Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

Interface com o Usuário

Medidas

Medidas

É possível especificar largura e altura com medidas exatas, embora não seja recomendável na maioria dos casos. Em geral, usa-se uma destas constantes para definir a largura e a altura:

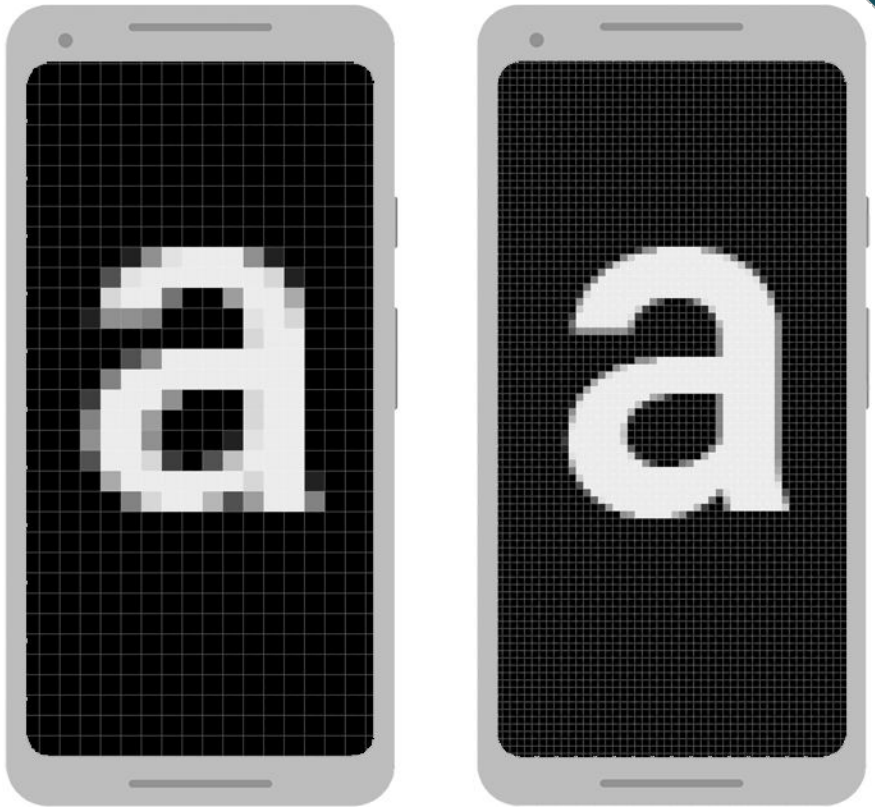
- **wrap_content**: instrui a exibição a se redimensionar de acordo com as medidas exigidas pelo conteúdo.
- **match_parent**: instrui a exibição a assumir o maior tamanho permitido pelo grupo de exibições pais.

Observação: ao usar **ConstraintLayout**, não use match_parent. Em vez disso, defina a dimensão como **0dp** para ativar um comportamento especial chamado de “corresponder restrições”, que geralmente é o mesmo que esperado de match_parent.

Medidas

A primeira armadilha a ser evitada é usar **pixels** para definir distâncias ou tamanhos.

Definir dimensões com pixels é um problema porque telas diferentes têm densidades pixels diferentes, de maneira que o mesmo número de pixels pode corresponder a diferentes tamanhos físicos nos vários dispositivos.



Medidas

Para preservar o tamanho visível da sua interface do usuário em telas com diferentes densidades, você precisa projetar sua IU com pixels de densidade independente (**dp**) como unidade de medida.

Um **dp** é uma unidade de **pixel virtual** aproximadamente do tamanho de um pixel em uma tela de densidade média (160 dpi, a densidade "básica").

O Android converte esse valor no número apropriado de pixels reais para cada densidade.

Medidas

Você precisa usar pixels escalonáveis (**sp**) como suas unidades para definir **tamanhos de texto**, mas nunca para tamanhos de layout.

Por padrão, a unidade **sp** é do mesmo tamanho que a **dp**, mas ela é redimensionada com base no tamanho de texto preferencial do usuário.

Medidas

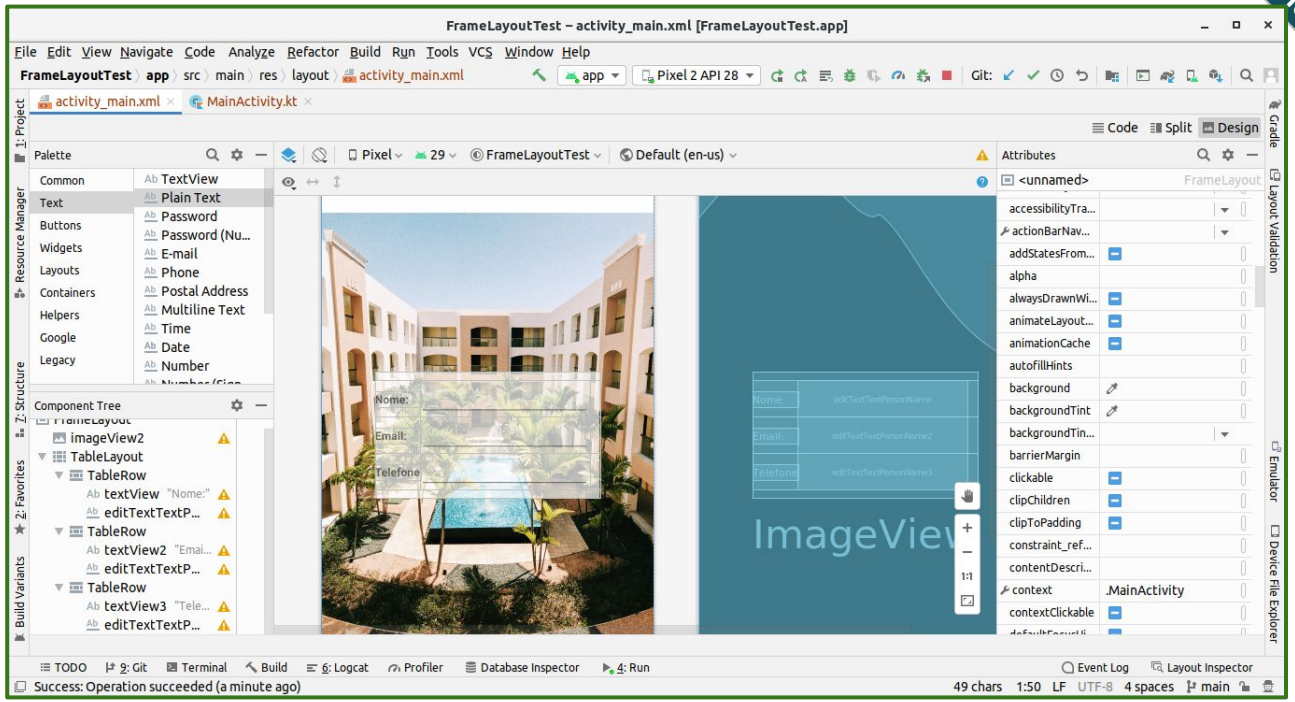
px → Correspondente ao número de pixels (pontos) da tela e a recomendação é evitar utilizar px e apenas usar em casos bem específicos e raros.

dip / **dp** → Density-Independent Pixels: essa unidade é relativa à resolução da tela e deve ser usada sempre que for definir medidas de componentes visuais. Por exemplo se a resolução da tela é de 160 dpi, significa que um dp representa 1 pixel em um total de 160.

sp → Scale-Independent Pixels: considera o tamanho da fonte que o usuário está utilizando. É recomendado que use essa unidade somente quando especificar o **tamanho de uma fonte**, para que esta seja automaticamente ajustada conforme as preferências da tela do usuário.

FrameLayout


4:55
FrameLayoutTest
Nome:
Email:
Telefone


FrameLayoutTest – activity_main.xml [FrameLayoutTest.app]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
activity_main.xml
MainActivity.kt
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
TextView
Plain Text
Password
E-mail
Phone
Postal Address
Multiline Text
Time
Date
Number
Component Tree
imageView2
TableLayout
TableRow
textView "Nome:"
editTextTextP...
textView2 "Emai...
textView3 "Tele...
Attributes
Code Split Design
Nome:
Email:
Telefone
ImageVie
TODO 9: Git Terminal Build 6: Logcat Profiler Database Inspector 4: Run
Event Log Layout Inspector
Success: Operation succeeded (a minute ago)

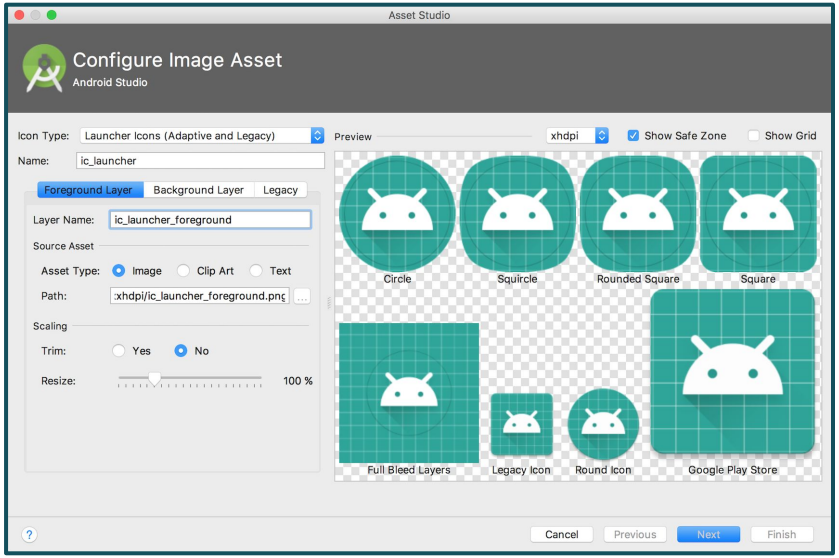
Ícones Adaptativos

Ícones Adaptativos

O Android 8.0 (API de nível 26) apresenta ícones de inicialização adaptáveis, que podem exibir uma variedade de formas em diferentes modelos de dispositivos.

Cada dispositivo fornece uma máscara, que o sistema usa para renderizar todos os ícones adaptáveis da mesma forma.

Um ícone de iniciador adaptável também é usado em atalhos, aplicativos de configurações, diálogos de compartilhamento e tela de visão geral.



Diálogos

Diálogos

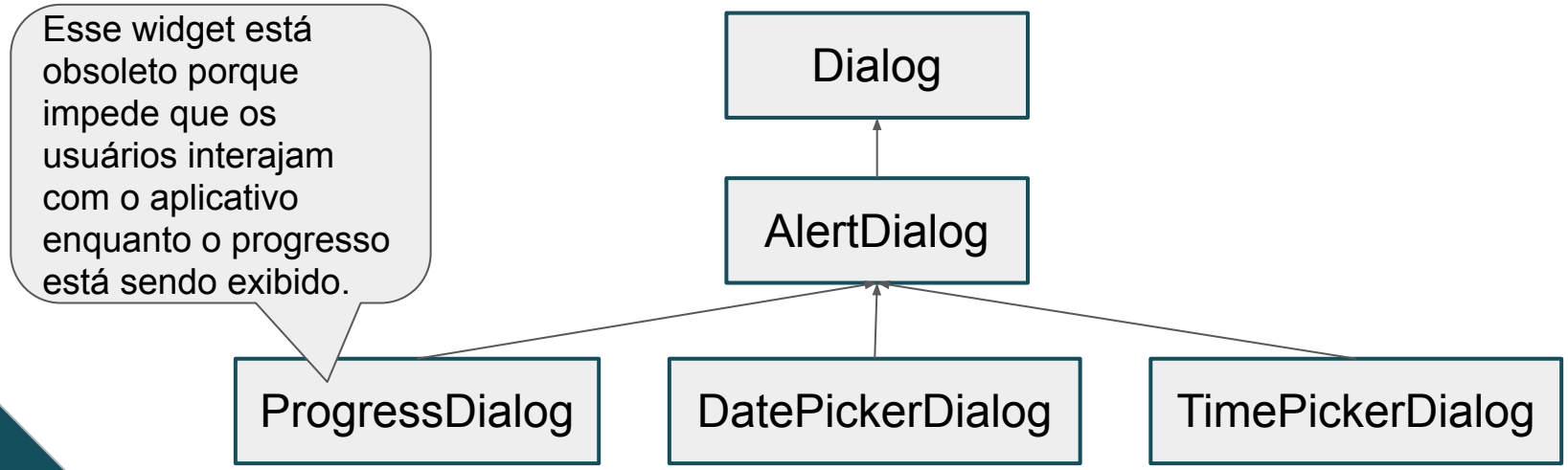
Os diálogos constituem uma parte importante dos aplicativos Android.

Uma caixa de diálogo é uma pequena janela que solicita que o usuário tome uma decisão, insira informações adicionais ou dê um feedback.

Ele não ocupa a tela inteira, mas parte dela, e pe
aos usuários que realizem uma ação específica
antes de prosseguir.

Diálogos

O Android fornece um conjunto de subclasses do Dialog que você pode utilizar para interagir com os usuários.



Diálogos

MainActivity.kt | activity_main.xml

```
15  class MainActivity : AppCompatActivity(), View.OnClickListener, DatePickerDialog.OnDateSetListener {
16
17      override fun onCreate(savedInstanceState: Bundle?) {
18
19          super.onCreate(savedInstanceState)
20          setContentView(R.layout.activity_main)
21
22          val btnAlertDialog :Button! = this.findViewById<Button>(R.id.btnAlertDialog)
23          btnAlertDialog.setOnClickListener(this)
24
25          val btnProgressBar :Button! = this.findViewById<Button>(R.id.btnProgressBar)
26          btnProgressBar.setOnClickListener(this)
27
28          val btnDatePicker :Button! = this.findViewById<Button>(R.id.btnDatePicker)
29          btnDatePicker.setOnClickListener(this)
30
31          val btnSnackBar :Button! = this.findViewById<Button>(R.id.btnSnackBar)
32          btnSnackBar.setOnClickListener(this)
33      }
```

1

2

3

Diálogos

MainActivity.kt | activity_main.xml

```
35    override fun onClick(view: View?) {
36
37        val button :Button  = view as Button
38        when(button.text) {
39
40            "AlertDialog" -> alertDialogTest()
41            "ProgressBar" -> progressBarTest()
42            "DatePicker" -> datePickerTest()
43            "SnackBar" -> snackBarTest()
44            "Clique Aqui" -> Toast.makeText( context: this,  text: "Snack Bar clicado", Toast.LENGTH_SHORT).show()
45        }
46    }
```

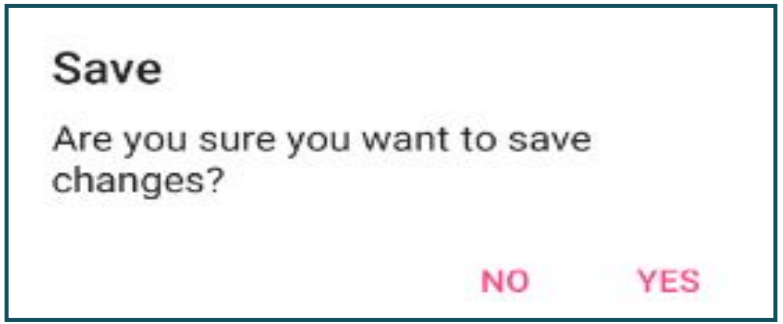
1

2

AlertDialog

Esta caixa de diálogo consiste em um título, vários botões e / ou uma lista de itens selecionáveis que podem incluir caixas de seleção ou botões de rádio.

O diálogo de alerta é capaz de construir a maioria das interfaces de usuário de diálogo; portanto, é o tipo de diálogo geralmente sugerido.



AlertDialog

MainActivity.kt | activity_main.xml

```
51  private fun alertDialogTest() {
52
53      val alertDialog = AlertDialog.Builder( context: this)
54      alertDialog.setTitle("Teste")
55      alertDialog.setMessage("Esta é uma mensagem de alerta bem simples")
56      alertDialog.setPositiveButton( text: "Tudo Bem",  listener: this)
57      alertDialog.setNegativeButton( text: "Nada Bem",  listener: this)
58      alertDialog.setCancelable(false) //diálogo "modal"
59      alertDialog.show()
60  }
61
62  override fun onClick(dialog: DialogInterface?, p1: Int) {
63
64      val alertDialog :AlertDialog  = dialog as AlertDialog
65      Toast.makeText( context: this,  text: "Botão ${alertDialog.getButton(p1).text} Clicado", Toast.LENGTH_SHORT).show()
66  }
```

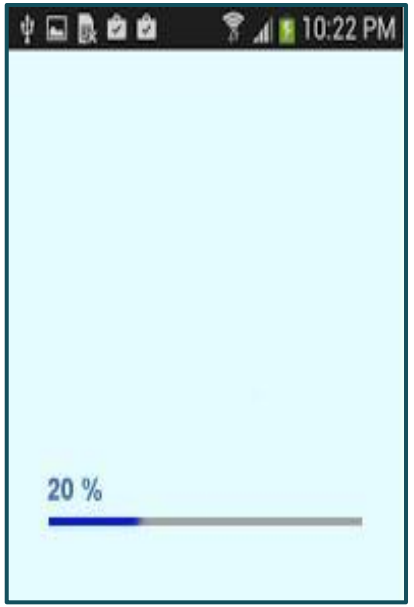
1

2

Progress Bar

Os desenvolvedores Android usam a classe Progress Bar para obter uma barra de progresso circular ou horizontal.

O objetivo dessa barra é fazer com que o usuário espere um certo tempo até que um determinado processo seja concluído.



ProgressBar

MainActivity.kt | activity_main.xml

```
68  private fun progressBarTest() {
69
70      val prgProgressBar :ProgressBar! = this.findViewById<ProgressBar>(R.id.prgProgressBar)
71      val lblProgressBar :TextView! = this.findViewById<TextView>(R.id.lblProgressBar)
72      Thread(Runnable {
73
74          for (i :Int in 0..100) {
75
76              this@MainActivity.runOnUiThread(Runnable {
77
78                  prgProgressBar.progress = i
79                  lblProgressBar.text = i.toString() + "%"
80              })
81              Thread.sleep( millis: 200)
82          }
83      }).start()
84  }
```

1

2

3

SeekBar

Uma SeekBar é uma extensão da ProgressBar arrastável.

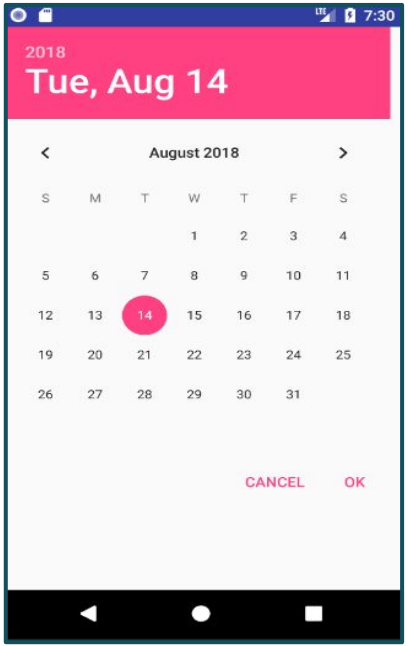
O usuário pode tocar o polegar e arrastá-lo para a esquerda ou direita para definir o nível de progresso atual ou ele / ela pode usar as teclas de seta.

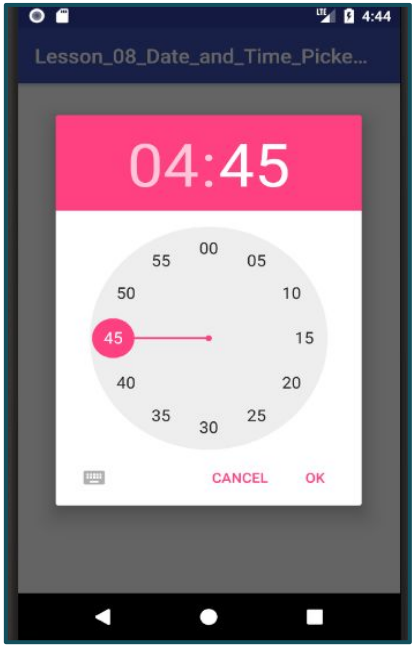
Date / Time Picker

O Android fornece controles para o usuário escolher uma data ou hora como caixas de diálogo prontas para uso.

Cada seletor fornece controles para selecionar cada parte da data (mês, dia, ano) ou hora (hora, minuto, AM / PM).

O uso desses seletores ajuda a garantir que seus usuários possam escolher uma hora ou data válida, formatada corretamente e ajustada à localidade do usuário.





DatePicker

MainActivity.kt | activity_main.xml

```
86      private fun datePickerTest() {
87
88          val calendar :Calendar  = Calendar.getInstance()
89          val dia :Int  = calendar.get(Calendar.DAY_OF_MONTH)
90          val mes :Int  = calendar.get(Calendar.MONTH)
91          val ano :Int  = calendar.get(Calendar.YEAR)
92
93          val datePicker :Unit  = DatePickerDialog( context: this,  listener: this, ano, mes, dia).show()
94      }
95
96      override fun onDateSet(datePicker: DatePicker?, ano: Int, mes: Int, dia: Int) {
97
98          Toast.makeText( context: this,  text: "Data Selecionada = $dia/${mes+1}/$ano", Toast.LENGTH_SHORT).show()
99      }
```

1

2

Snackbar

A classe Snackbar fornece um feedback leve sobre uma operação.

Snackbars mostram uma breve mensagem na parte inferior da tela em smartphones e inferior esquerdo em dispositivos maiores.

SnackBar

MainActivity.kt | activity_main.xml

```
101    private fun snackBarTest() {
102
103        val snackBar :Unit  = Snackbar
104            .make(this.findViewById(R.id.pnlMain),  text: "Teste de Snack Bar", Snackbar.LENGTH_LONG)
105            .setActionTextColor(Color.RED)
106            .setAction( text: "Clique Aqui",  listener: this)
107            .show()
108    }
```

Idiomas e Culturas

Idiomas e Culturas

Os apps incluem recursos que podem ser específicos para determinada cultura. Por exemplo, um app pode incluir strings próprias de uma cultura que são traduzidas para o idioma da localidade atual.

É recomendável manter os recursos específicos da cultura separados do restante do app.

O Android resolve recursos específicos de idioma e cultura com base na configuração de localidade do sistema.

Você pode oferecer compatibilidade com diferentes localidades usando o diretório de recursos no seu projeto Android.

Idiomas e Culturas

Você pode especificar recursos adaptados à cultura das pessoas que usam seu app e fornecer qualquer tipo de recurso apropriado para o idioma e a cultura dos seus usuários.

Por exemplo, a captura de tela a seguir mostra um app exibindo recursos drawable e de string na localidade padrão do dispositivo (en_US) e em espanhol (es_ES).

Menus

Menus

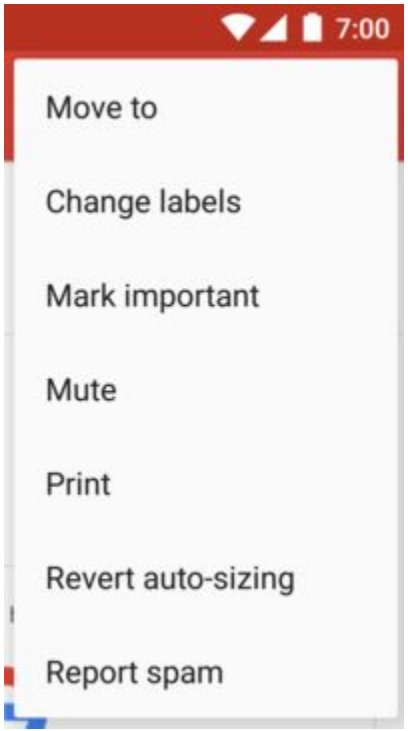
Menus são componentes comuns da interface do usuário em diversos tipos de aplicativos. Para fornecer uma experiência familiar e consistente ao usuário, você precisa usar APIs de Menu para apresentar ações de usuário e outras opções nas suas atividades.

Desde o Android 3.0 (API de nível 11), dispositivos Android não precisam mais fornecer um botão de Menu dedicado. Com essa alteração, os aplicativos Android migrarão de uma dependência do painel de menu de seis itens tradicional para fornecer uma barra de aplicativos para apresentar as ações comuns de usuário.

Menus

Apesar de o design e a experiência do usuário para alguns dos itens do menu terem passado por mudanças, a semântica para definir um conjunto de ações e opções ainda se baseia em APIs de Menu.

1. **Menu de opções e barra de aplicativos.**
2. Modo de ação contextual e menu de contexto.
3. Menu pop-up.



Menus

Para todos os tipos de menu, o Android fornece um formato XML padrão para definir os itens de menu.

Em vez de criar um menu no código da atividade, você precisa definir um menu e todos os seus itens em um recurso de menu XML.

Usar um recurso de menu é uma boa prática por alguns motivos:

- É mais fácil para visualizar a estrutura do menu em XML.
- Ele separa o conteúdo do menu do código comportamental do aplicativo.
- Ele permite criar configurações alternativas de menu para versões diferentes de plataforma, de tamanhos de tela e de outras configurações aproveitando a biblioteca de recursos do aplicativo.

Menus

Para definir o menu, crie um arquivo XML dentro do diretório res/menu/ do projeto e crie o menu com os seguintes elementos:

<menu> → Define um Menu, que é um contêiner para os itens de menu. Um elemento <menu> precisará ser o nó raiz para o arquivo e pode reter um ou mais elementos <item> e <group>.

<item> → Cria um MenuItem, que representa um único item em um menu. Esse elemento pode conter um elemento <menu> aninhado para criar um submenu.

<group> → Um contêiner invisível e opcional para os elementos <item>. Ele permite que você categorize itens de menu para que eles compartilhem propriedades como estado ativo e visibilidade. Para mais informações, consulte a seção Criação de grupos de menu.

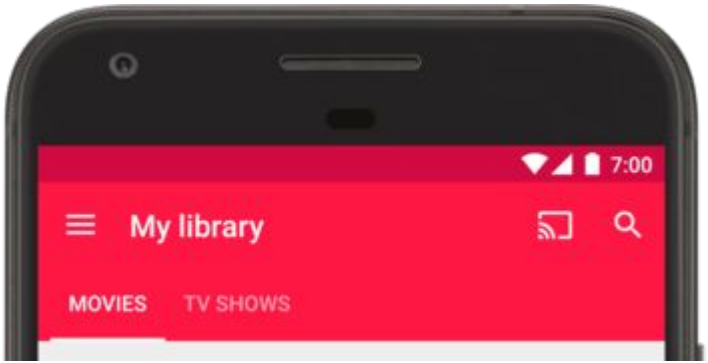
Menus

Caso tenha desenvolvido o aplicativo para Android 2.3.x (API de nível 10) ou inferior, os conteúdos do menu de opções aparecerão na parte inferior da tela, quando o usuário pressionar o botão Menu, como mostrado na figura.

Menus

Se você desenvolveu o aplicativo para Android 3.0 (API de nível 11) ou superior, os itens do menu de opções estão disponíveis na barra de aplicativos.

Por padrão, o sistema posiciona todos os itens nas ações adicionais, que o usuário pode revelar com o ícone de ações adicionais no lado direito da barra de aplicativos (ou pressionando o botão Menu no dispositivo, se disponível).



Menus

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

```kotlin
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.game_menu, menu)
    return true
}
```

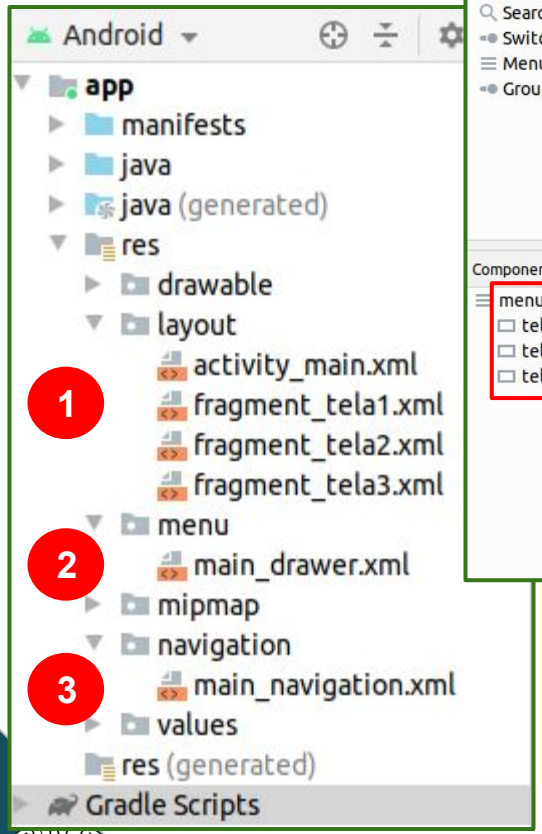
Menus

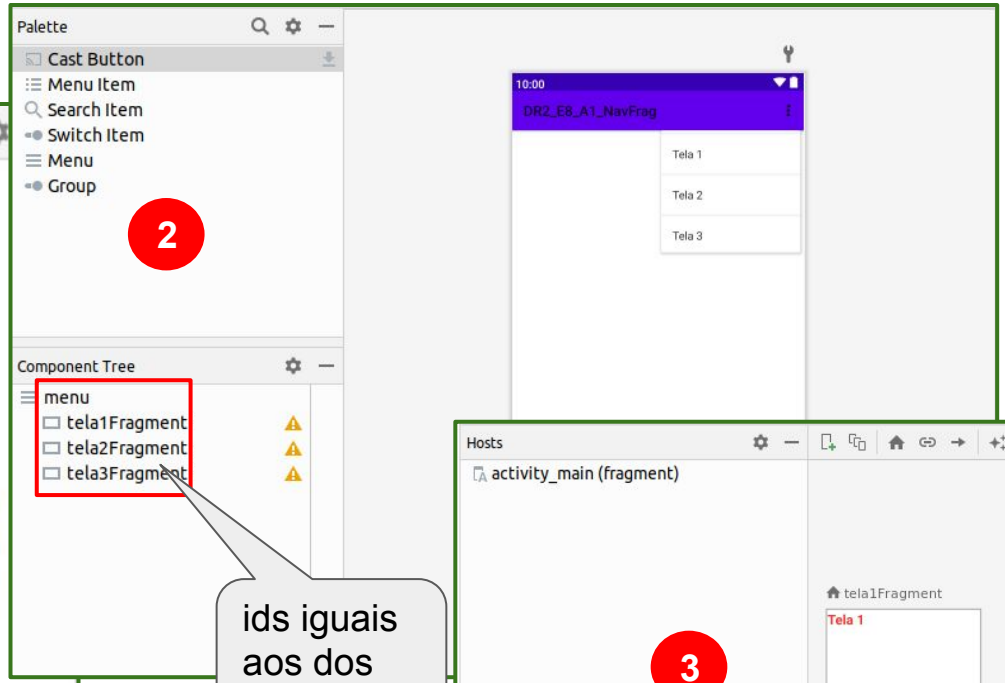
```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Handle item selection
    return when (item.itemId) {
        R.id.new_game -> {
            newGame()
            true
        }
        R.id.help -> {
            showHelp()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Drawer

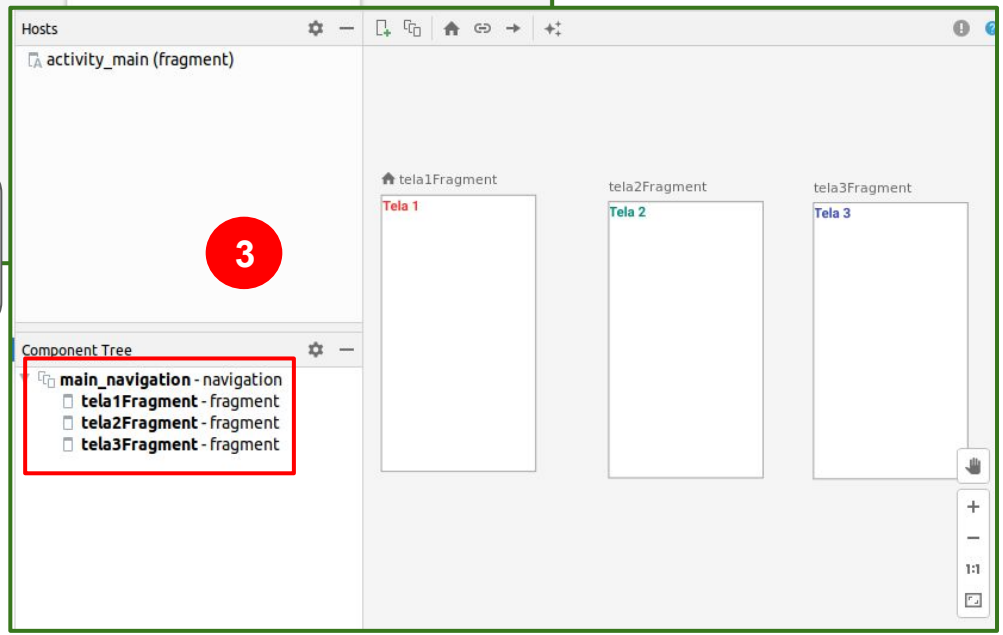
build.gradle (:app)

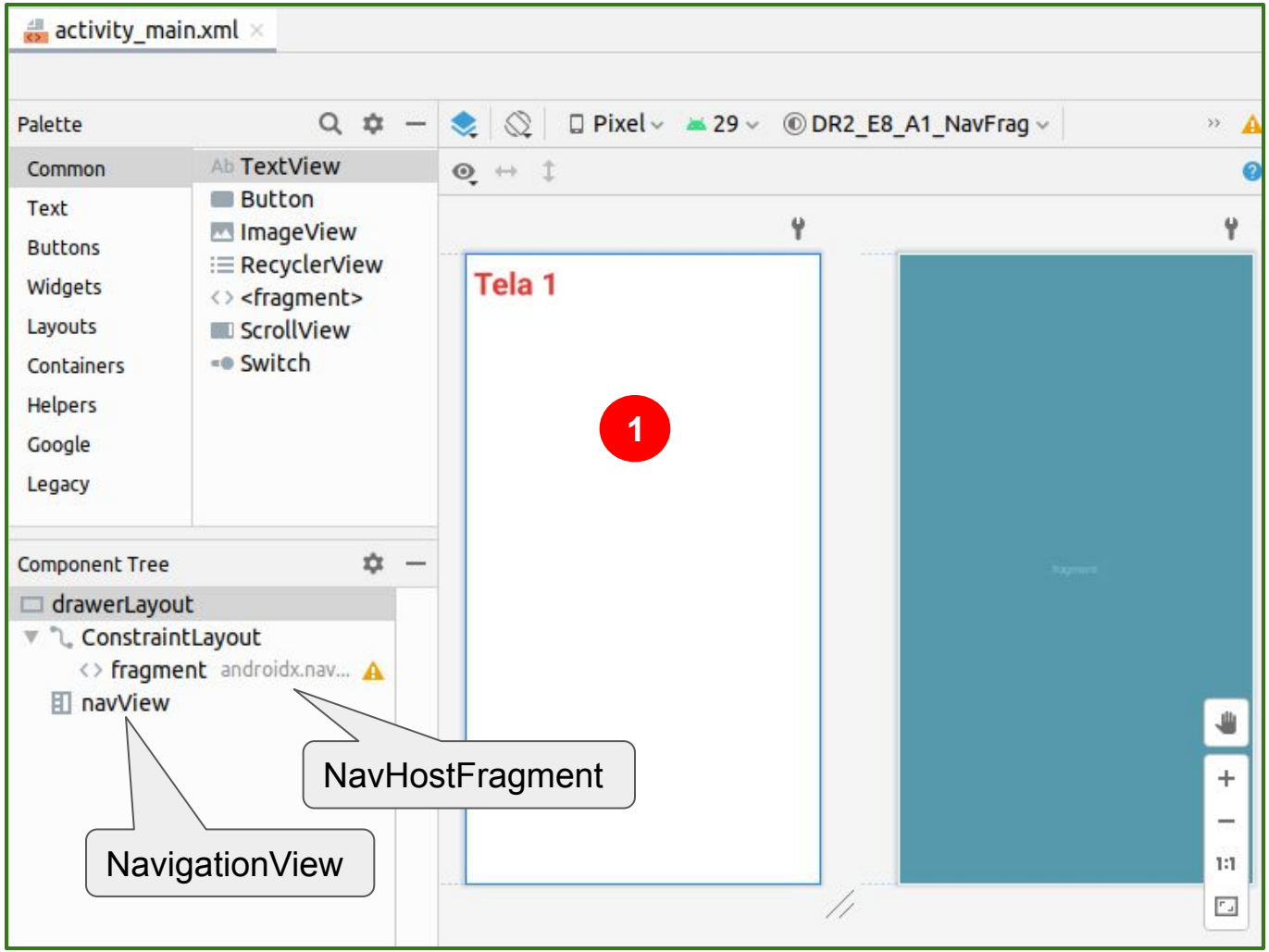
```
46      //--------------------------------------------------------------------------
47      //Navigation
48      def navigationVersion = "2.3.0"
49      implementation "androidx.navigation:navigation-fragment-ktx:$navigationVersion"
50      implementation "androidx.navigation:navigation-ui-ktx:$navigationVersion"
51  }
```





ids iguais aos dos fragments




activity_main.xml
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Pixel
29
DR2_E8_A1_NavFrag
Tela 1
1
Component Tree
drawerLayout
ConstraintLayout
fragment androidx.nav...
navView
NavHostFragment
NavigationView
1:1

activity_main.xml

```
1  <androidx.drawerlayout.widget.DrawerLayout
4      xmlns:android="http://schemas.android.com/apk/res/android"
5      xmlns:app="http://schemas.android.com/apk/res-auto"
6      xmlns:tools="http://schemas.android.com/tools"
2      android:id="@+id/drawerLayout"
8      android:layout_height="match_parent"
9      android:layout_width="match_parent">
10
11     <androidx.constraintlayout.widget.ConstraintLayout
12         android:layout_width="match_parent"
13         android:layout_height="match_parent"
14         tools:context=".MainActivity">
15
3      <fragment
17             android:id="@+id/fragment"
18             android:name="androidx.navigation.fragment.NavHostFragment"
19             android:layout_width="0dp"
20             android:layout_height="0dp"
21             app:defaultNavHost="true"
```

activity_main.xml

```
16      3 <fragment
17            android:id="@+id/fragment"
18            android:name="androidx.navigation.fragment.NavHostFragment"
19            android:layout_width="0dp"
20            android:layout_height="0dp"
21            app:defaultNavHost="true"
22            app:layout_constraintBottom_toBottomOf="parent"
23            app:layout_constraintEnd_toEndOf="parent"
24            app:layout_constraintStart_toStartOf="parent"
25            app:layout_constraintTop_toTopOf="parent"
26            app:navGraph="@navigation/main_navigation" />
27    </androidx.constraintlayout.widget.ConstraintLayout>
28    4 <com.google.android.material.navigation.NavigationView
29          android:id="@+id/navView"
30          android:layout_width="wrap_content"
31          android:layout_height="match_parent"
32          android:layout_gravity="start"
33          app:menu="@menu/main_drawer" />
34  </androidx.drawerlayout.widget.DrawerLayout>
```

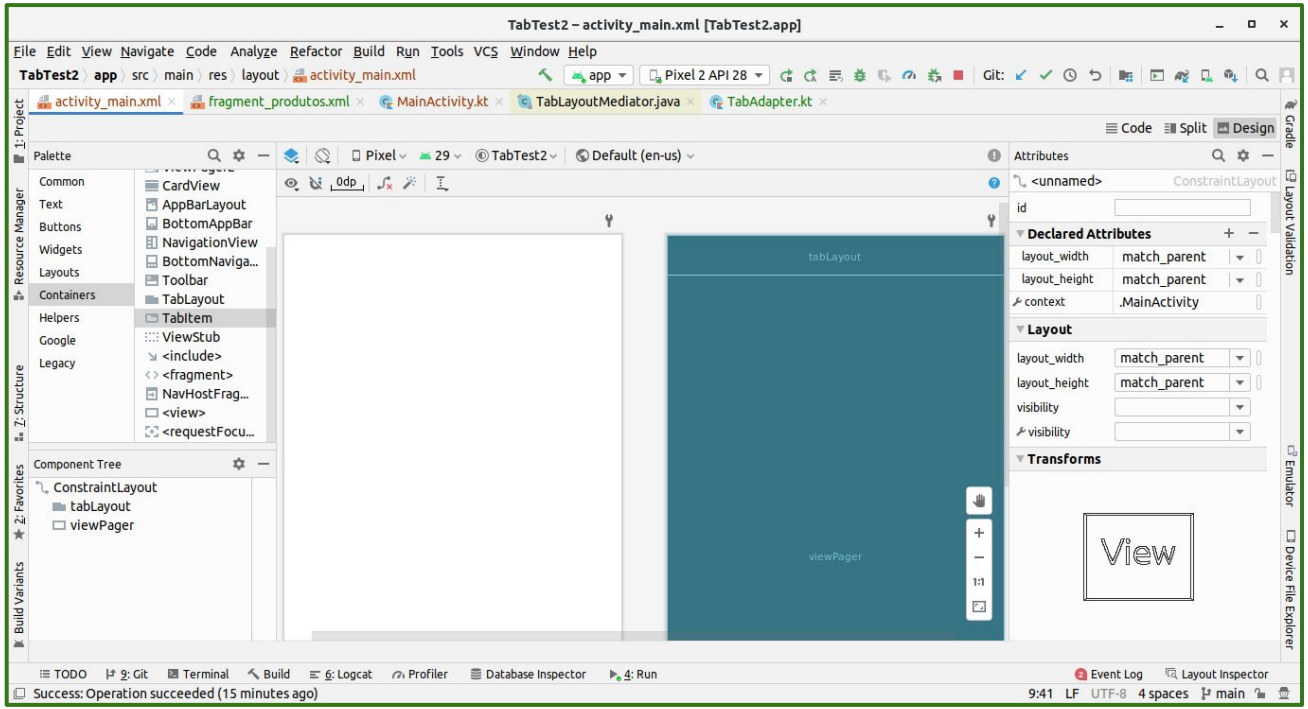
MainActivity.kt

```
11  class MainActivity : AppCompatActivity() {
12
13      lateinit var drawerLayout : DrawerLayout
14      lateinit var navController : NavController
15      lateinit var navView : NavigationView
16
17      override fun onCreate(savedInstanceState: Bundle?) {
18
19          super.onCreate(savedInstanceState)
20          setContentView(R.layout.activity_main)
21          //-------------------------------------------------------------------
22          drawerLayout = this.findViewById<DrawerLayout>(R.id.drawerLayout)
23          navController = this.findNavController(R.id.fragment)
24          navView = this.findViewById<NavigationView>(R.id.navView)
25          //-------------------------------------------------------------------
26          NavigationUI.setupActionBarWithNavController( activity: this,navController, drawerLayout)
27          NavigationUI.setupWithNavController(navView, navController)
28      }
29
30      override fun onSupportNavigateUp(): Boolean {
31
32          return NavigationUI.navigateUp(navController, drawerLayout)
33      }
```

1
2
3
4

TabLayout


4:21
TabTest2
FORNECEDORES
CONTATOS
PRODUTOS
Fornecedores


TabTest2 – activity_main.xml [TabTest2.app]
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
activity_main.xml
fragment_produtos.xml
MainActivity.kt
TabLayoutMediator.java
TabAdapter.kt
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Helpers
Google
Legacy
CardView
AppBarLayout
BottomAppBar
NavigationView
Toolbar
TabLayout
TabItem
ViewStub
Component Tree
ConstraintLayout
tabLayout
viewPager
tabLayout
viewPager
Attributes
Declared Attributes
layout_width
match_parent
layout_height
match_parent
context
.MainActivity
Layout
visibility
Transforms
View
TODO
9: Git
Terminal
Build
6: Logcat
Profiler
Database Inspector
4: Run
Event Log
Layout Inspector
Success: Operation succeeded (15 minutes ago)

TabAdapter.kt

```
7   class TabAdapter(fragmentActivity: FragmentActivity) : FragmentStateAdapter(fragmentActivity) {
8
9       override fun getItemCount(): Int = 3
10
11      override fun createFragment(position: Int): Fragment {
12
13          lateinit var fragment : Fragment
14          when(position) {
15
16              0 -> fragment = FornecedoresFragment()
17              1 -> fragment = ContatosFragment()
18              2 -> fragment = ProdutosFragment()
19          }
20          return fragment
21      }
22  }
```

1

2

3

MainActivity.kt

```
10  class MainActivity : AppCompatActivity() {
11      override fun onCreate(savedInstanceState: Bundle?) {
12          super.onCreate(savedInstanceState)
13          setContentView(R.layout.activity_main)
14
15          var tabLayout = findViewById<TabLayout>(R.id.tabLayout)
16          var viewPager2 = findViewById<ViewPager2>(R.id.viewPager)
17          viewPager2.adapter = TabAdapter( fragmentActivity: this)
18          TabLayoutMediator(tabLayout, viewPager2) { tab, position ->
19              viewPager2.setCurrentItem(tab.position, smoothScroll: true)
20              if (position == 2) tab.setText("Produtos")
21              if (position == 1) tab.setText("Contatos")
22              if (position == 0) tab.setText("Fornecedores")
23          }.attach()
24      }
25  }
```

1
2
3