Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

Intents

Intents

Um Intent é um objeto que fornece vínculos de runtime componentes separados, como duas Activities.

O Intent representa uma “intenção de fazer algo” do aplicativo.

Você pode usar intents para uma ampla variedade de tarefas, como por exemplo iniciar outra atividade.

MainActivity.kt

```kotlin
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        //-----------------------------------------------------------------------
        val txtTexto = this.findViewById<EditText>(R.id.txtTexto)
        //-----------------------------------------------------------------------
        val btnActivity = this.findViewById<Button>(R.id.btnActivity)
        btnActivity.setOnClickListener { it: View!

            val intent = Intent( packageContext: this, MainActivity2::class.java)
            intent.putExtra( name: "frase", txtTexto.text.toString())
            startActivity(intent)
        }
```

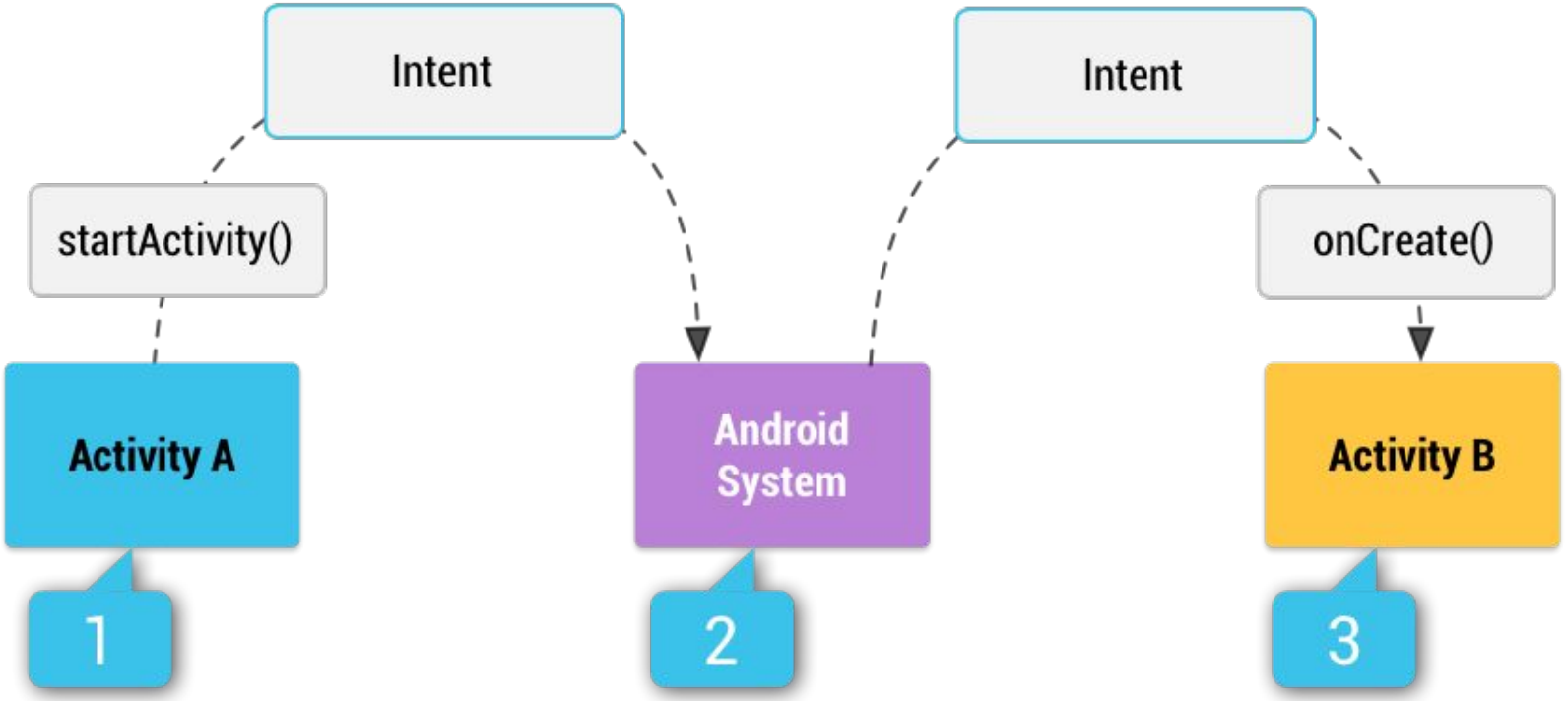
1

MainActivity.kt | MainActivity2.kt

```kotlin
class MainActivity2 : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main2)

        val frase = intent.getStringExtra( name: "frase")
        val lblFrase = this.findViewById<TextView>(R.id.lblFrase)
        lblFrase.text = frase
    }
}
```

2

1. Activity (A) cria um Intent com uma descrição de ação e a passa para startActivity().



2. O sistema Android busca, em todos os aplicativos, um filtro de intents que corresponda ao intent.
3. O sistema inicia a Activity correspondente (B) chamando seu método onCreate() e passando-lhe o Intent.

Tipos de Intents

Os intents explícitos especificam qual aplicativo atenderá ao intent, fornecendo o nome do pacote do aplicativo de destino ou o nome da classe de um componente totalmente qualificado.

Normalmente, usa-se um intent explícito para iniciar um componente no próprio aplicativo porque se sabe o nome de classe da atividade ou do serviço que se quer iniciar.

Por exemplo, iniciar uma nova Activity em resposta a uma ação do usuário ou iniciar um serviço para fazer o download de um arquivo em segundo plano.

Tipos de Intents

Os intents implícitos não nomeiam nenhum componente específico, mas declaram uma ação geral a realizar, o que permite que um componente de outro aplicativo a processe.

Por exemplo, se você quiser exibir ao usuário uma localização em um mapa, pode usar um intent implícito para solicitar que outro aplicativo capaz exiba uma localização especificada no mapa.

Intent Implícito

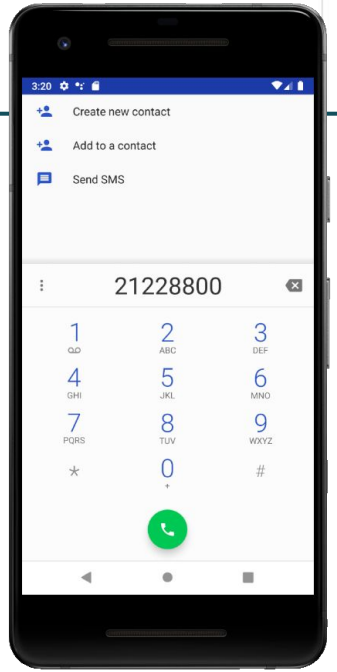
Ao criar um intent implícito, o sistema Android encontra o componente adequado para iniciar, comparando o conteúdo do intent aos filtros de intents declarados no arquivo de manifesto de outros aplicativos no dispositivo.

Se o intent corresponder a um filtro de intents, o sistema iniciará esse componente e entregará o objeto Intent.

Se diversos filtros de intents corresponderem, o sistema exibirá uma caixa de diálogo para que o usuário selecione o aplicativo que deseja usar.

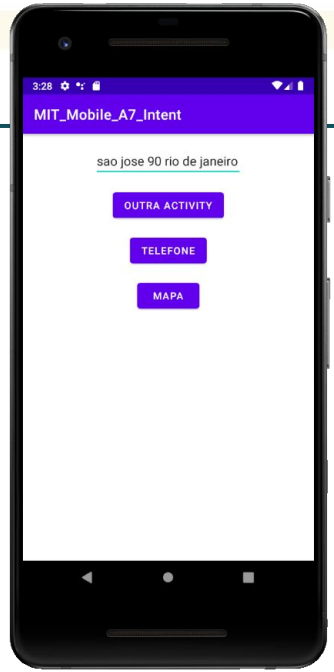
MainActivity.kt

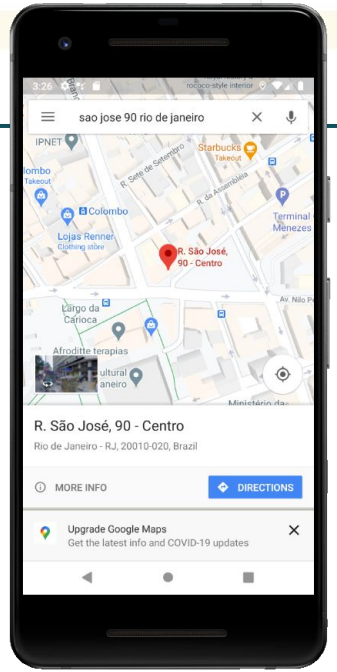
```
27      val btnTelefone = this.findViewById<Button>(R.id.btnTelefone)
28      btnTelefone.setOnClickListener {  it: View!
29
30          val callIntent: Intent = Uri.parse( uriString: "tel:${txtTexto.text.toString()}").let { number ->
31
32              Intent(Intent.ACTION_DIAL, number)
33          }
34          startActivity(callIntent)
35      }
```



MainActivity.kt

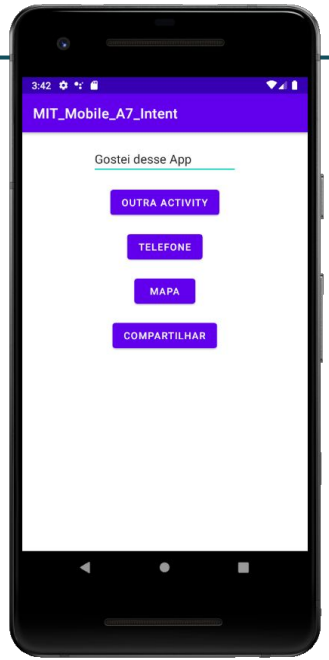
```
37          val btnMapa = this.findViewById<Button>(R.id.btnMapa)
38          btnMapa.setOnClickListener {  it: View!
39
40              // Map point based on address
41              val mapIntent: Intent = Uri.parse( uriString: "geo:0,0?q=${txtTexto.text.toString()}").let { location ->
42
43                  Intent(Intent.ACTION_VIEW, location)
44              }
45              startActivity(mapIntent)
46          }
47      }
```

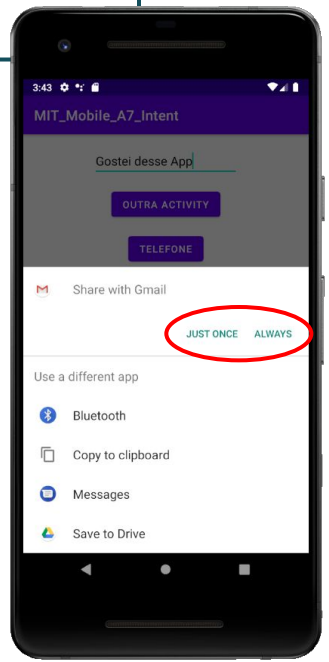




MainActivity.kt

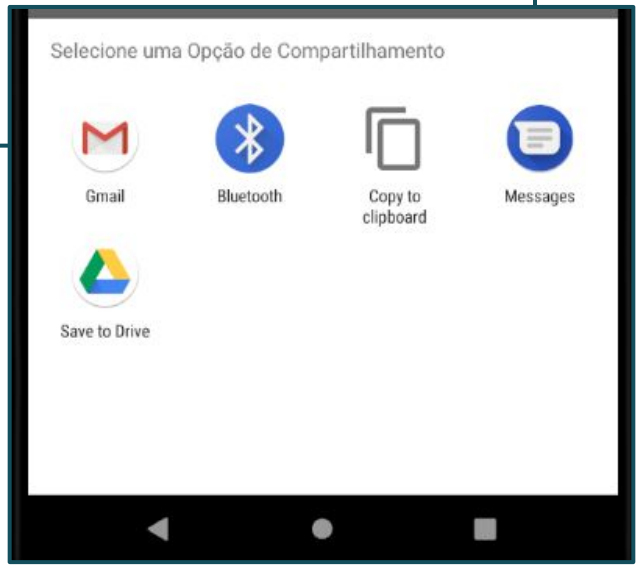
```
47      val btnCompartilhar = this.findViewById<Button>(R.id.btnCompartilhar)
48      btnCompartilhar.setOnClickListener {  it: View!
49
50          val shareIntent = Intent(Intent.ACTION_SEND)
51          shareIntent.setType("text/plain")
52          shareIntent.putExtra(Intent.EXTRA_TEXT, txtTexto.text.toString())
53          startActivity(shareIntent)
54      }
```

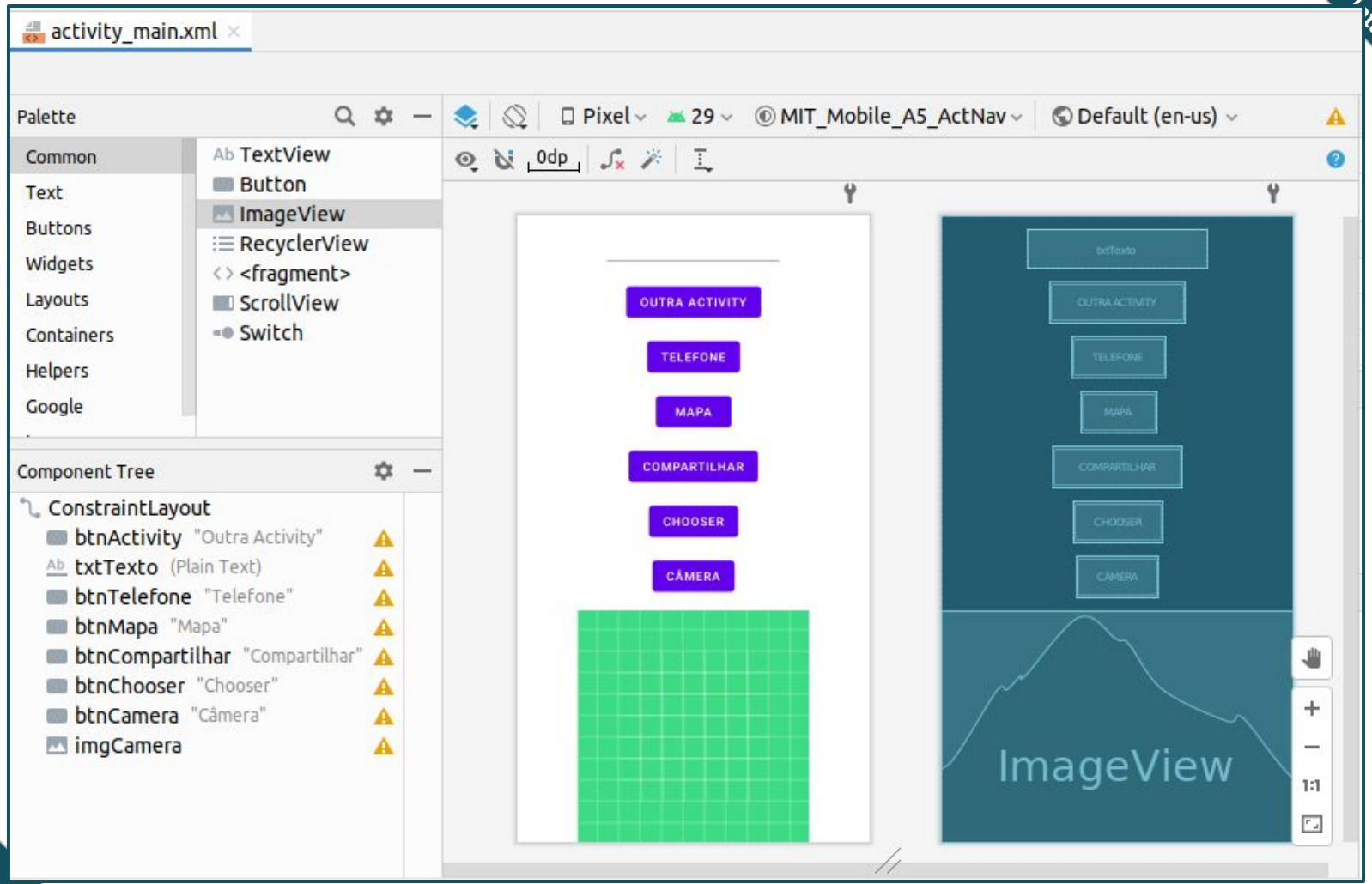




MainActivity.kt

```
56      val btnChooser = this.findViewById<Button>(R.id.btnChooser)
57      btnChooser.setOnClickListener {  it: View!
58
59          val shareIntent = Intent(Intent.ACTION_SEND)
60          shareIntent.setType("text/plain")
61          shareIntent.putExtra(Intent.EXTRA_TEXT, txtTexto.text.toString())
62          val title = "Selecione uma Opção de Compartilhamento"
63          val chooser = Intent.createChooser(shareIntent, title)
64          if (shareIntent.resolveActivity(packageManager) != null) {
65
66              startActivity(chooser)
67          }
68      }
```




activity_main.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
btnActivity "Outra Activity"
txtTexto (Plain Text)
btnTelefone "Telefone"
btnMapa "Mapa"
btnCompartilhar "Compartilhar"
btnChooser "Chooser"
btnCamera "Câmera"
imgCamera
Pixel
29
MIT_Mobile_A5_ActNav
Default (en-us)
0dp
OUTRA ACTIVITY
TELEFONE
MAPA
COMPARTILHAR
CHOOSER
CÂMERA
txtTexto
ImageView
1:1

MainActivity.kt

```
17  class MainActivity : AppCompatActivity() {
18
19      val CAMERA_PERMISSION_CODE = 100
20      val CAMERA_REQUEST = 1888
21
22      override fun onCreate(savedInstanceState: Bundle?) {
23
24          super.onCreate(savedInstanceState)
25          setContentView(R.layout.activity_main)
```

1

MainActivity.kt

```
80          val btnCamera = this.findViewById<Button>(R.id.btnCamera)
81          btnCamera.setOnClickListener {  it: View!
82
83              if (checkSelfPermission(Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
84
85                  requestPermissions(arrayOf(Manifest.permission.CAMERA), CAMERA_PERMISSION_CODE)
86              } else {
87
88                  val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
89                  startActivityForResult(cameraIntent, CAMERA_REQUEST)
90              }
91          }
```

2

MainActivity.kt

```
94  override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String?>,
95                                          grantResults: IntArray) {
96
97      super.onRequestPermissionsResult(requestCode, permissions, grantResults)
98      if (requestCode == CAMERA_PERMISSION_CODE) {
99
100         if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
101
102             Toast.makeText( context: this,  text: "Acesso à Câmera Concedido", Toast.LENGTH_LONG).show()
103             val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
104             startActivityForResult(cameraIntent, CAMERA_REQUEST)
105         } else {
106
107             Toast.makeText( context: this,  text: "Acesso à Câmera Negado", Toast.LENGTH_LONG).show()
108         }
109     }
110 }
```

1
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String?>,
grantResults: IntArray) {

2
if (requestCode == CAMERA_PERMISSION_CODE) {

3
if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
Toast.makeText(context: this, text: "Acesso à Câmera Concedido", Toast.LENGTH_LONG).show()
val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
startActivityForResult(cameraIntent, CAMERA_REQUEST)
} else {
Toast.makeText(context: this, text: "Acesso à Câmera Negado", Toast.LENGTH_LONG).show()
}

MainActivity.kt

```
112  override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
113
114      super.onActivityResult(requestCode, resultCode, data)
115      if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
116
117          val foto = data?.extras!!["data"] as Bitmap?
118          val imgCamera = this.findViewById<ImageView>(R.id.imgCamera)
119          imgCamera.setImageBitmap(foto)
120      }
121  }
```

1

2

Android Storage

Android Storage

A seguir estão as opções de armazenamento de dados:

SharedPreferences: esta técnica armazena dados primitivos específicos do aplicativo em pares chave-valor.

Armazenamento Interno: armazenar dados privados na memória do dispositivo usando a técnica de I/O de arquivo. Esses dados não poderão ser acessados por outros aplicativos).

Armazenamento Externo: armazenar dados públicos no armazenamento externo compartilhado (como um cartão SD).

Armazenamento Interno X Externo

Interno	Externo
Está sempre disponível.	Nem sempre está disponível, porque o usuário pode, em alguns casos, montar um armazenamento externo, como um armazenamento USB, e depois removê-lo do dispositivo.
Acessível somente pelo próprio aplicativo.	É legível por todos, portanto, os arquivos salvos aqui podem ser lidos fora do seu controle.
Quando o usuário desinstala seu aplicativo, o sistema remove todos os arquivos do armazenamento interno.	Quando o usuário desinstala seu aplicativo, o sistema remove os arquivos do seu aplicativo daqui, a menos que você os salve no diretório getExternalFilesDir ().
É a sua melhor opção quando você deseja ter certeza de que nem o usuário nem outros aplicativos podem acessar seus arquivos.	É o melhor lugar para salvar arquivos que não requeiram restrições de acesso e arquivos que você deseja compartilhar com outros aplicativos ou permitir que os usuários acessem de um computador.

Shared Preferences

A classe SharedPreferences fornece uma estrutura geral que permite salvar e recuperar pares de valores-chave persistentes de tipos de dados primitivos.

Você pode usar SharedPreferences para salvar quaisquer dados primitivos: booleans, floats, Ints, longs e strings.

Esses dados persistirão nas sessões do usuário (mesmo se seu aplicativo for encerrado).

Armazenamento Interno

Você pode salvar arquivos diretamente no armazenamento interno do dispositivo.

Por padrão, os arquivos salvos no armazenamento interno são privados para seu aplicativo (modo privado) e não podem ser acessados por outros aplicativos.

Quando o usuário desinstala seu aplicativo, esses arquivos são removidos.

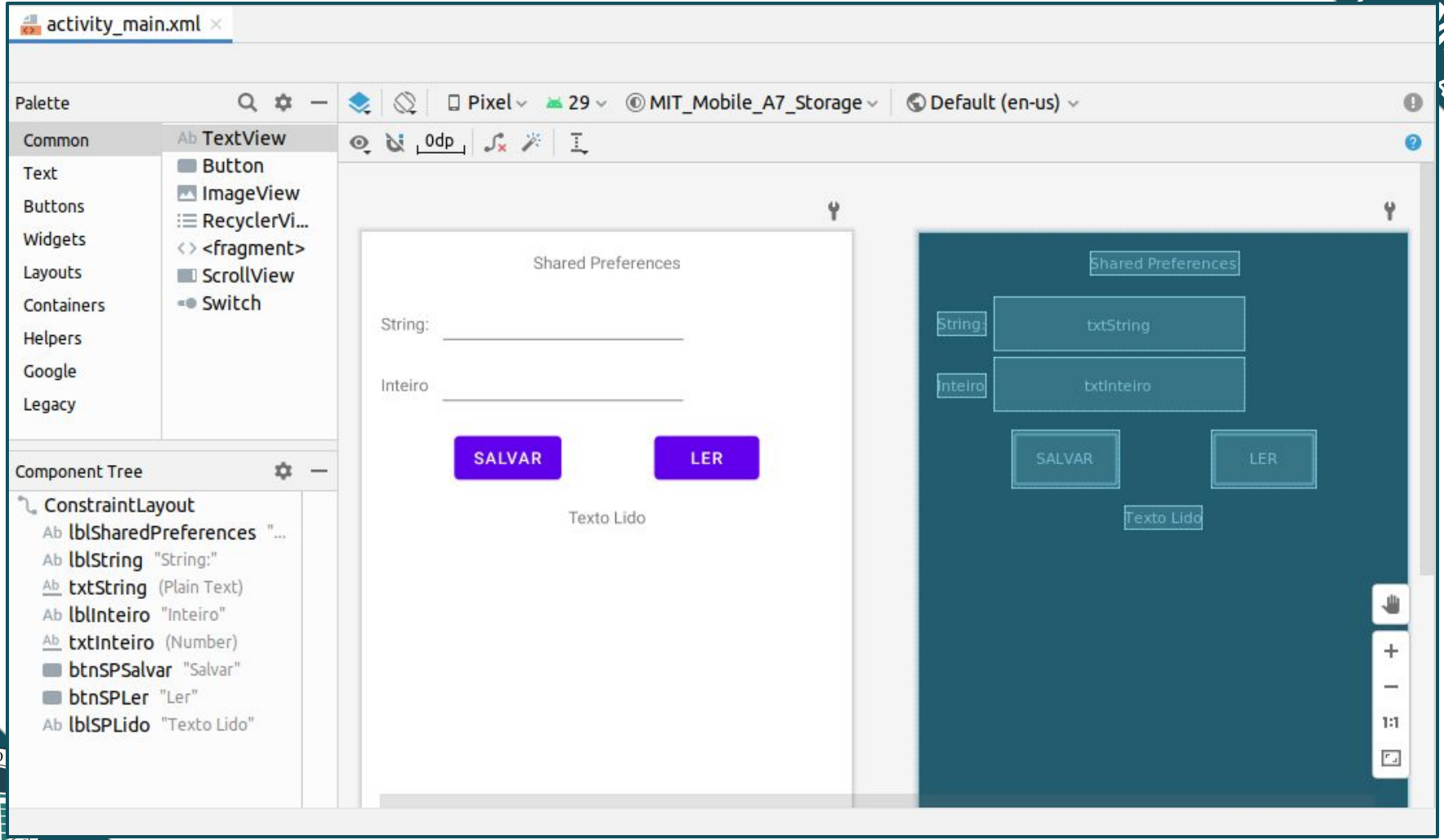
Para usar o armazenamento interno para gravar alguns dados no arquivo, chame o método openFileOutput() com o nome do arquivo e o modo.

Armazenamento Externo

Cada dispositivo compatível com Android suporta um "armazenamento externo" compartilhado que você pode usar para salvar arquivos.

Pode ser uma mídia de armazenamento removível (como um cartão SD) ou um armazenamento interno (não removível).

Os arquivos salvos no armazenamento externo podem ser lidos por todos e podem ser modificados pelo usuário ao habilitar o armazenamento em massa USB para transferir arquivos para um computador.


activity_main.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
TextView
Button
ImageView
RecyclerVi...
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
lblSharedPreferences "...
lblString "String:"
txtString (Plain Text)
lblInteiro "Inteiro"
txtInteiro (Number)
btnSPSalvar "Salvar"
btnSPLer "Ler"
lblSPLido "Texto Lido"
Pixel
29
MIT_Mobile_A7_Storage
Default (en-us)
0dp
Shared Preferences
String:
Inteiro
SALVAR
LER
Texto Lido
Shared Preferences
String:
txtString
Inteiro
txtInteiro
SALVAR
LER
Texto Lido
1:1

MainActivity.kt

```
17      val btnSPSalvar = this.findViewById<Button>(R.id.btnSPSalvar)
18  1   btnSPSalvar.setOnClickListener {  it: View!
19
20          val sp = this.getSharedPreferences( name: "mainPrefs", MODE_PRIVATE)
21          val txtString = this.findViewById<EditText>(R.id.txtString)
22          val txtInteiro = this.findViewById<EditText>(R.id.txtInteiro)
23          val editor = sp.edit()
24          editor.putString("dado1", txtString.text.toString())
25          editor.putInt("dado2", txtInteiro.text.toString().toInt())
26          editor.commit()
27          Toast.makeText( context: this,  text: "Salvo", Toast.LENGTH_SHORT).show()
28      }
29      val btnSPLer = this.findViewById<Button>(R.id.btnSPLer)
30  2   btnSPLer.setOnClickListener {  it: View!
31
32          val sp = this.getSharedPreferences( name: "mainPrefs", MODE_PRIVATE)
33          val lblSPLido = this.findViewById<TextView>(R.id.lblSPLido)
34          lblSPLido.setText("Dado 1 = ${sp.getString("dado1", "")} --- Dado 2 = ${sp.getInt("dado2", 0)}")
35          Toast.makeText( context: this,  text: "Lido", Toast.LENGTH_SHORT).show()
36      }
```

Shared Preferences

MainActivity.kt

```
42      val btnAISalvar = this.findViewById<Button>(R.id.btnAISalvar)
43      btnAISalvar.setOnClickListener {  it: View!
44
45          val txtAIString = this.findViewById<EditText>(R.id.txtAIString)
46          val fos : FileOutputStream = this.openFileOutput( name: "mainFile", MODE_PRIVATE)
47          fos.write(txtAIString.text.toString().toByteArray())
48          fos.close()
49          Toast.makeText( context: this,  text: "Salvo", Toast.LENGTH_SHORT).show()
50      }
51      val btnAILer = this.findViewById<Button>(R.id.btnAILer)
52      btnAILer.setOnClickListener {  it: View!
53
54          val fis : FileInputStream = this.openFileInput( name: "mainFile")
55          val bytes = fis.readBytes()
56          fis.close()
57          val lblAILido = this.findViewById<TextView>(R.id.lblAILido)
58          lblAILido.setText(String(bytes))
59          Toast.makeText( context: this,  text: "Lido", Toast.LENGTH_SHORT).show()
60      }
```

1

2

Arquivo Interno

MainActivity.kt

```
62      val btnAESalvar = this.findViewById<Button>(R.id.btnAESalvar)
63      btnAESalvar.setOnClickListener {  it: View!
64  1
65          val file = File(this.getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS),  child: "mainExt")
66          val fos = FileOutputStream(file)
67          val txtAEString = this.findViewById<EditText>(R.id.txtAEString)
68          fos.write(txtAEString.text.toString().toByteArray())
69          fos.close()
70          Toast.makeText( context: this,  text: "Salvo", Toast.LENGTH_SHORT).show()
71      }
72      val btnAELer = this.findViewById<Button>(R.id.btnAELer)
73      btnAELer.setOnClickListener {  it: View!
74
75  2       val file = File(this.getExternalFilesDir(Environment.DIRECTORY_DOCUMENTS),  child: "mainExt")
76          val fis = FileInputStream(file)
77          val bytes = fis.readBytes()
78          fis.close()
79          val lblAELido = this.findViewById<TextView>(R.id.lblAELido)
80          lblAELido.setText(String(bytes))
81          Toast.makeText( context: this,  text: "Lido", Toast.LENGTH_SHORT).show()
82      }
```

Arquivo Externo

Acessibilidade

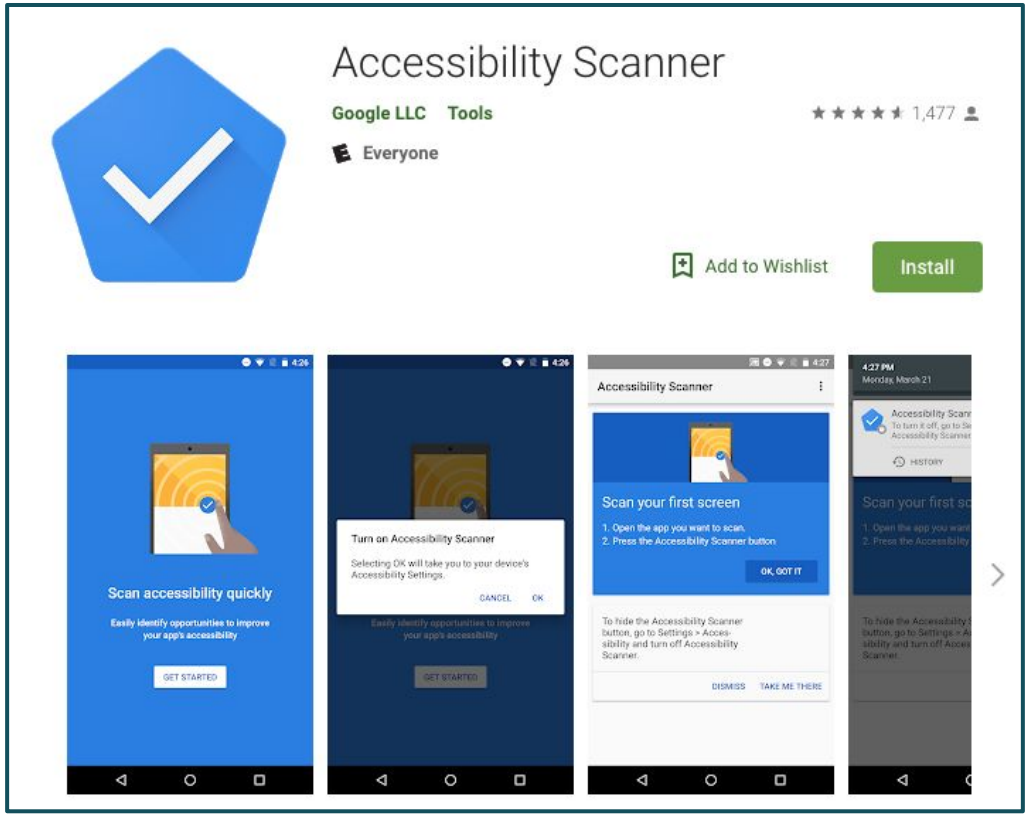
Acessibilidade

As deficiências comuns que afetam o uso de um dispositivo Android por uma pessoa incluem cegueira ou baixa visão, surdez ou deficiência auditiva, habilidades motoras restritas e daltonismo. E esta é apenas uma lista parcial.

Os aplicativos escritos com a acessibilidade em mente tornam a experiência do usuário melhor para todos: os atalhos do teclado no Gmail ajudam os usuários avançados, o alto contraste ajuda a olhar para uma tela com reflexos no fundo e os controles de voz ajudam você a controlar seu dispositivo enquanto cozinha.

Scanner de Acessibilidade

O Google oferece uma ferramenta de acessibilidade que sugere melhorias para aplicativos Android - como aumentar pequenos alvos de toque, aumentar o contraste e fornecer descrições de conteúdo - para que pessoas com necessidades de acessibilidade possam usar seu aplicativo com mais facilidade.

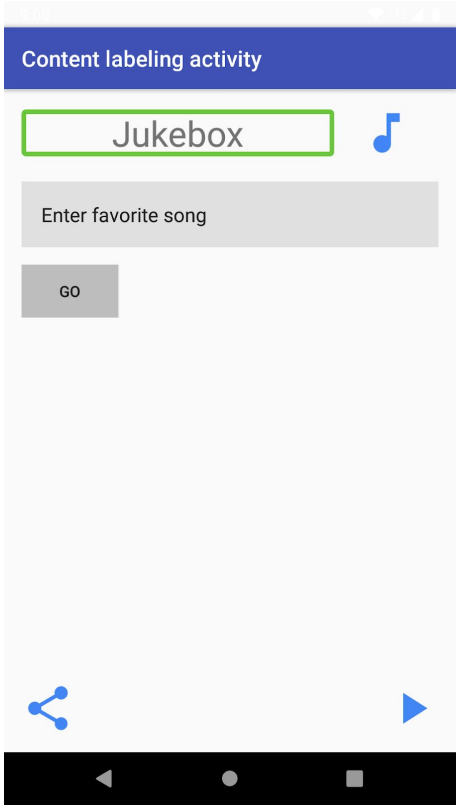


Rotulagem de Conteúdo

Quando um usuário com deficiência visual tenta navegar em um aplicativo pelo toque, o TalkBack anuncia todo o conteúdo acionável, desde que tenha rótulos significativos, úteis e descritivos.

Se esses rótulos estiverem ausentes, o TalkBack pode não ser capaz de explicar adequadamente a função de um controle específico para um usuário. Em alguns casos, o TalkBack pode pular completamente algum conteúdo.

> Dica: é uma prática recomendada usar **android: hint em** vez de **contentDescription** em **EditTexts** .



Rotulagem de Conteúdo

```xml
<ImageView
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:contentDescription="@null"
    android:src="@drawable/ic_music_note" />
```

```xml
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentStart="true"
    android:contentDescription="@string/share"
    android:background="@null"
    android:src="@drawable/ic_share" />
```

Rotulagem de Conteúdo

```
private void updateImageButton() {
    if (mPlaying) {
        mPlayPauseToggleImageView.setImageResource(R.drawable.ic_pause);
    } else {
        mPlayPauseToggleImageView.setImageResource(R.drawable.ic_play_arrow);
    }
}
```

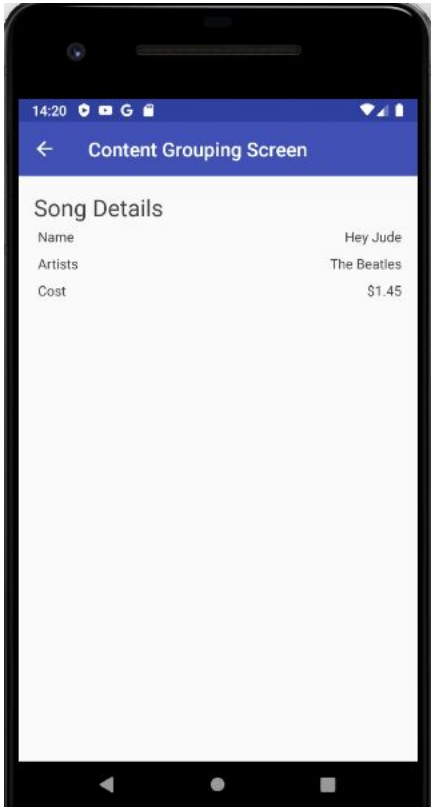
```
private void updateImageButton() {
    if (mPlaying) {
        mPlayPauseToggleImageView.setImageResource(R.drawable.ic_pause);
        mPlayPauseToggleImageView.setContentDescription(getString(R.string.pause));

    } else {
        mPlayPauseToggleImageView.setImageResource(R.drawable.ic_play_arrow);
        mPlayPauseToggleImageView.setContentDescription(getString(R.string.play));
    }
}
```

Agrupando Conteúdo

Às vezes, o feedback auditivo que o TalkBack fornece para elementos visuais em um aplicativo pode não refletir sua estrutura lógica e espacial.

Mesmo que os elementos possam ser ordenados visualmente de uma maneira sensata, eles podem ser falados fora de ordem.

Embora o usuário do TalkBack seja capaz de descobrir todo o conteúdo na tela, ele precisa deslizar várias vezes. Se os detalhes da música tivessem muito mais campos, a experiência rapidamente se tornaria tediosa.



Agrupando Conteúdo

Como os dados da música são compostos de apenas seis strings curtas, poderíamos fazer o TalkBack agrupar todos os seis itens em um único anúncio. Vamos experimentar essa abordagem.

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:focusable="true">
...
</RelativeLayout>
```

Agrupando Conteúdo

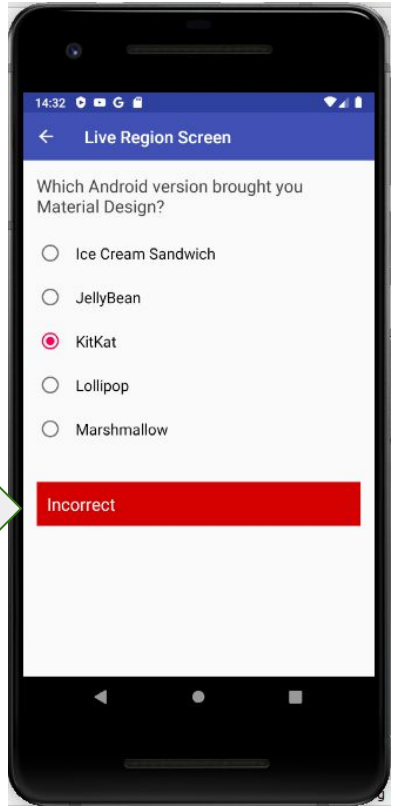
No exemplo que estamos considerando, ter um único anúncio TalkBack é melhor do que seis; mas esta solução tem seus limites:

1. Ele rapidamente se torna não escalável se o número de campos de detalhes da música aumentar ou se os campos contiverem muito texto.
2. Nem a versão original de seis anúncios nem a versão de anúncio único que acabamos de implementar levam em consideração o agrupamento natural de visualizações. Os dados são claramente organizados visualmente em duas colunas - os itens à esquerda e os valores correspondentes à direita. Esse agrupamento é evidente para um usuário que pode ver a tela, mas você deve garantir que isso seja igualmente óbvio para um usuário com deficiência visual com o TalkBack.

Usando uma Live Region

Até agora, todos os exemplos envolveram o uso do Talkback com visualizações nas quais o usuário se concentrou explicitamente. No entanto, às vezes você precisa descobrir um texto que é atualizado dinamicamente sem ter que navegar explicitamente até ele e se concentrar nele.

```
<TextView
    android:id="@+id/feedback_text_view"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@null"
    android:textColor="@color/white"
    android:padding="@dimen/standard_padding"
    android:textSize="@dimen/large_text"
    android:accessibilityLiveRegion="polite" />
```



Usando uma Live Region

Atribuímos ao nosso **accessibilityLiveRegion** o valor de "polite". Isso simplesmente significa que o TalkBack não interromperá nada que já esteja anunciando ao processar o evento gerado por uma região ao vivo.

Uma alternativa para "polite" é "assertive", que instrui o TalkBack a mover anúncios relacionados a uma região ao vivo na fila de eventos e interromper os anúncios em andamento.

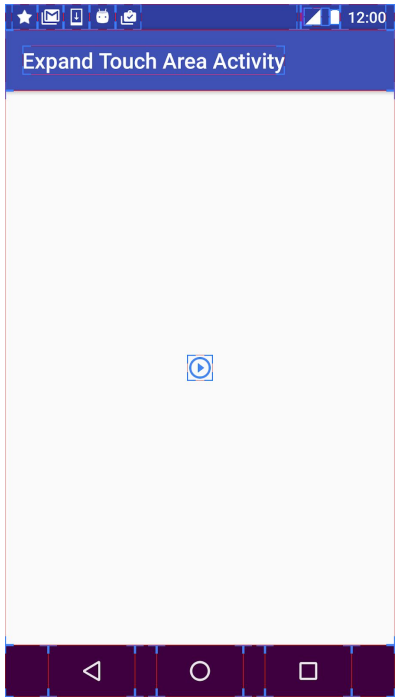
Você deve usar regiões ativas com moderação e quase sempre evitar regiões ativas "assertivas".

Áreas de Toque Acessíveis

Muitas pessoas têm dificuldade em se concentrar em pequenas áreas de toque na tela.

Isso pode ocorrer porque seus dedos são grandes ou porque eles têm uma condição médica que prejudica suas habilidades motoras.

Pequenas áreas de toque também tornam mais difícil para os usuários de leitores de tela navegar pelos aplicativos usando “explorar pelo toque”.




```
<ImageButton
    ...
    android:minWidth="48dp"
    android:minHeight="48dp" />
```

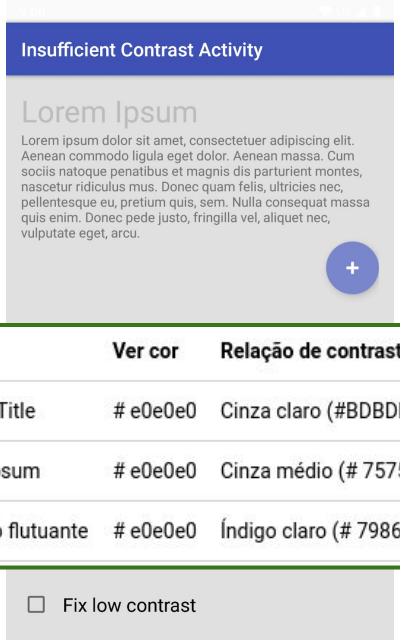
Contraste de Cor Adequado

Usuários com baixa visão não podem ler informações em uma tela se não houver contraste suficiente entre o **primeiro plano e o fundo**.

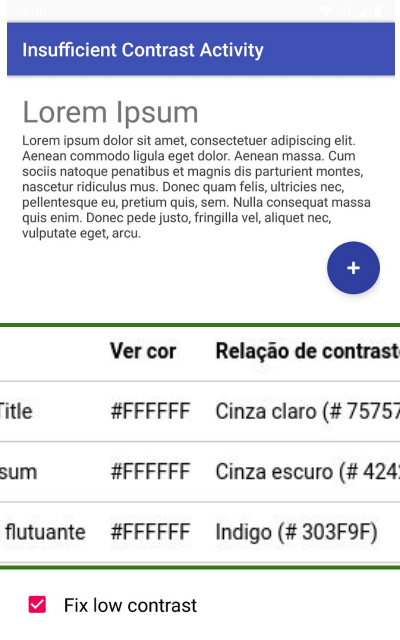
As taxas de contraste baixas entre as cores do primeiro plano e do plano de fundo podem fazer com que as visualizações fiquem desfocadas para alguns usuários, enquanto as taxas de contraste altas as tornam mais claras.

Diferentes situações de iluminação podem amplificar os problemas criados por baixas taxas de contraste.

Contraste de Cor Adequado



fundo	Ver cor	Relação de contraste	
Lorem ipsum Title	# e0e0e0	Cinza claro (#BDBDBD)	**1.42:1**
Texto lorem ipsum	# e0e0e0	Cinza médio (# 757575)	**3.49:1**
Botão de ação flutuante	# e0e0e0	Índigo claro (# 7986CB)	**2.61:1**



fundo	Ver cor	Relação de contraste	
Lorem ipsum Title	#FFFFFF	Cinza claro (# 757575)	**4.61:1**
Texto lorem ipsum	#FFFFFF	Cinza escuro (# 424242)	**10.05:1**
Botão de ação flutuante	#FFFFFF	Indigo (# 303F9F)	**8.98:1**

As Diretrizes de acessibilidade de conteúdo da Web recomendam uma taxa de contraste mínima de **4,5: 1** para todo o texto, com uma taxa de contraste de **3,0: 1** considerada aceitável para texto grande ou em negrito, e você deve tentar atender ou exceder essas taxas de contraste em seus aplicativos.

Exercício

1. Instalar o **Accessibility Scanner**.
2. Explorar o CodeLabs de Acessibilidade.
3. Passar o Accessibility Scanner no seu projeto.