

FIAP

NABA



ARQUITETURA NATIVA ANDROID

DESENVOLVIMENTO AVANÇADO ANDROID
MOBILE DEVELOPMENT

MÉTODO EXTREME PROGRAMMING (XP)

O XP (*Extreme Programming*) é um método de desenvolvimento de software criado em 1997 considerado leve, não prescritivo, e que procura fundamentar as suas práticas por um conjunto de valores.

O método XP permite a troca de informações (*feedbacks*) constantes, o processo de adaptação se torna menos árduo, ou seja, é possível a aceitação de novos requisitos de usuário sem causar problemas ao projeto, como atraso na entrega ou validações de tarefas.

Em relação aos processos tradicionais, as alterações de requisitos são alvos de crítica para muitas equipes de desenvolvimento, por não permitirem adaptações rápidas.

TEST DRIVEN DEVELOPMENT

A técnica surgiu dentro da metodologia XP com o nome de Teste a Priori ou Teste Antes (*Test-first*), em alusão à proposta estranha de se testar o código antes mesmo de escrevê-lo.

O TDD é tido como uma prática central na metodologia, sendo um dos fatores que a torna viável. Isso porque, dentre outros fatores, ele ajuda a manter controlado o crescimento do custo das mudanças ao longo do projeto.

A prática do TDD mantém a solução mais facilmente modificável durante o desenvolvimento, permitindo revisões constantes e viabilizando a estratégia adaptativa como um todo.

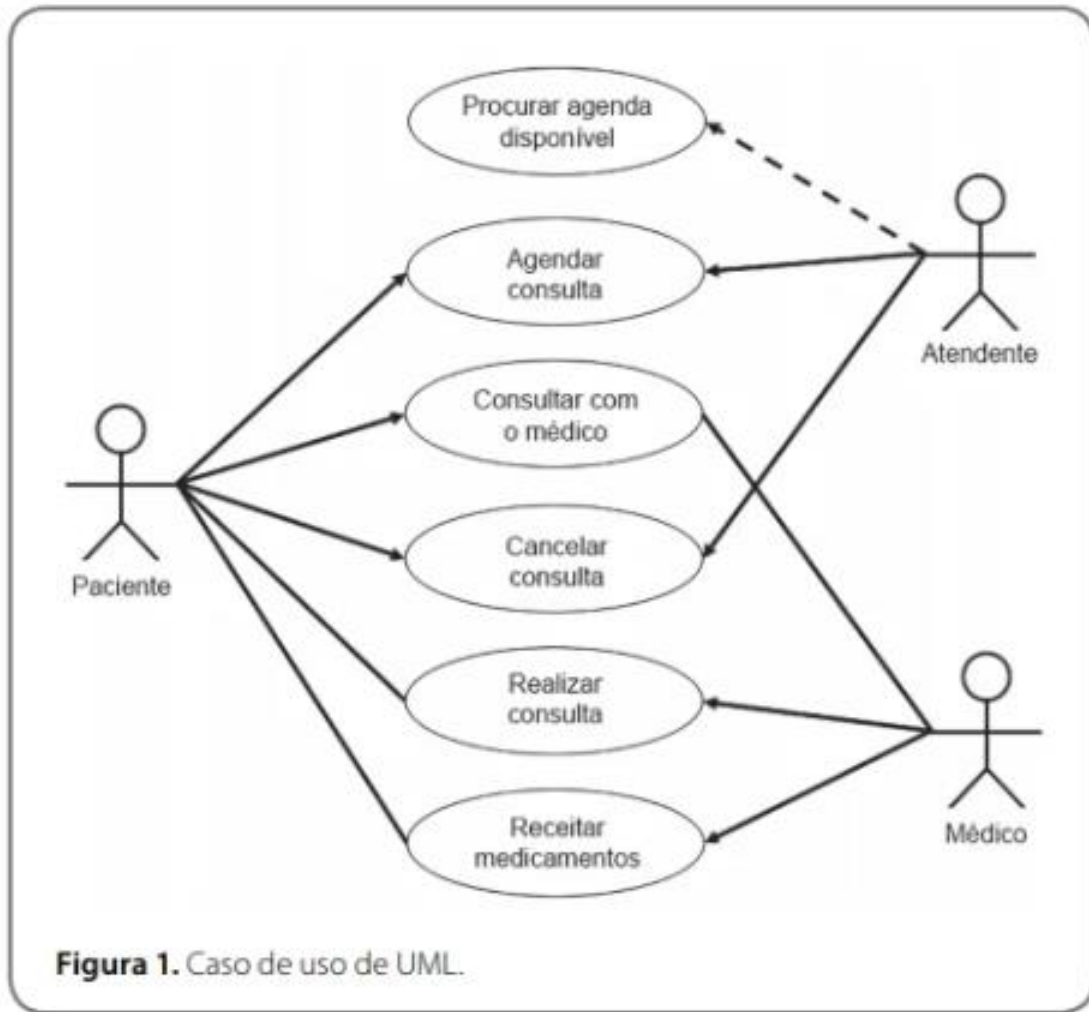
REFATORAÇÃO

É o processo de reestruturação do código de computador existente — alterando a **fatoração** — sem alterar seu comportamento externo. A refatoração visa melhorar o design, estrutura e/ou implementação do software, preservando sua funcionalidade .

As vantagens potenciais da refatoração podem incluir **melhor legibilidade** do código e **complexidade reduzida**; estes podem melhorar a **manutenibilidade** do código fonte e criar uma arquitetura interna ou **modelo de objeto mais simples, limpo ou expressivo** para melhorar a **extensibilidade** .

Outro objetivo potencial para refatoração é o **desempenho aprimorado**; engenheiros de software enfrentam um desafio contínuo para escrever programas que funcionem mais rápido ou usem menos memória.

DIAGRAMA DE CASO DE USO

**Paciente:**

agenda uma consulta;
consulta com o médico;
cancela uma consulta.

Atendente:

procura disponibilidade de data e horário na agenda;
marca consulta para o paciente;
cancela consulta a pedido do paciente.

Médico:

realiza uma consulta;
receita medicamentos.

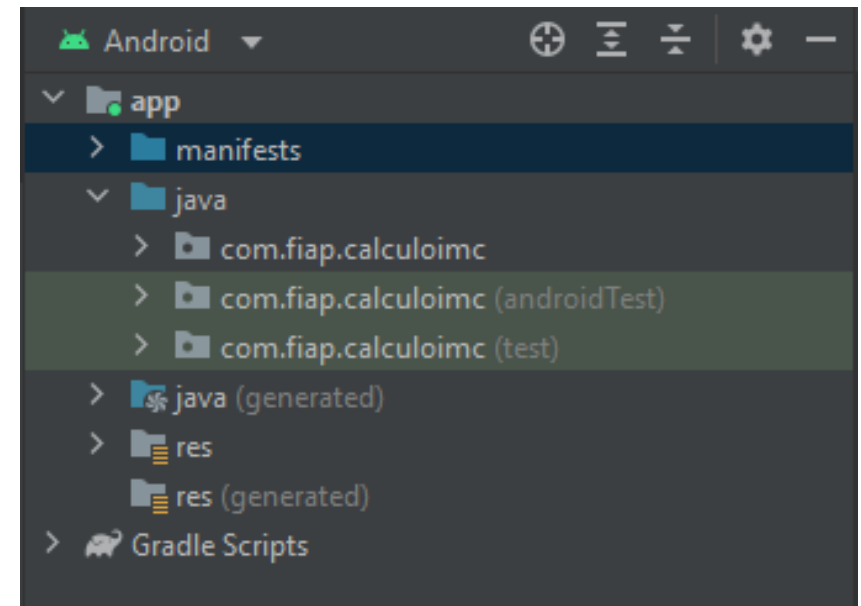
O QUE TESTAR NO ANDROID?

O que você deve testar depende de fatores como o tipo de aplicativo, a equipe de desenvolvimento, a quantidade de código legado e a arquitetura usada. As seções a seguir descrevem o que um iniciante pode querer considerar ao planejar o que testar em seu aplicativo.

Um projeto típico no Android Studio contém dois diretórios que realizam testes dependendo do ambiente de execução. Organize seus testes nos seguintes diretórios, conforme descrito:

O **androidTest** diretório deve conter os testes executados em **dispositivos reais ou virtuais**. Esses testes incluem **testes de integração**, testes de ponta a ponta e outros testes em que a JVM sozinha não pode validar a funcionalidade do seu aplicativo.

O **test** diretório deve conter os testes executados em sua **máquina local**, como **testes de unidade**. Em contraste com o acima, estes podem ser testes executados em uma JVM local



TESTES UNITÁRIOS ESSENCIAIS

Ao seguir as melhores práticas, certifique-se de usar testes de unidade nos seguintes casos:

Testes de unidade para **ViewModels** ou apresentadores.

Testes de unidade para a camada de **dados**, **especialmente repositórios**. A maior parte da camada de dados deve ser independente de plataforma. Isso permite que duplicatas de teste substituam módulos de banco de dados e fontes de dados remotas em testes. Veja o guia sobre como usar duplos de teste no Android

Testes de unidade para outras camadas independentes de plataforma, como a camada de **domínio**, como com casos de uso e interatores.

Testes de unidade para **classes utilitárias**, como manipulação de strings e matemática.

<https://developer.android.com/training/testing/fundamentals/what-to-test>

CASOS DE TESTE DE BORDA

Os testes de unidade devem se concentrar em casos normais e extremos. Casos de borda são cenários incomuns que testadores humanos e testes maiores provavelmente não detectarão. Os exemplos incluem o seguinte:

- Operações matemáticas usando números negativos, zero e condições de contorno.
- Todos os possíveis erros de conexão de rede.
- Dados corrompidos, como JSON malformatado.
- Simulando armazenamento completo ao salvar em um arquivo.
- Objeto recriado no meio de um processo (como uma atividade quando o dispositivo é girado).

ITENS A SEREM EVITADOS EM TESTES UNITÁRIOS

Alguns testes unitários devem ser evitados devido ao seu baixo valor:

- Testes que verificam a operação correta da estrutura (*framework*) ou de uma biblioteca, não do seu código.
- Os pontos de entrada da estrutura, como *atividades*, *fragmentos* ou *serviços* , não devem ter lógica de negócios, portanto, o teste de unidade não deve ser uma prioridade. Testes de unidade para atividades têm pouco valor, porque cobririam principalmente o código do framework e exigem uma configuração mais complexa. Testes instrumentados, como testes de interface do usuário, podem abranger essas classes.

DUPLO TESTES NO ANDROID

Ao projetar a estratégia de teste para um elemento ou sistema, existem três aspectos de teste relacionados:

Escopo : Quanto do código o teste toca? Os testes podem verificar um único método, todo o aplicativo ou algo intermediário. O escopo testado está *em teste* e geralmente se refere a ele como o *Assunto em Teste* , embora também o *Sistema em Teste* ou a *Unidade em Teste*.

Velocidade : Quão rápido o teste é executado? As velocidades de teste podem variar de milissegundos a vários minutos.

Fidedigno : Quão "no mundo real" é o teste? Por exemplo, se parte do código que você está testando precisar fazer uma solicitação de rede, o código de teste realmente faz essa solicitação de rede ou falsifica o resultado? Se o teste realmente falar com a rede, isso significa que ele tem maior fidelidade. A desvantagem é que o teste pode levar mais tempo para ser executado, pode resultar em erros se a rede estiver inativa ou pode ser caro de usar.

DUPLO TESTES NO ANDROID

FAKE - Um duplo de teste que tem uma implementação "funcional" da classe, mas é implementado de uma maneira que o torna bom para testes, mas inadequado para produção. Exemplo: um banco de dados na memória. As falsificações não exigem uma estrutura de simulação e são leves. Eles são preferidos .

MOCK - Um dublê de teste que se comporta como você o programa para se comportar e que tem expectativas sobre suas interações. As simulações falharão nos testes se suas interações não corresponderem aos requisitos definidos por você. Os mocks geralmente são criados com uma estrutura de mocking para conseguir tudo isso. Exemplo: Verifique se um método em um banco de dados foi chamado exatamente uma vez.

STUB - Um dublê de teste que se comporta como você o programa para se comportar, mas não tem expectativas sobre suas interações. Geralmente criado com uma estrutura de simulação. As falsificações são preferidas aos stubs por simplicidade.

DUPLO TESTES NO ANDROID

DUMMY - Um duplo de teste que é passado, mas não usado, como se você só precisasse fornecê-lo como um parâmetro.

Exemplo: uma função vazia passada como retorno de chamada de clique.

SPY - Um wrapper sobre um objeto real que também rastreia algumas informações adicionais, semelhantes a mocks. Eles geralmente são evitados por adicionar complexidade. As falsificações ou simulações são, portanto, preferidas aos espiões.

SHADOW - Falso usado em Robolectric.

CONSTRUINDO AMBIENTE DE TESTES UNITÁRIOS

Os testes de unidade geralmente são simples, mas sua configuração pode ser problemática quando a unidade em teste não é projetada com o foco em teste:

- O código que você deseja verificar precisa estar acessível a partir de um teste. Por exemplo, você não pode testar um método privado diretamente. Em vez disso, você testa a classe usando suas APIs públicas.
- Para executar testes de unidade isoladamente, as dependências da unidade em teste devem ser substituídas por componentes que você controla, como **FAKE** ou outros **TESTES DUPLOS**.

CONSTRUINDO AMBIENTE DE TESTES UNITÁRIOS

Para fazer isso, abra o arquivo do módulo do seu aplicativo **build.gradle** e especifique as seguintes bibliotecas como dependências. Use a **testImplementation** função para indicar que eles se aplicam ao conjunto de origem de teste local e não ao aplicativo:

```
dependencies {  
    // Required -- JUnit 4 framework  
    testImplementation "junit:junit:$jUnitVersion"  
    // Optional -- Robolectric environment  
    testImplementation "androidx.test:core:$androidXTestVersion"  
    // Optional -- Mockito framework  
    testImplementation "org.mockito:mockito-core:$mockitoVersion"  
    // Optional -- mockito-kotlin  
    testImplementation "org.mockito.kotlin:mockito-kotlin:$mockitoKotlinVersion"  
    // Optional -- Mockk framework  
    testImplementation "io.mockk:mockk:$mockkVersion"  
}
```

CONSTRUINDO AMBIENTE DE TESTES UNITÁRIOS

Para fazer isso, abra o arquivo do módulo do seu aplicativo **build.gradle** e especifique as seguintes bibliotecas como dependências. Use a **testImplementation** função para indicar que eles se aplicam ao conjunto de origem de teste local e não ao aplicativo:

```
dependencies {  
    // Required -- JUnit 4 framework  
    testImplementation "junit:junit:$jUnitVersion"  
    // Optional -- Robolectric environment  
    testImplementation "androidx.test:core:$androidXTestVersion"  
    // Optional -- Mockito framework  
    testImplementation "org.mockito:mockito-core:$mockitoVersion"  
    // Optional -- mockito-kotlin  
    testImplementation "org.mockito.kotlin:mockito-kotlin:$mockitoKotlinVersion"  
    // Optional -- Mockk framework  
    testImplementation "io.mockk:mockk:$mockkVersion"  
}
```


FRAMEWORKS DE TESTE

O JUnit é uma estrutura de teste de unidade para a linguagem de programação Java.

JUnit tem sido importante no desenvolvimento de desenvolvimento orientado a testes e é uma família de estruturas de teste de unidade que é coletivamente conhecida como xUnit que se originou com SUnit.



```
@RunWith(AndroidJUnit4::class.java)
@MediumTest
class MyServiceTest {
    @get:Rule
    val serviceRule = ServiceTestRule()

    @Test fun testWithStartedService() {
        serviceRule.startService(
            Intent(ApplicationProvider.getApplicationContext(),
                MyService::class.java))

        // Add your test code here.
    }

    @Test fun testWithBoundService() {
        val binder = serviceRule.bindService(
            Intent(ApplicationProvider.getApplicationContext(),
                MyService::class.java))
        val service = (binder as MyService.LocalBinder).service
        assertTrue(service.doSomethingToReturnTrue()).isTrue()
    }
}
```

FRAMEWORKS DE TESTE

O **Espresso** está testando a estrutura do google para testes de interface do usuário.

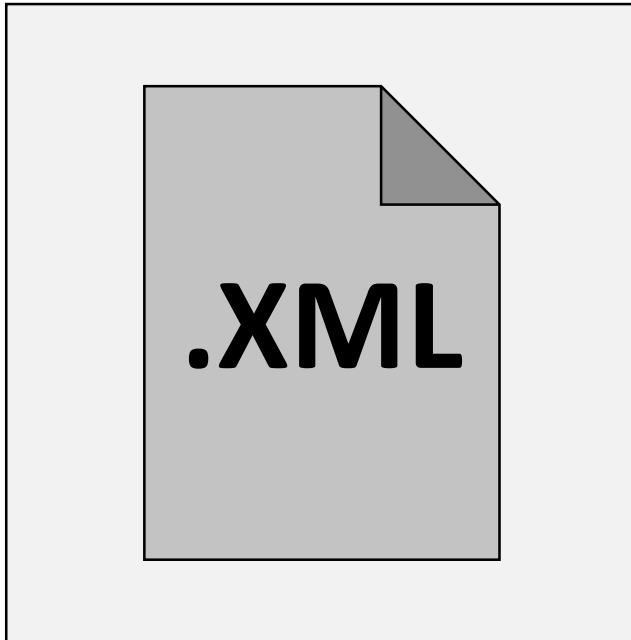
Ele fornece APIs para testes de interface do usuário em um único aplicativo.

O teste de interface do usuário garante que o usuário não tenha uma interação ruim ou encontre um comportamento inesperado.

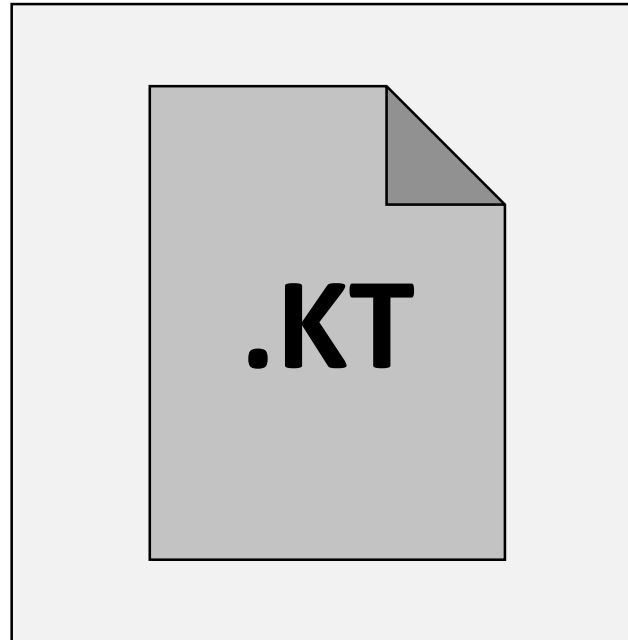


```
@Test
fun greeterSaysHello() {
    onView(withId(R.id.name_field)).perform(typeText("Steve"))
    onView(withId(R.id.greet_button)).perform(click())
    onView(withText("Hello Steve!")).check(matches(isDisplayed()))
}
```

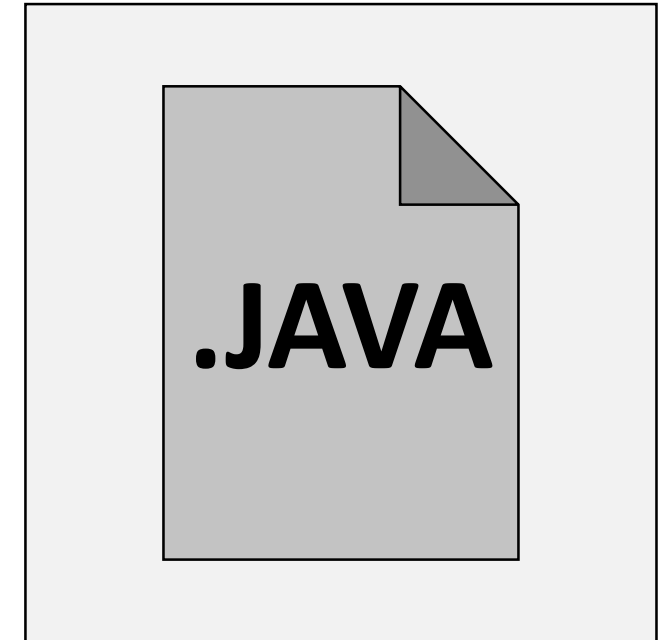
ANDROID STUDIO (EXTENSÃO DOS ARQUIVOS TESTE)



```
activity_main.xml
content_main.xml
fragment_first.xml
fragment_second.xml
```

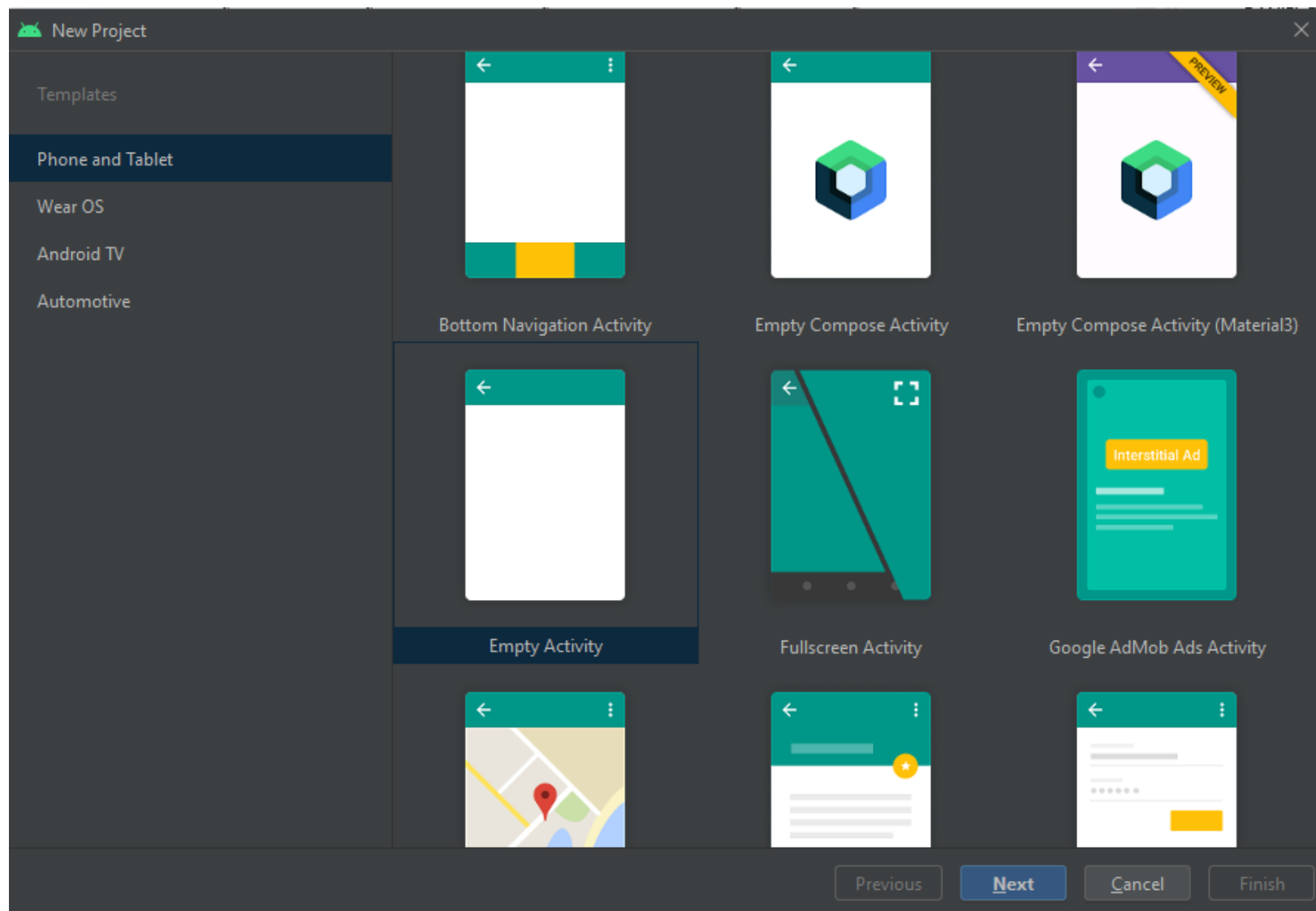


```
signOut
Task
TaskColor
TasklistFragment.kt
```

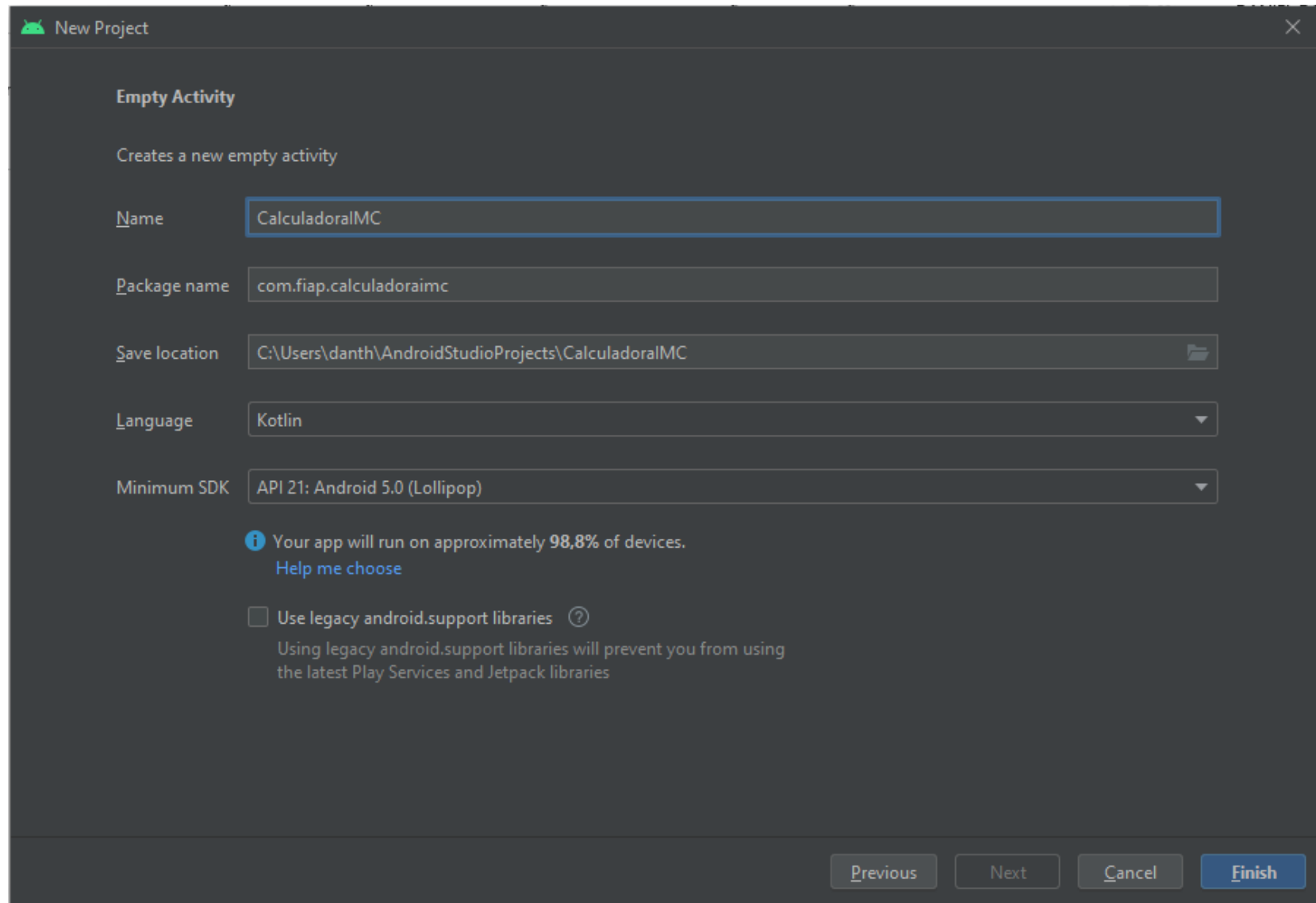


```
java
└─ com.fiap.androidtest
    └─ MainActivity
    └─ com.fiap.androidtest (androidTest)
        └─ ExampleInstrumentedTest
    └─ com.fiap.androidtest (test)
        └─ ExampleUnitTest
```

CRIAÇÃO DE UM PROJETO CALCULADORA IMC



CRIAÇÃO DE UM PROJETO CALCULADORA IMC



New Project

Empty Activity

Creates a new empty activity

Name

Package name

Save location

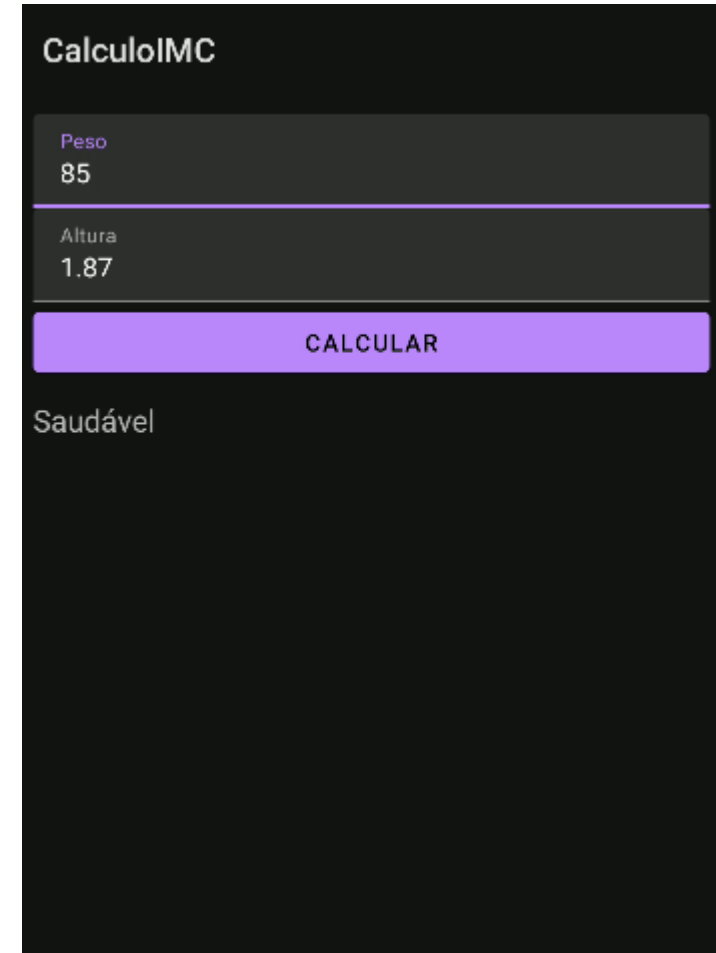
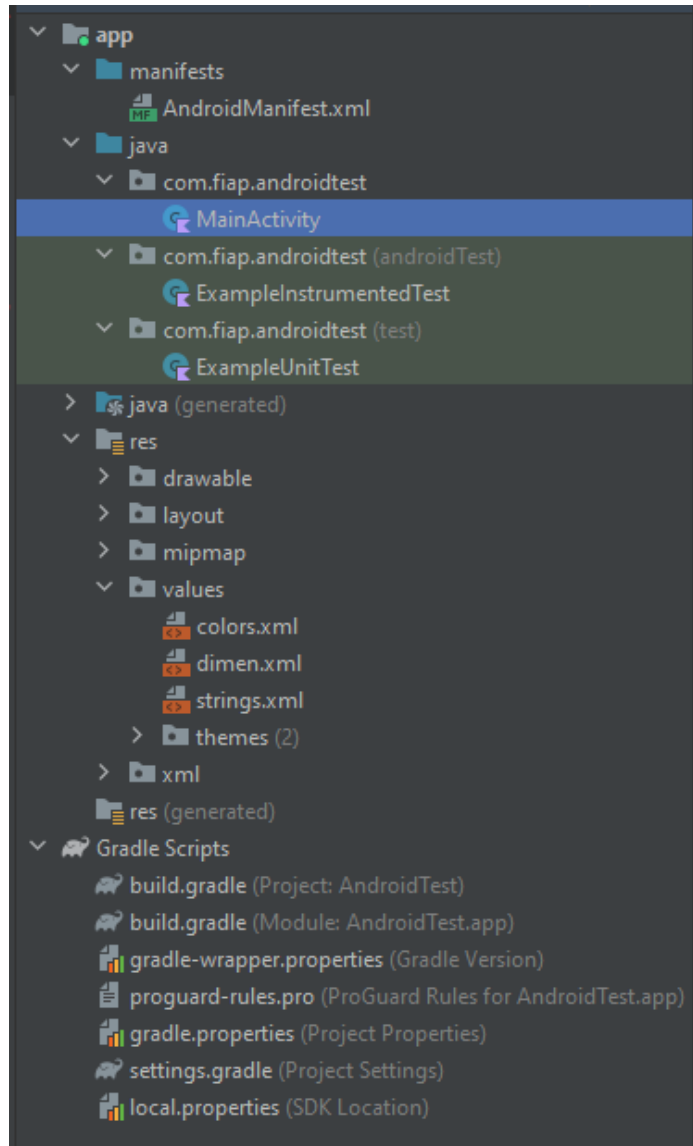
Language

Minimum SDK

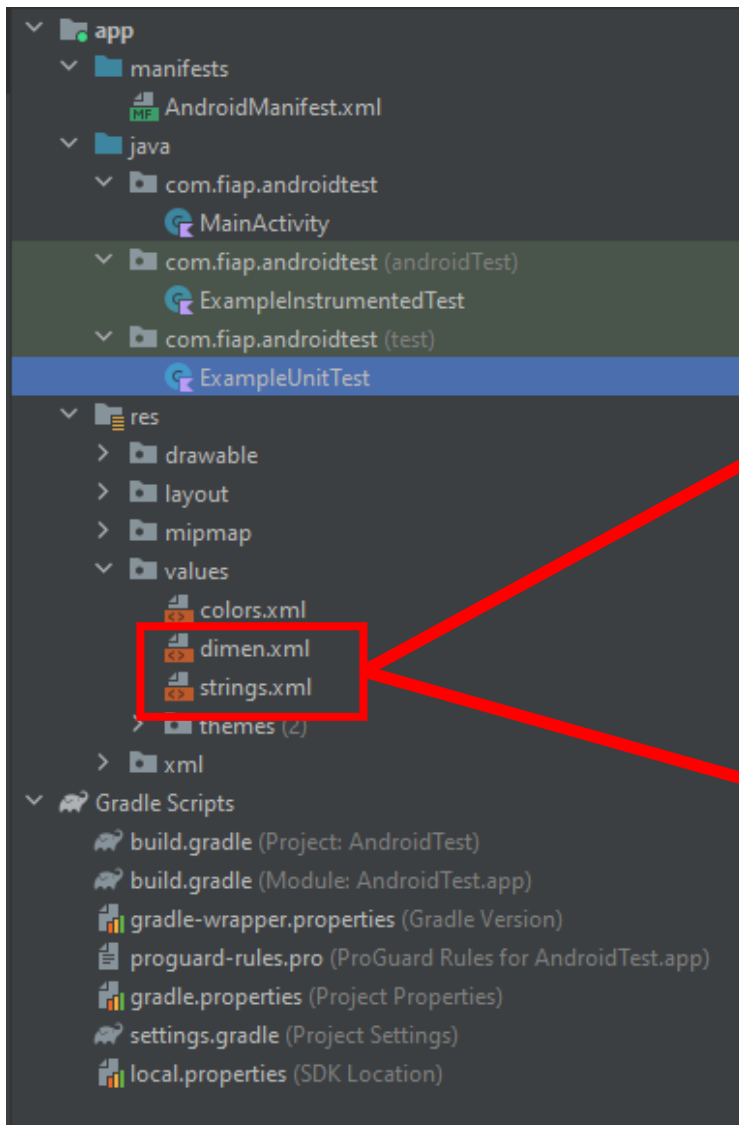
i Your app will run on approximately **98,8%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries **?**
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

TELA DO PROJETO



ARQUIVOS .XML



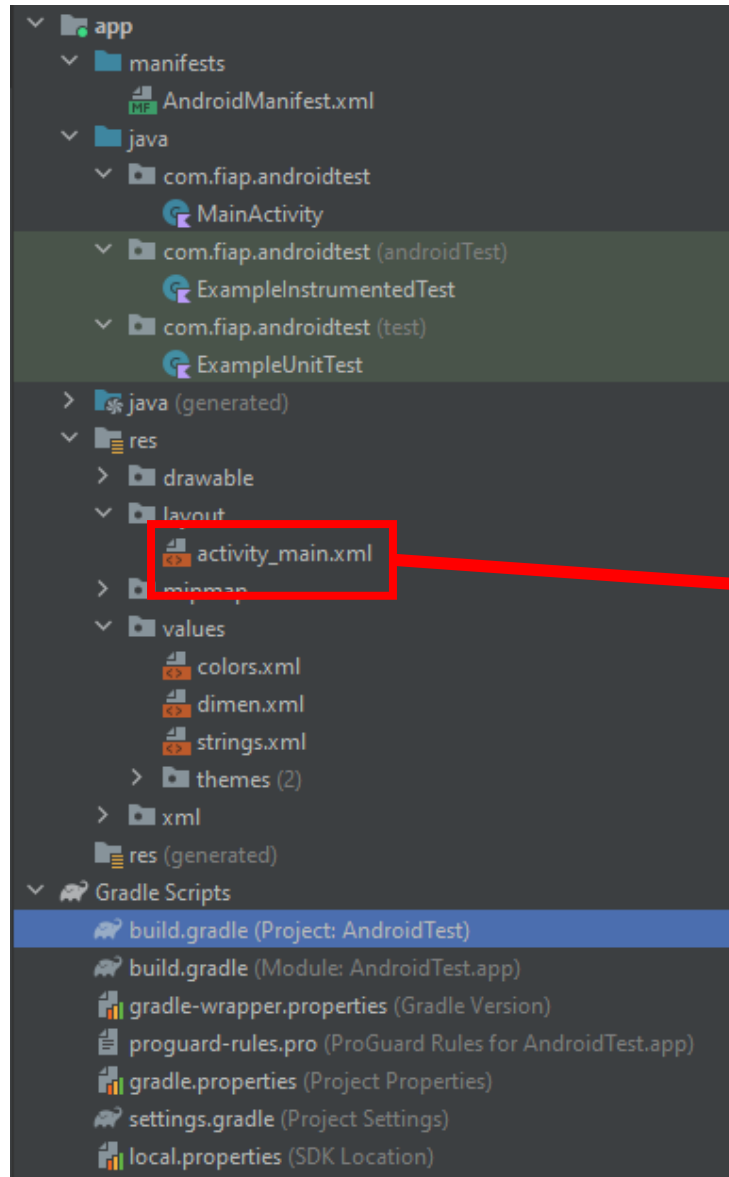
DIMEN.XML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <dimen name="padding_main_screen">10dp</dimen>
  <dimen name="label_imc">18sp</dimen>
</resources>
```

STRINGS.XML

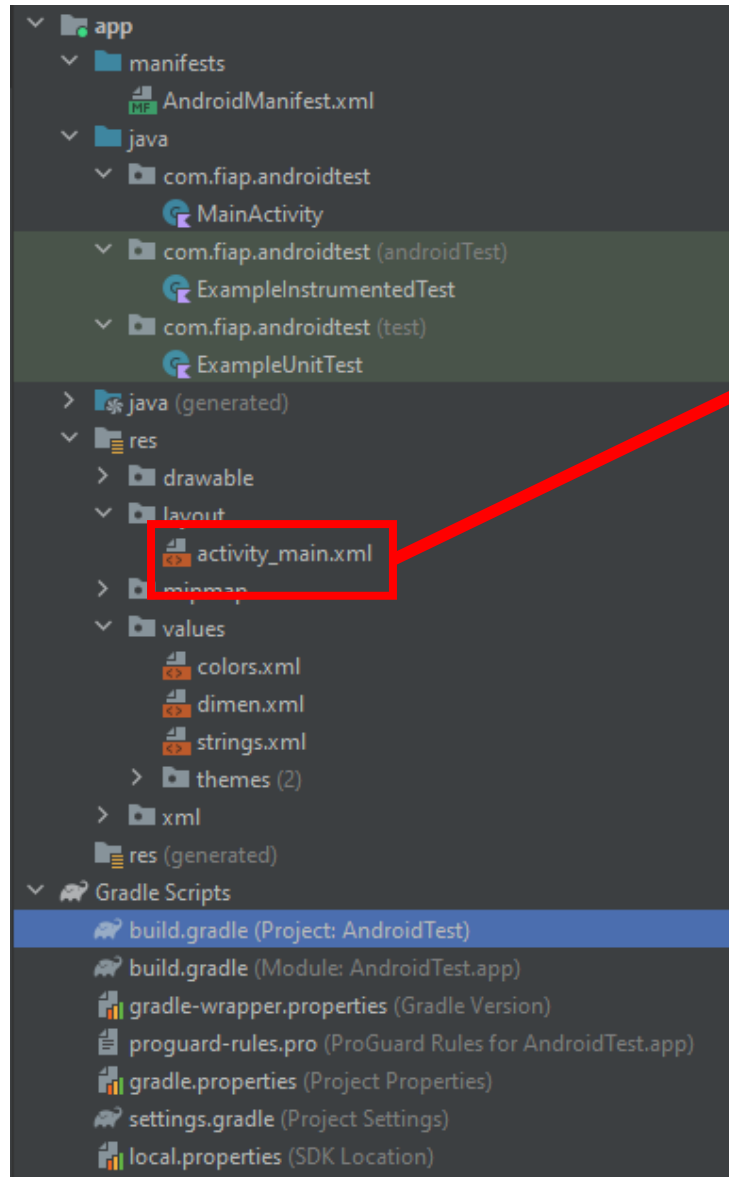
```
<resources>
  <string name="app_name">CalculoIMC</string>
  <string name="peso">Peso</string>
  <string name="altura">Altura</string>
  <string name="calcular">Calcular</string>
</resources>
```

ARQUIVOS .XML (ACTIVITY_MAIN 1/2)



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="@dimen/padding_main_screen"
    tools:context=".MainActivity">
    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/peso">
        <com.google.android.material.textfield.TextInputEditText
            android:inputType="numberDecimal"
            android:id="@+id/edt_peso"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
    </com.google.android.material.textfield.TextInputLayout>
    <com.google.android.material.textfield.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/altura">
        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/edt_altura"
            android:inputType="numberDecimal"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"/>
    </com.google.android.material.textfield.TextInputLayout>
```


ARQUIVOS .XML (ACTIVITY_MAIN 2/2)

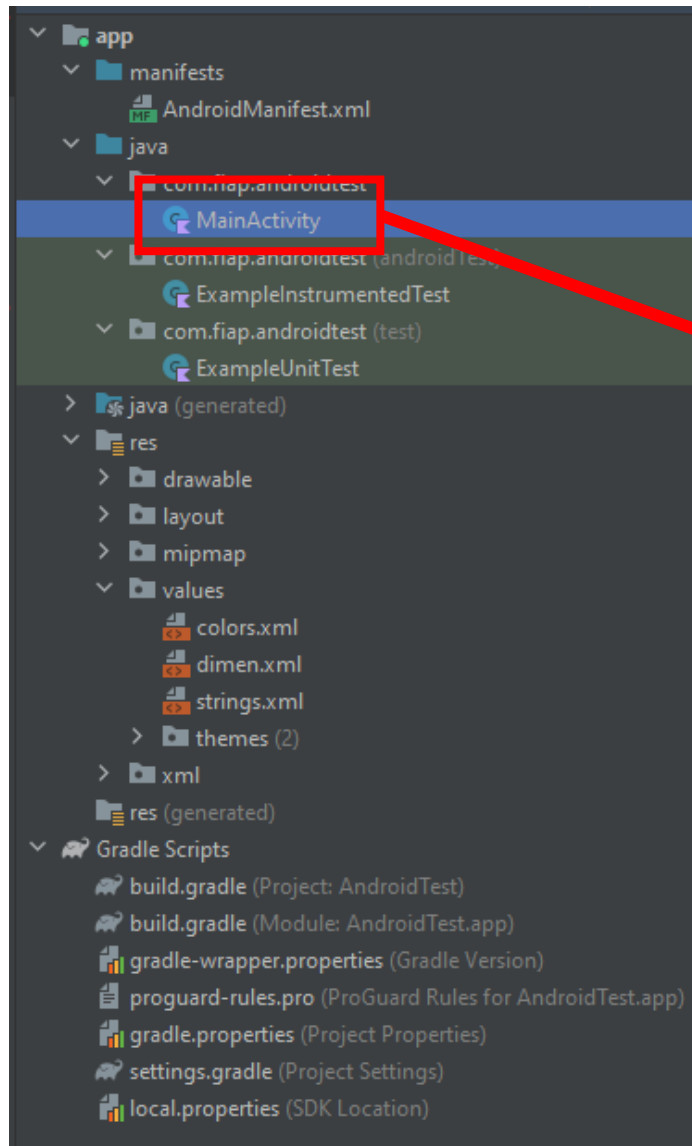


```
<Button
    android:id="@+id/btn_make_calc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/calcular"/>

<TextView
    android:textSize="@dimen/label_imc"
    android:layout_marginTop="10dp"
    android:id="@+id/txt_result_imc"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

</LinearLayout>
```

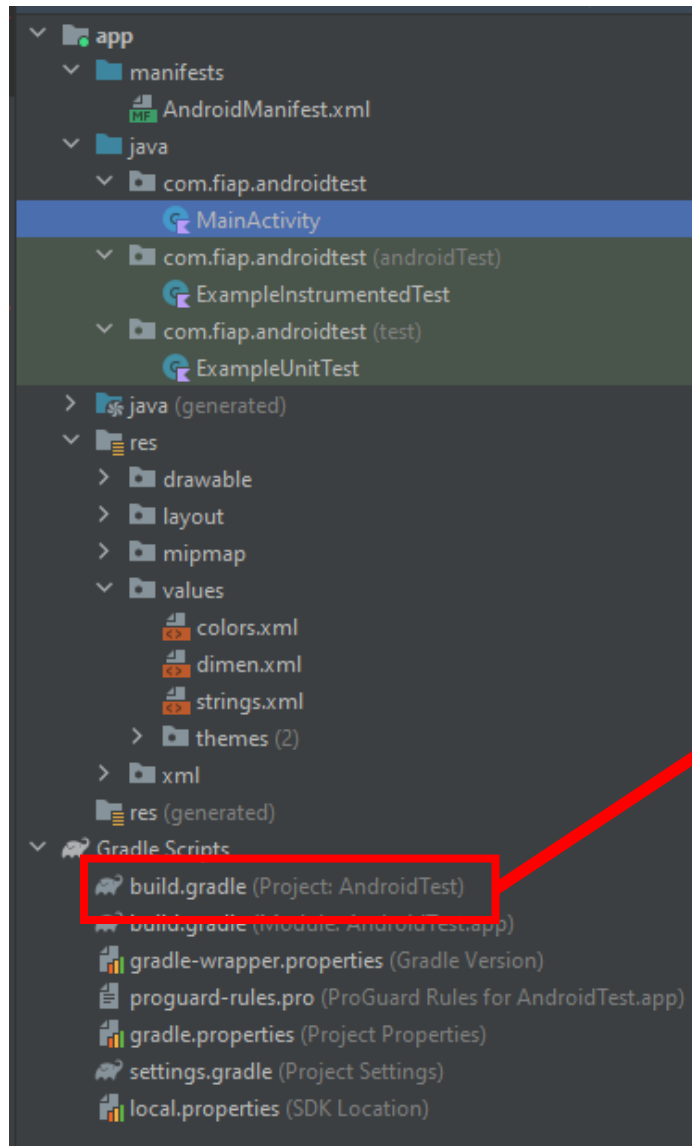
ARQUIVOS .KT



```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*
```

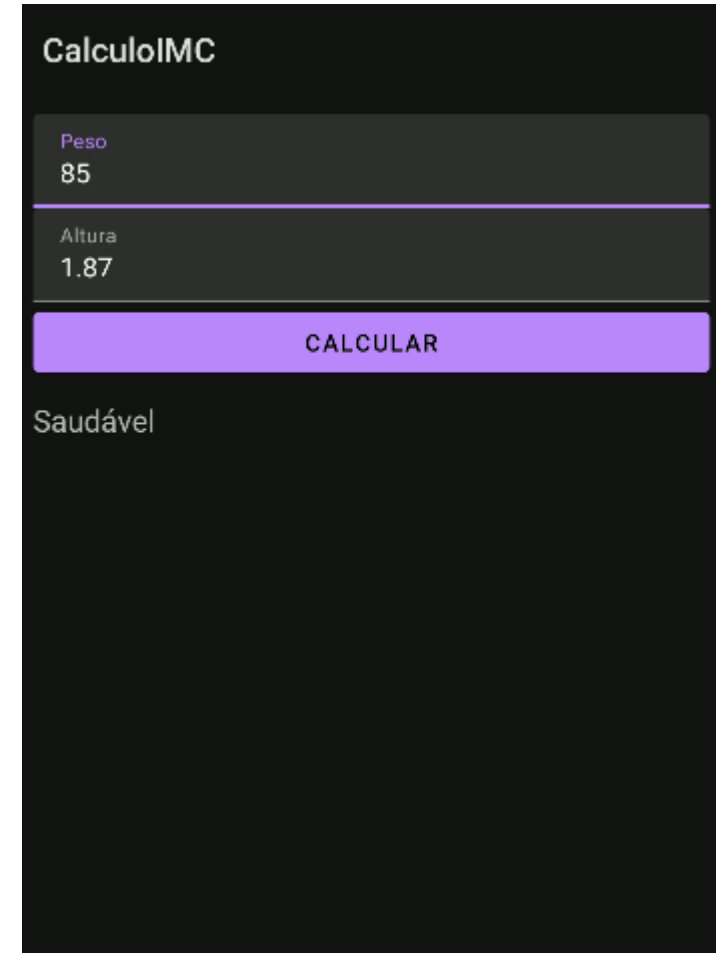
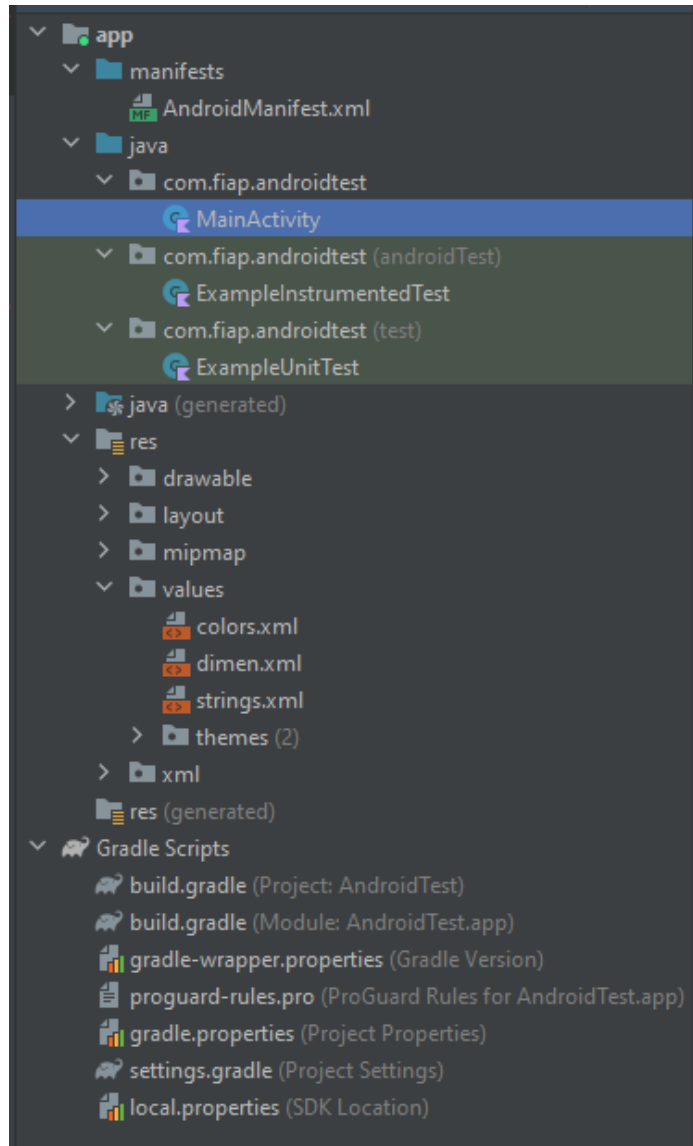
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        btn_make_calc.setOnClickListener {
            val pesoTxt = edt_peso.text.toString()
            val alturaTxt = edt_altura.text.toString()
            val peso = pesoTxt.toFloat()
            val altura = alturaTxt.toFloat()
            val imc = peso / (altura * altura)
            if (imc < 16){
                txt_result_imc.text = "Magreza grave"
            } else if (imc < 17){
                txt_result_imc.text = "Magreza moderada"
            } else if (imc < 18.5){
                txt_result_imc.text = "Magreza leve"
            } else if (imc < 25){
                txt_result_imc.text = "Saudável"
            } else if (imc < 30){
                txt_result_imc.text = "Sobrepeso"
            } else if (imc < 35){
                txt_result_imc.text = "Obesidade Grau I"
            } else if (imc < 40){
                txt_result_imc.text = "Obesidade Grau II (severa)"
            } else {
                txt_result_imc.text = "Obesidade Grau III (mórbida)"
            }
        }
    }
}
```

BUILD.GRADLE (PROJECT:ANDROIDTEST)

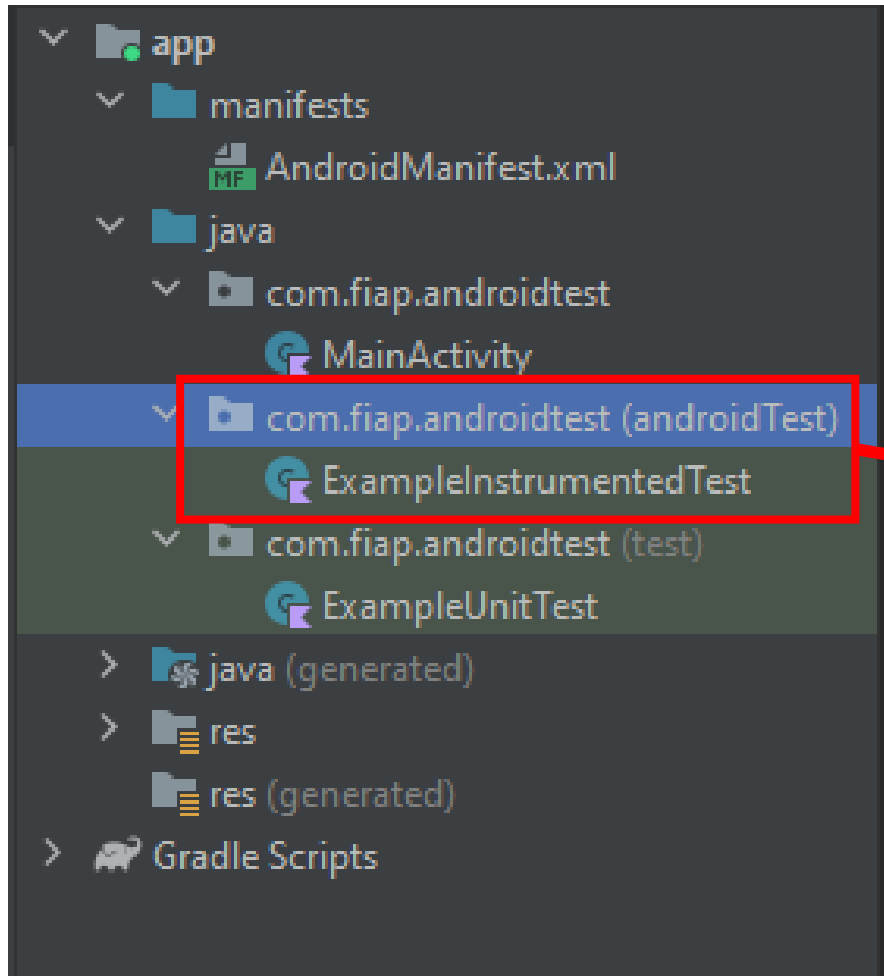


```
plugins {  
    id 'com.android.application'  
    id 'org.jetbrains.kotlin.android'  
    id 'kotlin-android-extensions'  
}
```

TELA DO PROJETO



ARQUIVOS DE TESTE



```
package com.fiap.androidtest

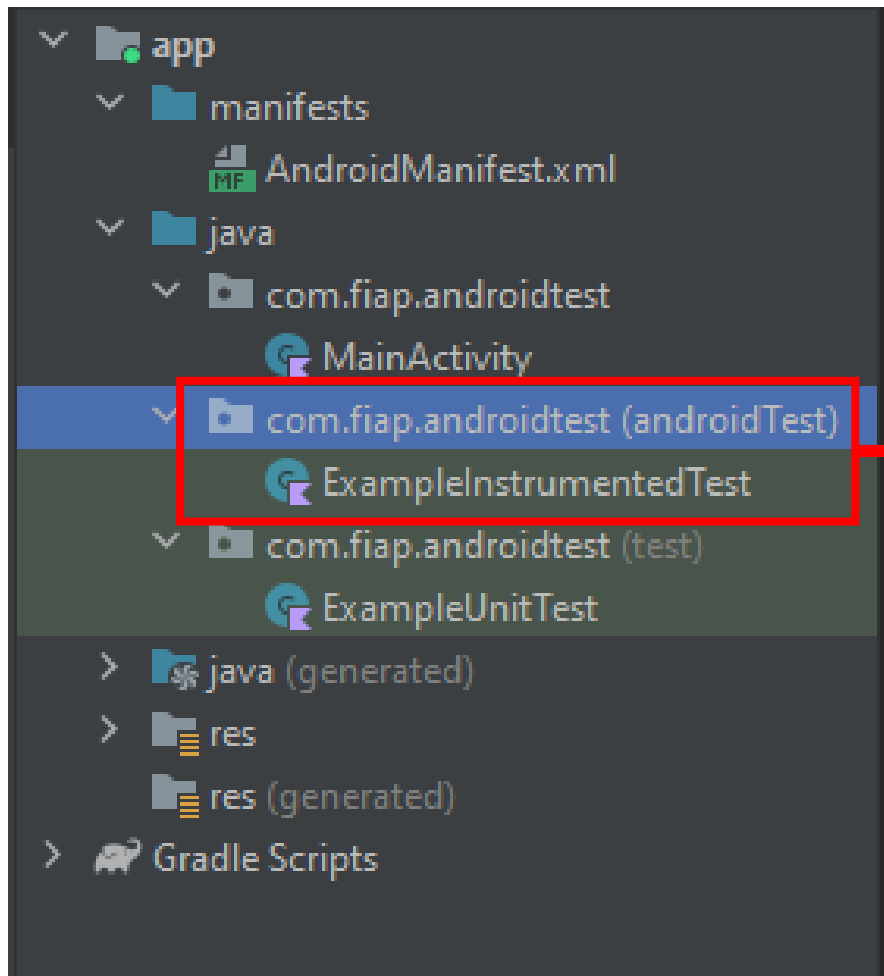
import androidx.test.platform.app.InstrumentationRegistry
import androidx.test.ext.junit.runners.AndroidJUnit4

import org.junit.Test
import org.junit.runner.RunWith

import org.junit.Assert.*

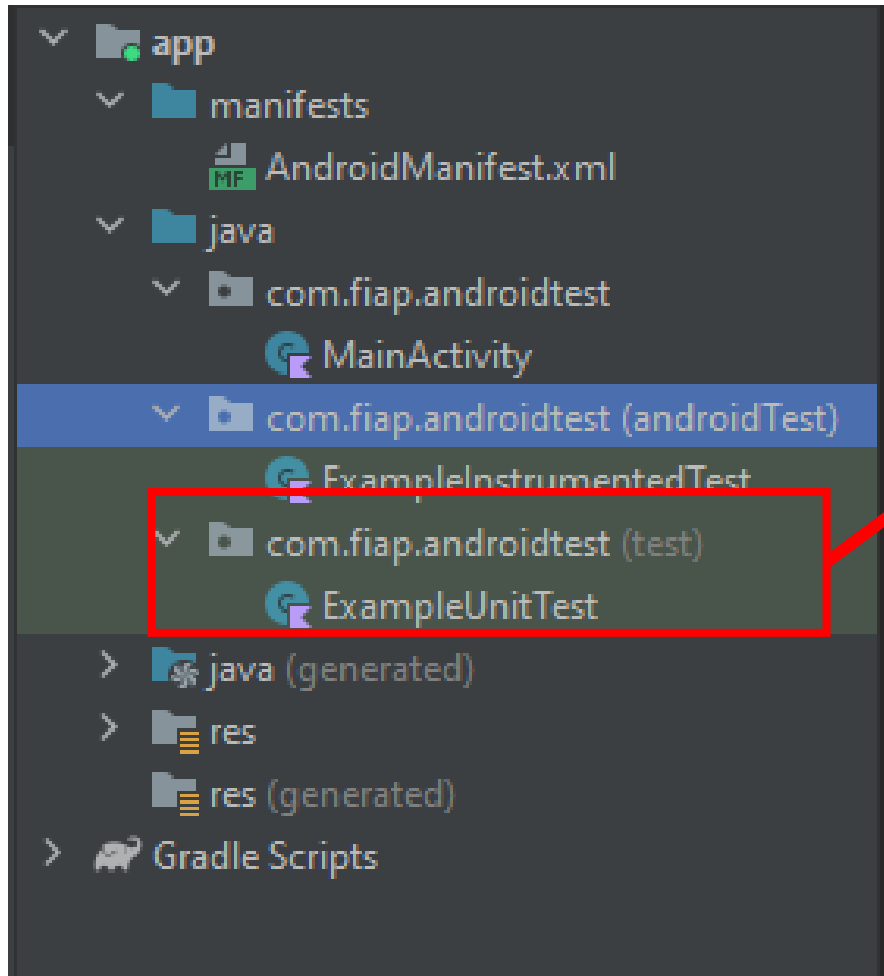
/**
 * Instrumented test, which will execute on an Android device.
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
    @Test
    fun useAppContext() {
        // Context of the app under test.
        val appContext = InstrumentationRegistry.getInstrumentation().targetContext
        assertEquals("com.fiap.androidtest", appContext.packageName)
    }
}
```

ARQUIVOS DE TESTE



O **androidTest** diretório deve conter os testes executados em **dispositivos reais ou virtuais**. Esses testes incluem **testes de integração**, testes de ponta a ponta e outros testes em que a JVM sozinha não pode validar a funcionalidade do seu aplicativo.

ARQUIVOS DE TESTE

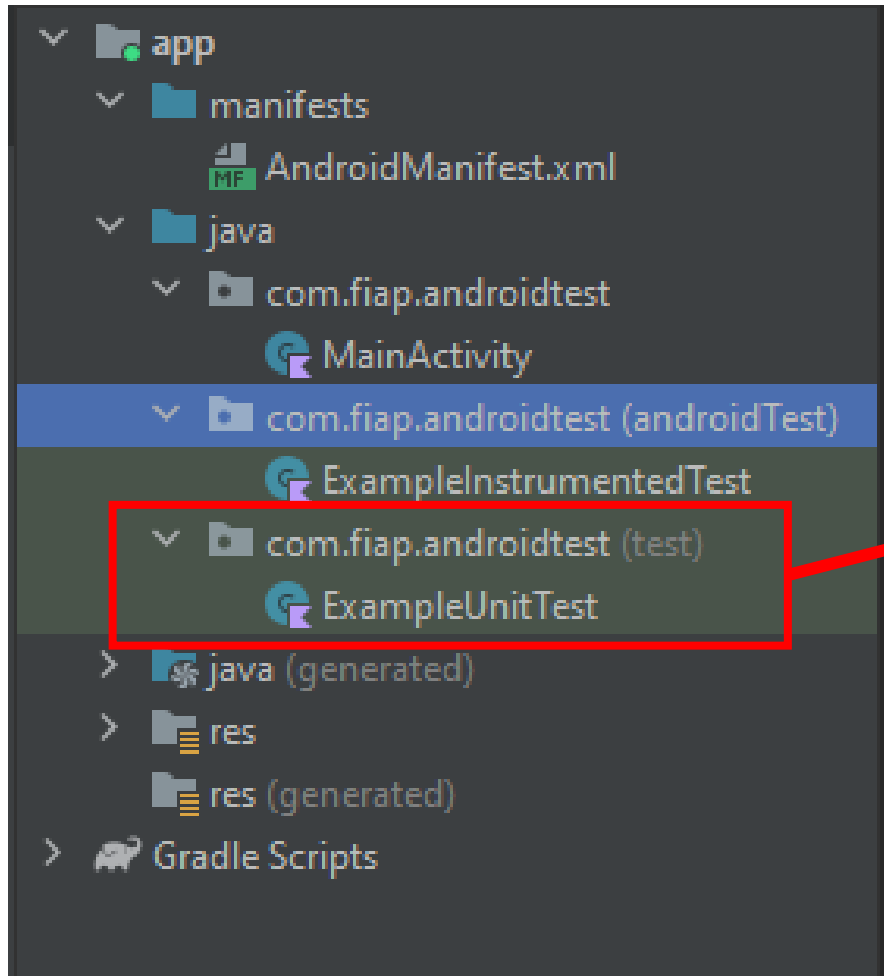


```
package com.fiap.androidtest

import org.junit.Test
import org.junit.Assert.*

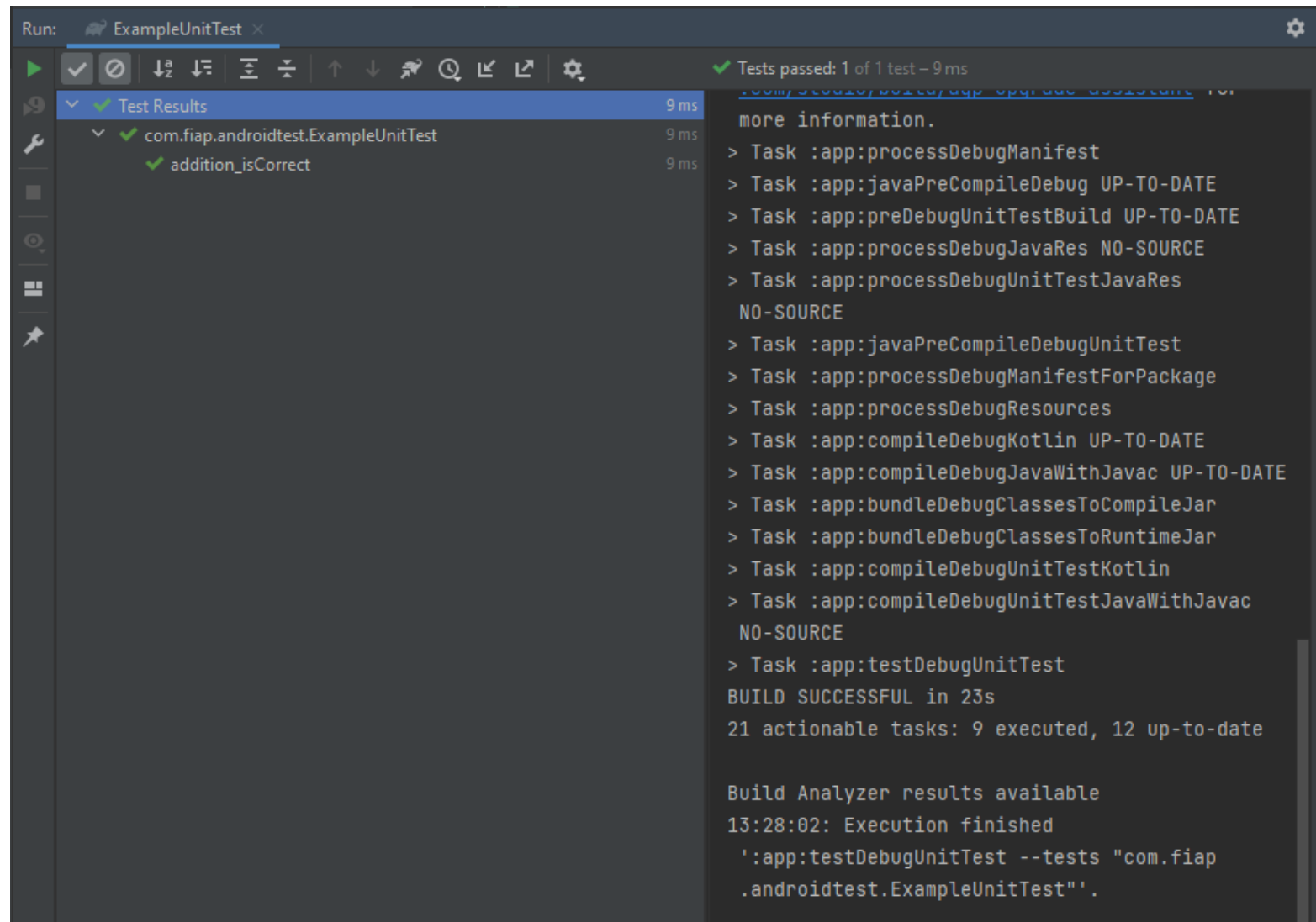
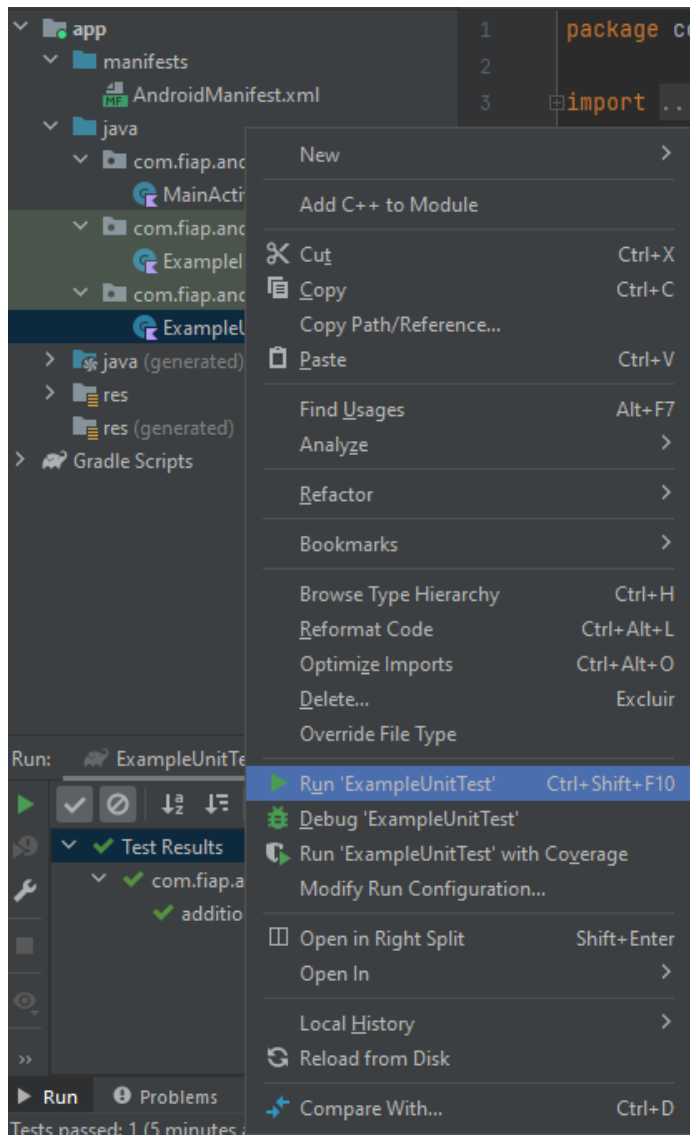
/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * See [testing documentation](http://d.android.com/tools/testing).
 */
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, 2 + 2)
    }
}
```

ARQUIVOS DE TESTE

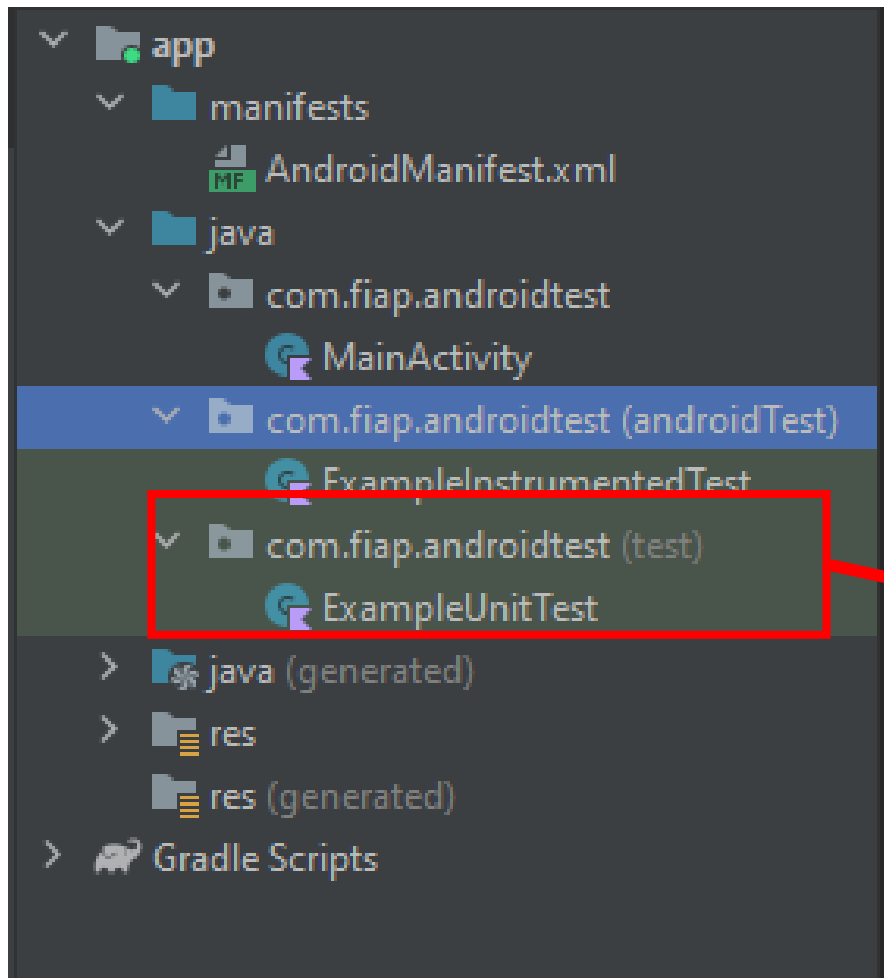


O **test** diretório deve conter os testes executados em sua **máquina local**, como **testes de unidade**. Em contraste com o acima, estes podem ser testes executados em uma JVM local

UTILIZANDO O ARQUIVO DE TESTES E O RESULTADO

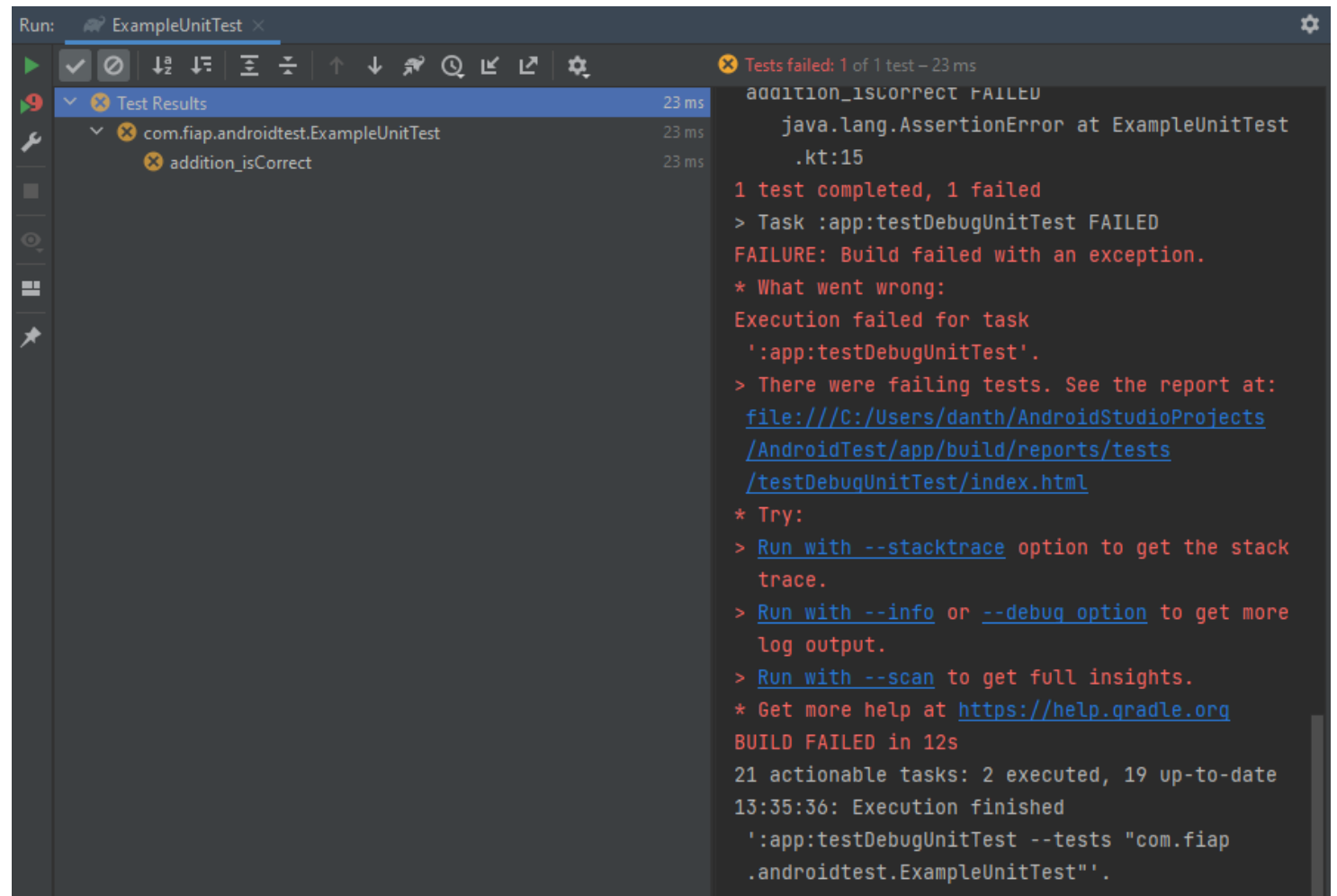
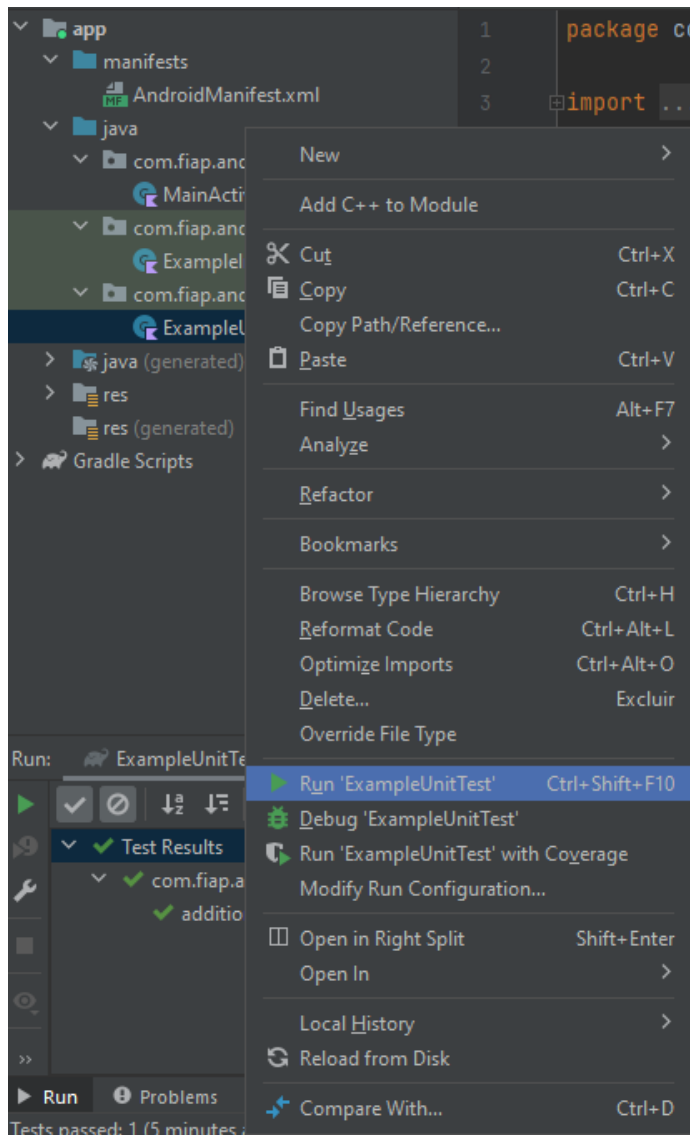


ARQUIVOS DE TESTE



```
1 package com.fiap.androidtest
2
3 import ...
4
5
6
7 /**
8  * Example local unit test, which will execute on the development machine
9  *
10  * See [testing documentation](http://d.android.com/tools/testing).
11  */
12 class ExampleUnitTest {
13     @Test
14     fun addition_isCorrect() {
15         assertEquals(expected: 4, actual: 3 + 2)
16     }
17 }
```

UTILIZANDO O ARQUIVO DE TESTES E O RESULTADO



ExampleUnitTest.kt

UTILIZANDO O ARQUIVO DE TESTES E O RESULTADO

The screenshot displays an IDE interface with a failed unit test. The top pane shows a side-by-side comparison of expected and actual values. The bottom pane shows the test results and the full exception stack trace.

Comparison Failure:

Expected	Actual
4	5

Test Results:

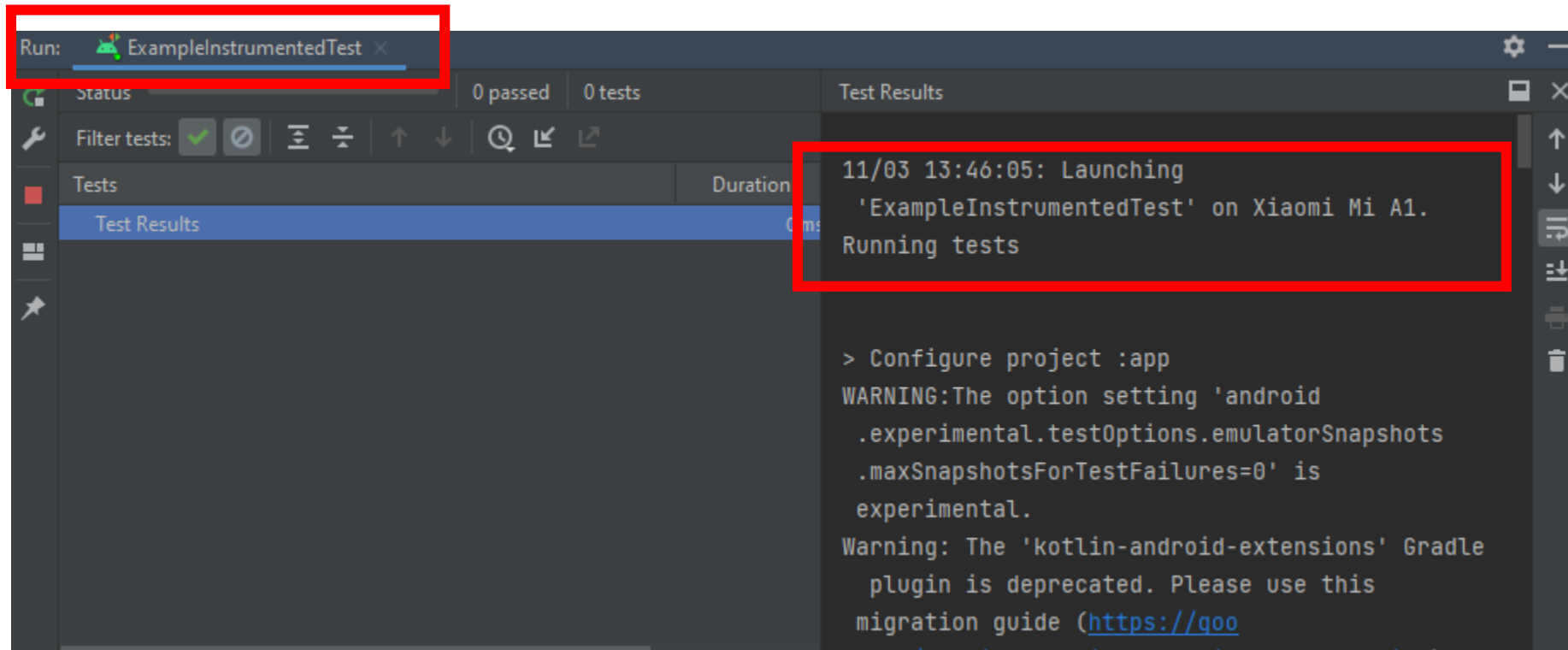
- Test Results: 23 ms
- com.fiap.androidtest.ExampleUnitTest: 23 ms
- addition_isCorrect: 23 ms

Exception Stack Trace:

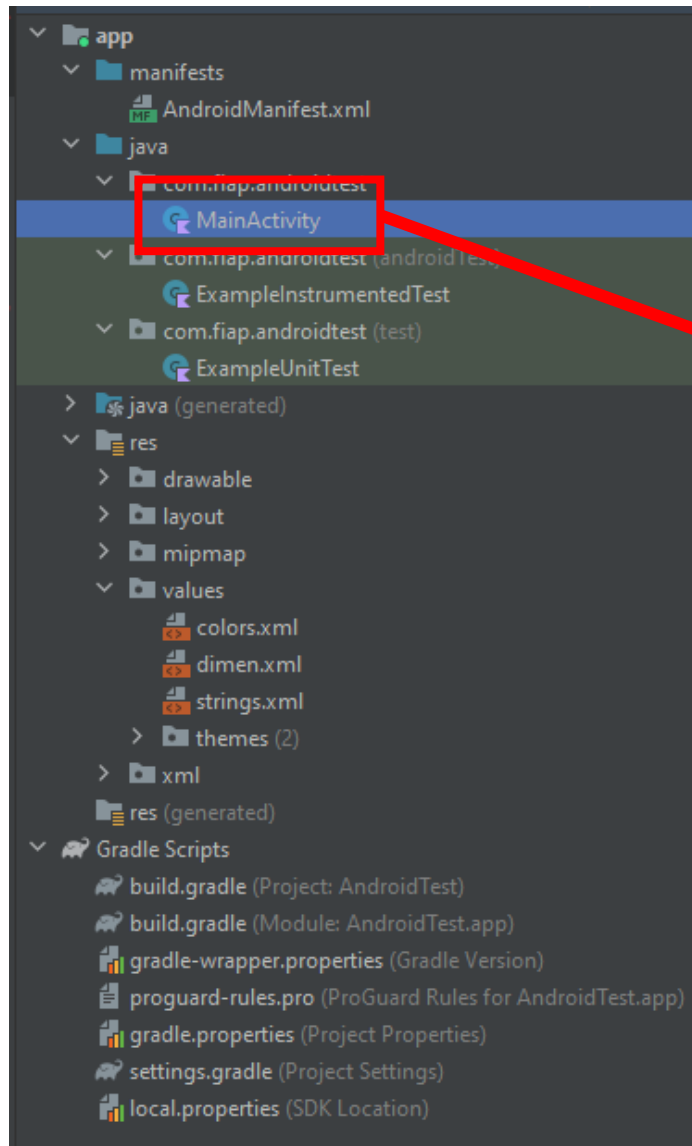
```
java.lang.AssertionError: expected:<4> but was:<5>
    at org.junit.Assert.failNotEquals(Assert.java:835)
    at com.fiap.androidtest.ExampleUnitTest.addition_isCorrect(ExampleUnitTest.kt:15)
    at worker.org.gradle.process.internal.worker.GradleWorkerMain.run(GradleWorkerMain.java:69)
    at worker.org.gradle.process.internal.worker.GradleWorkerMain.main(GradleWorkerMain.java:74)
```

UTILIZANDO O ARQUIVO DE TESTES E O RESULTADO

ExampleInstrumentedTest.kt



ARQUIVOS .KT



```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*
```

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        btn_make_calc.setOnClickListener {
            val pesoTxt = edt_peso.text.toString()
            val alturaTxt = edt_altura.text.toString()
            val peso = pesoTxt.toFloat()
            val altura = alturaTxt.toFloat()
            val imc = peso / (altura * altura)
            if (imc < 16){
                txt_result_imc.text = "Magreza grave"
            } else if (imc < 17){
                txt_result_imc.text = "Magreza moderada"
            } else if (imc < 18.5){
                txt_result_imc.text = "Magreza leve"
            } else if (imc < 25){
                txt_result_imc.text = "Saudável"
            } else if (imc < 30){
                txt_result_imc.text = "Sobrepeso"
            } else if (imc < 35){
                txt_result_imc.text = "Obesidade Grau I"
            } else if (imc < 40){
                txt_result_imc.text = "Obesidade Grau II (severa)"
            } else {
                txt_result_imc.text = "Obesidade Grau III (mórbida)"
            }
        }
    }
}
```

PRINCÍPIO DE COESÃO ALTA E ACOPLAMENTO BAIXO

Princípio de coesão alta e acoplamento baixo

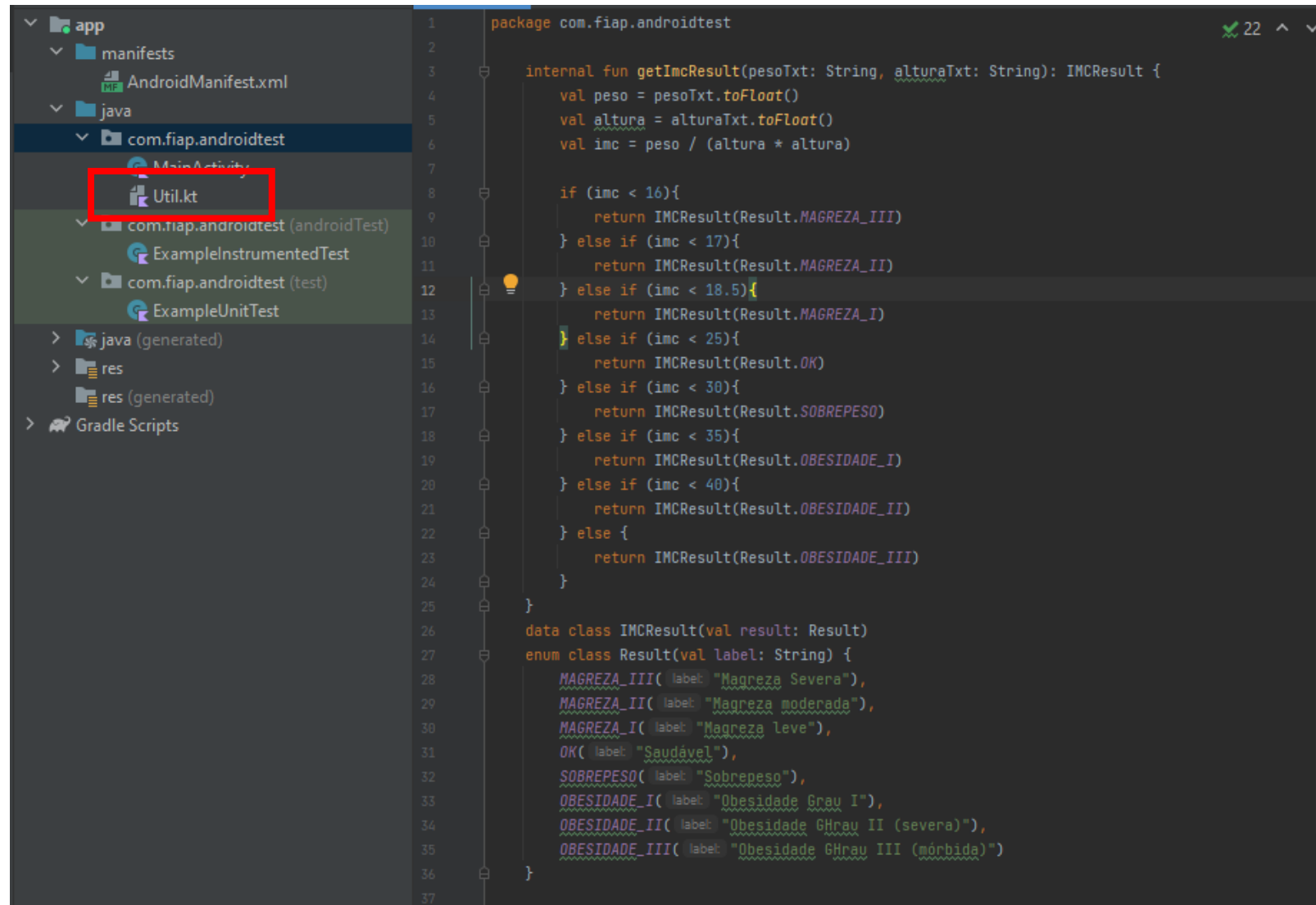
Uma maneira de caracterizar uma base de código modular seria usando as propriedades de **acoplamento** e **coesão**. O **acoplamento** mede o grau de dependência dos módulos entre si. Nesse contexto, a **coesão** mede como os elementos de um único módulo são funcionalmente relacionados. Como regra geral, você precisa se esforçar para ter um **acoplamento baixo e uma coesão alta**:

O **acoplamento baixo** significa que os módulos precisam ter o máximo de independência possível, de modo que as mudanças em um módulo tenham zero ou pouco impacto nos outros. Os módulos não devem ter conhecimento sobre o funcionamento interno uns dos outros.

Coesão alta significa que os módulos têm que abranger um conjunto de códigos que atua como um sistema. Eles precisam ter responsabilidades definidas de forma clara e ficar dentro dos limites de um determinado conhecimento do domínio. Vamos usar um aplicativo de e-book como exemplo. Pode ser inadequado combinar códigos relacionados a livros e pagamentos no mesmo módulo desse app, porque esses são dois domínios funcionais diferentes.

Dica: se dois módulos dependerem muito do conhecimento um do outro, pode ser um sinal de que eles precisam atuar como um único sistema. Por outro lado, se duas partes de um módulo não interagem entre si com frequência, é provável que elas precisem ser módulos separados.

CRIAR ARQUIVO UTIL.KT



CRIAR ARQUIVO UTIL.KT (1/2)

```
package com.fiap.androidtest

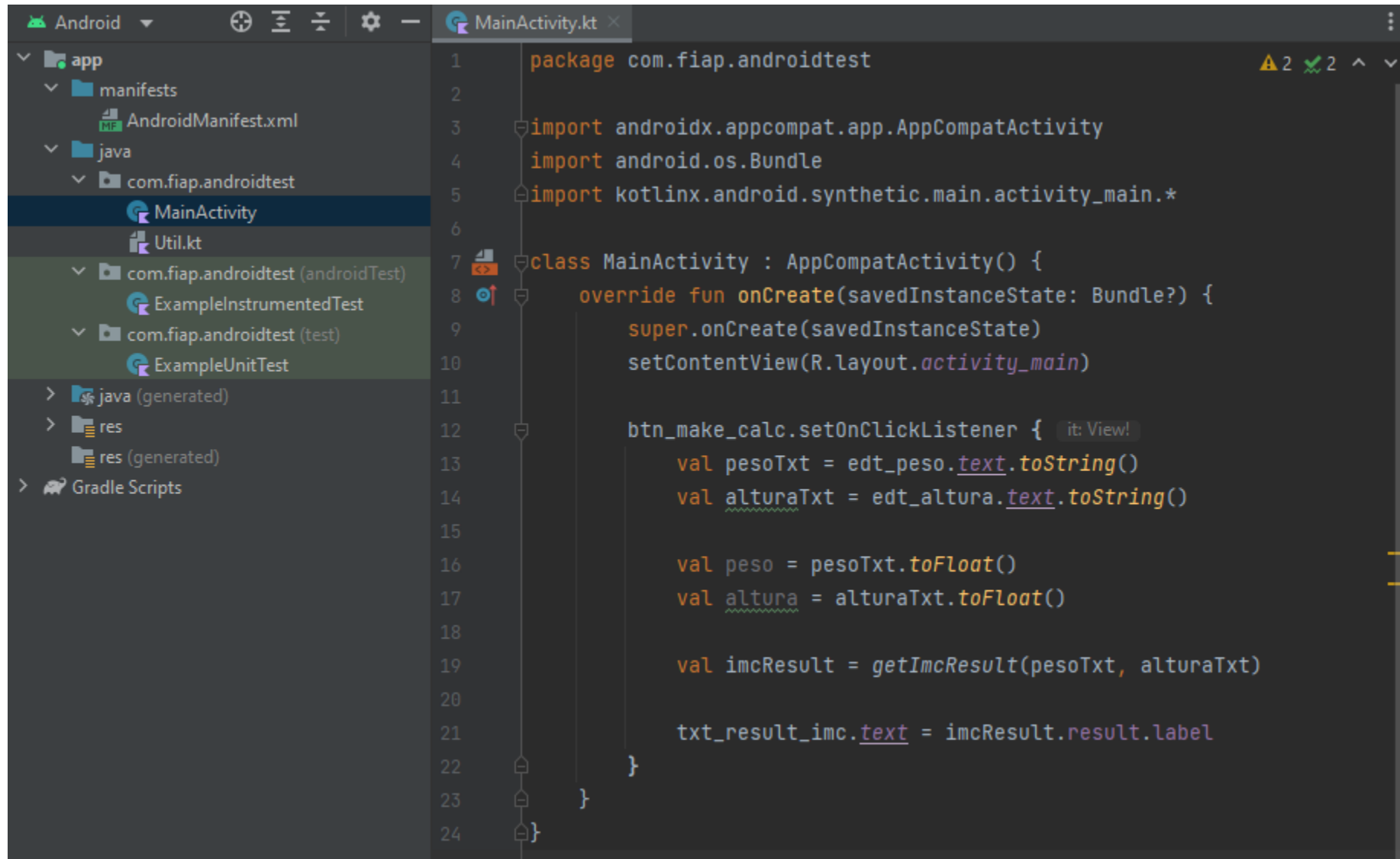
internal fun getImcResult(pesoTxt: String, alturaTxt: String): IMCResult {
    val peso = pesoTxt.toFloat()
    val altura = alturaTxt.toFloat()
    val imc = peso / (altura * altura)

    if (imc < 16){
        return IMCResult(Result.MAGREZA_III)
    } else if (imc < 17){
        return IMCResult(Result.MAGREZA_II)
    } else if (imc < 18.5){
        return IMCResult(Result.MAGREZA_I)
    } else if (imc < 25){
        return IMCResult(Result.OK)
    } else if (imc < 30){
        return IMCResult(Result.SOBREPESO)
    } else if (imc < 35){
        return IMCResult(Result.OBESIDADE_I)
    } else if (imc < 40){
        return IMCResult(Result.OBESIDADE_II)
    } else {
        return IMCResult(Result.OBESIDADE_III)
    }
} ....
```

CRIAR ARQUIVO UTIL.KT (2/2)

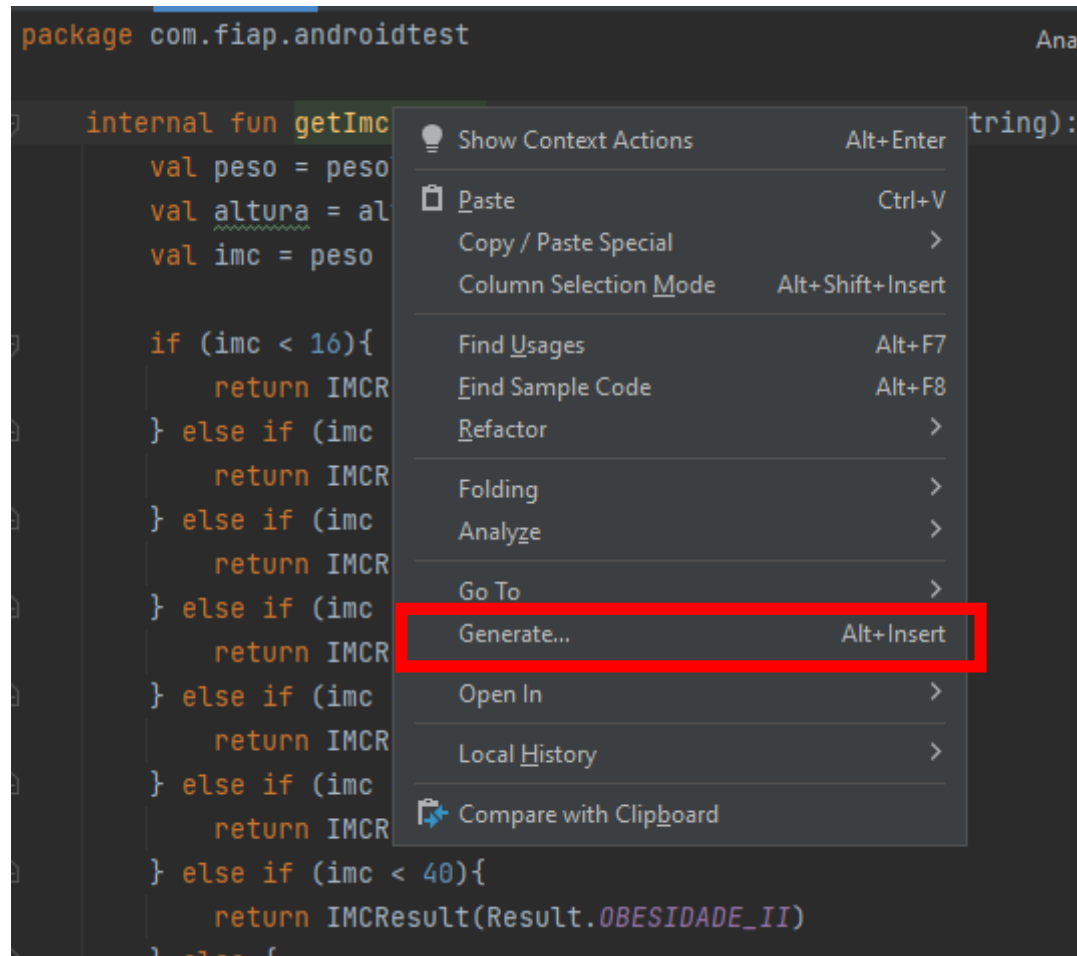
```
data class IMCResult(val result: Result)
    enum class Result(val label: String) {
        MAGREZA_III("Magreza Severa"),
        MAGREZA_II("Magreza moderada"),
        MAGREZA_I("Magreza leve"),
        OK("Saudável"),
        SOBREPESO("Sobrepeso"),
        OBESIDADE_I("Obesidade Grau I"),
        OBESIDADE_II("Obesidade GHrau II (severa)"),
        OBESIDADE_III("Obesidade GHrau III (mórbida)")
    }
```

ARQUIVOS .KT (ACTIVITY_MAIN)

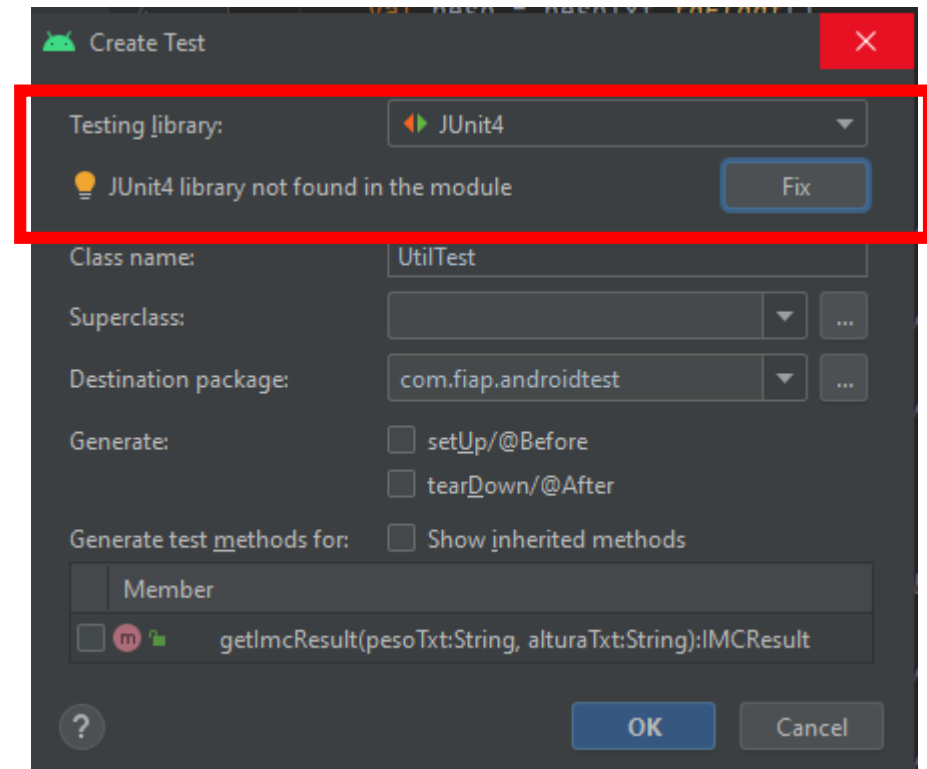
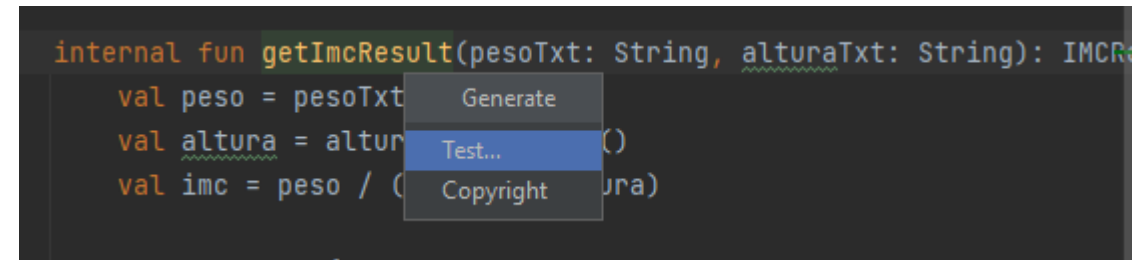


```
1 package com.fiap.androidtest
2
3 import androidx.appcompat.app.AppCompatActivity
4 import android.os.Bundle
5 import kotlinx.android.synthetic.main.activity_main.*
6
7 class MainActivity : AppCompatActivity() {
8     override fun onCreate(savedInstanceState: Bundle?) {
9         super.onCreate(savedInstanceState)
10        setContentView(R.layout.activity_main)
11
12        btn_make_calc.setOnClickListener { it: View!
13            val pesoTxt = edt_peso.text.toString()
14            val alturaTxt = edt_altura.text.toString()
15
16            val peso = pesoTxt.toFloat()
17            val altura = alturaTxt.toFloat()
18
19            val imcResult = getImcResult(pesoTxt, alturaTxt)
20
21            txt_result_imc.text = imcResult.result.label
22        }
23    }
24 }
```

ARQUIVOS UTIL.KT (CRIANDO O ARQUIVO TESTE UTILTEST)

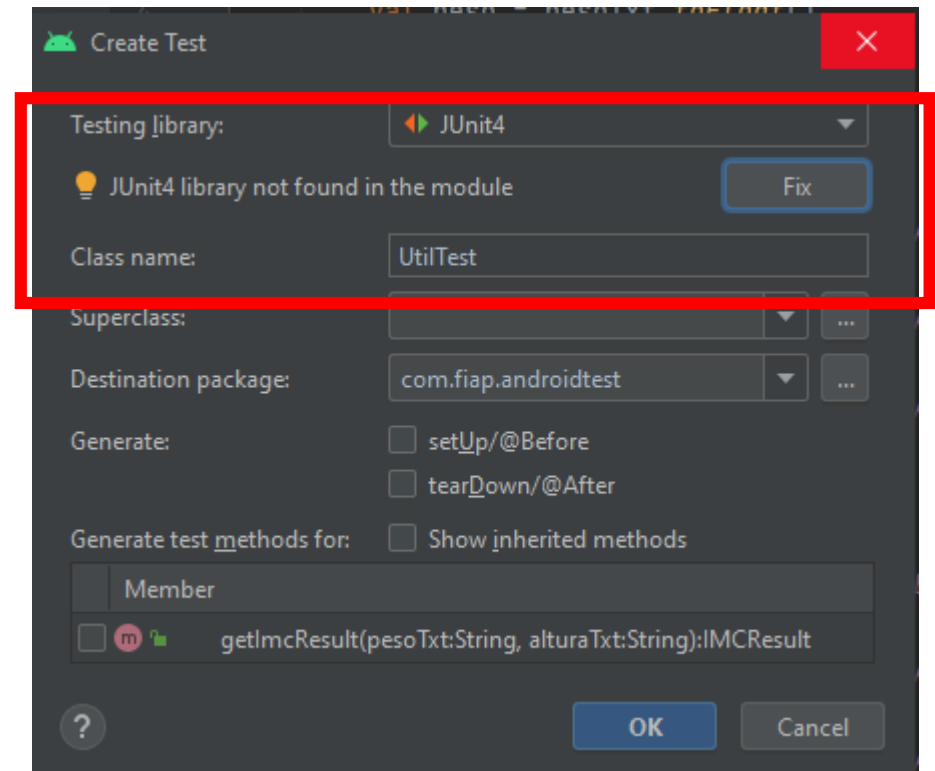


Botão da direita do mouse

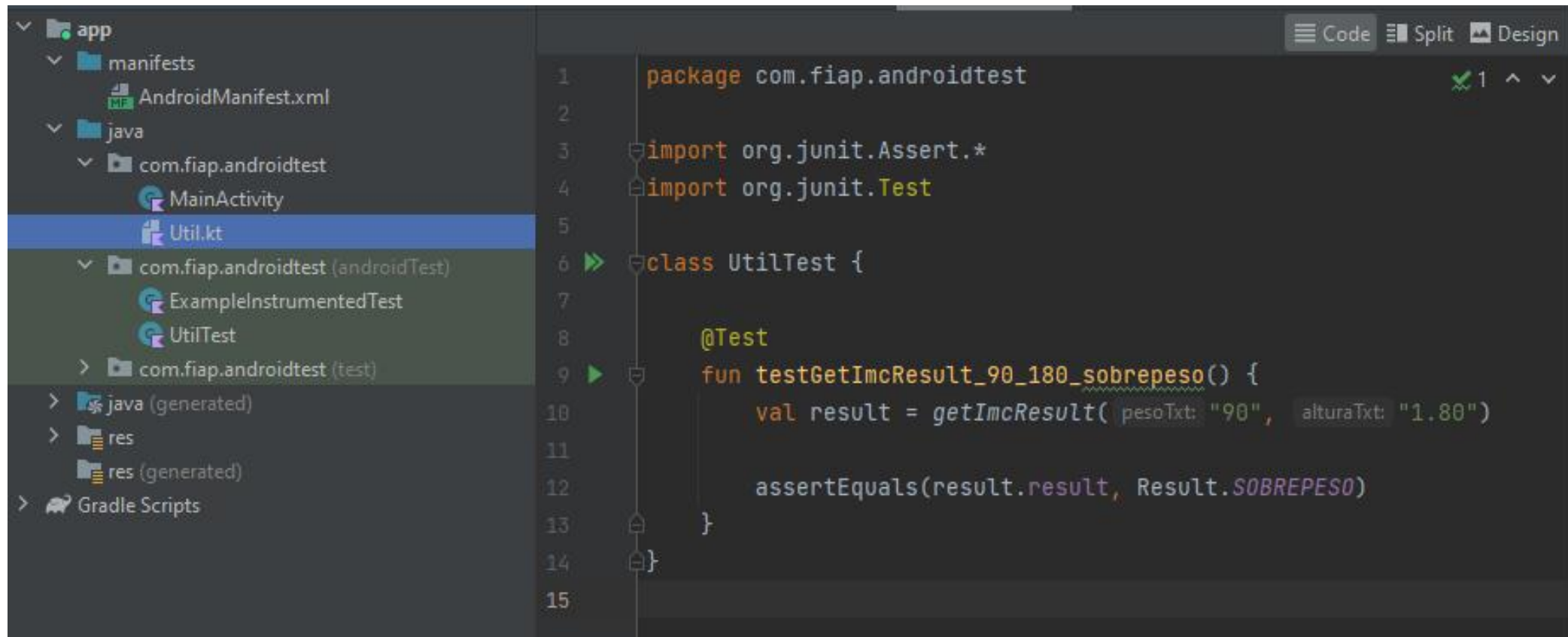


ARQUIVOS UTIL.KT (CRIANDO O ARQUIVO TESTE UTILTEST)

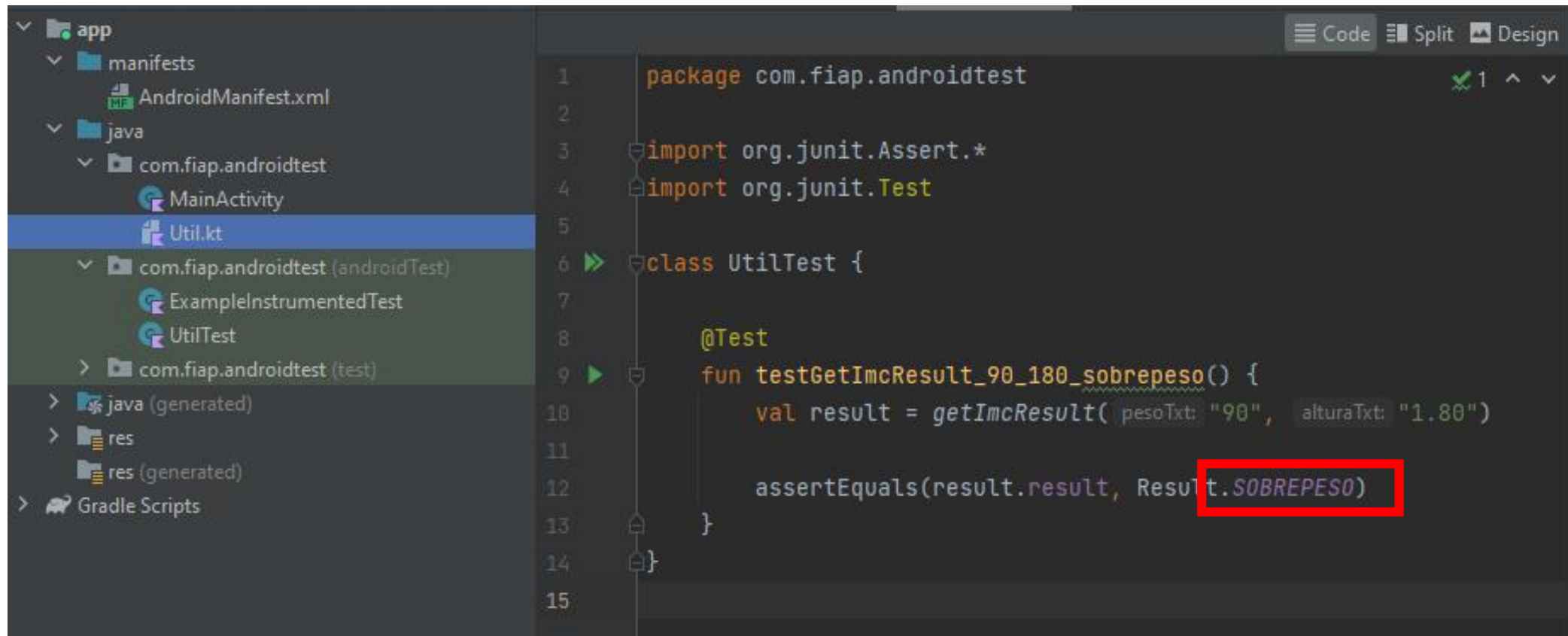
**MUDE A BIBLIOTECA PARA JUNIT4;
CASO A BIBLIOTECA NÃO ESTEJA NO MODULO
CLIQUE NO BOTÃO FIX;
ALTERE O NOME PARA UTILTEST.KT.
SERÁ CRIADA UMA CLASSE DE TESTE E ESTARÁ
LOCALIZADA NA PASTA androidTest.**



ARQUIVOS UTIL.KT (CRIANDO O ARQUIVO TESTE UTILTEST)



ARQUIVOS UTIL.KT (TESTE DO RESULTADO)

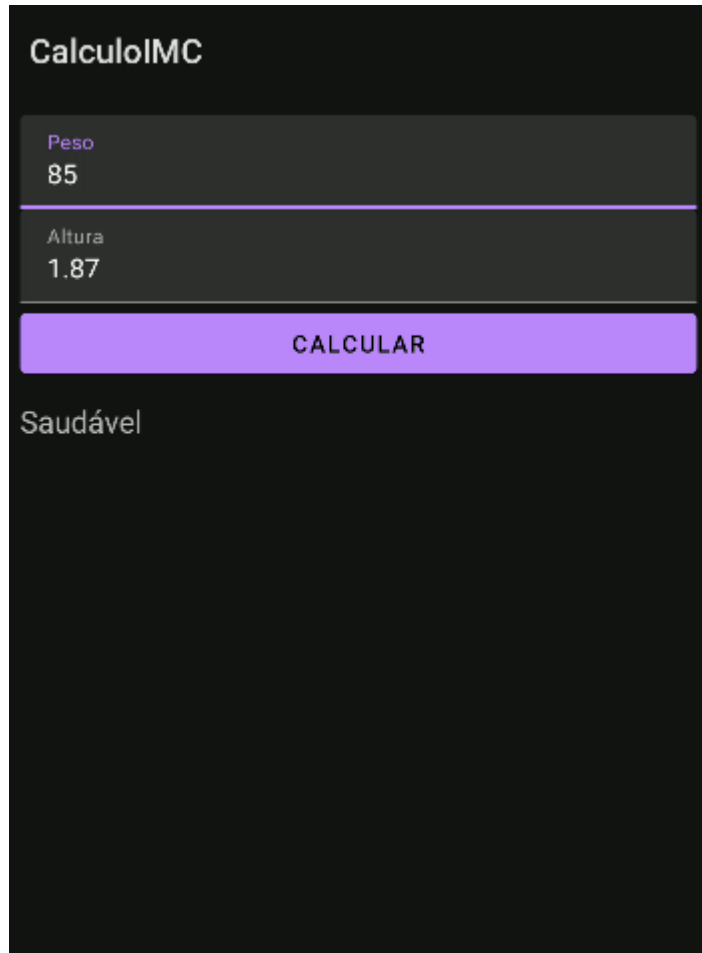


The screenshot shows the Android Studio interface. On the left, the 'app' folder is expanded, showing the 'com.fiap.androidtest' package with 'Util.kt' selected. The main editor displays the code for 'UtilTest.kt'. The code includes package declarations, imports for JUnit, and a test method 'testGetImcResult_90_180_sobrepeso()' that calls 'getImcResult()' and asserts the result is 'Result.SOBREPESO'. The 'Result.SOBREPESO' value is highlighted with a red rectangle.

```
1 package com.fiap.androidtest
2
3 import org.junit.Assert.*
4 import org.junit.Test
5
6 class UtilTest {
7
8     @Test
9     fun testGetImcResult_90_180_sobrepeso() {
10         val result = getImcResult( pesoTxt: "90", alturaTxt: "1.80")
11
12         assertEquals(result.result, Result.SOBREPESO)
13     }
14 }
15
```

**FAÇA UM TESTE, TROQUE O VALOR DO RESULT.SOBREPESO POR RESULT.OK;
E OBSERVE O QUE IRÁ RESULTAR.**

EXERCÍCIO - TELA DO PROJETO (ADICIONAR UM NOVO TESTE)



The screenshot shows a mobile application interface for a BMI calculator. At the top, the title 'CalculoIMC' is displayed in white text on a dark background. Below the title, there are two input fields: 'Peso' (Weight) with the value '85' and 'Altura' (Height) with the value '1.87'. A large blue button labeled 'CALCULAR' is positioned below the input fields. At the bottom of the screen, the text 'Saudável' (Healthy) is visible in white.

CASO VOCÊ APERTE O BOTÃO CALCULAR E NÃO TIVER DADOS NOS CAMPOS IRÁ QUEBRAR A APP.

MODIFIQUE OS DOIS ARQUIVOS CONFORME NAS IMAGENS DOS PRÓXIMOS SLIDES;

CORRIJA O ERRO QUE APARECE;

E OBSERVE O QUE SERÁ APRESENTADO NO CONSOLE.

ARQUIVOS UTIL.KT (MODIFICANDO O ARQUIVO)

```
26      data class IMCResult(val result: Result)
27      enum class Result(val label: String) {
28          MAGREZA_III( label: "Magreza Severa"),
29          MAGREZA_II( label: "Magreza moderada"),
30          MAGREZA_I( label: "Magreza leve"),
31          OK( label: "Saudável"),
32          SOBREPESO( label: "Sobrepeso"),
33          OBESIDADE_I( label: "Obesidade Grau I"),
34          OBESIDADE_II( label: "Obesidade GHrau II (severa)",
35          OBESIDADE_III( label: "Obesidade GHrau III (mórbida)",
36          BLANK( label: "Insira os valores de peso e altura corretamente!")
37      }
38
```

ARQUIVOS UTILTEST.KT (MODIFICANDO O ARQUIVO)

```
1  package com.fiap.androidtest
2
3  import org.junit.Assert.*
4  import org.junit.Test
5
6  class UtilTest {
7
8      @Test
9      fun testGetImcResult_90_180_sobrepeso() {
10         val result = getImcResult( pesoTxt: "90", alturaTxt: "1.80")
11
12         assertEquals(result.result, Result.SOBREPESO)
13     }
14
15     @Test
16     fun testGetImcResult_embranco_aviso() {
17         val semPeso = getImcResult( pesoTxt: "", alturaTxt: "1.80")
18         assertEquals(semPeso.result, Result.BLANK)
19
20         val semAltura = getImcResult( pesoTxt: "90", alturaTxt: "")
21         assertEquals(semAltura.result, Result.BLANK)
22     }
23 }
24
25
```

EXERCÍCIO

**CRIE UM REPOSITÓRIO REMOTO (GITHUB), COM NOME TESTE CALCULADORA IMC;
SIGA TODOS OS PASSOS FEITOS NOS SLIDES E AS RECOMENDAÇÕES DURANTE A AULA;
NO SLIDE 48 EXISTE UMA IMPLEMENTAÇÃO A SER FEITA, INSIRA NO SEU CÓDIGO;
COMENTE TODO O CÓDIGO CRIADO;
CRIE UM README PARA O REPOSITÓRIO EXPLICANDO SEU FUNCIONAMENTO;
ENVIE SOMENTE O LINK DO REPOSITÓRIO NA TAREFA.**

FIAP