



Introdução ao Kotlin e Primeiros Passos

- **Kotlin é uma linguagem moderna e concisa.**

Tem se destacado como linguagem de programação em ascensão nos últimos anos.

- **Oficialmente suportada pelo Google para desenvolvimento Android.**

Em 2017, o time do Android da Google oficialmente anunciou o Kotlin como a principal linguagem de programação para o desenvolvimento de aplicações Android.

- **Compatível com Java e roda na JVM (Java Virtual Machine).**

Pode ser executada em máquinas virtuais Java (JVM), além de outras plataformas, como o Android. Além disso, Kotlin tem uma sintaxe semelhante ao Java e muitas das suas características e conceitos de programação são parecidas.



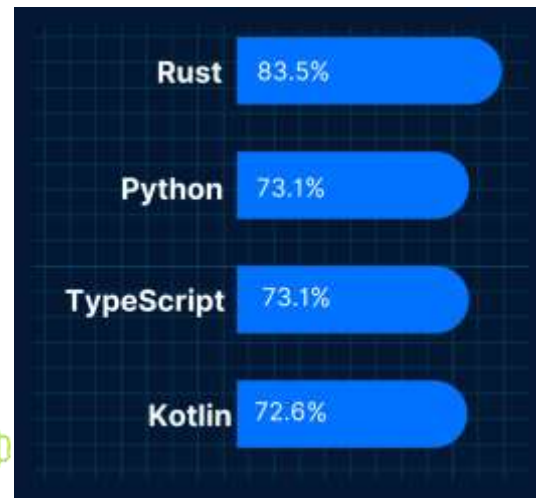
Histórico da Linguagem

- **Criada pela JetBrains em 2010.**

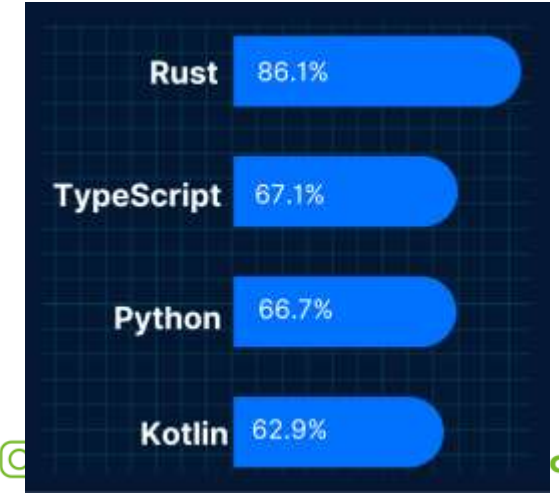
2010, JetBrains, mesma criadora Pycharm, decidiu criar uma nova linguagem de programação para a JVM (Java Virtual Machine)

- **Apresentada publicamente em 2011 (JVMLS).**
- **Lançamento oficial da versão 1.0: fevereiro de 2016.**
- **Desde 2017, anunciado como linguagem oficial dev. Android**
- **Ganhou destaque nas Developer Surveys (2019 e 2020).**

StackOverflow,
2019



StackOverflow,
2020

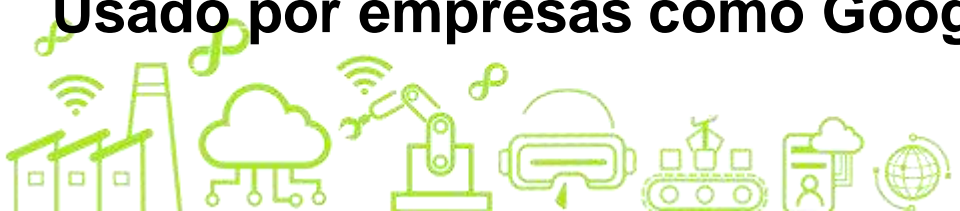


Onde Kotlin é Utilizado

Plataformas:

- **Android:** popular por sua sintaxe concisa e interoperabilidade com o Java. Mais fácil e eficiente para desenvolver aplicativos Android.
- **Desktop:** Kotlin + Compose, app serve tanto pra mobile, quanto pra desktops.
- **Web:** Cada vez mais sendo adotado para o desenvolvimento web ou sistemas de servidor, com diversos frameworks e bibliotecas disponíveis para ajudar o desenvolvedor.
- **Back-end:** Em Android, o código é executado em uma máquina virtual Android; já em sistemas back-end, é num servidor de aplicativos ou máquina virtual Java.

Usado por empresas como Google, Pinterest, Uber, Netflix e Trello.



Por que Aprender Kotlin?

Sintaxe limpa e fácil de entender.

Interoperabilidade com Java.

Suporte a múltiplos paradigmas (OO e funcional).

Popularidade crescente e oportunidades no mercado.

Documentação bem intuitiva

Exemplos detalhados

Suporte ativo da JetBrains

Comunidade crescente



Kotlin x Flutter

Kotlin: Linguagem de programação. (assim como Java, Dart, Swift etc.)

Linguagem compatível com Java, desenvolvedores podem usar bibliotecas existentes em Java para seus projetos.



Flutter: Framework baseado em Dart. (como Android, React Native etc.)

Framework específico criado pela google para desenvolvimento de aplicativos mobile e é escrito na linguagem de programação Dart.

Flutter é ideal para apps multiplataforma.

se o objetivo do projeto é entregar um aplicativo para diferentes plataformas com apenas uma base de código

Kotlin pode ser usado com o Kotlin Multiplatform (KMM).

Kotlin Multiplatform (KMM)

Kotlin Multiplatform Mobile (KMM) é um SDK desenvolvido pela JetBrains que simplifica a criação de aplicativos móveis para Android e iOS com uma base de código compartilhada.

Permite que você escreva a lógica de negócios, como manipulação de dados, chamadas de rede e operações de armazenamento, em Kotlin comum, enquanto a interface do usuário permanece nativa para cada plataforma.



Kotlin Multiplatform (KMM)

Código Compartilhado: Você desenvolve a lógica de negócios em Kotlin, que é compartilhada entre as plataformas.

Código Específico da Plataforma: As partes específicas, como a interface do usuário e APIs nativas, são implementadas nas linguagens nativas — Kotlin/Java para Android e Swift/Objective-C para iOS.

Compilação: No Android, o código Kotlin é compilado para bytecode JVM. No iOS, o Kotlin/Native compila o código para binários nativos que podem ser executados diretamente no dispositivo iOS.



KMP Compose

Kotlin Multiplatform com Jetpack Compose, também conhecido como Compose Multiplatform ou Compose for Multiplatform. Essa tecnologia permite que você crie interfaces de usuário nativas para várias plataformas (Android, iOS, desktop e web) usando uma única base de código Kotlin, aproveitando o poder do Jetpack Compose.



Criando nosso primeiro Projeto – Android Studio

Opção A:

Veja se há “Jetbrains Toolbox”

Opção instalar Android Studio



Opção B:

Busque “Android Studio Download”

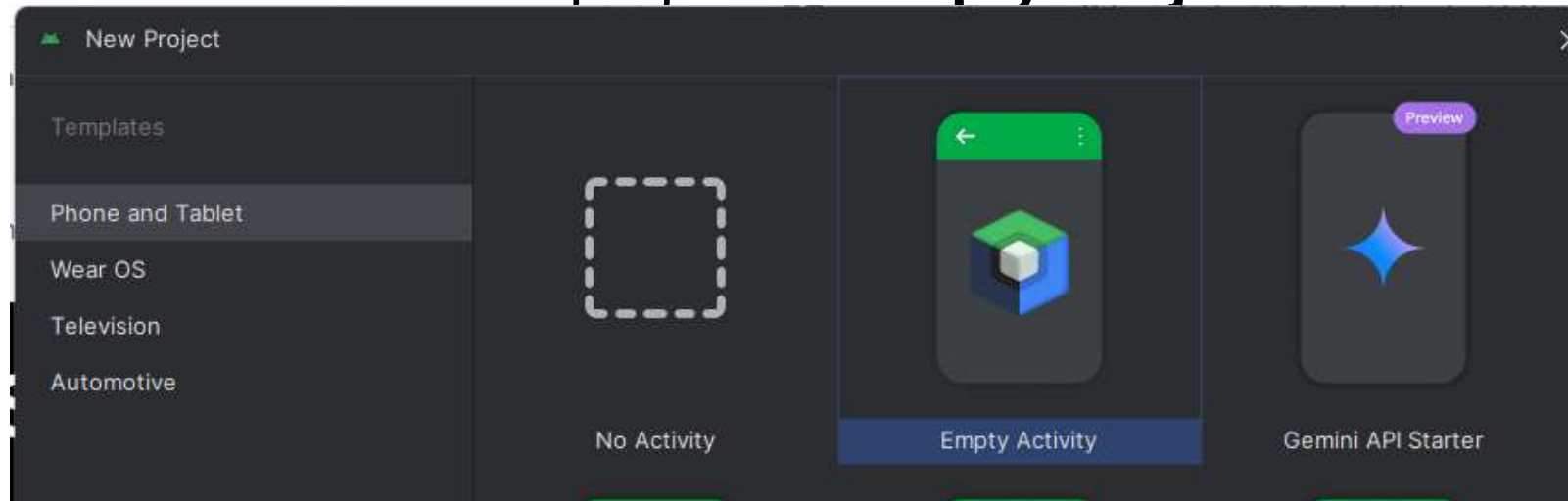


Vai demorar um pouco, ele vai fazer os downloads necessários do Android, tenha pelo menos 3GB livres




Criando nosso primeiro Projeto – nome: “Teste”

1º Primeiro: Seleccione Novo projeto > “Empty Project”



2º Segundo: Mude o nome para “Test” e Inicie o projeto.

Name	Test
Package name	com.example.test
Save location	C:\Users\jonatas.lopes\AndroidStudioProjects\Test2
Minimum SDK	API 24 ("Nougat"; Android 7.0)
<p> Your app will run on approximately 98,6% of devices. Help me choose</p>	
Build configuration language ?	Kotlin DSL (build.gradle.kts) [Recommended]



Vizualizar Pastas

Arquivo .ks “MainActivity”

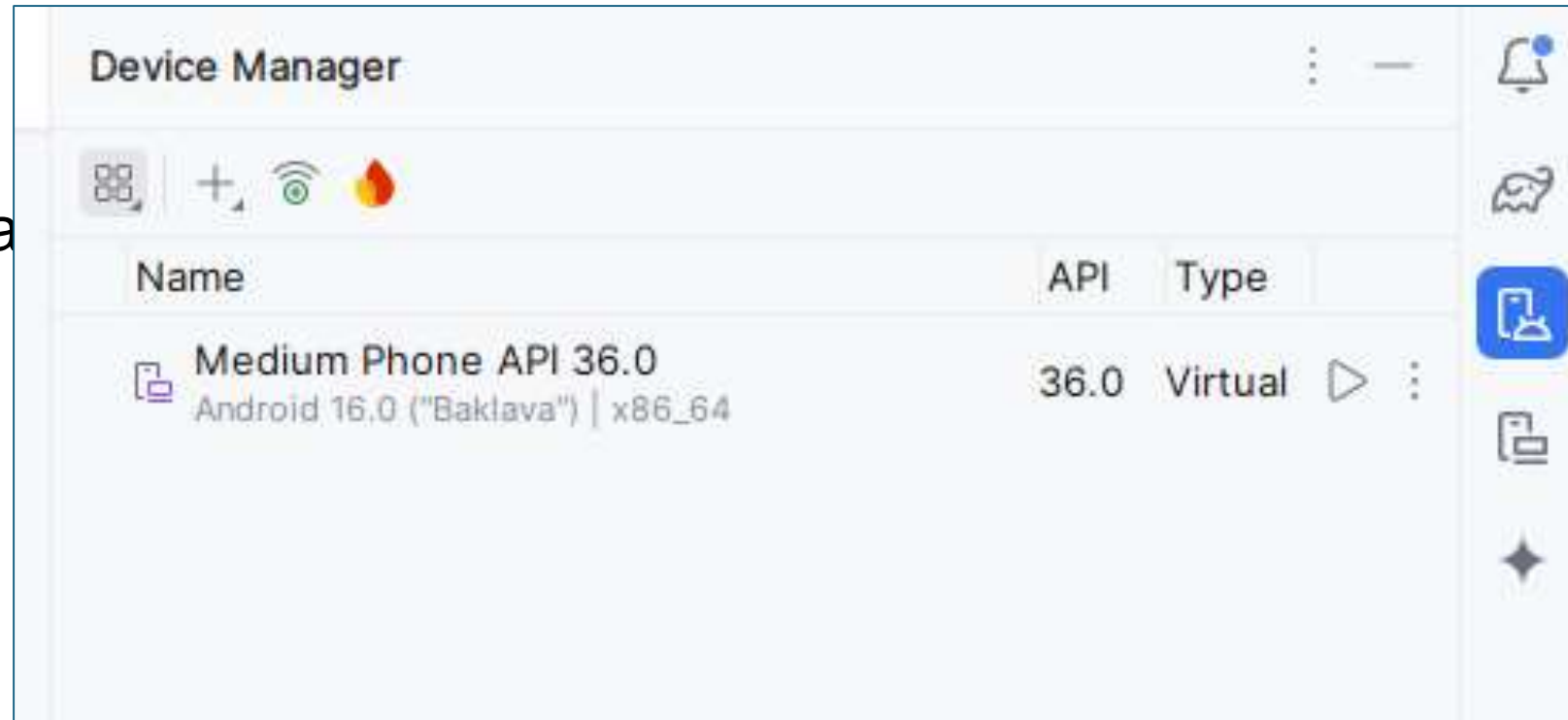
```
Android >
└─ app
   └─ manifests
   └─ kotlin+java
      └─ com.example.test
         └─ ui.theme
            └─ MainActivity.kt
            └─ com.example.test (androidTest)
            └─ com.example.test (test)
         └─ res
            └─ res (generated)
         └─ Gradle Scripts

MainActivity.k
1  import androidx.activity.enableEdgeToEdge
2  import androidx.compose.foundation.layout.fillMaxSize
3  import androidx.compose.foundation.layout.padding
4  import androidx.compose.material3.Scaffold
5  import androidx.compose.material3.Text
6  import androidx.compose.runtime.Composable
7  import androidx.compose.ui.Modifier
8  import androidx.compose.ui.tooling.preview.Preview
9  import com.example.test.ui.theme.TestTheme
10
11 class MainActivity : ComponentActivity() {
12     override fun onCreate(savedInstanceState: Bundle?) {
13         super.onCreate(savedInstanceState)
14         enableEdgeToEdge()
15         setContent {
16             TestTheme {
17                 Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
18                     Greeting(
19                         name = "Android",
20                         modifier = Modifier.padding(innerPadding)
21                     )
22                 }
23             }
24         }
25     }
26 }
27
28 @Composable
29 fun Greeting(name: String, modifier: Modifier = Modifier) {
30     Text(
31         text = "Hello $name!",
32         modifier = modifier
33     )
34 }
35
36 @Preview(showBackground = true)
37 @Composable
38 fun GreetingPreview() {
39     TestTheme {
40         Greeting(name = "Android")
41     }
42 }
```



Device Manager

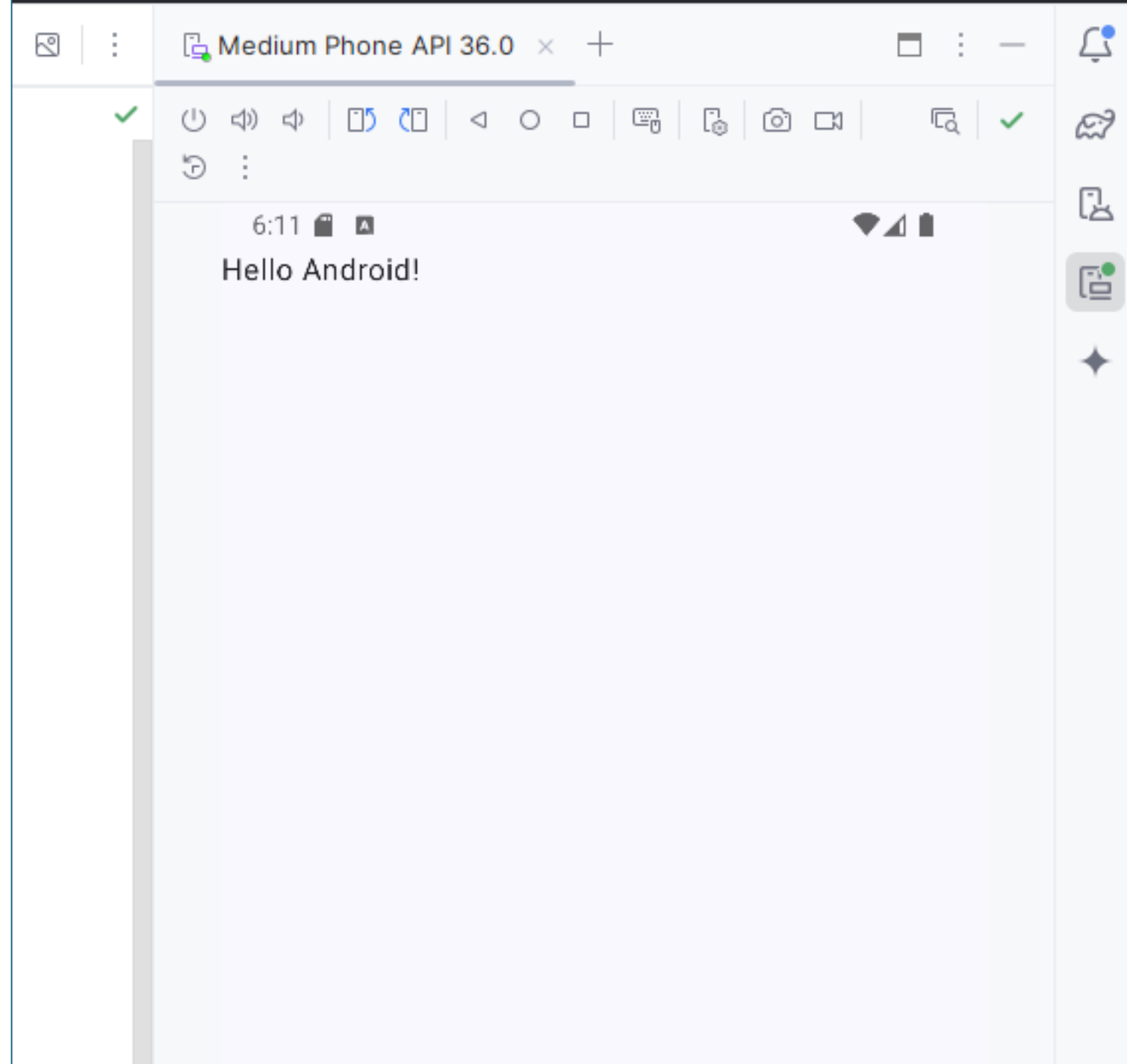
Aqui você visualiza e modifica seus dispositivos emulados (celulares).



Emulador

**Quando você roda a
aplicação**

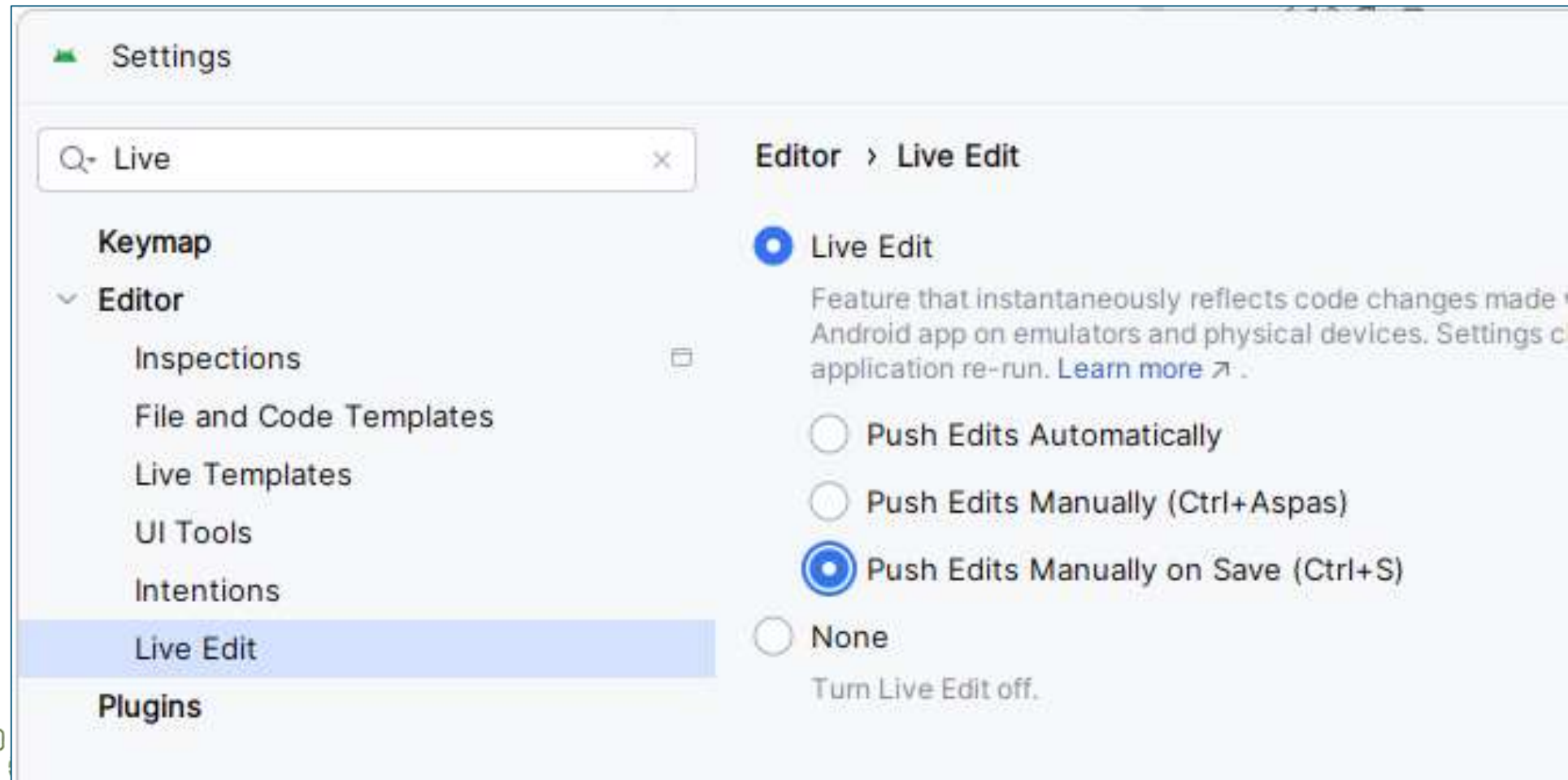
O emulador mostra como se
fosse a tela do celular



Live Edit

Vá para **Settings > Editor > Live Edit**

Modifique o refresh do seu emulador



Jetpack compose

Jetpack Compose é o novo jeito de criar interfaces no Android.

Antes: usávamos XML para o layout e Kotlin para a lógica.

Agora: com Jetpack Compose, fazemos tudo em Kotlin de forma declarativa.

Declarar = dizer o que você quer ver na tela, não como desenhar.

```
<ScrollView
  android:id="@+id/scroll_view"
  android:layout_width="match_parent"
  android:layout_height="@dp"
>

<androidx.constraintlayout.widget.ConstraintLayout
  android:id="@+id/content_layout"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
>

  <!-- Scrollable Content -->

</androidx.constraintlayout.widget.ConstraintLayout>

</ScrollView>
```

XML

```
Column(
  modifier = Modifier.verticalScroll(rememberScrollState())
) {

  // Scrollable Content

}
```

Compose

val / var

val = valor fixo (imutável) -> Use **val** quando o valor não muda.

var = variável (mutável) -> Use **var** quando o valor pode mudar depois.

Pense em val como uma etiqueta de nome e var como um balde que pode mudar de conteúdo.

```
val popcorn = 5    // There are 5 boxes of popcorn
val hotdog = 7     // There are 7 hotdogs
var customers = 10 // There are 10 customers in the queue

// Some customers leave the queue
customers = 8
println(customers)
// 8
```



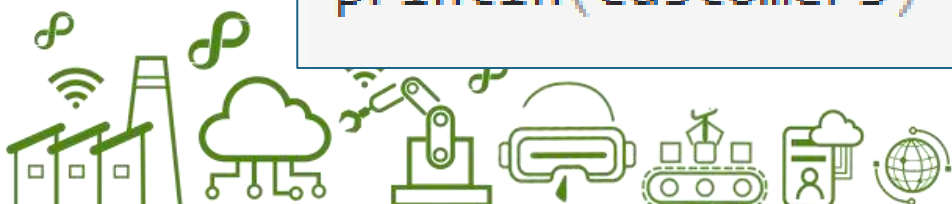
val / var

```
var customers = 10

// Some customers leave the queue
customers = 8

customers = customers + 3 // Example of addition: 11
customers += 7            // Example of addition: 18
customers -= 3            // Example of subtraction: 15
customers *= 2            // Example of multiplication: 30
customers /= 3            // Example of division: 10

println(customers) // 10
```



Tipagem



Category	Basic types	Example code
Integers	<code>Byte</code> , <code>Short</code> , <code>Int</code> , <code>Long</code>	<code>val year: Int = 2020</code>
Unsigned integers	<code>UByte</code> , <code>UShort</code> , <code>UInt</code> , <code>ULong</code>	<code>val score: UInt = 100u</code>
Floating-point numbers	<code>Float</code> , <code>Double</code>	<code>val currentTemp: Float = 24.5f</code> , <code>val price: Double = 19.99</code>
Booleans	<code>Boolean</code>	<code>val isEnabled: Boolean = true</code>
Characters	<code>Char</code>	<code>val separator: Char = ','</code>
Strings	<code>String</code>	<code>val message: String = "Hello, world!"</code>

Coleções

Lists

Sets

Maps

Tipo de coleção	Descrição
Listas	Coleções ordenadas de itens
Conjuntos	Coleções únicas e não ordenadas de itens
Mapas	Conjuntos de pares de chave-valor onde as chaves são únicas e mapeadas para apenas um valor

As listas armazenam itens na ordem em que são adicionados e permitem itens duplicados.

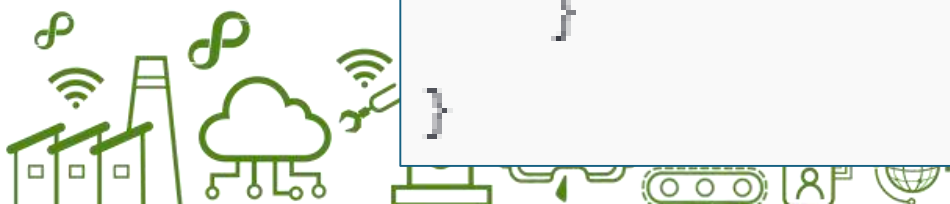
Para criar uma lista somente leitura (List), use a **listOf()**

Para criar uma lista mutável (MutableList), use a **mutableListOf()**




```
val frutas = listOf("Maçã", "Banana", "Uva")
```

```
@Composable
fun ListaDeFrutas() {
    val frutas = listOf("Maçã", "Banana", "Uva")
    Column {
        frutas.forEach { fruta ->
            Text("Fruta: $fruta")
        }
    }
}
```



Listas, Funções

As listas são ordenadas, portanto, para acessar um item em uma lista, use o operador de acesso indexado = **lista[i]**

```
val readOnlyShapes = listOf("triangle", "square", "circle")
println("The first item in the list is: ${readOnlyShapes[0]}")
// The first item in the list is: triangle
```

Para obter o número de itens em uma lista, use **.count()**

```
val readOnlyShapes = listOf("triangle", "square", "circle")
println("This list has ${readOnlyShapes.count()} items")
// This list has 3 items
```



Listas, Funções

.first() = pega o primeiro item da lista.

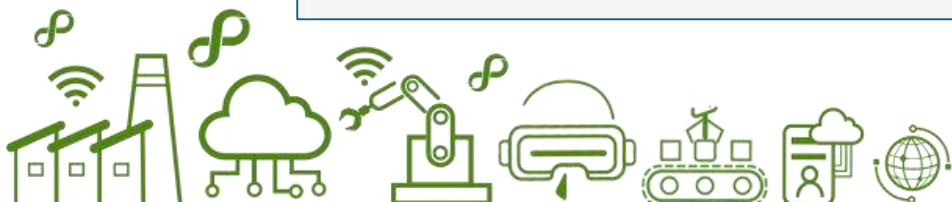
.last() = pega o último item da lista.

```
val readOnlyShapes = listOf("triangle", "square", "circle")
println("The first item in the list is: ${readOnlyShapes.first()}")
// The first item in the list is: triangle
```

Para saber se um item está ou não na lista, utilize __item__ in __lista__

Isso retorna um booleano:

```
val readOnlyShapes = listOf("triangle", "square", "circle")
println("circle" in readOnlyShapes)
// true
```



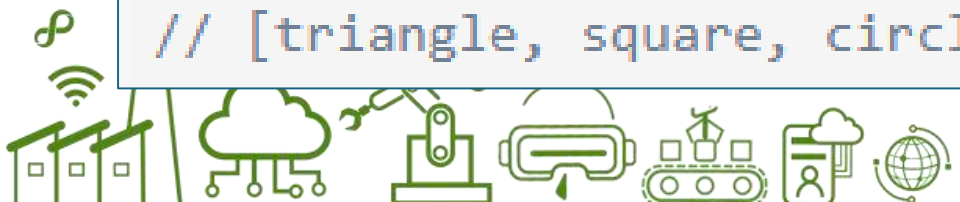
Listas, Funções

.add(item) = adiciona o item no fim da lista.

.remove(item) = deleta o item da lista. (independentemente de lugar)

```
val shapes: MutableList<String> = mutableListOf("triangle", "square", "circle")
// Add "pentagon" to the list
shapes.add("pentagon")
println(shapes)
// [triangle, square, circle, pentagon]

// Remove the first "pentagon" from the list
shapes.remove("pentagon")
println(shapes)
// [triangle, square, circle]
```



Sets: *não são ordenados e armazenam apenas itens únicos*

Para criar um conjunto somente leitura (Set), use a **setOf()**

Para criar um conjunto mutável (MutableSet), use a **mutableSetOf()**

```
// Conjunto somente leitura
val readOnlyFruit = setOf ( "maçã" , "banana" , "cereja" , "cereja" )
// Conjunto mutável com declaração de tipo explícita
val fruta : MutableSet < String > = mutableSetOf ( "maçã" , "banana" , "cer

println ( readOnlyFruit )
// [maçã, banana, cereja]
```

Perceba que o item duplicado “cereja” é descartado.

Funções: **Count(), ...in..., add(), remove()**



Sets, Funções

.count() = obter numero de itens.

```
val readOnlyFruit = setOf("apple", "banana", "cherry", "cherry")
println("This set has ${readOnlyFruit.count()} items")
// This set has 3 items
```

item in set = retorna se o item está no set ou não.

```
val readOnlyFruit = setOf("apple", "banana", "cherry", "cherry")
println("banana" in readOnlyFruit)
// true
```

add(item) = adiciona o item no fim do set.

.remove(item) = deleta o item do set.

```
val fruit: MutableSet<String> = mutableSetOf("apple", "banana", "cherry", "c
fruit.add("dragonfruit") // Add "dragonfruit" to the set
println(fruit)           // [apple, banana, cherry, dragonfruit]

fruit.remove("dragonfruit") // Remove "dragonfruit" from the set
println(fruit)             // [apple, banana, cherry]
```



Maps: *armazenam itens como pares chave-valor.*

Para criar um mapa somente leitura (Map), use **mapOf()**

Para criar um mapa mutável (MutableMap), use **mutableMapOf()**

Para declarar o tipo, adicione colchetes <> após o Map.

Por exemplo: `MutableMap<String, Int>`. *Chaves tipo String e valores tipo Int*

```
// Read-only map
val readOnlyJuiceMenu = mapOf("apple" to 100, "kiwi" to 190, "orange" to 100)
println(readOnlyJuiceMenu)
// {apple=100, kiwi=190, orange=100}

// Mutable map with explicit type declaration
val juiceMenu: MutableMap<String, Int> = mutableMapOf("apple" to 100, "kiwi" to 190, "orange" to 100)
println(juiceMenu)
// {apple=100, kiwi=190, orange=100}
```



Maps, funções

Obter o número de itens, use *mapa.count()*

Obter o valor do item, *mapa[chave]*

Obter lista de chaves, *mapa.keys()*

Obter lista de valores, *mapa.values()*

Verificar se tem um item, use o *chave/valor in mapa* (retorna boolean)

Verificar se tem um item, use *mapa.containsKey(chave)* (retorna boolean)

Adicionar item, use *mapa[nova_chave] = novo_valor*

Remover item, use *mapa.remove(chave)*



Jetpack compose - Funções e Componentes

Funções são blocos de código que podem ser reutilizados.

Em Compose, usamos funções com o **@Composable** para criar a interface (UI).

```
@Composable
fun Saudacao() {
    Text("Bem-vindo ao Jetpack Compose!")
}
```

@Preview: ver an

```
@Preview
@Composable
fun PreviewSaudacao() {
    Saudacao()
}
```



Jetpack compose – Componentes básicos

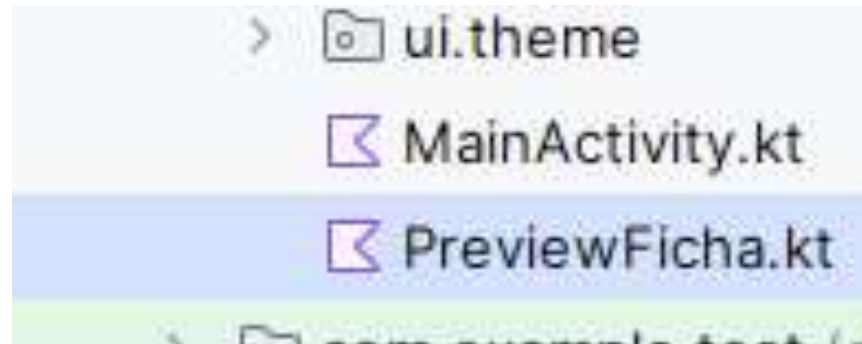
@Composable = "isso é interface", **Column** = "empilha na vertical",
Text = "mostra texto".

```
@Composable  
fun Ficha() {  
    Column {  
        Text("Nome: Ana")  
        Text("Idade: 25")  
    }  
}
```



Mão no Massa

1 – No projeto Kotlin, crie um novo arquivo “**PreviewFicha.kt**”



2 – Adicione os imports: <https://donthpad.com/NDS03/kotlin/1/1>

```
package com.example.test

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
```



3 – Adicione a Função FichaPessoal. <https://donthpad.com/NDS03/kotlin/1/2>

```
@Composable
fun FichaPessoal(nome: String, idade: Int) {
    Column(modifier = Modifier.padding(16.dp)) {
        Text("Olá, meu nome é $nome e tenho $idade anos.")
    }
}
```

4 – Adicione o **@Preview**. <https://donthpad.com/NDS03/kotlin/1/3>

Aqui chamamos a
função e passamos os
valores.

```
@Preview
@Composable
fun PreviewFicha() {
    FichaPessoal("Maria", 28)
}
```

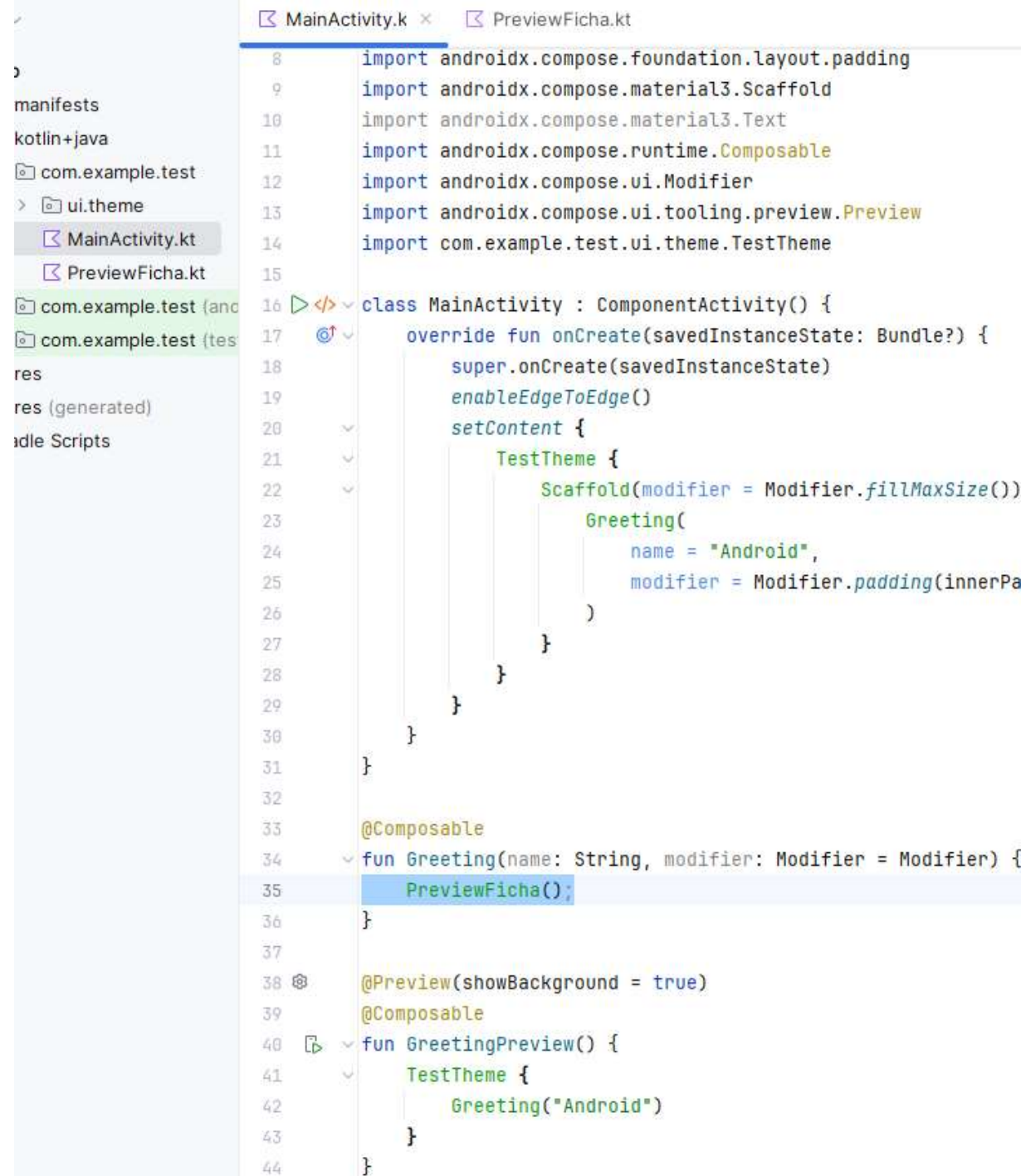


Mão no Massa

5 – Adicione na MainActivity

->

```
1 package com.example.test
2
3 import androidx.compose.foundation.layout.Column
4 import androidx.compose.foundation.layout.padding
5 import androidx.compose.material3.Text
6 import androidx.compose.runtime.Composable
7 import androidx.compose.ui.Modifier
8 import androidx.compose.ui.tooling.preview.Preview
9 import androidx.compose.ui.unit.dp
10
11 @Composable
12 fun FichaPessoal(nome: String, idade: Int) {
13     Column(modifier = Modifier.padding(16.dp)) {
14         Text("Olá, meu nome é $nome e tenho $idade anos.")
15     }
16 }
17
18
19 @Preview
20 @Composable
21 fun PreviewFicha() {
22     FichaPessoal("Maria", 28)
23 }
24
```

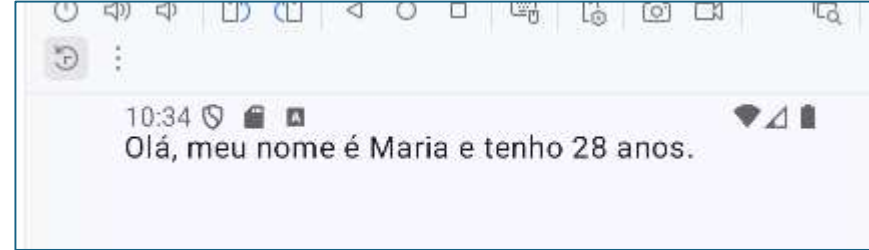


```
MainActivity.kt x PreviewFicha.kt
8 import androidx.compose.foundation.layout.padding
9 import androidx.compose.material3.Scaffold
10 import androidx.compose.material3.Text
11 import androidx.compose.runtime.Composable
12 import androidx.compose.ui.Modifier
13 import androidx.compose.ui.tooling.preview.Preview
14 import com.example.test.ui.theme.TestTheme
15
16 class MainActivity : ComponentActivity() {
17     override fun onCreate(savedInstanceState: Bundle?) {
18         super.onCreate(savedInstanceState)
19         enableEdgeToEdge()
20         setContent {
21             TestTheme {
22                 Scaffold(modifier = Modifier.fillMaxSize()) {
23                     Greeting(
24                         name = "Android",
25                         modifier = Modifier.padding(innerPa
26                     )
27                 }
28             }
29         }
30     }
31 }
32
33 @Composable
34 fun Greeting(name: String, modifier: Modifier = Modifier) {
35     PreviewFicha()
36 }
37
38 @Preview(showBackground = true)
39 @Composable
40 fun GreetingPreview() {
41     TestTheme {
42         Greeting("Android")
43     }
44 }
```

Mão no Massa

6 – Rodar o projeto

->



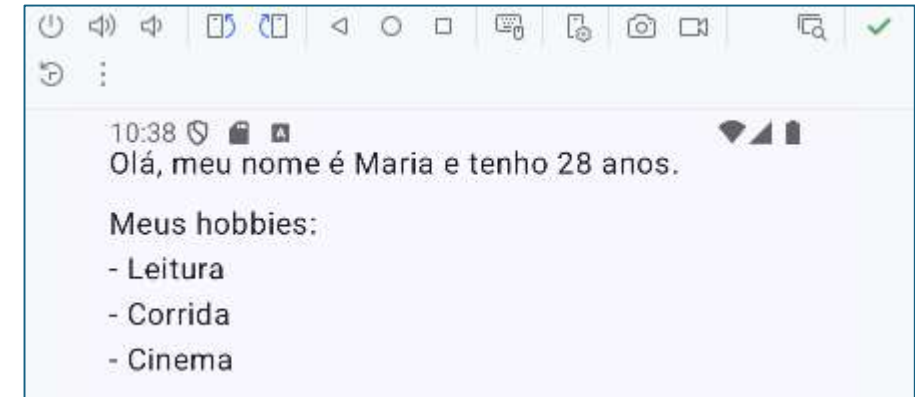
7 – Adicionar mais campos no arquivo PreviewFicha.ks

```
package com.example.test

import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp

@Composable
fun FichaPessoal(nome: String, idade: Int, hobbies: List<String>) {
    Column(modifier = Modifier.padding(16.dp)) {
        Text("Olá, meu nome é $nome e tenho $idade anos.")
        Spacer(modifier = Modifier.height(8.dp))
        Text("Meus hobbies:")
        hobbies.forEach { hobby ->
            Text("- $hobby")
        }
    }
}

@Preview
@Composable
fun PreviewFicha() {
    FichaPessoal("Maria", 28, listOf("Leitura", "Corrida", "Cinema"))
}
```



Obrigado!



    @fpftech.educacional