UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

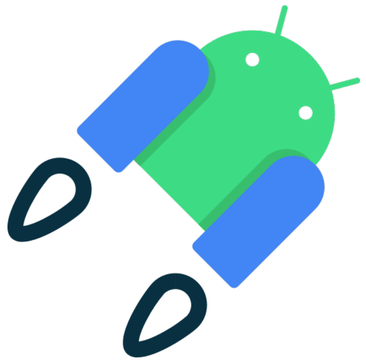
Layouts

QXD0102 - Desenvolvimento para Dispositivos Móveis

Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Conteúdo

- Views e ViewGroups
- ConstraintLayout
- Estilos, Temas, Material Design
- Recursos alternativos
- RecyclerView



Views e ViewGroups

View e ViewGroups

Revisão

- Existem duas maneiras de criar interfaces
  - Declarativa
    - Arquivos XML são usado para declarar as interfaces
  - Programática
    - São construídas através de código Kotlin/Java

View e ViewGroups

O Futuro

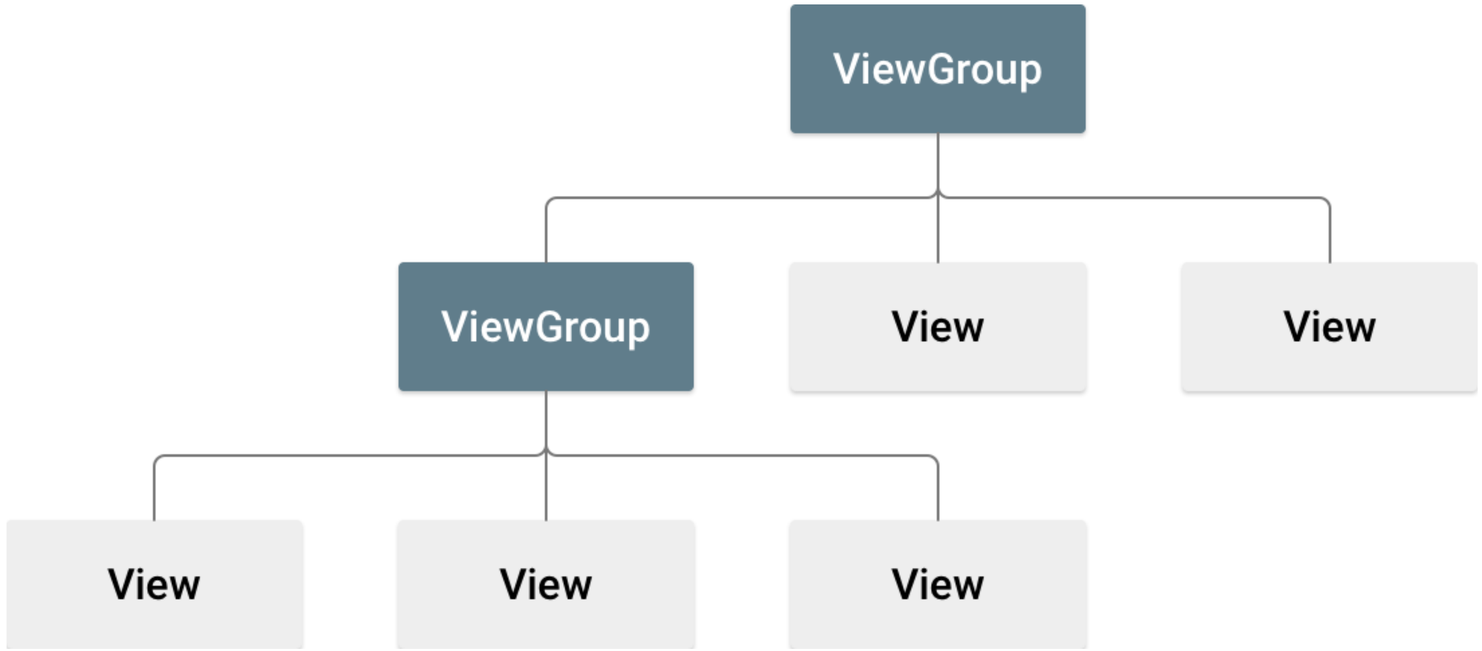
```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```

Recentemente anunciado a primeira versão estável para Julho.

View e ViewGroups

Revisão

- Um layout define a estrutura da interface de uma app
  - É definido por meio de uma hierarquia de View e ViewGroup



View e ViewGroups

Revisão

View vs ViewGroup

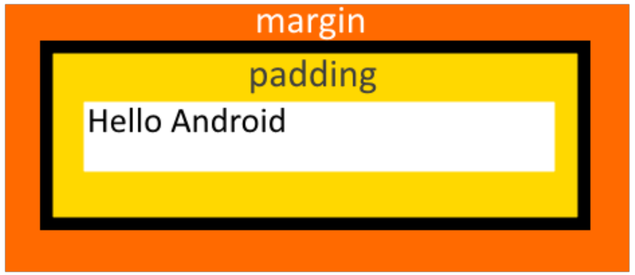
- Views geralmente mostram componentes com os quais o usuário pode interagir
  - Também chamada de “widget”. Ex: Button e TextView
- ViewGroups são contêineres invisíveis que definem a estrutura das views neles contidos
  - Também chamados de “layouts”

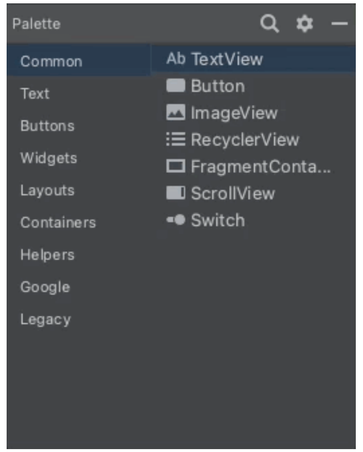
View e ViewGroups

Revisão

View

- É responsável por uma área retangular na tela
- É possível definir propriedades, listeners, visibilidade e foco



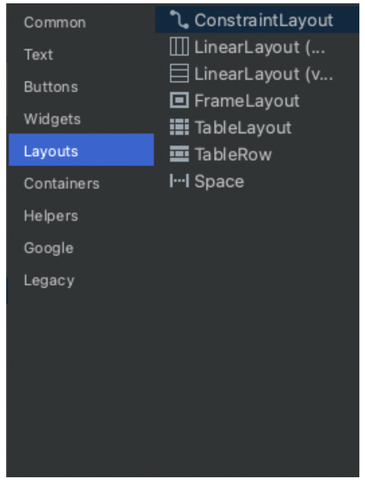


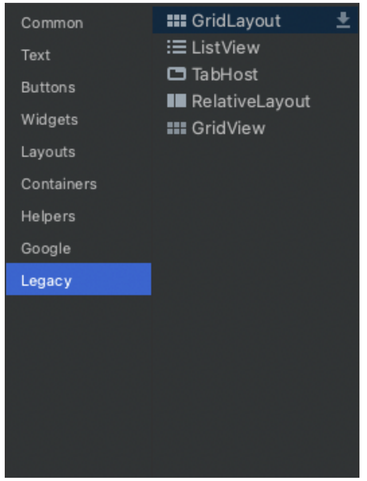
View e ViewGroups

Revisão

ViewGroup

- É uma subclasse da classe View
- Define como as views dentro do ViewGroup serão organizadas na tela





Views e ViewGroups

Atributos

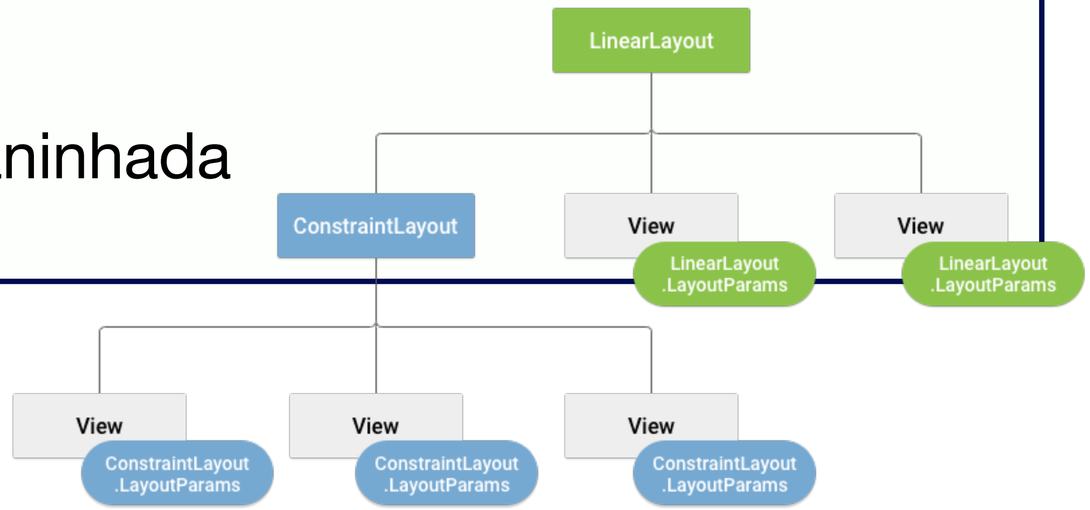
- Cada View e ViewGroup possuem seus atributos XML
- Alguns atributos são específicos para um objeto View
- Alguns atributos são comuns a todos objetos do tipo View já que eles herdam de View

Views e ViewGroups

Atributos

Parâmetros de Layout

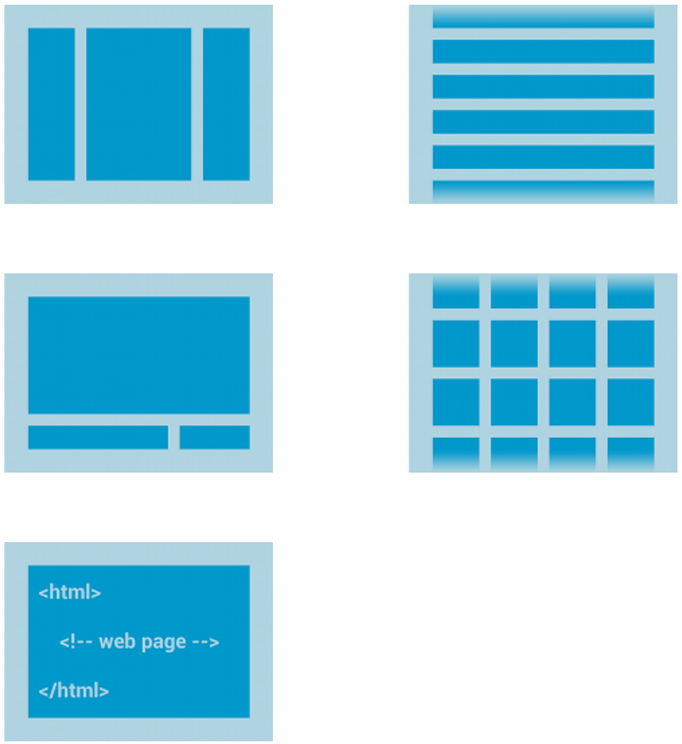
- São os atributos começado com layout_
- Definem os parâmetros de layout da View de acordo com a ViewGroup que ela pertence
- ViewGroup.LayoutParams é uma classe aninhada em cada classe de Layout



Views e ViewGroups

- LinearLayout
- Relative Layout
- WebView
- ~~ListView~~
- ~~GridView~~

RecyclerView



ConstraintLayout

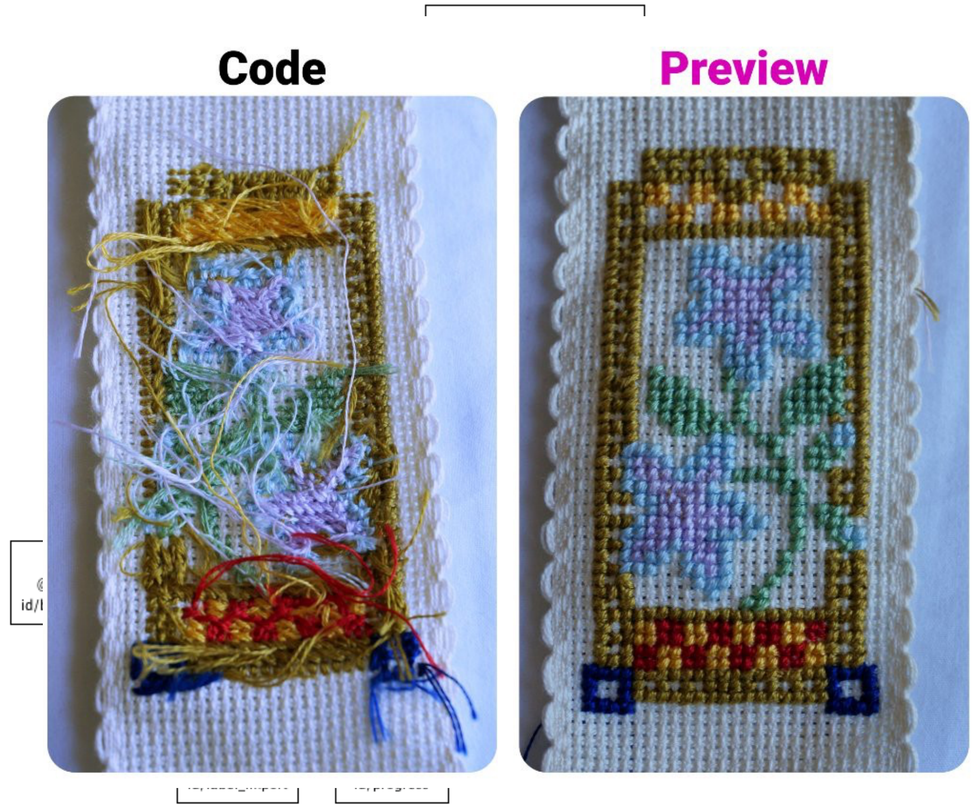
ConstraintLayout

- Adicionado em 2017 é compatível com API Level 9 (GingerBread)
- Atualmente faz parte do Android Jetpack
- Permite a criação de layouts complexos porém mantendo sua hierarquia plana (*flat*)
  - Um dos layouts mais caros
  - Nunca usá-lo aninhado

ConstraintLayout

ConstraintLayout

- Interfaces complexas com poucos elementos aninhados
- Aumento da legibilidade e menos *boilerplate code*



ConstraintLayout

- Cada view deve ter pelo menos:
  - Uma limitação (*constraint*) vertical
  - Uma limitação (*constraint*) horizontal
- Cada limitação representa uma conexão or alinhamento com outra view, layout pai ou com um guia invisível

ConstraintLayout

Configuração

- É necessário adicionar o seguinte repositório no build.gradle

```
repositories {
        google()
    }
```

- Adicionar a seguinte dependência no arquivo build.gradle do módulo em que o layout será utilizado

```
dependencies {
    implementation("androidx.constraintlayout:constraintlayout:2.0.4")
}
```

ConstraintLayout

- Cada view deve ter pelo menos:
  - Uma limitação (*constraint*) vertical
  - Uma limitação (*constraint*) horizontal
- Cada limitação representa uma conexão or alinhamento com outra view, layout pai ou com um guia invisível

ConstraintLayout

Tipos de restrições

- Dimensões
- Posicionamento relativo
- Margens
- Posicionamento de centralização
- Posicionamento circular
- Comportamento relacionado a visibilidade
- Chains
- Helpers virtuais
- Otimizações

Confiram todos os detalhes em:
https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout

ConstraintLayout

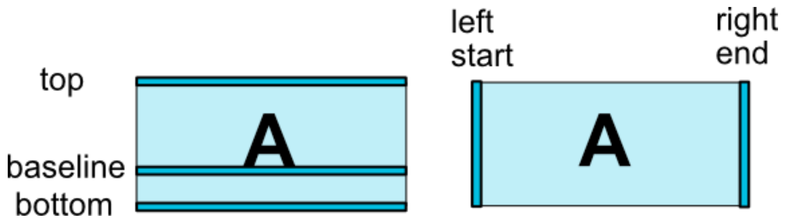
Posicionamento relativo

- Existem 3 possíveis maneiras de definir as dimensões, usando android:layout_width e android:layout_height
  - Usando uma dimensão específica
  - Using WRAP_CONTENT
  - Using 0dp ou MATCH_CONSTRAINT
- Usando MACH_CONSTRAINT é possível configurar a largura e altura, mínimas e máximas, usando o valor em dimensão ou em porcentagem
  - layout_constraintWidth_min, layout_constraintHeight_min, layout_constraintWidth_max e layout_constraintHeight_max
  - app:layout_constraintWidth_default, app:layout_constraintHeight_default, layout_constraintWidth_percent e layout_constraintHeight_percent

ConstraintLayout

Posicionamento relativo

- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

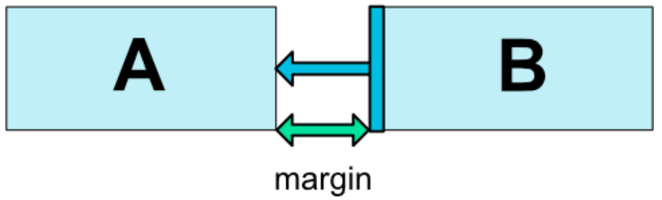



```
<Button android:id="@+id/buttonB" ...
                app:layout_constraintLeft_toLeftOf="parent" />
```

ConstraintLayout

Margens

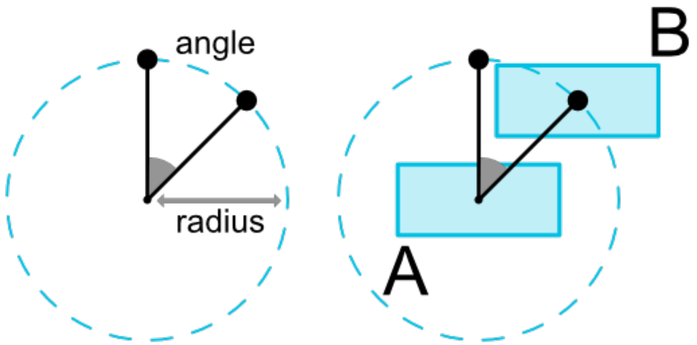
- android:layout_marginStart
- android:layout_marginEnd
- android:layout_marginLeft
- android:layout_marginTop
- android:layout_marginRight
- android:layout_marginBottom

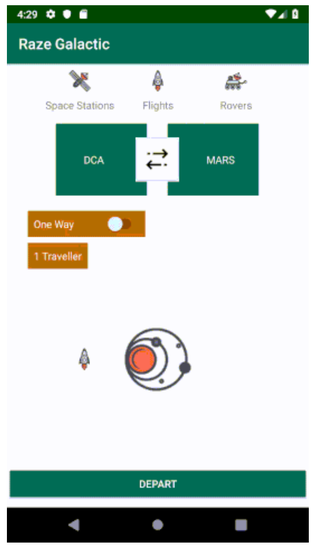


ConstraintLayout

Posicionamento circular

- layout_constraintCircle : referência para a outra view
- layout_constraintCircleRadius : a distância para a outra view
- layout_constraintCircleAngle : o ângulo de alinhamento entre as views, em graus [0, 360]

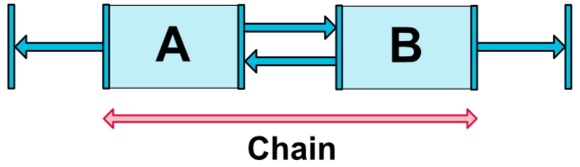


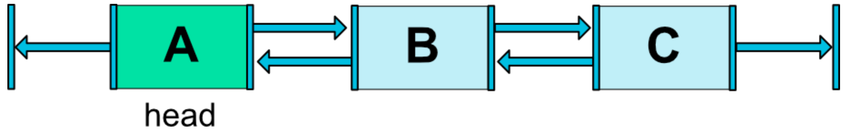


ConstraintLayout

Chains

- Fornecem comportamento semelhante ao de um grupo em um único eixo (horizontal ou verticalmente)
- Conjunto de widgets ligados entre si por meio de uma conexão bidirecional
- São controladas por atributos definidos no primeiro elemento da cadeia (a "cabeça" da cadeia)

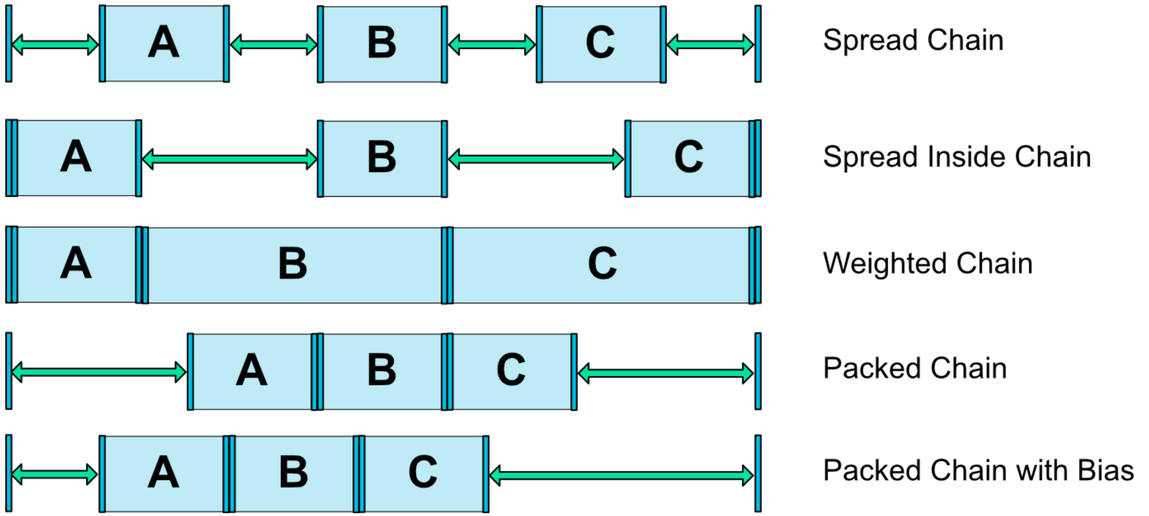




ConstraintLayout

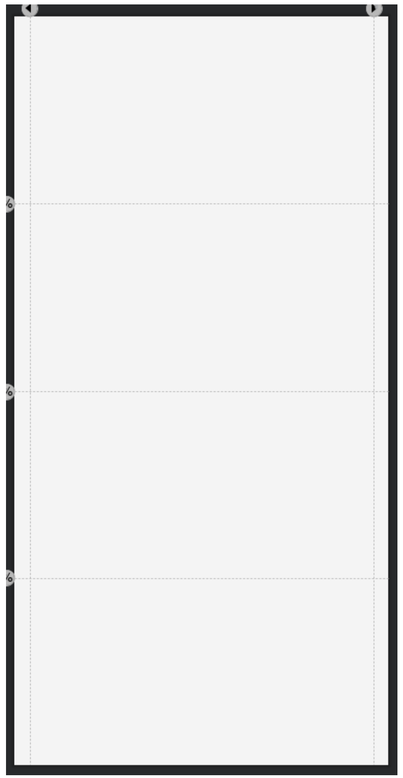
Estilo da Chains

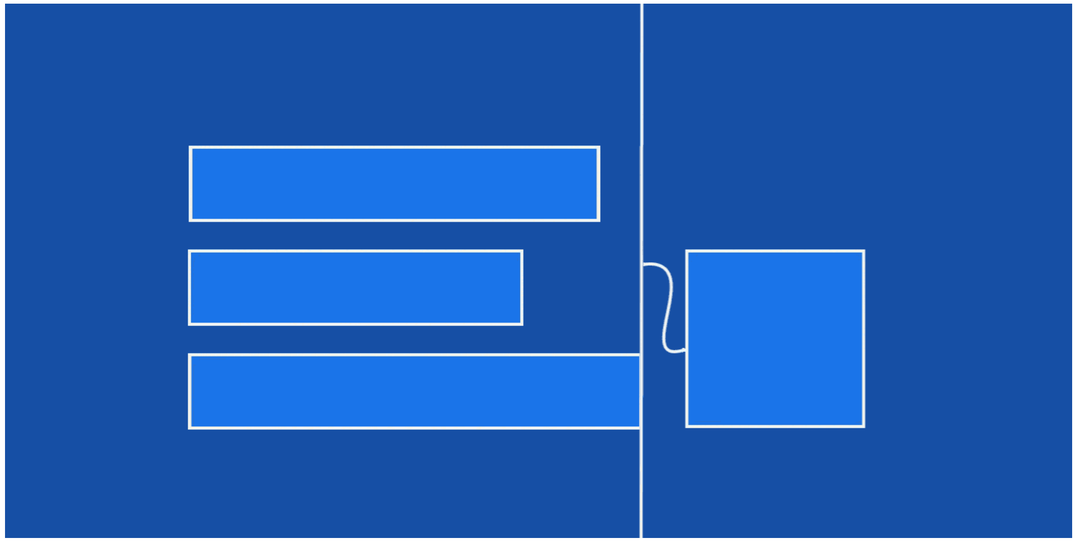
- Ao definir o atributo layout_constraintHorizontal_chainStyle ou layout_constraintVertical_chainStyle no primeiro elemento de uma cadeia, o comportamento da cadeia mudará de acordo com o estilo especificado
  - CHAIN_SPREAD - os elementos serão espalhados (estilo padrão)
    - Cadeia ponderada - no modo CHAIN_SPREAD, se alguns widgets forem configurados para MATCH_CONSTRAINT, eles dividirão o espaço disponível
  - CHAIN_SPREAD_INSIDE - semelhante, mas os pontos finais da cadeia não serão espalhados
  - CHAIN_PACKED - os elementos da cadeia serão empacotados juntos.
    - O atributo de bias horizontal ou vertical do filho afetará o posicionamento dos elementos compactados



ConstraintLayout

Guidelines e Barreiras





Mais informações: https://medium.com/swlh/constraint-layout-8b13ace936df

Estilos, Temas, Material Design

Estilos, Temas, Material Design

- Compartilham a mesma estrutura, chave -> valor
  - Mesma sintaxe <style>

```
<style name="Widget.Plaid.Button.InlineAction" parent="…">
  <item name="android:gravity">center_horizontal</item>
  <item name="android:textAppearance">@style/TextAppearance.CommentAuthor</item>
  <item name="android:drawablePadding">@dimen/spacing_micro</item>
</style>
```

```
<style name="Theme.Plaid" parent="…">
  <item name="colorPrimary">@color/teal_500</item>
  <item name="colorSecondary">@color/pink_200</item>
  <item name="android:windowBackground">@color/white</item>
</style>
```

Estilos, Temas, Material Design

Estilos

- Especificam atributos de uma view específica
  - Ex: Botões
- Todos esses atributos poderiam ser postos no arquivo de layout

Temas

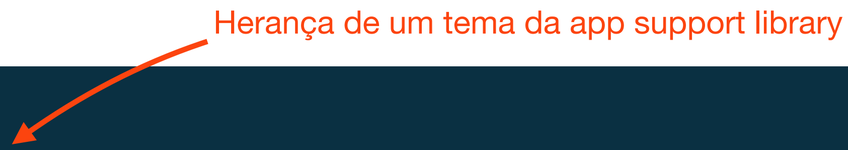
- Coleção de atributos aplicados a um app, activity ou hierarquia de views
- Atribuem nomes semânticos, ex *colorPrimary*

Foram feitos para trabalhar juntos

Estilos, Temas, Material Design

Estilos

- Exemplo de estilo




```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="GreenText" parent="TextAppearance.AppCompat">
        <item name="android:textColor">#00FF00</item>
    </style>
</resources>
```

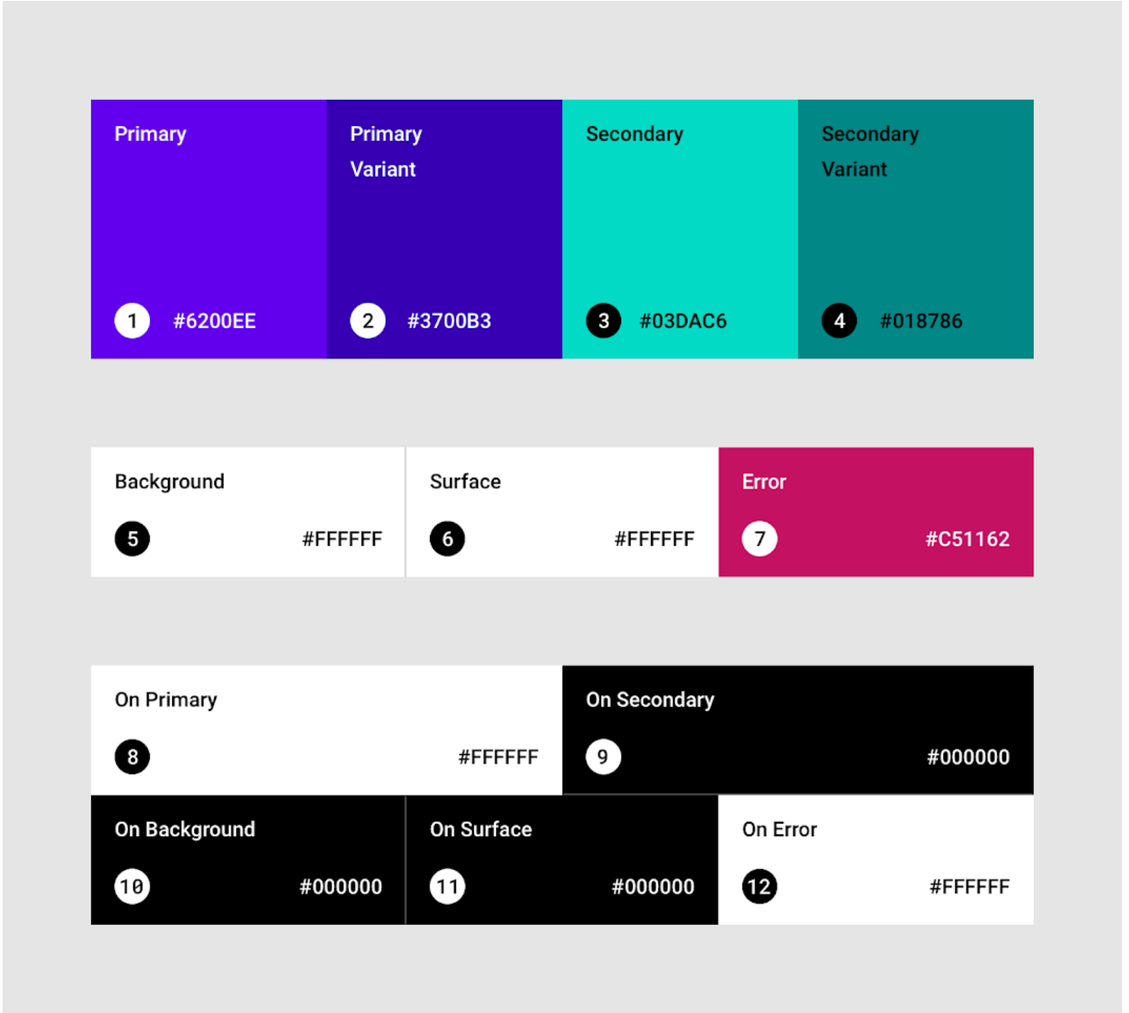
- Como aplicá-lo

```
<TextView
    style="@style/GreenText"
    ... />
```

Estilos, Temas, Material Design

Temas

#	Name	Theme Attribute
1	Primary	colorPrimary
2	Primary Variant	colorPrimaryVariant
3	Secondary	colorSecondary
4	Secondary Variant	colorSecondaryVariant
5	Background	colorBackground
6	Surface	colorSurface
7	Error	colorError
8	On Primary	colorOnPrimary
9	On Secondary	colorOnSecondary
10	On Background	colorOnBackground
11	On Surface	colorOnSurface
12	On Error	colorOnError

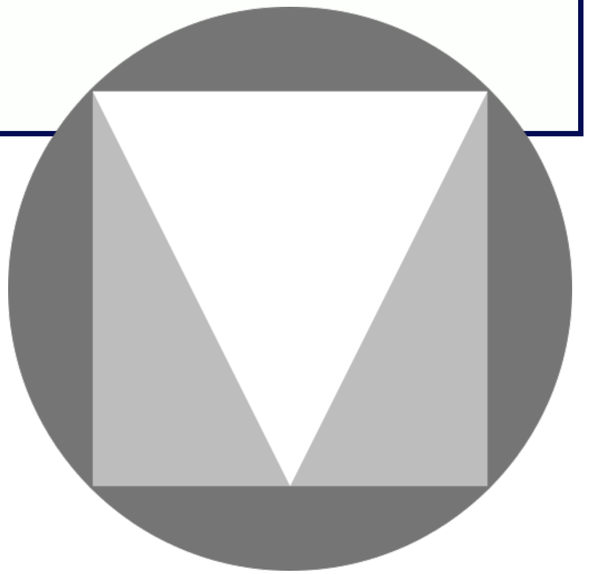


Estilos, Temas, Material Design

Material Design

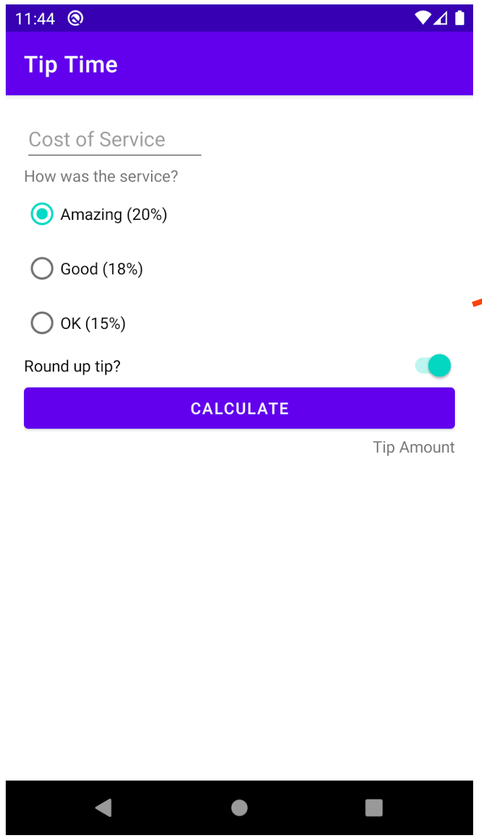
Material Componentes

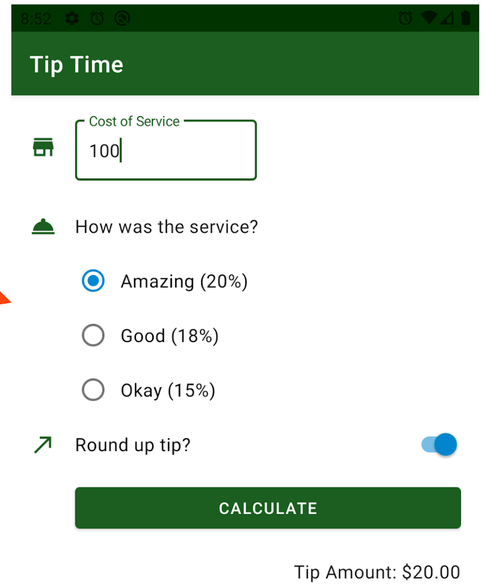
- São widgets facilitam a implementação do Material Design em um app
- Contém a documentação ensinado como utilizá-los e customizá-los
- Confere uma experiência mais consistente com outros aplicativos no dispositivo do usuário



Estilos, Temas, Material Design

Tip Calculator





Codelab: Create XML layouts for Android

Codelab: Create a more polished user experience

Estilos, Temas, Material Design

Mais informações

- Android styling: themes vs styles
- Android styling: common theme attributes
- Android Styling: prefer theme attributes
- Material Design: The color system
- Material Design: Color Tool

Recursos alternativos

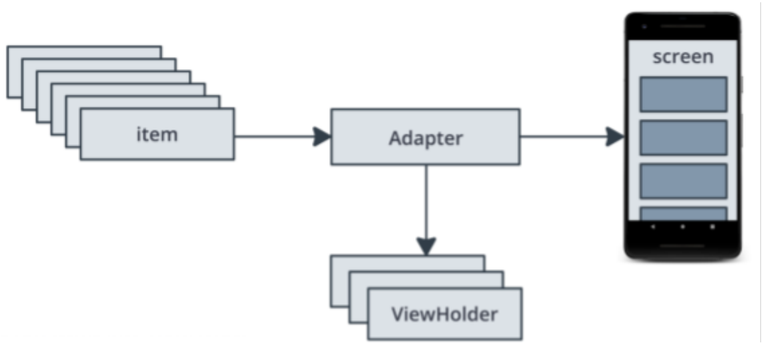
Recursos alternativos

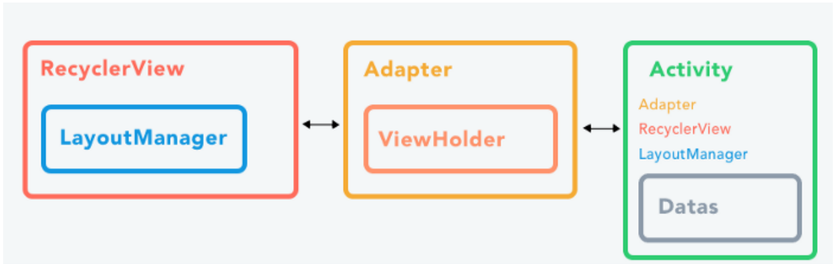
- Lista de possíveis qualificadores de recursos alternativos

RecycleView

RecycleView

- RecyclerView é o ViewGroup que contém as views correspondentes aos seus dados
- Cada elemento da lista é associado a um *view holder*
  - É criado sem nenhum dado associado a ele
  - Uma RecyclerView vincula um elemento ao um view holder
  - É definido ao extender a classe RecyclerView.ViewHolder
- Requisita as views e as vincula com os dados ao chamar o métodos de um adapter
  - Um adapter é definido ao extender a classe RecyclerView.Adapter
- O *layout manager* organizaram os elements individuais na lista





RecycleView

- Mais informações

Por hoje é só