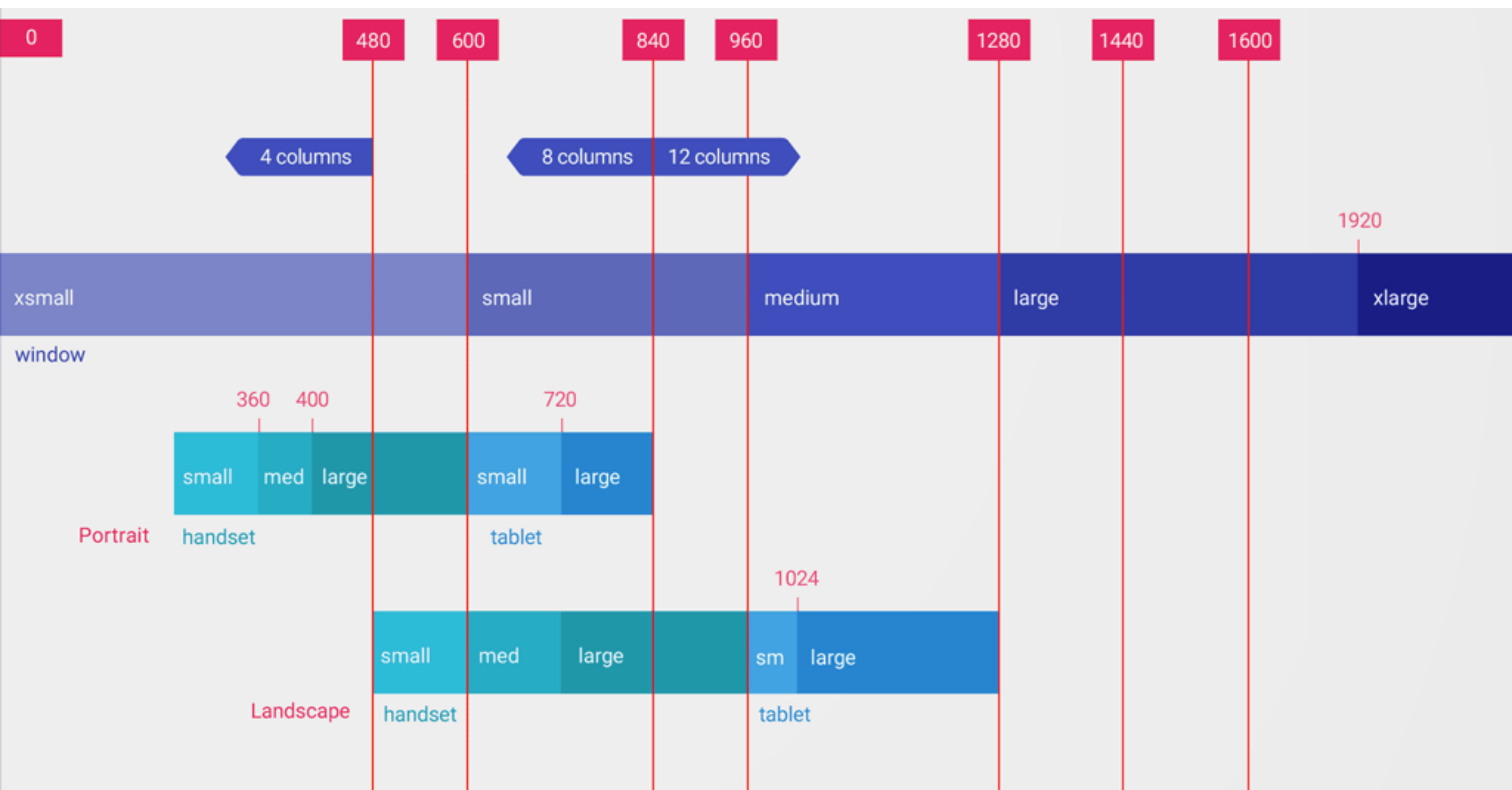
Por qual razão
usamos containers?

*Density-independent
pixels*

$$px = dp * (dpi / 160)$$

Density Qualifiers

Qualifier	Descrição
ldpi	~120dpi
mdpi	~160dpi (baseline)
hdpi	~240dpi
xhdpi	~320dpi
xxhdpi	~480dpi
xxxhdpi	~640dpi
nodpi	todas as densidades (não faz <i>scaling</i>)
tvdpi	~213dpi (apenas para TV apps)



<https://developer.android.com/training/multiscreen/screensizes.html#alternative-layouts>

Declarando Layouts

```
<?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" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

<http://developer.android.com/guide/topics/ui/overview.html>

Layouts

- Correspondem a ViewGroups genéricos
- Definem uma estrutura visual para interface com o usuário em uma activity ou app widget
- Podem ser declarados em XML ou instanciados em tempo de execução

Quais as vantagens de
utilizar XML para
declarar layouts?

Criando Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<ViewGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+[package:]id/resource_name"
    android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
    android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
    [ ViewGroup-specific attributes ] >
    <View
        android:id="@+[package:]id/resource_name"
        android:layout_height=["dimension" | "fill_parent" | "wrap_content"]
        android:layout_width=["dimension" | "fill_parent" | "wrap_content"]
        [ View-specific attributes ] >
        <requestFocus/>
    </View>
    <ViewGroup >
        <View />
    </ViewGroup>
    <include layout="@layout/layout_resource"/>
</ViewGroup>
```

Mais detalhes em

<http://developer.android.com/guide/topics/resources/layout-resource.html>

Exemplo

```
<?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" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

<http://developer.android.com/guide/topics/ui/overview.html>

Carregando o resource

- Ao compilar a aplicação, cada arquivo XML é compilado em um resource View
- Ao iniciar Activity, devemos carregar o layout correspondente no código, utilizando setContentView:

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main_layout);  
}
```

Atributos

- Objetos View e ViewGroup suportam diversos atributos XML
- Alguns atributos são específicos a um tipo de objeto, como textSize para TextView
- Alguns são comuns a todos os objetos, como id
- Outros atributos são considerados parâmetros de layout, como width e height

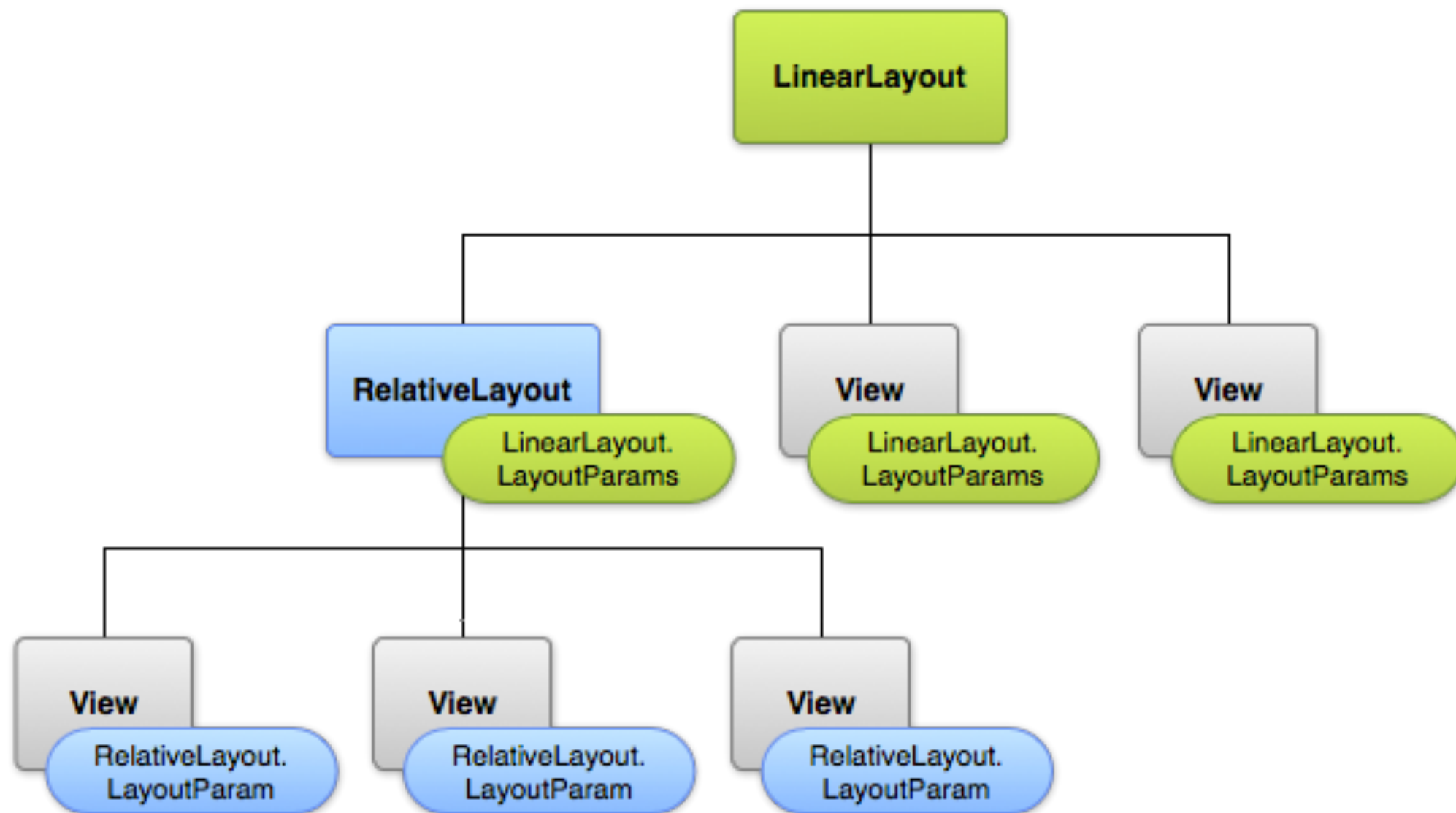
ID

- Identificador único, referenciado no código como um inteiro, mas geralmente assinalado como string no XML
- Sintaxe: **`android:id="@+id/my_button"`**
- Criamos uma instância do objeto View ao criar nossa Activity:
 - **`Button b = (Button) findViewById(R.id.my_button);`**
- Importante definir IDs quando criando layouts relativos

Layout Parameters

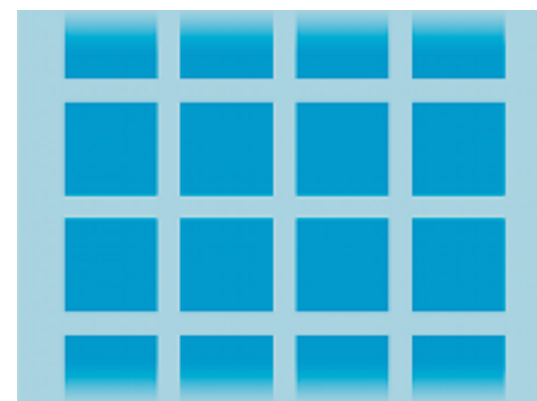
- Os atributos com nome iniciado em `layout_` definem parâmetros para a View que são apropriados para o ViewGroup pai
- Todo ViewGroup e View deve definir largura e altura (`layout_width` e `layout_height`)
 - *wrap_content* vs. *match_parent*
 - *density-independent pixel units (dp)*
- Existem vários outros parâmetros opcionais, para definir margens, bordas, etc.

Layout Parameters



<http://developer.android.com/guide/topics/ui/declaring-layout.html>

Layouts



LinearLayout

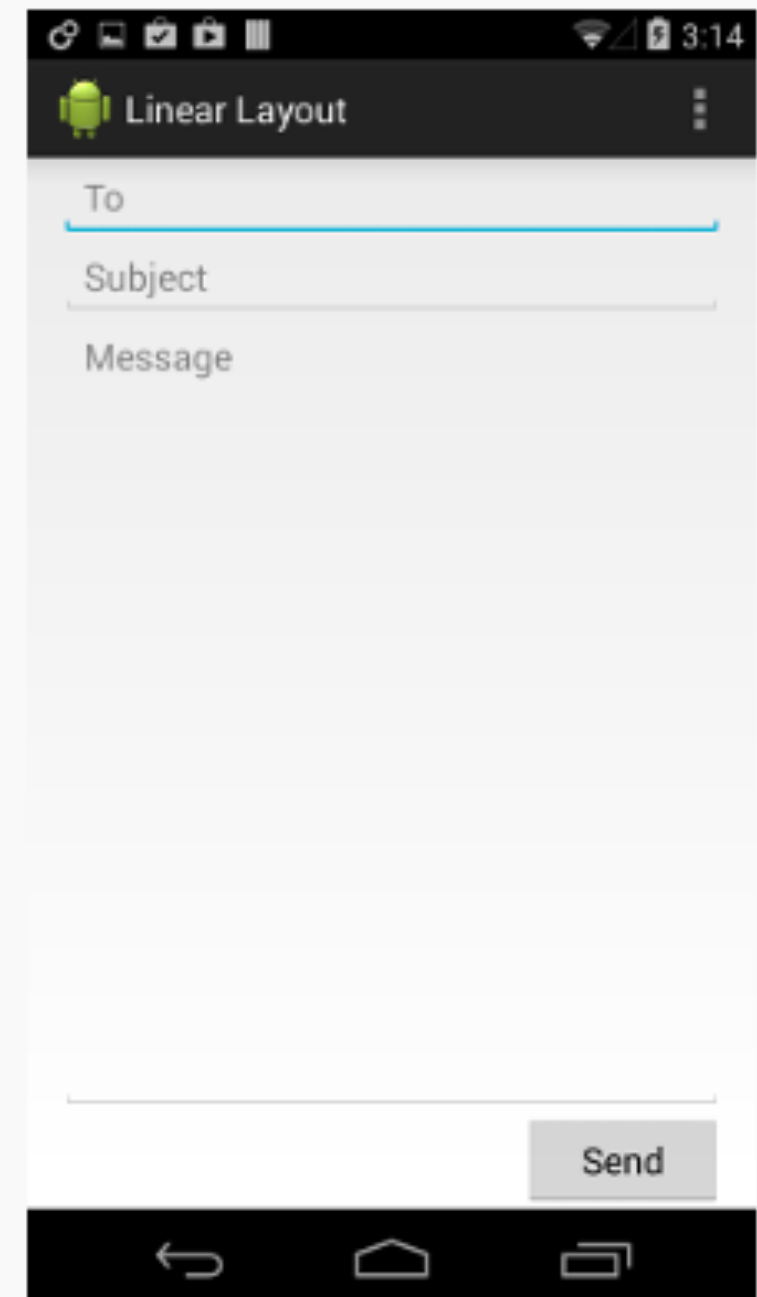
- Arranjam views em linhas, horizontais ou verticais
- Barra de rolagem é criada caso tamanho da janela exceda tamanho da tela
- `layout_weight`: 'importância' da view em termos de espaço ocupado



```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/a
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>

```



RelativeLayout

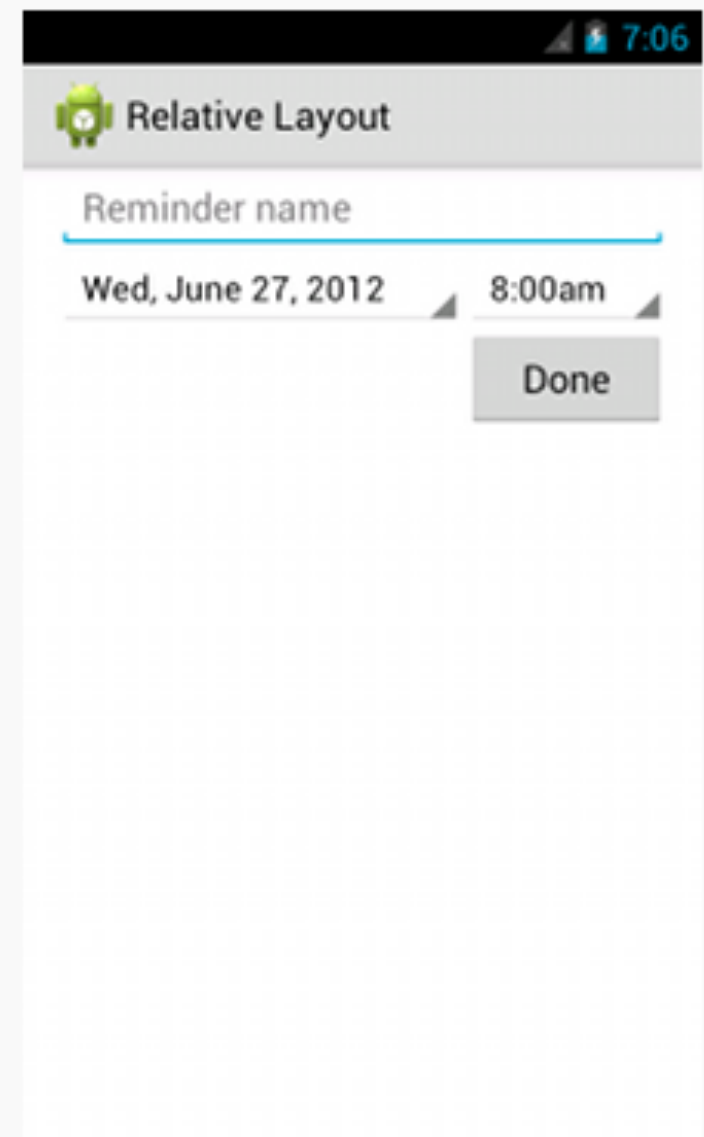
- Exibe filhas em posições relativas
- A posição pode ser relativa a elementos irmãos ou ao elemento pai
 - `layout_alignParentTop`
 - `layout_centerVertical`
 - `layout_below`
 - `layout_toRightOf`



```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@+id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>

```



TableLayout...

ConstraintLayout...

Quick Demo

Views

- Blocos básicos de componentes de interação com usuário
- Ocupam espaço retangular na tela
- São responsáveis por seu próprio desenho e por eventos direcionados a ela

Operações comuns

- Definir visibilidade
 - mostrar ou esconder a View
- Definir estado (checked/unchecked)
- Definir listeners
 - código que deve ser executado quando eventos específicos ocorrem
- Definir propriedades
 - opacidade, background, rotation
- Gerenciando foco do teclado, por exemplo

Fontes dos Eventos

- Interação com o usuário
 - toque,
 - teclado, trackball
- Sistema
 - mudanças no ciclo de vida

Input Controls

- Button
- Text field
 - Autocomplete
- ToggleButton
- Checkbox
- Spinner
- Pickers
- ...



Buttons

- Texto ou ícone (ou ambos) que comunica uma ação
- A ação ocorre ao usuário tocar o botão
- A ação ocorrida no clique (onClick) pode ser definida via XML, por meio de atributos
- Ou ainda por meio de listeners, como já visto...



ToggleButton e Switch

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```



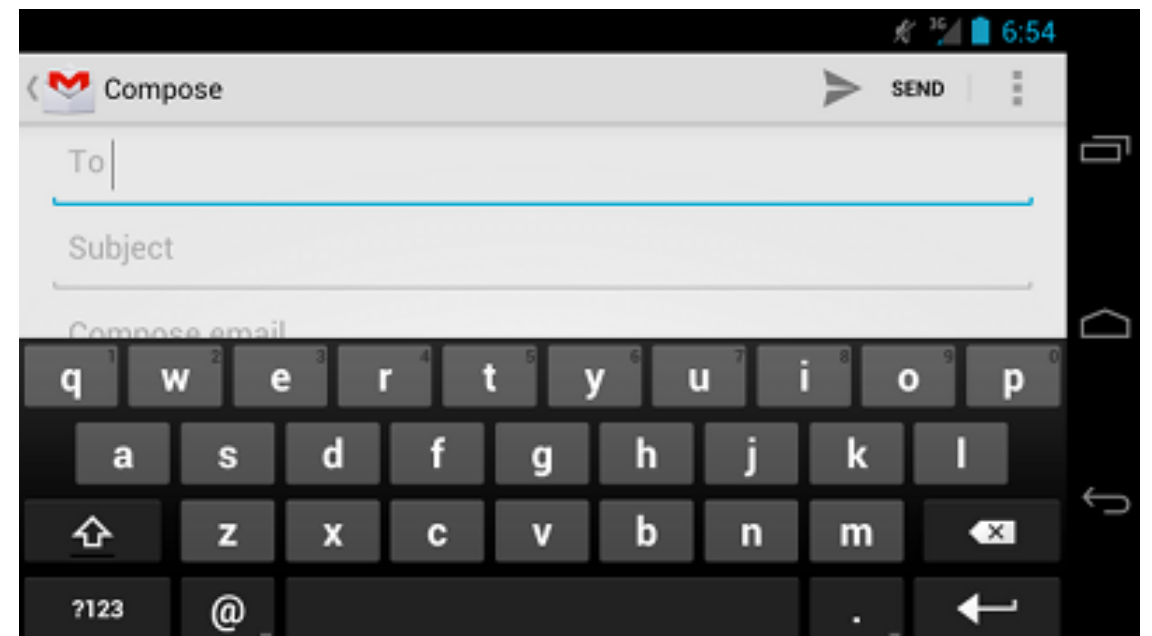
Toggle buttons



Switches (Android 4.0+)

Text Fields

- Permitem que o usuário digite texto
- Podem ser single line ou multi-line
- Pode especificar o tipo de teclado a ser utilizado
- Pode controlar outros comportamentos, como tudo maiúsculo ou apenas primeiras letras (contatos)
- Pode especificar uma ação, como 'buscar' ou 'enviar'



Tipos de Teclado



Padrão de entrada



Padrão de emails



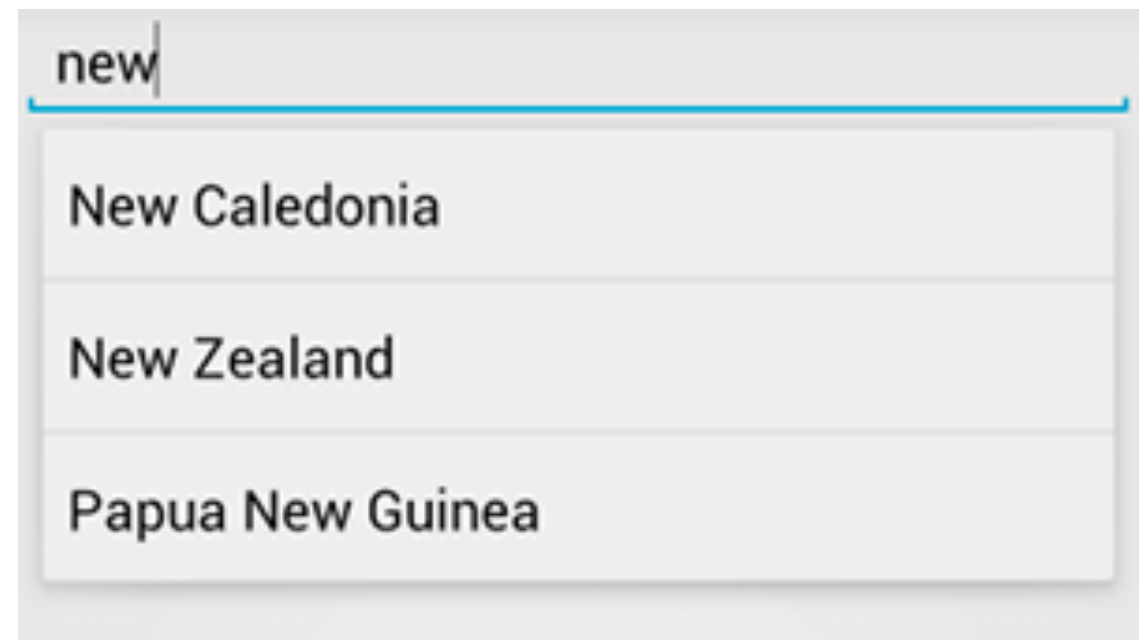
Padrão de telefone



android:imeOptions="actionSend"

AutoCompleteTextView

- Uma caixa de texto (TextView) editável
- Fornece sugestões na medida que o usuário digita texto
- Alternativa para um drop down muito grande
- A fonte de dados pode variar



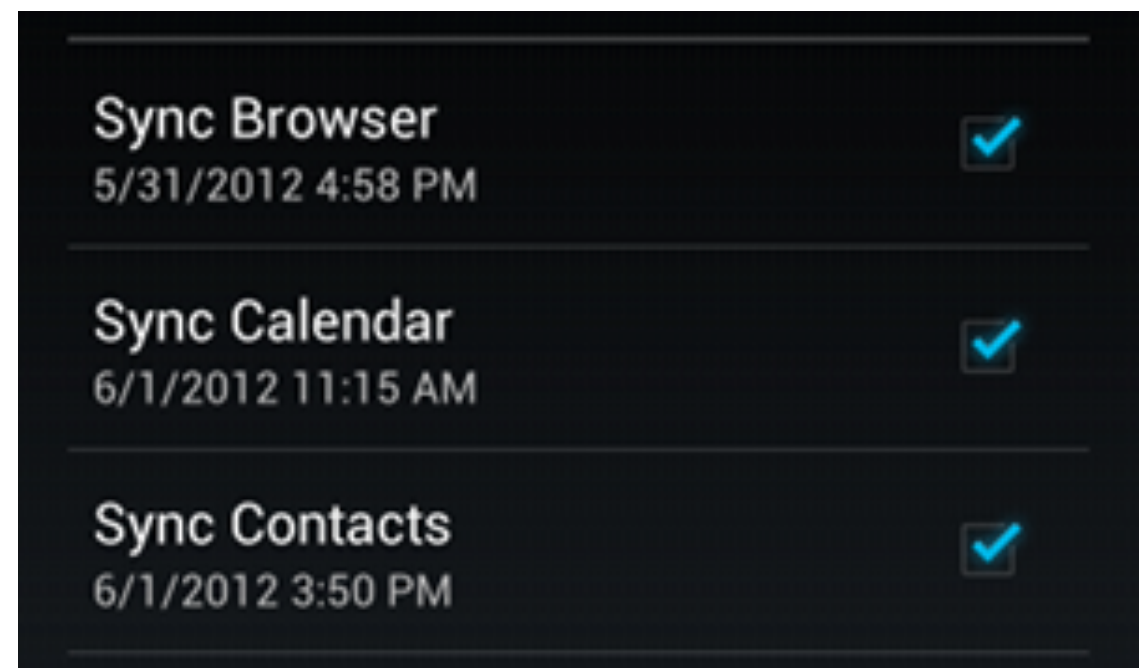
```
// Get a reference to the AutoCompleteTextView
AutoCompleteTextView textView = (AutoCompleteTextView) findViewById(R.id.autocomplete_country);

// Create an ArrayAdapter containing country names
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    R.layout.list_item, COUNTRIES);

// Set the adapter for the AutoCompleteTextView
textView.setAdapter(adapter);
```

Checkbox

- Permitem escolher uma ou mais opções de um conjunto
- Normalmente apresentados em uma lista vertical
- Resposta a eventos definida no XML ou via listeners

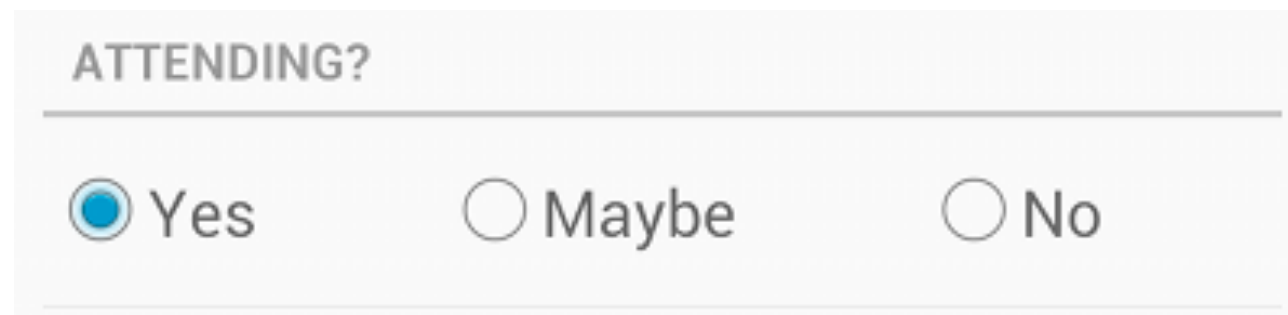


ViewGroup

- Agrupar views relacionadas
 - Por ex.: Radio Groups
- Dá suporte à composição de views
- View invisível que contem outras views
- Usada para agrupar e organizar conjuntos de views
- Classe base para View Containers e Layouts

RadioGroup

- ViewGroup contendo um conjunto de radio buttons (ou checkboxes)
- Apenas um botão pode ser selecionado a cada momento
- Se não é necessário exibir todas as opções, pode-se usar um Spinner
- Responde a eventos assim como os demais elementos já vistos

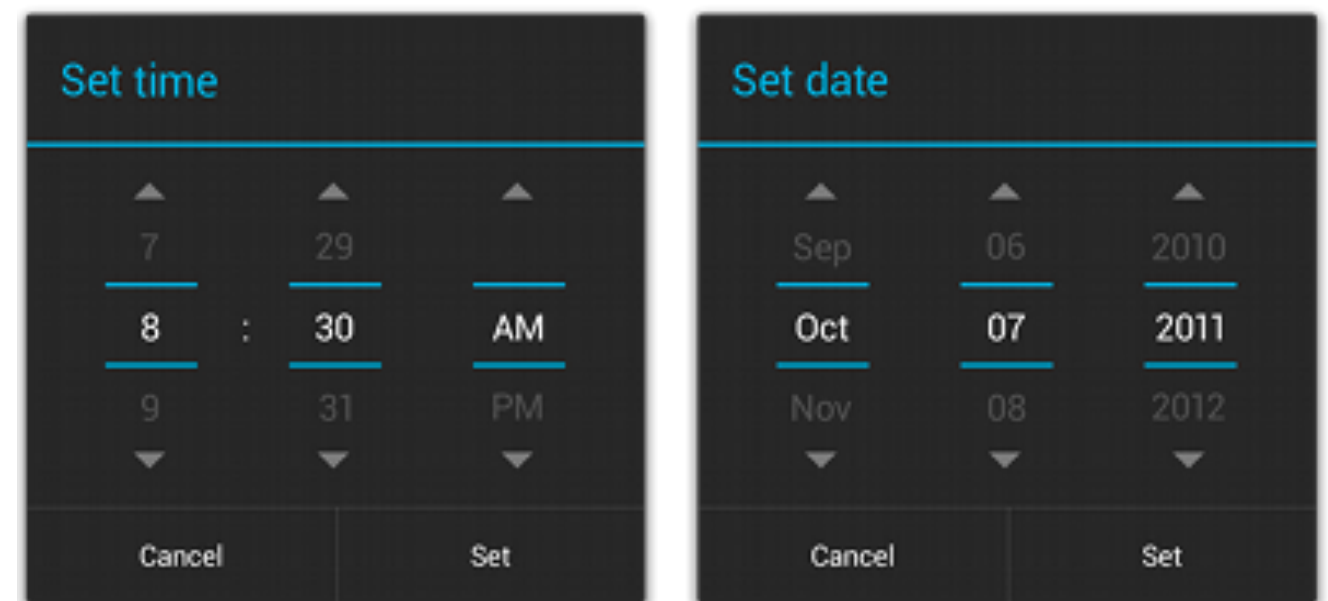


ATTENDING?

☒ Yes ☐ Maybe ☐ No

Pickers

- Permitem que usuário escolha um horário ou data específica
- Permitem seleção independente
- Ajuda a garantir que será utilizada uma data ou hora válida e formatada corretamente



Outros

- Ratingbar
 - Ideia de associar estrelas para avaliação
- WebView
 - Container para exibir páginas web
- MapView
 - mostra um mapa do Google Maps ‘embutido’
 - Definir localizações, posição, nível de zoom, etc.

Gerenciando Eventos

- Existem várias maneiras de interceptar eventos gerados pela interação com o usuário
- Dentro das várias classes View que usamos para montar o layout, existem diversos métodos de callback
- Podemos utilizar estas interfaces de ‘event listeners’ para capturar interação com o usuário
- Em alguns casos vamos querer estender a classe View, para adicionar algum comportamento mais rebuscado para o seu botão

Gerenciando Eventos

- Diversas interfaces de listeners definidas pela classe View
 - **`View.OnClickListener.onClick()`**
 - **`View.OnLongClickListener.onLongClick()`**
 - **`View.OnFocusChangeListener.onFocusChange()`**
 - **`View.OnKeyListener.onKey()`**
 - **`View.OnTouchListener.onTouch()`**

Mais detalhes em <http://developer.android.com/guide/topics/ui/ui-events.html>

Quick Demo