

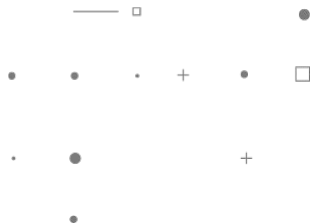


SISTEMAS DE INFORMAÇÃO  
**DESENVOLVIMENTO ANDROID**



# MOBILE DEVELOPMENT

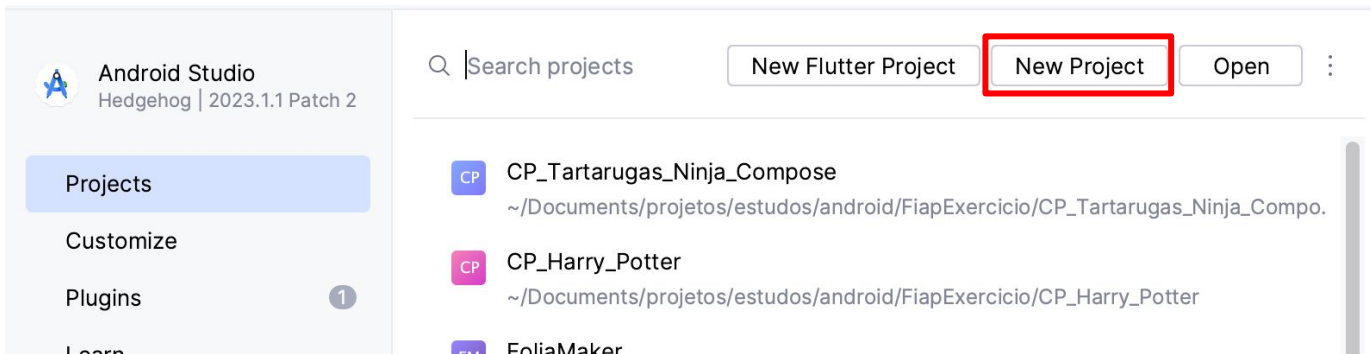




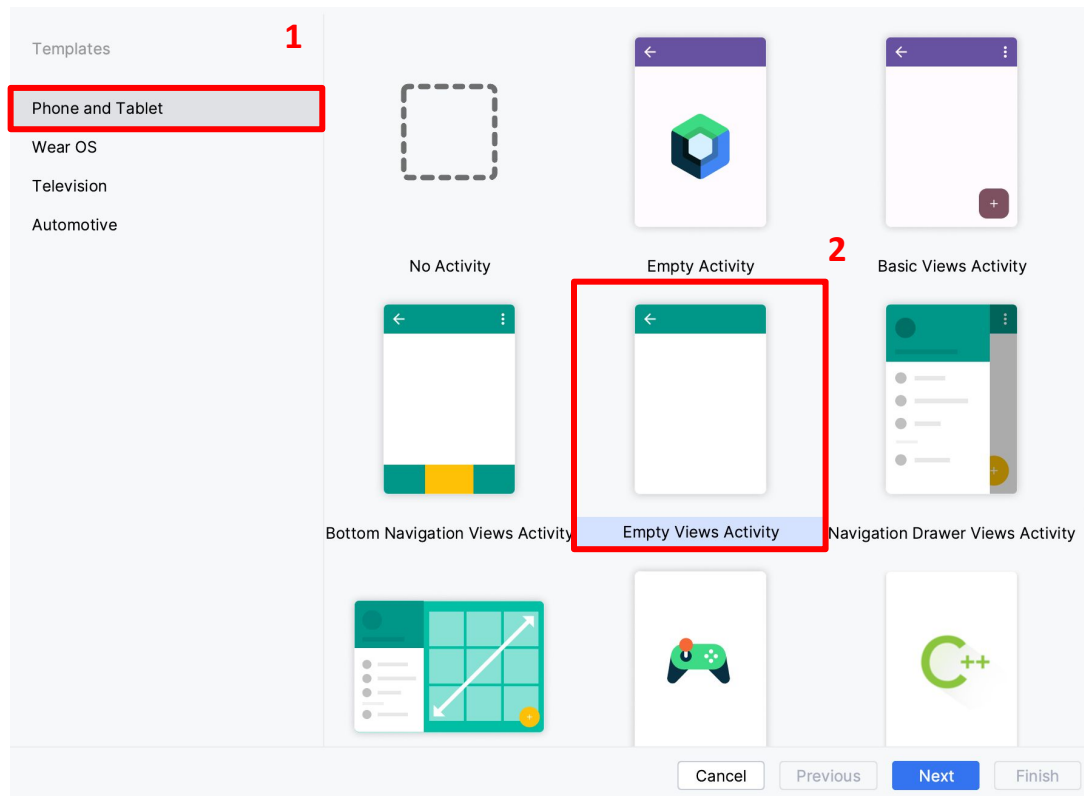
**HORA DE COMEÇAR:**  
QUAL VAI SER O APP DE HOJE?



# CRIANDO O PROJETO



# CRIANDO O PROJETO



# CRIANDO O PROJETO

### Empty Views Activity

Creates a new empty activity


Name


Package name

Save location

Language

Minimum SDK

 Your app will run on approximately 81,2% of devices.  
[Help me choose](#)

Build configuration language 

# Ping Pong X - v0



PingPongX

Começar a partida

Preencha os campos

COMEÇAR

Desenvolvido por Heider Lopes

PingPongX

0 X 0

Casa

Visitante

PONTO

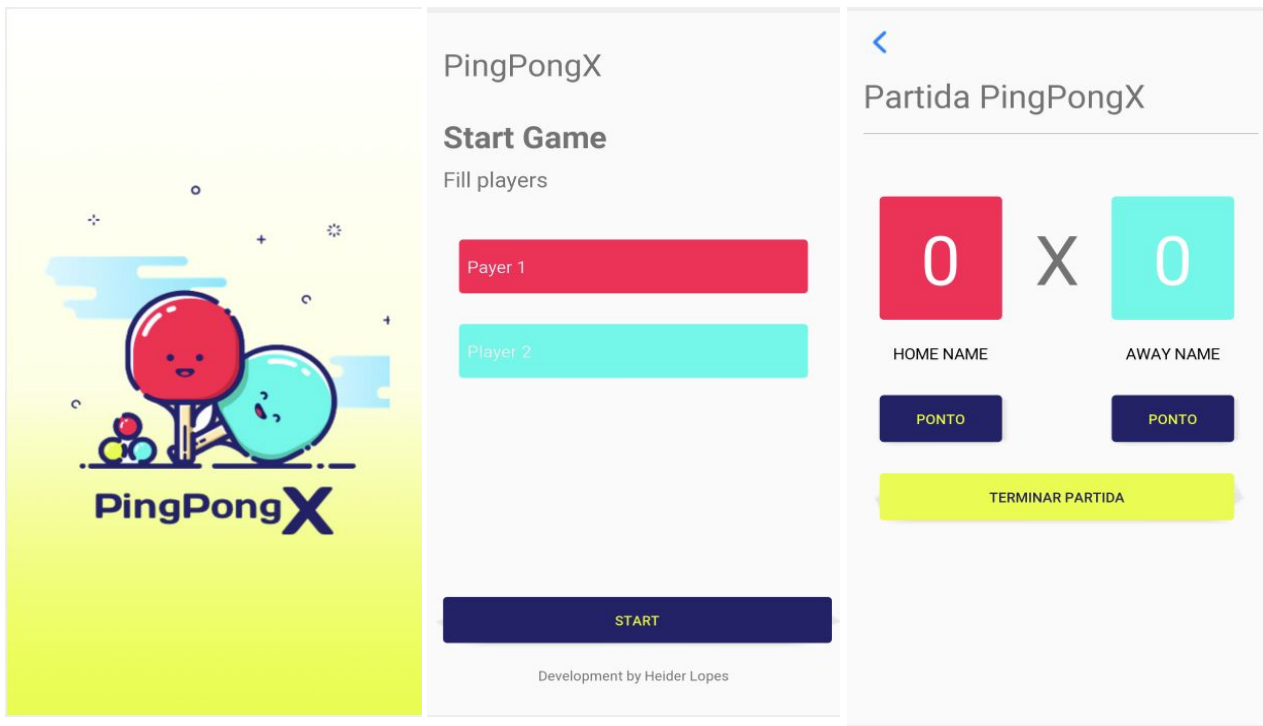
PONTO

TERMINAR PARTIDA

REVANCHE

# Ping Pong X - v1

Altere o layout do aplicativo para que ele fique conforme as imagens abaixo:







## Habilitando View Binding

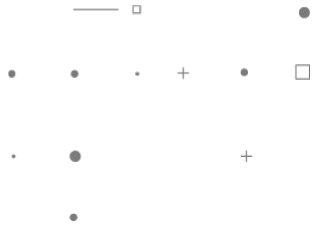


## HABILITANDO VIA VIEWBINDING

---

Abra o arquivo **build.gradle (app)** e adicione a seguinte configuração:

```
buildFeatures {  
    viewBinding = true  
}
```



## ADICIONANDO AS IMAGENS AO PROJETO

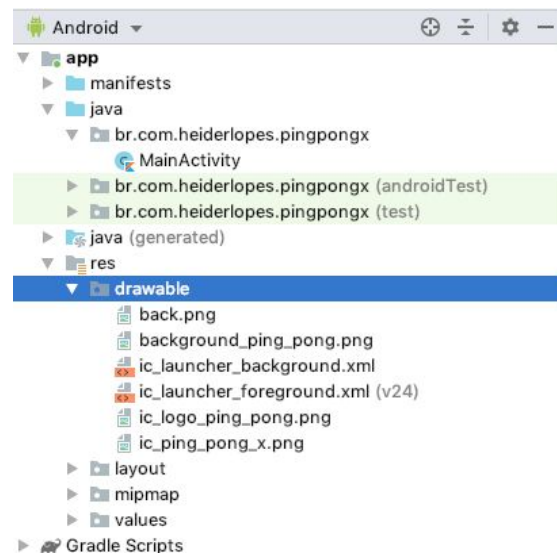


## Adicionando os arquivos do aplicativo

As imagens que serão utilizadas no aplicativo encontram-se em:

[https://github.com/heiderlopes/placar\\_ping\\_pong.git](https://github.com/heiderlopes/placar_ping_pong.git)

Adicionar as imagens na pasta drawable



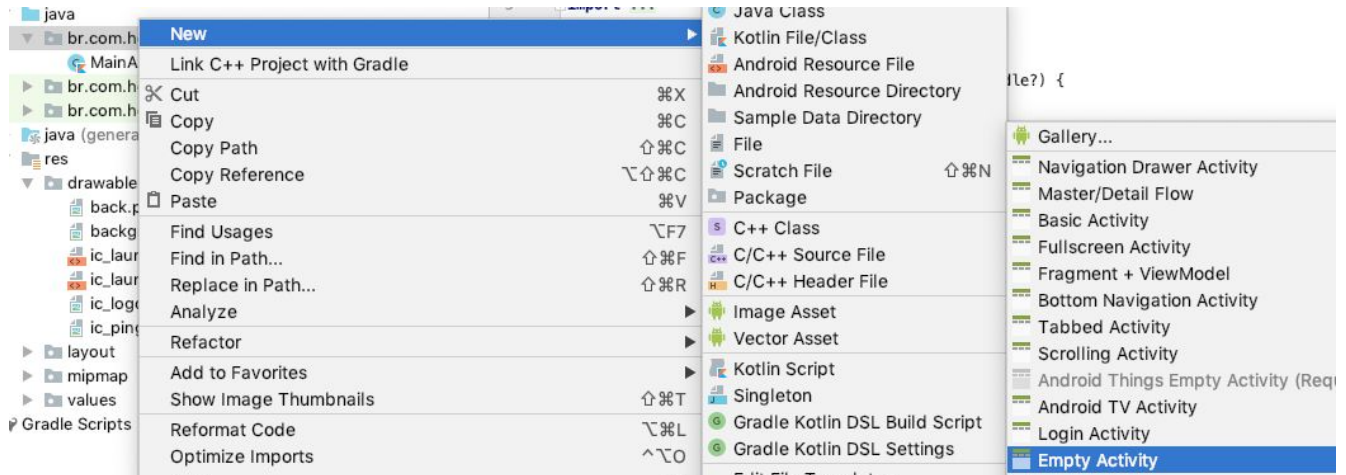
## NOSSA PRIMEIRA TELA: CRIANDO A SPLASHSCREEN



## Adicionando a SplashScreen

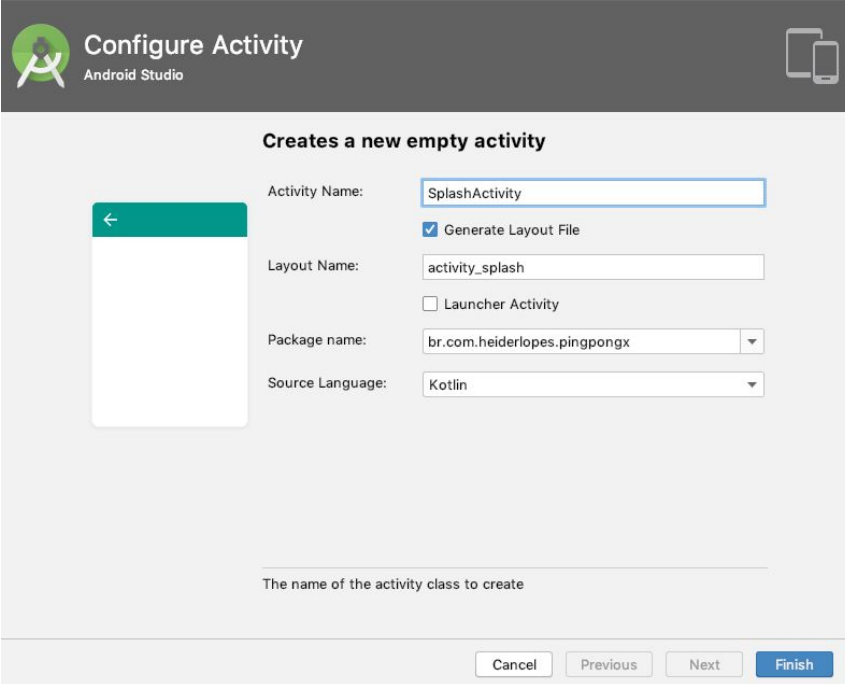
Para isso, precisamos adicionar uma nova activity em nosso aplicativo.

Clique com o botão direito sobre o pacote, **New** → **Activity** → **Empty Activity**



## Adicionando a SplashScreen

Defina o nome da activity como **SplashActivity**



**Configure Activity**  
Android Studio

**Creates a new empty activity**

Activity Name:

☒ Generate Layout File

Layout Name:

☐ Launcher Activity

Package name:

Source Language:

The name of the activity class to create

Cancel Previous Next Finish

## Alterando o background

---

O primeiro passo é definirmos o background da nossa splash, para isso, abra o arquivo **activity\_splash.xml** e e na tag raíz (**ConstraintLayout**) adicione o background conforme código abaixo:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/background_ping_pong"
    tools:context=".SplashActivity">

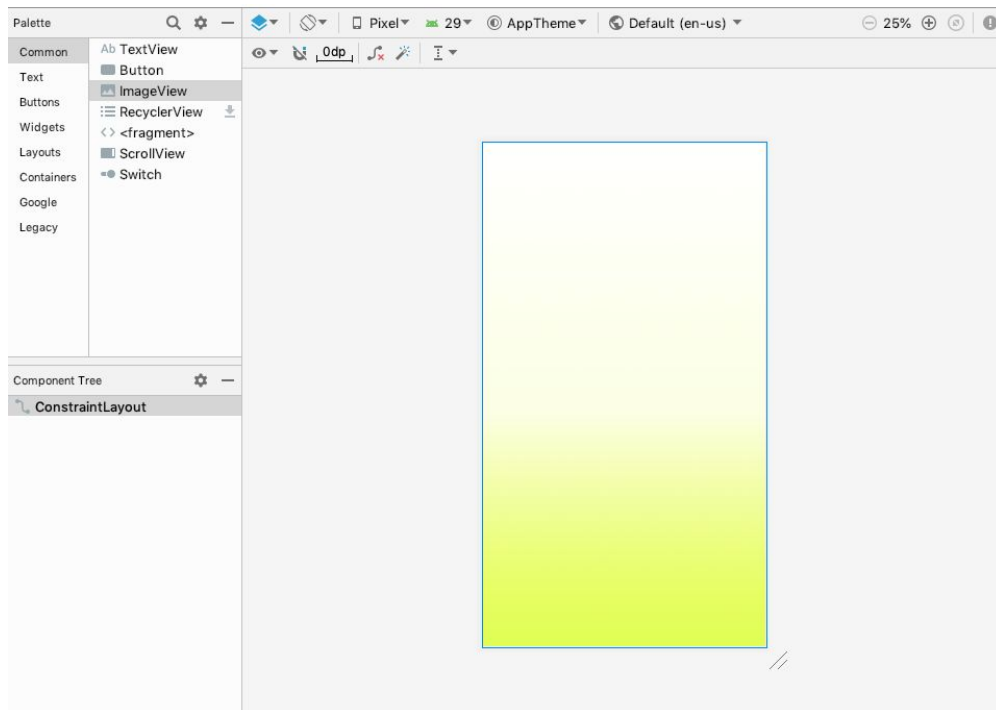
</androidx.constraintlayout.widget.ConstraintLayout>
```





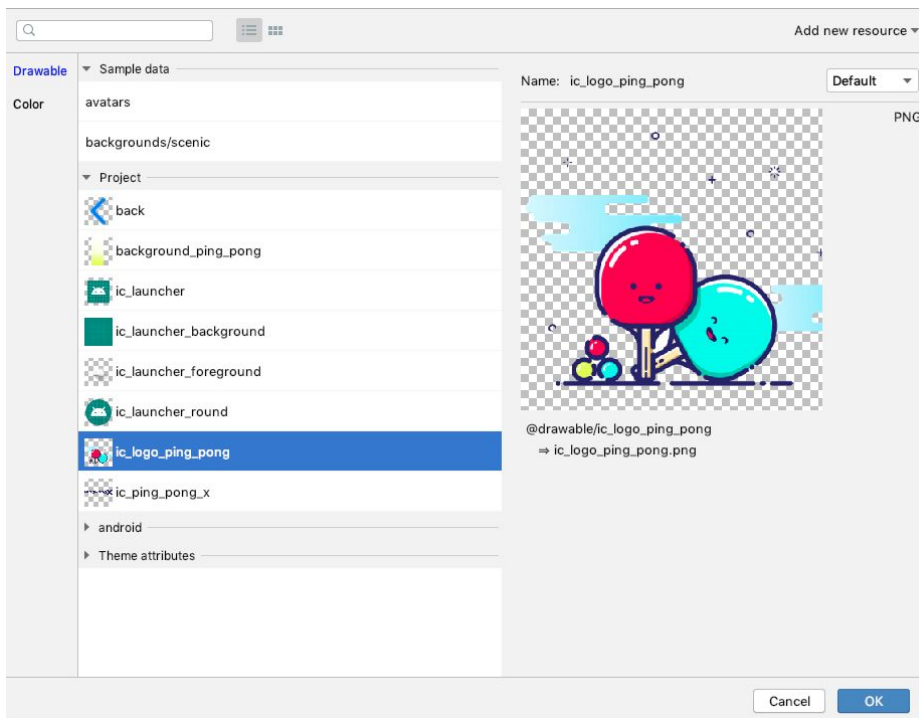
## Adicionando as imagens

Arraste um ImageView para dentro do nosso layout.



## Adicionando as imagens

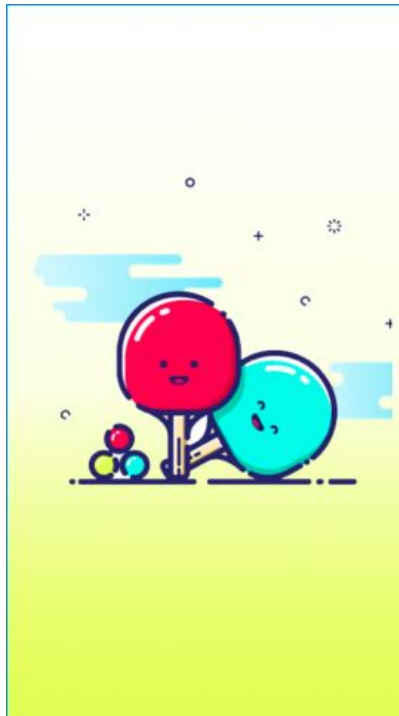
Selecione a imagem **ic\_logo\_ping\_pong**



## Adicionando as imagens

---

Clique e arraste a imagem inserida até o centro da tela.



## Adicionando as imagens

---

Repita o processo para adicionar a imagem **ic\_ping\_pong\_x** conforme imagem abaixo:



Rode o aplicativo para ver o resultado



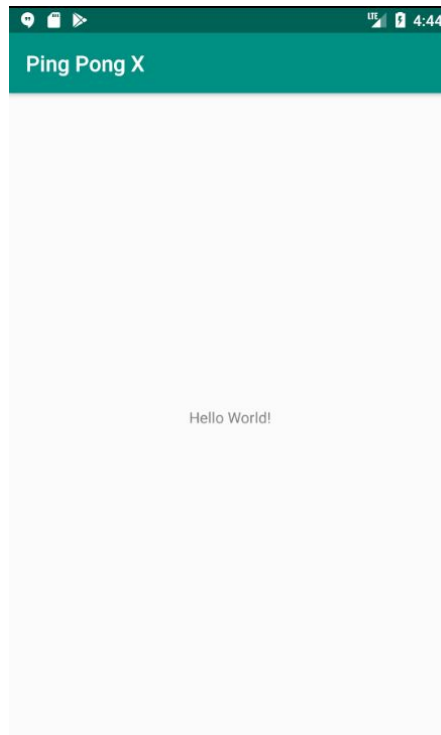
Expectativa



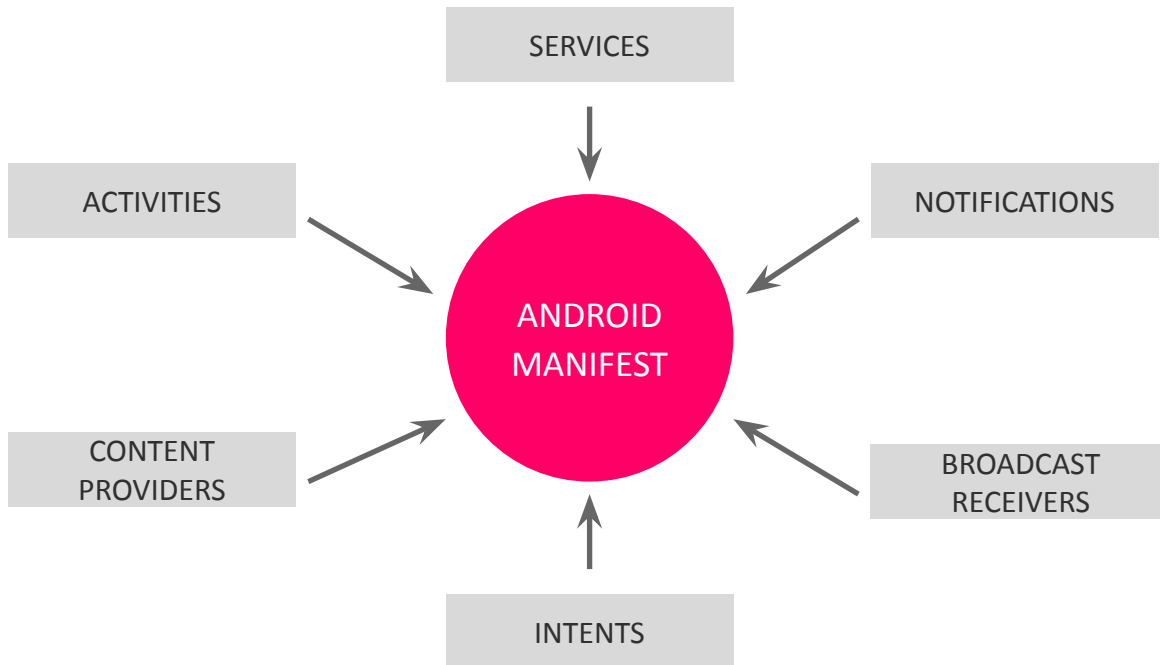
# Rode o aplicativo para ver o resultado

---

**Realidade**



## Alterando o AndroidManifest.xml



**Android Manifest** é um arquivo XML (AndroidManifest.xml) que **define e integra os componentes de uma aplicação** vistos acima.

## Alterando o AndroidManifest.xml

---

Quando rodamos nosso aplicativo a tela exibida não foi a nossa **SplashActivity** e sim a **MainActivity**.

Para alterarmos a tela que será disparada ao iniciar o aplicativo devemos realizar a seguinte alteração no arquivo **AndroidManifest.xml**



# Alterando o AndroidManifest.xml

---

O arquivo atualmente está da seguinte forma:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.heiderlopes.pingpongx">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SplashActivity"></activity>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Alterando o AndroidManifest.xml

---

Devemos deixar da seguinte forma:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.heiderlopes.pingpongx">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity">
        </activity>
    </application>
</manifest>
```

## O que mudamos no AndroidManifest

---

As atividades do aplicativo, os serviços e os broadcast receivers são ativados pelos intents. O **intent é uma mensagem definida por um objeto Intent que descreve uma ação a ser realizada**, inclusive dados usados em ações, a categoria do componente que executará a ação e outras instruções.

Quando um aplicativo envia um intent ao sistema, este último localiza um componente do aplicativo que possa processar o intent com base nas declarações do filtro de intents do arquivo de manifesto do aplicativo. O sistema lança uma instância do componente correspondente e passa o objeto Intent a esse componente. Caso mais de um aplicativo possa processar o intent, o usuário pode escolher qual usar.

Um componente do aplicativo pode ter vários filtros de intents (definidos com o elemento `<intent-filter>`), cada um descrevendo um recurso diferente do componente.

Para mais informações, consulte o documento Intents e filtros de intents.

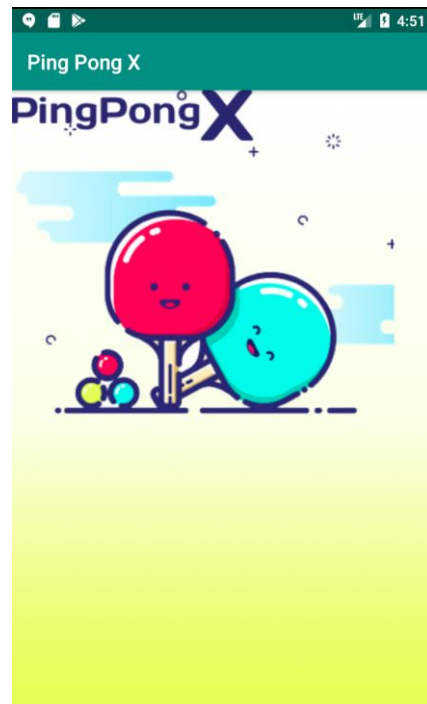
<https://developer.android.com/guide/topics/manifest/manifest-intro?hl=pt-br>

## O que mudamos no AndroidManifest

---

No exemplo anterior, alteramos o filtro de intenção para abrir a nossa **SplashActivity** em vez de rodar a **MainActivity**.

Rode o aplicativo novamente e confira o resultado ao lado



## Como resolvemos a organização do nosso layout

---

Podemos observar que o nosso layout não ficou igual ao que desenhamos, pois, estamos utilizando o **ConstraintLayout** como nosso organizador de layout.

Neste tipo de layout precisamos definir uma constraint horizontal e uma vertical para nossa view.

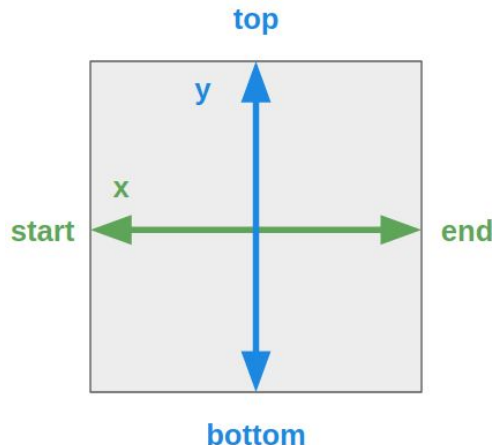
Quando selecionamos uma das views teremos o seguinte resultado



## Como resolvemos a organização do nosso layout

O **Constraint Layout** opera de uma maneira similar ao RelativeLayout, ou seja, todo o alinhamento é feito de **relações entre as Views**, a diferença é que somos capazes de indicar o posicionamento que queremos manter as Views por meio dos seus eixos.

Os eixos são definidos como X e Y, sendo X o eixo de início (esquerdo) e fim (direito) e o Y o de topo (cima) e inferior (baixo).



## As Constraints

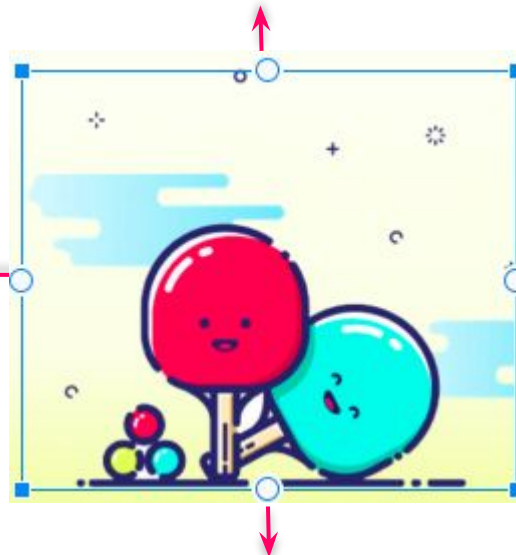
---

1- Create top constraint

4 - Create left constraint

2- Create right constraint

3 - Create bottom constraint



## Como resolvemos a organização do nosso layout

---

Criando as constraint para centralizar a imagem na tela.

Posicione o cursor do mouse sobre o **Create Top Constraint** e arraste para o topo da tela

Posicione o cursor do mouse sobre o **Create Bottom Constraint** e arraste para a parte de baixo da tela

Posicione o cursor do mouse sobre o **Create Left Constraint** e arraste para o extremo esquerdo da tela

Posicione o cursor do mouse sobre o **Create Right Constraint** e arraste para o extremo direito da tela



## Como resolvemos a organização do nosso layout

Teremos o seguinte resultado no nosso layout:



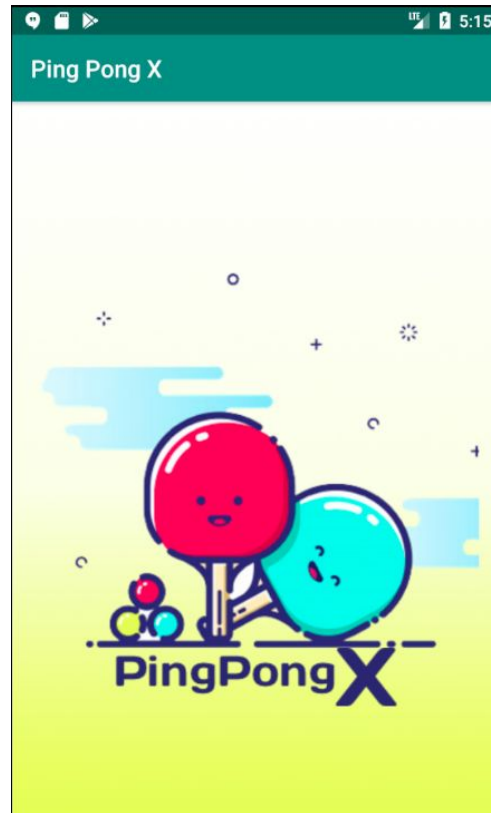
## Como resolvemos a organização do nosso layout

O próximo passo é alterarmos as constraint referente à imagem pingpongx.

Faça com que fique alinhada à imagem anterior.



Rode o aplicativo para ver o novo resultado





## UM POUCO MAIS DE TEORIA

### CONHECENDO MAIS SOBRE CONSTRAINTLAYOUT



## Entendendo as Constraints

---

Constraints significa algo como “restrições” ou “limitações”, e são essas restrições que são as bases para o funcionamento deste layout manager.

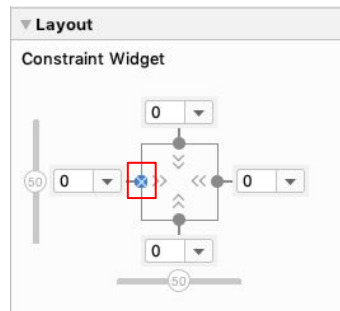
Como vimos, para definir a posição de uma view no ConstraintLayout, você deve adicionar ao **menos uma constraint horizontal e uma vertical para a view**.

Cada constraint representa uma conexão ou alinhamento em relação à outra view, o layout parent ou mesmo uma linha-guia invisível (abordaremos no decorrer do curso).

Quando você arrasta e solta uma view no Layout Editor, ela fica exatamente onde você a deixar, mesmo que não possua constraint alguma. No entanto, isso é apenas para tornar o seu trabalho mais fácil quando estiver posicionando os elementos; se uma view não possui constraints, ela **ficará no canto superior esquerdo da tela automaticamente (0,0)**.

## Removendo as Constraints

Para remover uma constraint, selecione a view e então clique no constraint handle. No exemplo abaixo, estamos removendo a Constraint da esquerda.



Ou remova todas constraints selecionando a view e depois clicando em **“Clear All Constraints”**.

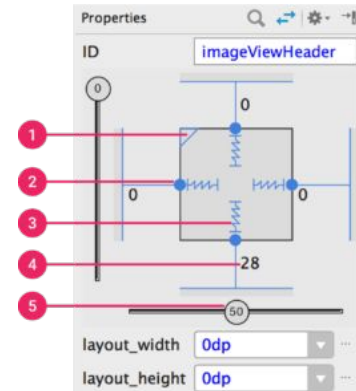


## Ajustando os tamanhos dos componentes

Você pode usar os cantos do seu componente para redimensioná-lo, mas isso não é recomendado, uma vez que adiciona um valor de largura e altura literal à ele, impedindo-o de se ajustar às diferentes resoluções de telas.

Para selecionar modos de redimensionamento mais espertos, clique no componente e abra a janela de propriedades dele no lado direito do editor.

Próximo ao topo da janela de propriedades está o **View Inspector**, que inclui controles para muitas propriedades de layout, como mostrado na figura abaixo (disponível apenas para layouts usando ConstraintLayout):



# CRIANDO A ANIMAÇÃO

---

Abra o arquivo **activity\_splash.xml** e adicione os seguintes nomes para os componentes:



Altere os ids das views com seus respectivos nomes:

1 - ivLogo

2 - ivLogoName



## CRIANDO A ANIMAÇÃO

---

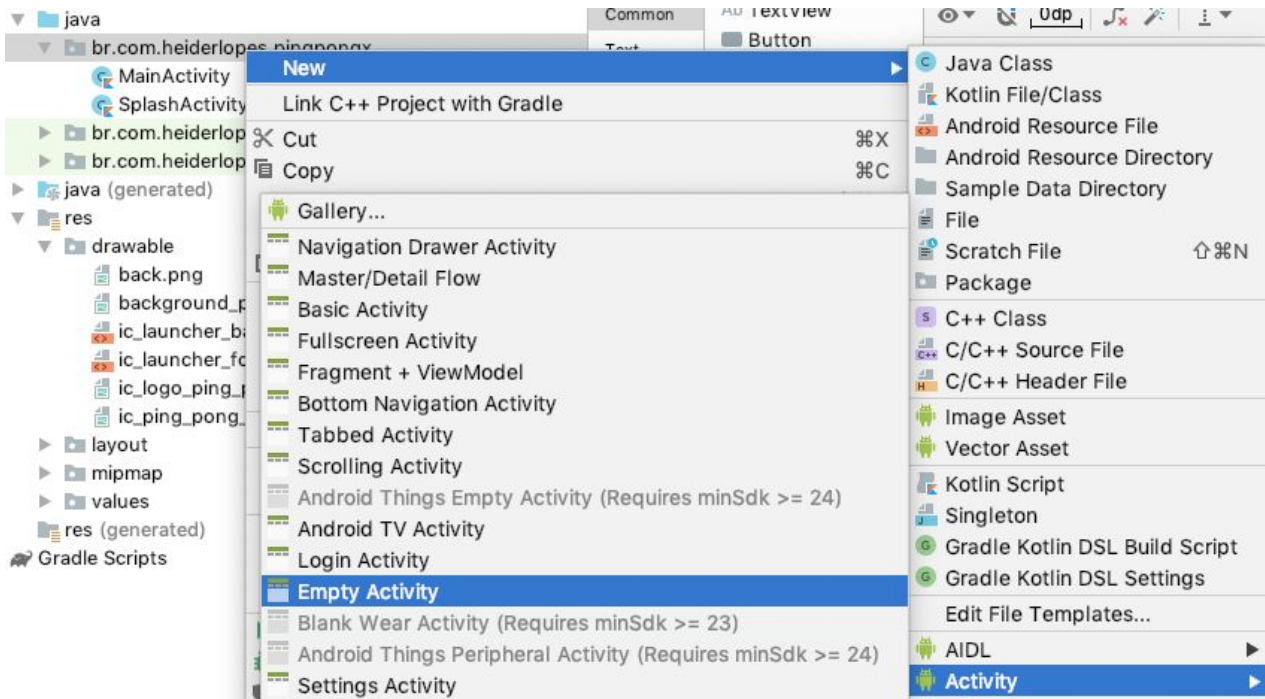
Abra o arquivo **SplashActivity.kt** e adicione o seguinte código:

```
class SplashActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivitySplashBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        binding = ActivitySplashBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
    }  
}
```

## CRIANDO A SEGUNDA TELA FORMULÁRIOS DOS JOGADORES

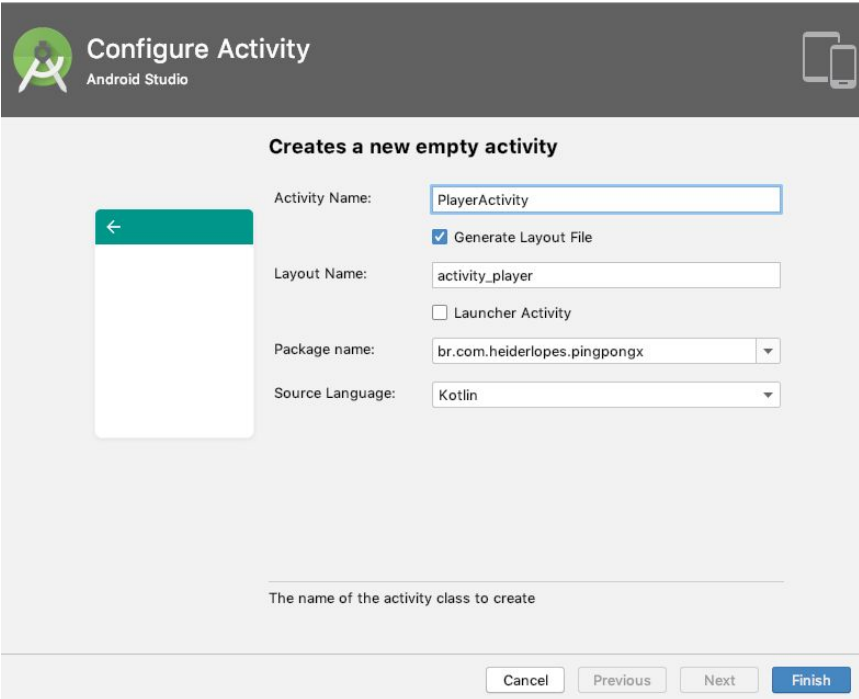
## Criando a PlayerActivity

Clique com o botão direito sobre sobre o pacote da aplicação, **New** → **Activity** → **Empty Activity**



# Criando a PlayerActivity

Dê o nome de **PlayerActivity**



**Configure Activity**  
Android Studio

**Creates a new empty activity**

Activity Name:

☒ Generate Layout File

Layout Name:

☐ Launcher Activity

Package name:

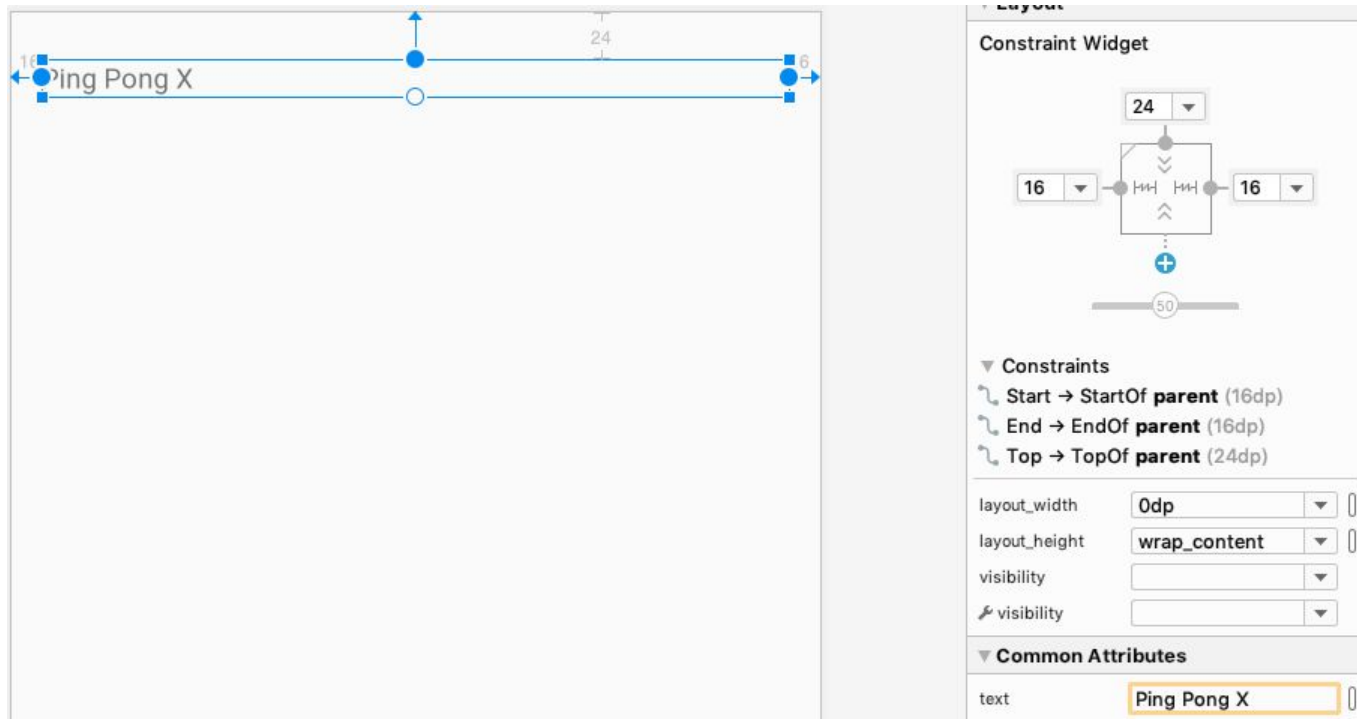
Source Language:

The name of the activity class to create

Cancel Previous Next Finish

## Criando a PlayerActivity

Crie o seguinte componente com suas respectivas **Constraints**



The image shows the Android Studio IDE with a layout editor on the left and a properties panel on the right. The layout editor displays a horizontal bar with a blue circle on the left and a blue square on the right. The text "Ping Pong X" is centered. The layout is defined by constraints: Start to Start of parent (16dp), End to End of parent (16dp), and Top to Top of parent (24dp). The right panel shows the "Constraint Widget" configuration with these constraints and attributes.

**Constraint Widget**

24

16 16

50

**Constraints**

- Start → StartOf **parent** (16dp)
- End → EndOf **parent** (16dp)
- Top → TopOf **parent** (24dp)

layout\_width 0dp

layout\_height wrap\_content

visibility

visibility

**Common Attributes**

text Ping Pong X

## Criando a PlayerActivity

Crie o seguinte componente com suas respectivas **Constraints**

The screenshot shows the Android Studio IDE with the design of a **PlayerActivity** component. The main view displays a text view labeled "Ping Pong X" and a button labeled "Começar a partida". The button is positioned below the text view, with a vertical dimension line indicating a 32dp offset from the bottom of the text view.

The right-hand panel shows the **Constraint Widget** configuration. It includes a diagram of the widget with a 32dp offset and two 0dp offsets. Below the diagram, the **Constraints** section lists:

- Start -> StartOf textView (0dp)
- End -> EndOf textView (0dp)
- Top -> BottomOf textView (32dp)

The **Common Attributes** section shows the text "Começar a partida".



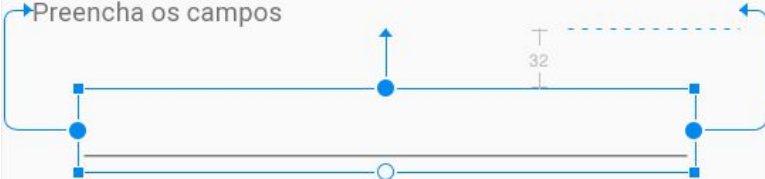
## Criando a PlayerActivity

Crie o seguinte componente com suas respectivas **Constraints**

Ping Pong X

Começar a partida

Preencha os campos

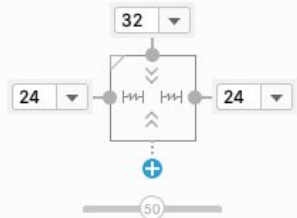


### id: etPlayer1

► Declared Attributes + -

▼ Layout

Constraint Widget



▼ Constraints

- Start → StartOf **textView4** (24dp)
- End → EndOf **textView4** (24dp)
- Top → BottomOf **textView4** (32dp)

layout\_width 0dp

layout\_height wrap\_content

visibility

visibility



## Criando a PlayerActivity

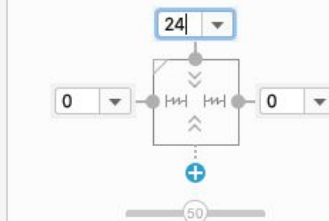
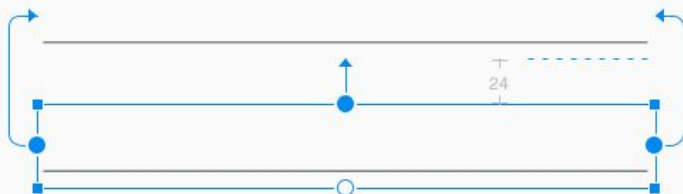
Crie o seguinte componente com suas respectivas **Constraints**

**id: etPlayer2**

Ping Pong X

Começar a partida

Preencha os campos



### ▼ Constraints

Start → StartOf `editText` (0dp)

End → EndOf `editText` (0dp)

Top → BottomOf `editText` (24dp)

layout\_width 0dp

layout\_height wrap\_content

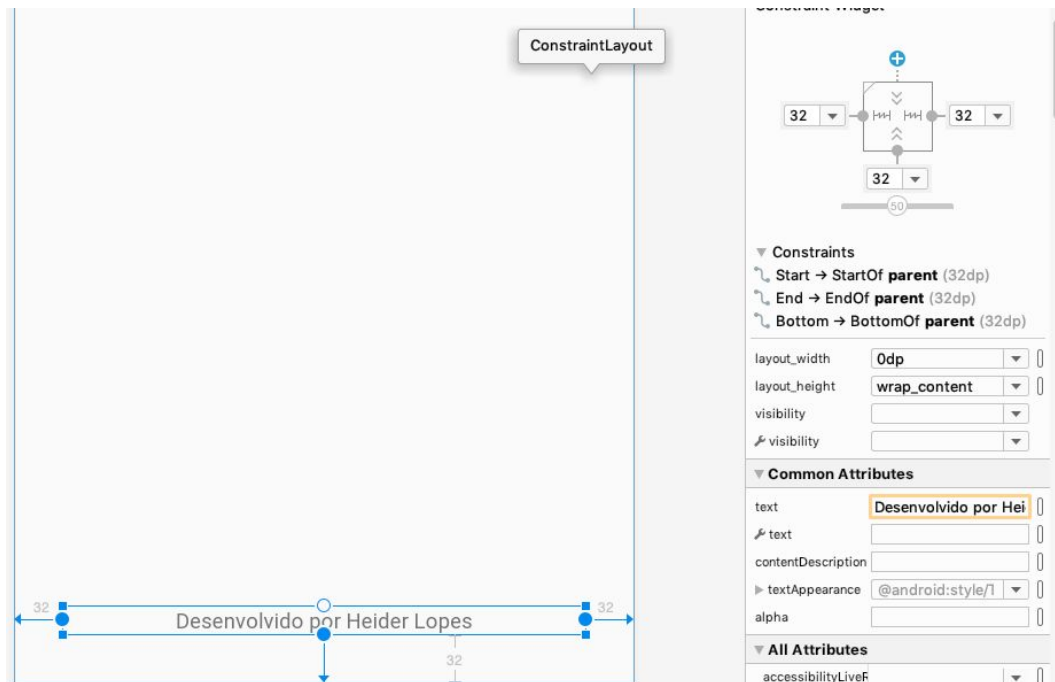
visibility

visibility

### ▼ Common Attributes

## Criando a PlayerActivity

Crie o seguinte componente com suas respectivas **Constraints**



## Criando a PlayerActivity

Crie o seguinte componente com suas respectivas **Constraints**

The image shows the Android Studio IDE with a design view on the left and a code view on the right. The design view shows a button with the text "COMEÇAR" and a width of 8. The code view shows the XML for the button, with constraints and attributes.

**Design View:**

- Button text: COMEÇAR
- Width: 8

**Code View:**

```
id="btStart"
Declared Attributes
Layout
Constraint Widget
Constraints (3)
layout_width="0dp"
layout_height="wrap_content"
visibility="visible"
visibility="visible"
Common Attributes
style="@android:style/..."
stateListAnimator="@android:anim/button..."
onClick=""
elevation=""
```

## Criando a PlayerActivity



Nossa player activity até o momento

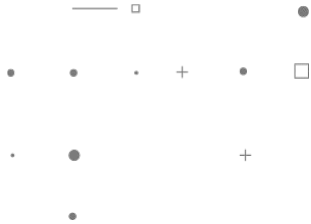
PingPongX

Começar a partida

Preencha os campos

COMEÇAR

Desenvolvido por Heider Lopes

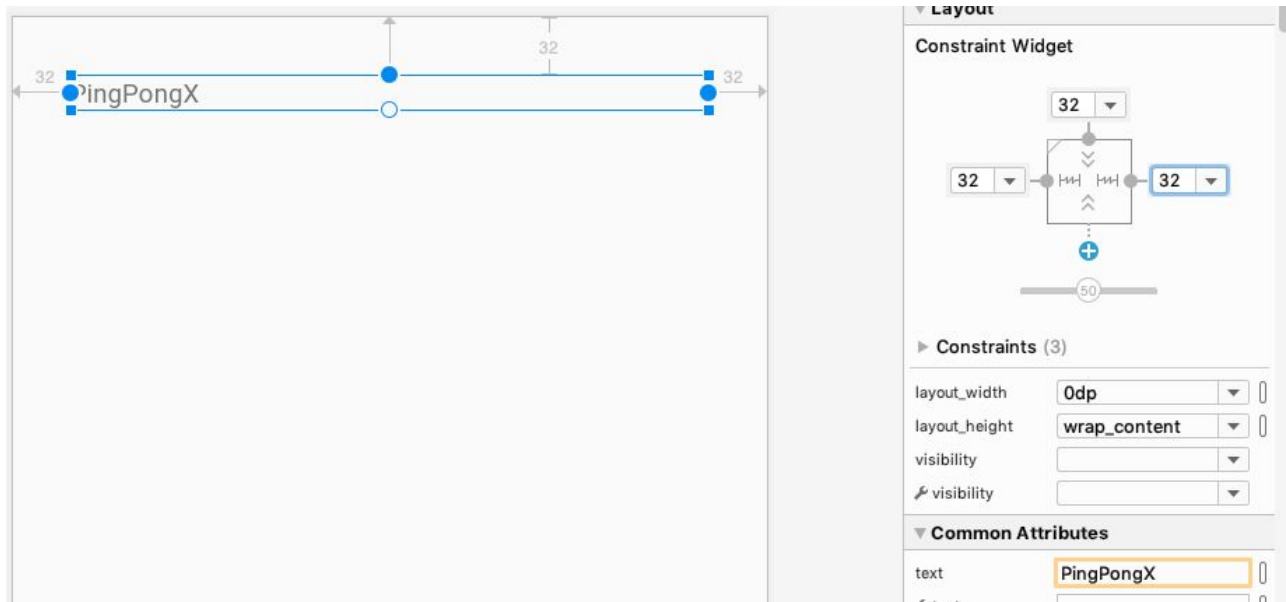


## Criando o layout da MainActivity



## Criando a MainActivity

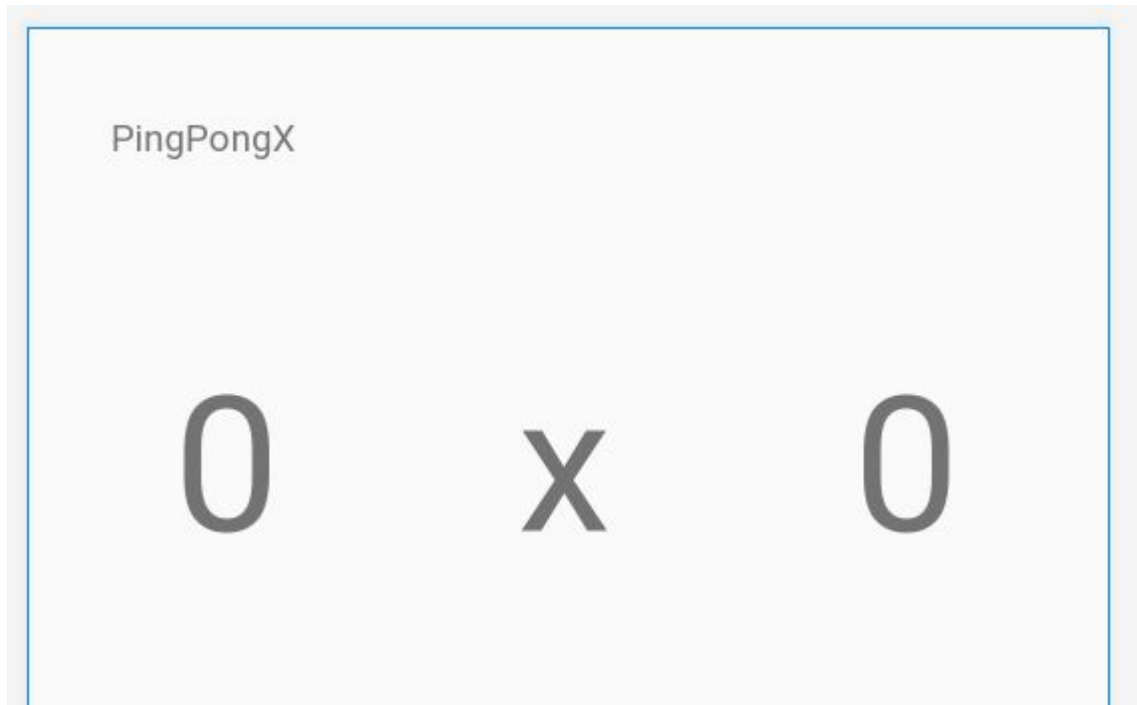
Crie o seguinte componente com suas respectivas **Constraints**



## Criando a MainActivity

---

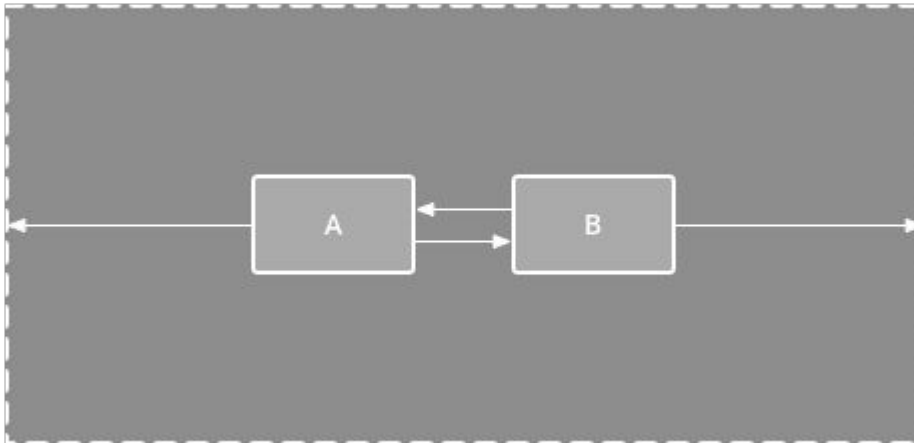
Crie os elementos abaixo com **layout\_width** de **120dp** e **textSize** de **72sp**



## Ligando grupo de componentes

---

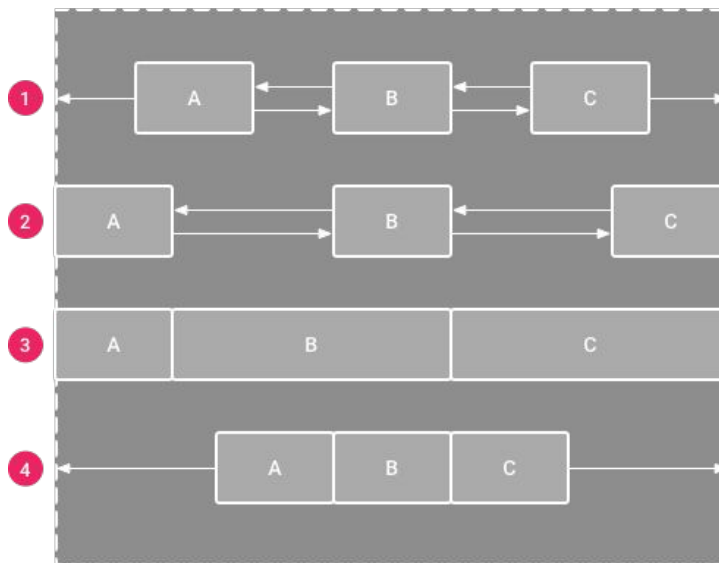
Um grupo de componentes pode ser ligado usando constraints bi-direcionais uns com os outros, como mostrado na imagem abaixo, onde temos dois componentes com constraints horizontais um no outro, criando uma corrente ou cadeia horizontal (**chain**).





## Ligando grupo de componentes

Uma corrente permite que você distribua os componentes horizontalmente ou verticalmente com os seguintes estilos:



## Ligando grupo de componentes

---

**Spread:** default, os componentes são distribuídos de maneira uniforme, após suas margens serem calculadas;

**Spread inside:** são respeitadas as constraints de cada extremidade e o restante do espaço é distribuído uniformemente;

**Weighted:** quando a corrente é definida como Spread ou Spread Inside, você pode colocar o tamanho de seus componentes como “match constraint” para que eles ocupem todo o espaço disponível uniformemente. Caso deseje que um componente ocupe mais espaço que o outro, você pode definir pesos diferenciados para eles usando as propriedades `layout_constraintHorizontal_weight` e `layout_constraintVertical_weight`, assim como funcionava no `LinearLayout` clássico;

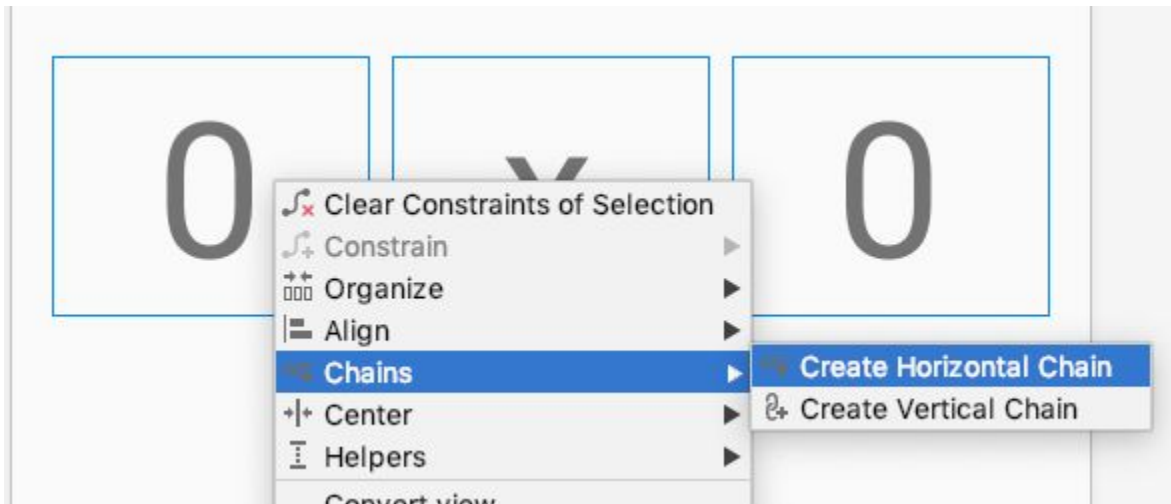
**Packed:** os componentes ficam grudados uns nos outros.

Para criar uma corrente de componentes rapidamente, selecione todos eles e depois com o clique direito do mouse, escolha **Center Horizontally** ou **Center Vertically**, para criar a corrente na respectiva orientação.

## Criando a MainActivity

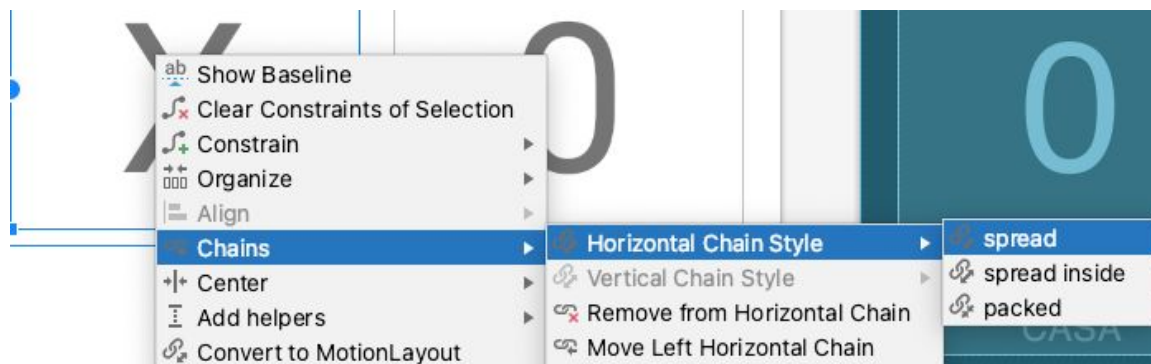
---

Selecione os três elementos criados (0 x 0) e clique com o botão direito sobre o primeiro selecionado e crie uma **horizontal chains**



## Criando a MainActivity

Podemos alterar o **chain mode** para termos uma outra exibição dos elementos criados:



## Criando a MainActivity

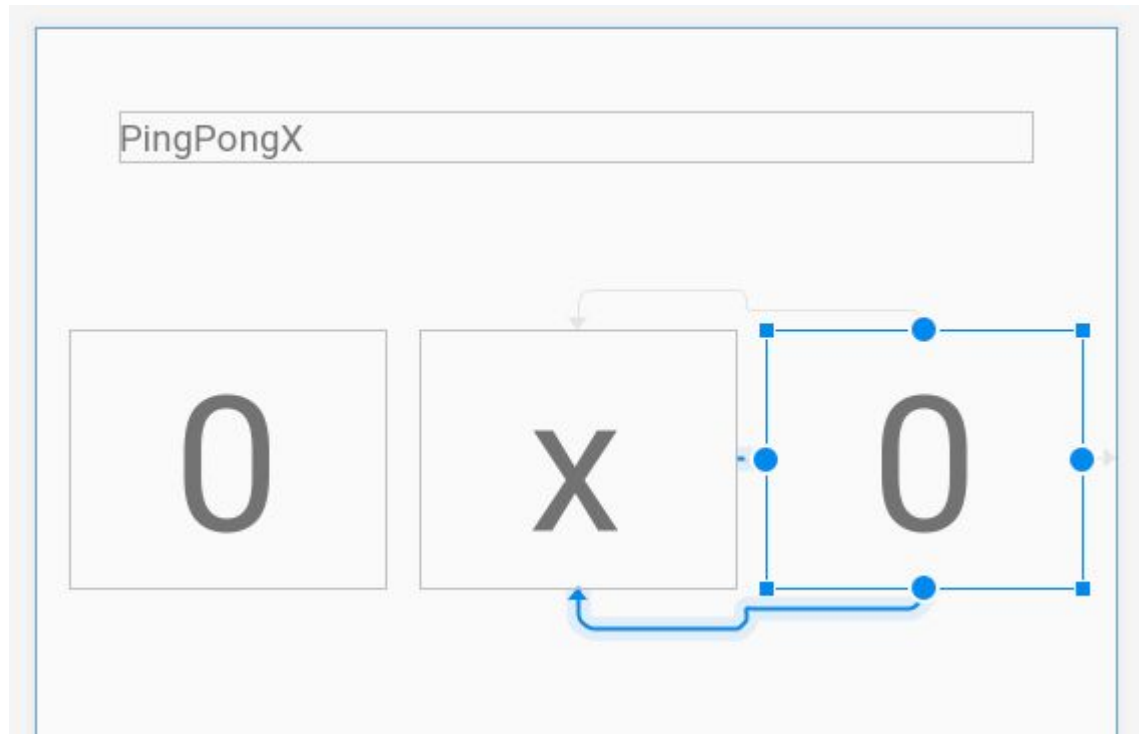
---

Teremos os seguintes resultados:



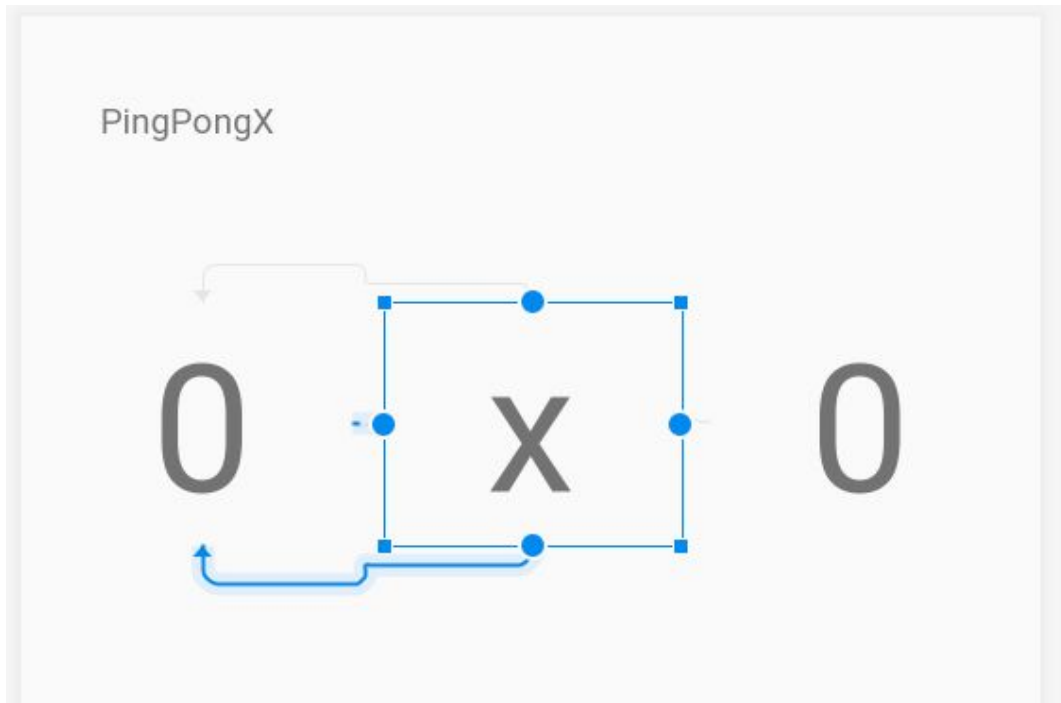
## Criando a MainActivity

Configurando as **Constraints**



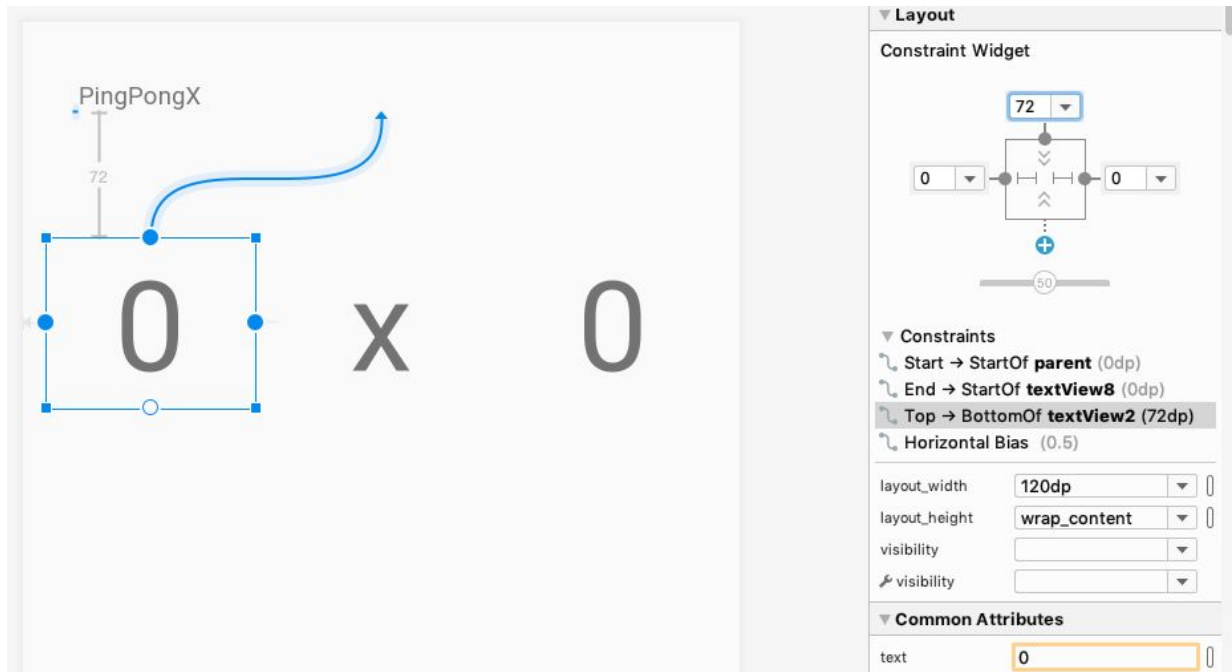
## Criando a MainActivity

Configurando as **Constraints**



## Criando a MainActivity

Configurando as **Constraints**



The screenshot displays the Android Studio interface during the configuration of a `TextView` widget. On the left, the **Layout** editor shows a visual representation of the widget with its constraints. A blue box highlights the `TextView` widget, and a blue arrow points from the `TextView` widget in the **Constraints** panel to the widget in the layout editor. The `TextView` widget is positioned at the top of the layout, with a height of 72dp. The **Constraints** panel on the right shows the following constraints:

- Start** → StartOf **parent** (0dp)
- End** → StartOf **textView8** (0dp)
- Top** → BottomOf **textView2** (72dp)
- Horizontal Bias** (0.5)

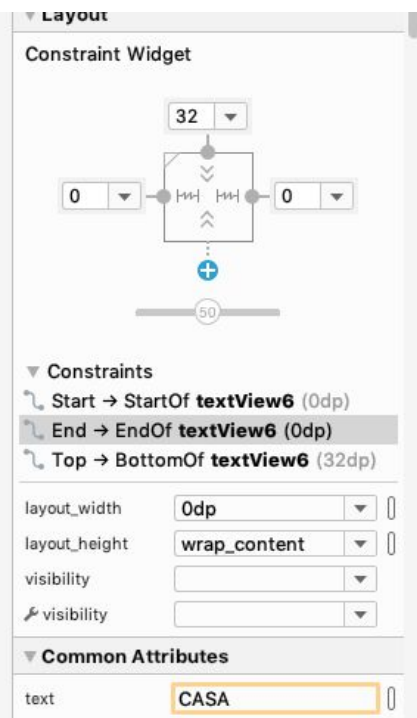
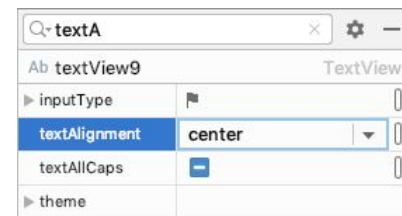
The **Common Attributes** section shows the following values:

- layout\_width**: 120dp
- layout\_height**: wrap\_content
- visibility**: (default)
- text**: 0



## Criando a MainActivity

Configurando as **Constraints**



## Criando a MainActivity

Configurando as **Constraints**

The screenshot shows the Android Studio IDE with the layout design of a PingPong game interface. The main canvas displays a title "PingPongX" at the top, followed by three large boxes containing "0", "X", and "0". Below these are two boxes labeled "CASA" and "VISITANTE". Blue lines represent constraints connecting the boxes to each other and to the parent layout. On the right, the "Layout" tab is active, showing the "Constraint Widget" with a visual representation of the constraints and a list of specific constraint rules.

**Constraint Widget**

Start → StartOf textView7 (0dp)  
End → EndOf textView7 (0dp)  
Top → TopOf textView9 (0dp)  
Bottom → BottomOf textView9 (0...

layout\_width 0dp  
layout\_height wrap\_content  
visibility  
visibility

**Common Attributes**

text VISITANTE

# Criando a MainActivity

## Configurando as **Constraints**

The screenshot shows the Android Studio IDE with the following components:

- Main Canvas:** Displays a score screen for a game called "PingPongX". It shows a score of 0 x 0. The left side is labeled "CASA" and the right side is labeled "VISITANTE". A "PONTO" (Point) box is located at the bottom left.
- Constraint Widget:** Shows a diagram of a widget with constraints. The width is set to 72dp. The height is set to 0dp. The widget is constrained to the top and bottom of the parent container.
- Constraints:** Lists the constraints applied to the widget:
  - Start → StartOf **textView6** (0dp)
  - End → EndOf **textView6** (0dp)
  - Top → BottomOf **textView9** (72dp)
- Common Attributes:** Shows the common attributes for the widget:
  - style: @android:style/
  - stateListAnimator: @android:anim/butto

# Criando a MainActivity

## Configurando as **Constraints**

The design view shows a layout for a PingPongX game. It features two large '0' digits representing scores, separated by an 'X'. Below the left '0' is the text 'CASA', and below the right '0' is 'VISITANTE'. At the bottom, there are two buttons labeled 'PONTO'. Blue lines indicate constraints: the right '0' is constrained to the top and bottom of the right 'PONTO' button, and the left '0' is constrained to the top and bottom of the left 'PONTO' button. A dashed line also connects the two 'PONTO' buttons.

The constraints view on the right shows the configuration for the right score display (textView7). The constraints are:

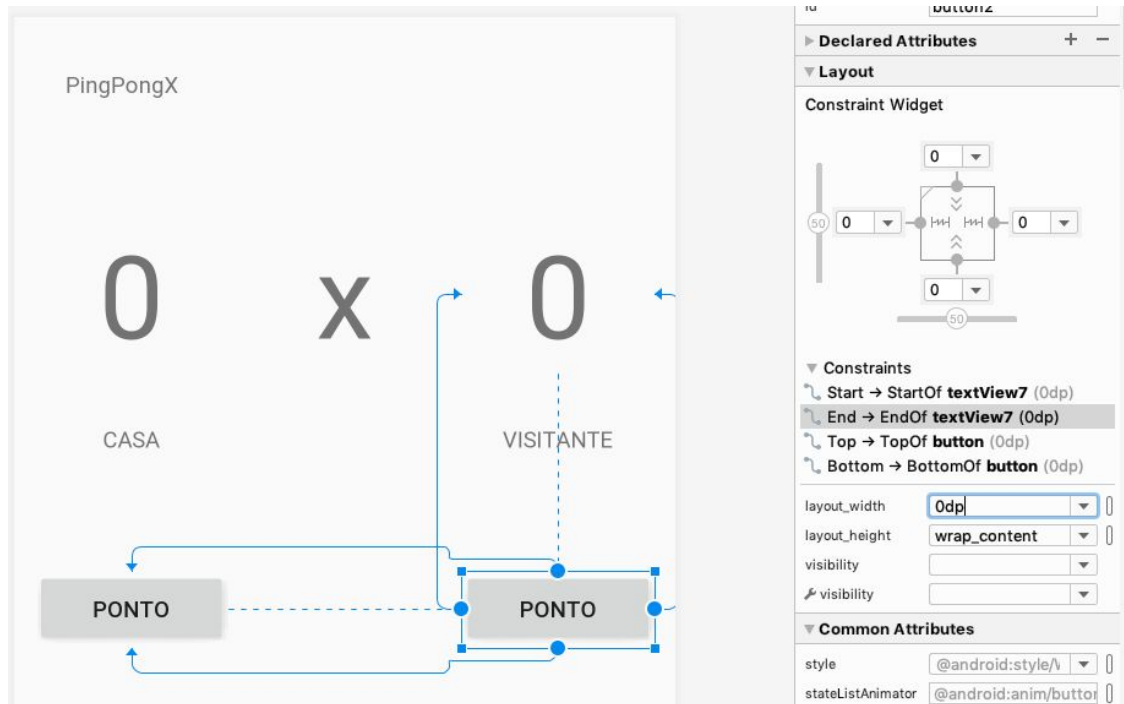
- Start → StartOf textView7 (0dp)
- End → EndOf textView7 (0dp)
- Top → TopOf button (0dp)
- Bottom → BottomOf button (0dp)

The common attributes for the widget are:

- layout\_width: wrap\_content
- layout\_height: wrap\_content
- visibility: visible
- style: @android:style/
- stateListAnimator: @android:anim/button
- onClick:

# Criando a MainActivity

## Configurando as **Constraints**



# Criando a MainActivity

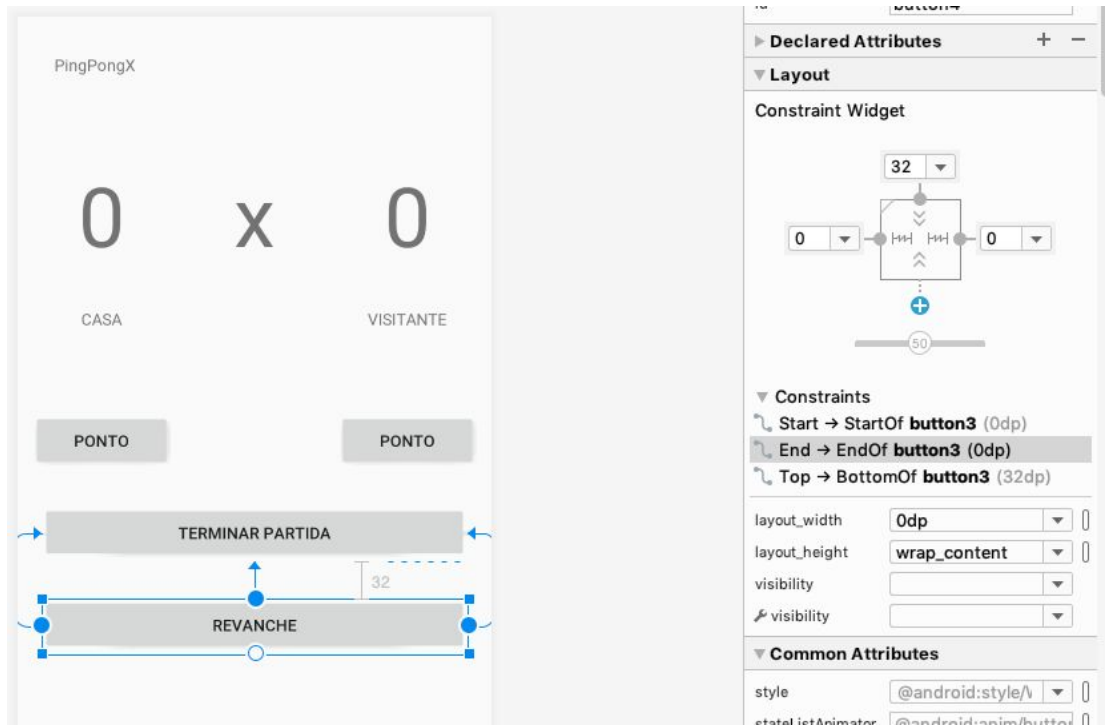
## Configurando as **Constraints**

The image displays the Android Studio interface for designing the MainActivity layout. The left pane shows a visual representation of the layout, titled "PingPongX". It features a score display "0 x 0" with "CASA" and "VISITANTE" labels below the scores. At the bottom, there is a button labeled "TERMINAR PARTIDA". The right pane shows the "Constraint Widget" configuration for the "TERMINAR PARTIDA" button. The constraints are as follows:

- Declared Attributes:** + -
- Layout:** Constraint Widget
- Constraints:**
  - Start → StartOf **button** (8dp)
  - End → EndOf **button2** (8dp)
  - Top → BottomOf **button** (32dp)
- Properties:**
  - layout\_width: 0dp
  - layout\_height: wrap\_content
  - visibility: visible

# Criando a MainActivity

## Configurando as **Constraints**



# Criando a MainActivity

## Configurando as **Constraints**



Altere os ids das views com seus respectivos nomes:

- 1 - tvPlayerOneScore
- 2 - tvPlayerTwoScore
- 3 - tvPlayerOneName
- 4 - tvPlayerTwoName
- 5 - btPlayerOneScore
- 6 - btPlayerTwoScore
- 7 - btFinishMatch
- 8 - btRevenge





## Programando nossas Activities

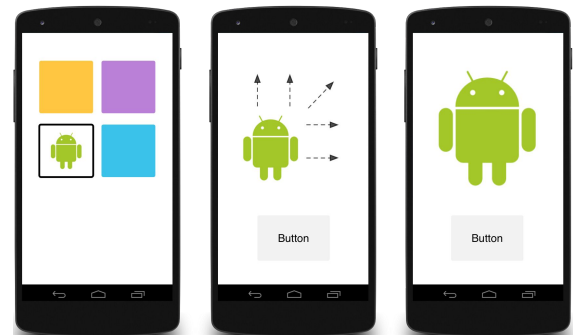


## O que são Activities?

As **Activities** são **componentes** independentes **que representam as interfaces do seu aplicativo**.

Uma **activity** nada mais é que uma classe que herda da classe **android.app.Activity** (ou alguma subclasse dela, como por exemplo a **AppCompatActivity**).

Na classe que herda Activity devemos sobrescrever o método **onCreate(Bundle bundle)**. Através desse método iremos realizar a inicialização necessária para executar nosso app, por exemplo, chamando o método **setContentView(R.layout.seulayout)** para definir a interface do usuário.





Já ouvi falar de Intents?



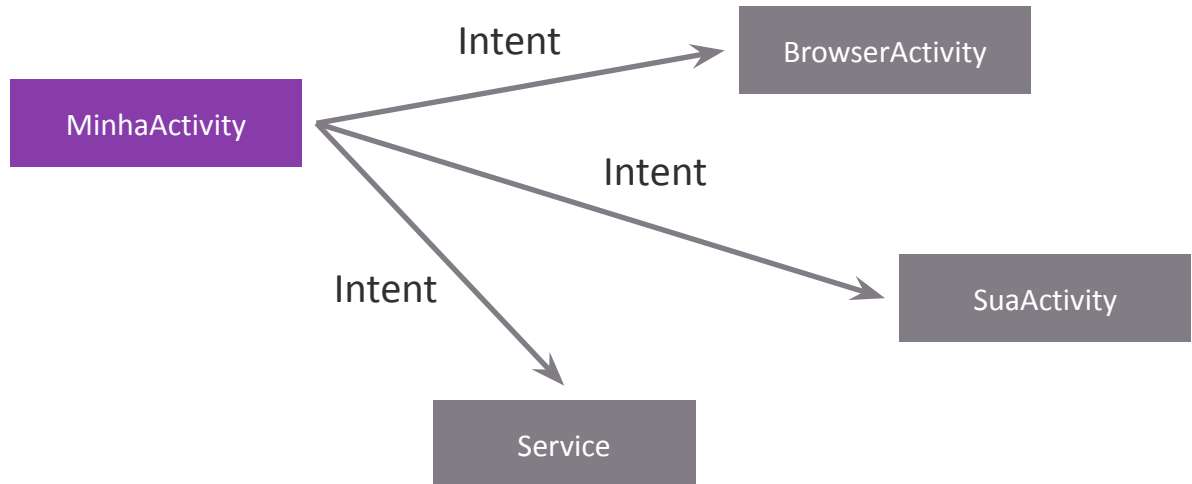
# Intents

---

**Intents** são mensagens assíncronas trocadas entre componentes de uma mesma aplicação ou entre componentes de aplicações distintas;

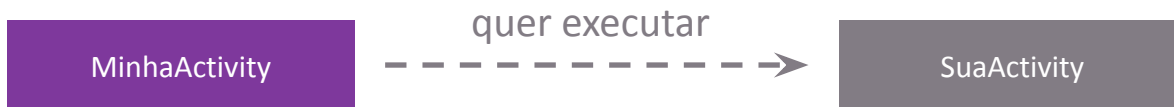
Tais componentes podem ser **Activities**, **Services** ou **Broadcast Receivers**.

Um **Intent** pode conter informações que serão utilizadas na comunicação entre dois componentes;



## Comunicação explícita entre Activities

Suponha que a **MinhaActivity** queira executar a **SuaActivity** (dentro do mesmo dispositivo):



Então, a **MinhaActivity** deverá instanciar um Intent e acionar a **SuaActivity** enviando-lhe o Intent criado:

```
// Cria o Intent
val meuIntent = Intent(this, SuaActivity::class.java)

// Inicia a Activity enviando o meuIntent
startActivity(meuIntent)
```



## Programando nossa Splash

---

Abra o arquivo **SplashActivity.kt** e adicione o seguinte código para que a tela de fique alguns minutos e, em seguida, mude para tela de configuração de jogadores.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
  
    binding = ActivitySplashBinding.inflate(layoutInflater)  
    setContentView(binding.root)  
  
    Handler(Looper.getMainLooper()).postDelayed({  
        val nextScreen = Intent(this, PlayerActivity::class.java)  
        startActivity(nextScreen)  
        finish()  
    }, 2000)  
}
```

Através do Handler estamos dizendo para trocar de tela após **2000** milissegundos.

## Programando nossa Splash

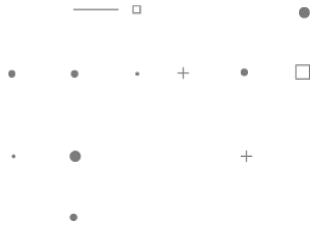
---

Através das linhas abaixo estamos criando uma intenção que será responsável em chamar a **PlayerActivity**:

```
val nextScreen = Intent(this, PlayerActivity::class.java)  
startActivity(nextScreen)
```

Estamos chamando o método finish para finalizar a **SplashActivity**:

```
finish()
```



## Como passar parâmetros através das Intents?





## Passando parâmetros entre Activities

```
val meuIntent = Intent(this, SuaActivity::class.java)

// Associa valores ao Bundle
meuIntent.putExtra("mensagem", "Boa Noite!!!! ")

startActivity(meuIntent)
```

MinhaActivity

meuIntent

SuaActivity

Bundle

Nome	Valor
Mensagem	Boa noite

```
// Obtém o Bundle
```

```
val params = intent.extras
```

```
// Obtém os valores associados ao Bundle
```

```
val msg = params?.getString("mensagem")
```

## Passando parâmetros entre Activities

Parâmetros podem ser associados ao **Intent** e encaminhados para a **Activity** destino;

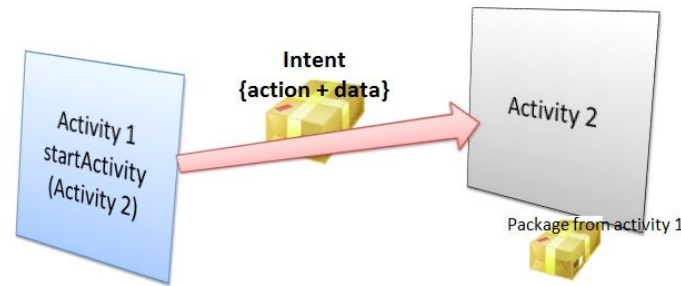
Para tanto, basta utilizar o método **putExtra** da classe Intent:

**putExtra(String P1, ? P2)**

Onde:

**P1:** nome do parâmetro

**P2:** valor do parâmetro (? indica o tipo de dados... String, int, boolean...)



## Recuperando parâmetros entre Activities

Os valores são encapsulados em um objeto do tipo Bundle que é recuperado na Activity destino conforme abaixo:

```
val params = intent.extras
```

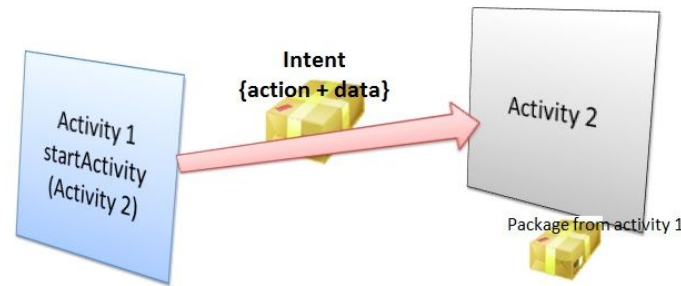
Já os valores armazenados no Bundle são recuperados pelo método. Por exemplo:

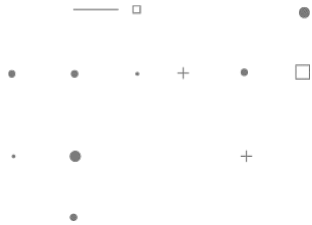
```
val nomeDoAtributo = params?.getString("nome_do_atributo")
```

Onde:

**?:** indica o tipo de dados... String, int, ...

**P1:** nome do parâmetro





## ENVIANDO OS NOMES DO JOGADORES



## Enviando os dados digitados

---

Ao clicar no botão **btStart** da **PlayerActivity** iremos enviar os nomes dos jogadores para a **MainActivity**

```
class PlayerActivity : AppCompatActivity() {
    private lateinit var binding: ActivityPlayerBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityPlayerBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btStart.setOnClickListener {
            val proximaTela = Intent(this, MainActivity::class.java)
            proximaTela.putExtra("PLAYER1", binding.etPlayer1.text.toString())
            proximaTela.putExtra("PLAYER2", binding.etPlayer2.text.toString())
            startActivity(proximaTela)
            finish()
        }
    }
}
```



## Como passar parâmetros através das Intents?



## Recuperando os dados digitados

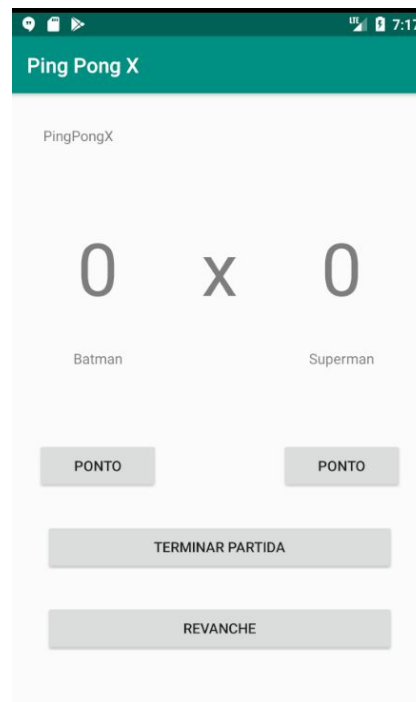
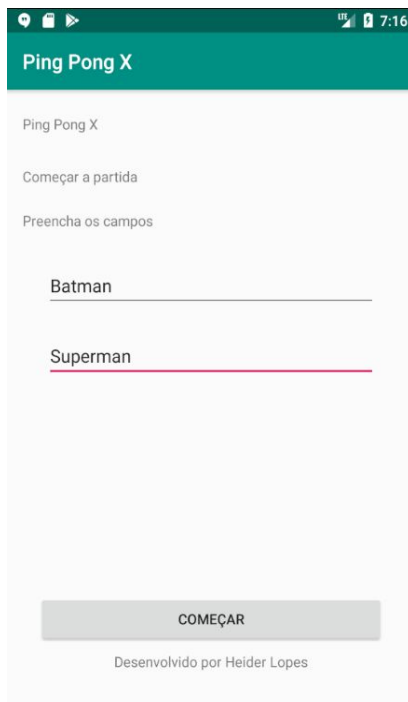
---

Na **MainActivity** adicione o seguinte código:

```
class MainActivity : AppCompatActivity() {  
  
    private lateinit var binding: ActivityMainBinding  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        binding = ActivityMainBinding.inflate(layoutInflater)  
        setContentView(binding.root)  
  
        setUpExtras(savedInstanceState)  
    }  
  
    private fun setUpExtras(savedInstanceState: Bundle?) {  
        binding.tvPlayerOneName.text = intent.getStringExtra("PLAYER1")  
        binding.tvPlayerTwoName.text = intent.getStringExtra("PLAYER2")  
    }  
}
```

## Recuperando os dados digitados

Rode a aplicação e confirme que os dados digitados na **PlayerActivity** estão sendo exibidos na **MainActivity**





## Programando o placar

---

```
private var playerOneScore = 0
private var playerTwoScore = 0

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    setUpExtras(savedInstanceState)

    setUpListeners()
}
```

## Programando o placar

---

```
private fun setUpListeners() {  
    binding.btPlayerOneScore.setOnClickListener {  
        playerOneScore++  
        setUpScorePlayerOne()  
    }  
    binding.btPlayerTwoScore.setOnClickListener {  
        playerTwoScore++  
        setUpScorePlayerTwo()  
    }  
  
    binding.btFinishMatch.setOnClickListener {  
        finish()  
    }  
  
    binding.btRevenge.setOnClickListener {  
        revenge()  
    }  
}
```

## Programando o placar

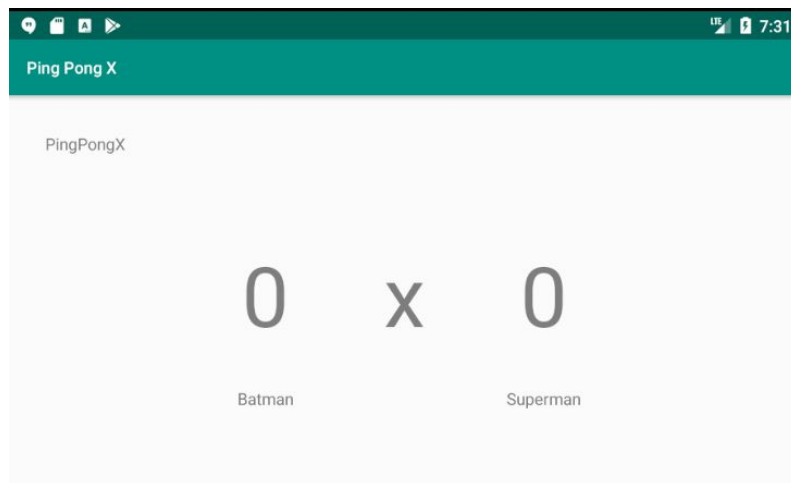
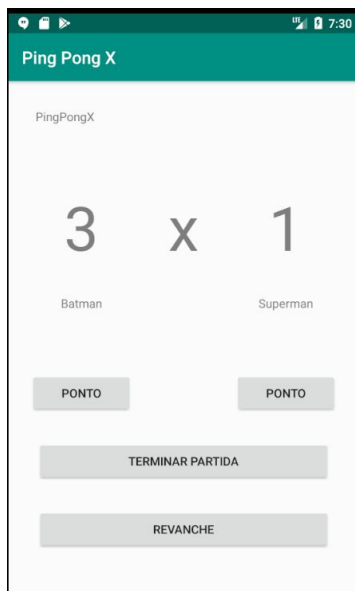
---

```
private fun setUpScorePlayerOne() {  
    binding.tvPlayerOneScore.text = playerOneScore.toString()  
}  
  
private fun setUpScorePlayerTwo() {  
    binding.tvPlayerTwoScore.text = playerTwoScore.toString()  
}  
  
private fun revenge() {  
    playerOneScore = 0  
    playerTwoScore = 0  
    setUpScorePlayerOne()  
    setUpScorePlayerTwo()  
}
```

## Recuperando os dados digitados

Tudo rodando, porém quando rotacionamos nosso device, perdemos o placar.

Para que isso não aconteça iremos salvar os valores antes da activity ser destruída e carregamos novamente quando ela for criada:



## CICLO DE VIDA DA ACTIVITY



## Ciclo de Vida das Activities

---

As Activities possuem vários estados internos, elas são criadas, iniciadas, pausadas, reiniciadas e destruídas.

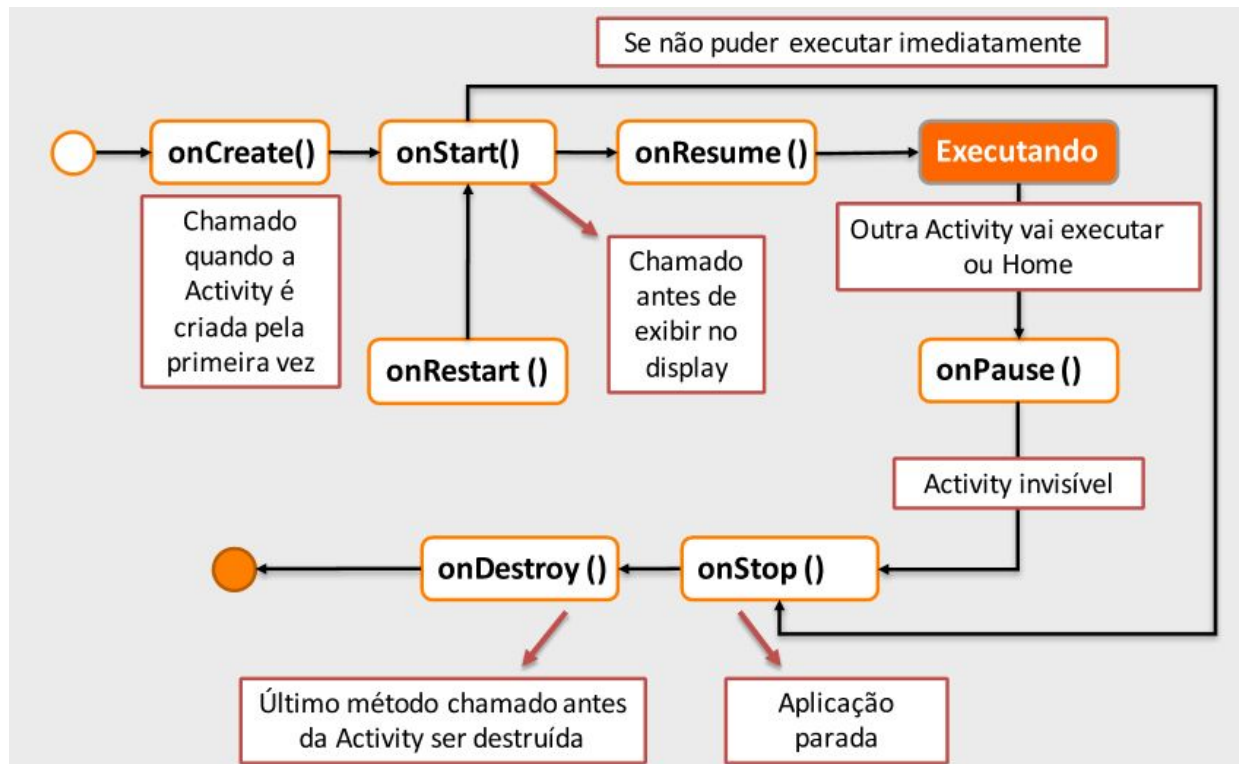
Esses eventos que se passam em uma **Activity** são conhecidos como **ciclo de vida (lifecycle)**.

Em cada ponto podemos colocar uma determinada ação.

Pode ser uma configuração quando uma Activity é iniciada ou a limpeza de variáveis e objetos quando ela é pausada.

A Activity é configurada no momento que é criada, após isso, ela é iniciada, e fica visível para o usuário, assim como quando é reiniciada após ser pausada.

## Ciclo de Vida



## Ciclo de Vida das Activities

---

**onCreate:** É o primeiro método a ser executado quando uma Activity é lançada. Geralmente utilizamos para carregar os layouts XML e outras operações de inicialização. É executado somente uma vez durante a “vida útil” da Activity.

**onStart:** É chamado imediatamente após a onCreate() – e também quando uma Activity que estava em background volta a ter foco. Podemos por exemplo, se caso estivermos usando o GPS, certificar que ele já está pronto para uso.

**onRestart:** Esse método é chamado quando uma activity foi parada temporariamente e está sendo iniciada outra vez.

**onResume:** É acionado quando a Activity se inicia e quando é reiniciada. Ele é acionado sempre que a Activity voltar para o primeiro plano. Podemos utilizá-lo para obter Intents e dados extras. A diferença entre o onStart e onResume é que o onStart() só é chamado quando a Activity não estava mais visível na tela e volta a ter o foco, enquanto o onResume() sempre é chamado nas “retomadas de foco”.



## Ciclo de Vida das Activities

---

**onPause:** é acionado, quando a Activity deixa o primeiro plano. Isso pode significar que uma janela de diálogo está sendo mostrada na tela, ou pode significar que este é o primeiro passo para que a Activity seja parada. Isso faz do onPause o lugar ideal para tarefas como parar animações, salvar dados e liberar recursos do sistema. Tudo que for liberado no método aqui deverá ser reconfigurado no método onResume. Esse método pode ser acionado se o usuário pressionar o botão Home ou o botão para sair da aplicação ou caso receba uma ligação.

**onStop:** é chamado quando a Activity não está mais visível para o usuário. Isso pode acontecer porque ela está sendo destruída ou porque outra Activity foi reiniciada e está em sua frente. Aqui é o lugar para liberar todos os recursos que não são mais utilizados pelo usuário.

**onDestroy:** é chamado quando a Activity vai ser destruída. É a última chamada que a Activity receberá antes de ser finalizada. Ele pode ser chamado automaticamente pelo sistema operacional para liberar recursos ou chamado pela própria aplicação após chamar o método finish da Activity. Depois de ter sido executado a Activity sai da pilha e seu processo é finalizado.

## O que acontece quando rotacionamos o device?

---

Quando giramos o nosso aparelho o Android **destrói a activity atual** e **recria logo em seguida**.

Ele faz isso porque precisa recriar todas as view se aplicar espaçamentos e margens para a nova orientação. Durante esse processo o Android chama o método **onSaveInstanceState(bundle)**.

Se salvarmos os valores no **Bundle** no método **onSaveInstanceState(bundle)** podemos recuperá-lo no **onCreate(bundle)**.

Se for a primeira vez que a Activity for executada esse valor é nulo (null), devemos verificar esse valor antes de tentarmos recuperar os dados, pois poderemos ter um crash no app caso tentarmos acessar um valor null.

## Recuperando os dados digitados

---

Salvando os dados

```
override fun onSaveInstanceState(outState: Bundle) {  
    super.onSaveInstanceState(outState)  
    outState.putInt("PLAYER_ONE_SCORE", playerOneScore)  
    outState.putInt("PLAYER_TWO_SCORE", playerTwoScore)  
}
```

## Recuperando os dados digitados

---

Altere o método **setUpExtras** para recuperar os valores:

```
private fun setUpExtras(savedInstanceState: Bundle?) {  
    binding.tvPlayerOneName.text = intent.getStringExtra("PLAYER1")  
    binding.tvPlayerTwoName.text = intent.getStringExtra("PLAYER2")  
  
    if(savedInstanceState != null) {  
        playerOneScore = savedInstanceState.getInt("PLAYER_ONE_SCORE")  
        playerTwoScore = savedInstanceState.getInt("PLAYER_TWO_SCORE")  
        setUpScorePlayerOne()  
        setUpScorePlayerTwo()  
    }  
}
```



## RETORNO DE EXECUÇÃO DE UMA ACTIVITY



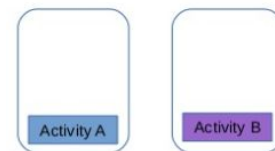
## O retorno de uma execução

Suponha agora que a **SuaActivity** já executou e um resultado deve ser retornado para a **MinhaActivity**:



Para tanto, ao iniciar o Intent é preciso utilizar o método:  
**startActivityForResult(Intent P1, int P2)**

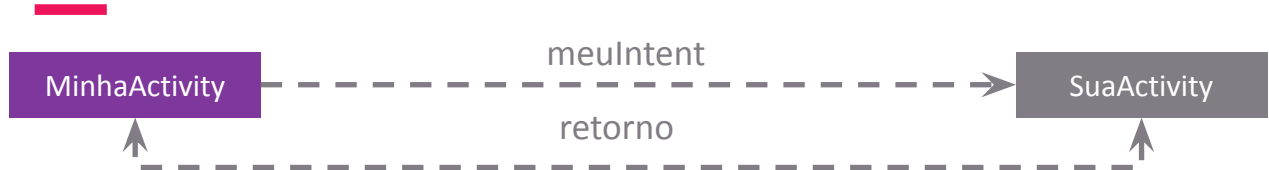
Onde **P1** é o Intent e **P2** pode ser qualquer número para identificar a requisição;



Quando o componente acionado terminar a execução o método abaixo (sobrescrevê-lo) é iniciada no componente que emitiu o Intent:

```
@Override  
public void onActivityResult(int  
    requestCode, int resultCode,  
    Intent data) {  
}
```

## O retorno de uma execução



1

```
val meuIntent = Intent (this, SuaActivity::class.java)
//Inicia a Activity esperando um retorno
startActivityForResult(meuIntent, 1)
```

2

```
override fun finish() {
    val ret = Intent()
    ret.putExtra("retorno", "Obrigado!")
    //Indica que a execução foi OK
    setResult(RESULT_OK, ret)
    super.finish()
}
```

## O retorno de uma execução

3

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (resultCode == RESULT_OK && requestCode == 1) {

        //Verifica se existe o parâmetro retorno
        if (data?.hasExtra("retorno") == true) {
            Toast.makeText(
                this,
                data?.extras?.getString("retorno"),
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}
```





## IMPLEMENTANDO O RETORNO DE UMA EXECUÇÃO



## O retorno de uma execução

---

Crie um arquivo chamado **Constants.kt** e adicione os seguintes valores:

```
const val KEY_RESULT_EXTRA_PLAYER_ONE_NAME =  
"KEY_RESULT_EXTRA_PLAYER_ONE_NAME"
```

```
const val KEY_RESULT_EXTRA_PLAYER_TWO_NAME =  
"KEY_RESULT_EXTRA_PLAYER_TWO_NAME"
```

```
const val KEY_RESULT_EXTRA_PLAYER_ONE_SCORE =  
"KEY_RESULT_EXTRA_PLAYER_ONE_SCORE"
```

```
const val KEY_RESULT_EXTRA_PLAYER_TWO_SCORE =  
"KEY_RESULT_EXTRA_PLAYER_TWO_SCORE"
```

## O retorno de uma execução

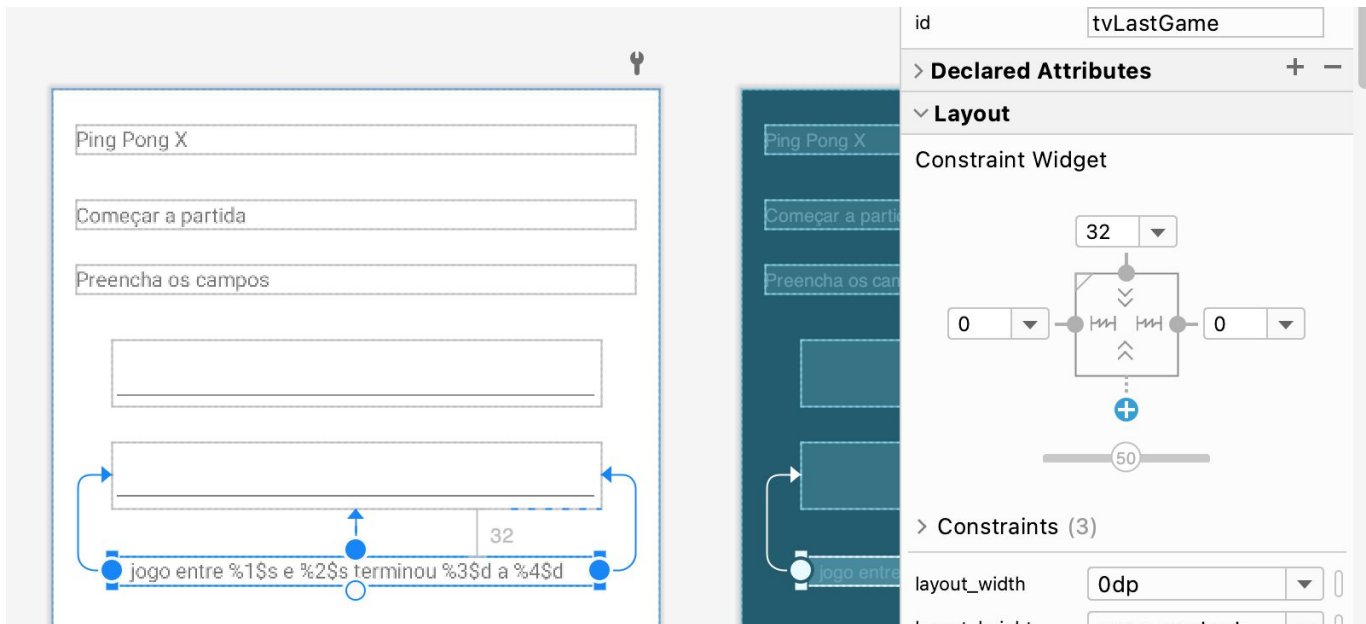
---

Abra o arquivo **strings.xml** e adicione a seguinte string:

```
<string name="message_to_share">O jogo entre %1$s e %2$s terminou  
%3$d a %4$d</string>
```

## O retorno de uma execução

Abra o arquivo **activity\_player.xml** e adicione um novo **TextView** chamado **tvLastGame**



## O retorno de uma execução

---

Abra o arquivo **PlayerActivity.kt** e adicione o seguinte código:

```
private val previewRequest =  
  
registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {  
    if (it.resultCode == RESULT_OK) {  
        val lastResult = getString(R.string.message_to_share,  
            it.data?.getStringExtra(KEY_RESULT_EXTRA_PLAYER_ONE_NAME),  
            it.data?.getStringExtra(KEY_RESULT_EXTRA_PLAYER_TWO_NAME),  
            it.data?.getIntExtra(KEY_RESULT_EXTRA_PLAYER_ONE_SCORE, 0),  
            it.data?.getIntExtra(KEY_RESULT_EXTRA_PLAYER_TWO_SCORE, 0))  
        binding.tvLastGame.text = lastResult  
    }  
}
```

## O retorno de uma execução

---

Dentro do arquivo **PlayerActivity.kt** altere o código do listener do botão **btStart**

```
binding.btStart.setOnClickListener {  
    val nextScreen = Intent(this, MainActivity::class.java)  
  
    nextScreen.putExtra(MainActivity.KEY_PLAYER1_EXTRA,  
binding.etPlayer1.text.toString())  
    nextScreen.putExtra(MainActivity.KEY_PLAYER2_EXTRA,  
binding.etPlayer2.text.toString())  
    //startActivity(nextScreen)  
  
    previewRequest.launch(nextScreen)  
}
```

## O retorno de uma execução

---

Dentro do arquivo **MainActivity.kt** altere o código do listener do botão **btFinishMatch**

```
binding.btFinishMatch.setOnClickListener {  
    val ret = Intent()  
    ret.putExtra(KEY_RESULT_EXTRA_PLAYER_ONE_NAME, binding.tvPlayerOneName.text.toString())  
    ret.putExtra(KEY_RESULT_EXTRA_PLAYER_TWO_NAME, binding.tvPlayerTwoName.text.toString())  
    ret.putExtra(KEY_RESULT_EXTRA_PLAYER_ONE_SCORE,  
binding.tvPlayerOneScore.text.toString().toInt())  
    ret.putExtra(KEY_RESULT_EXTRA_PLAYER_TWO_SCORE,  
binding.tvPlayerTwoScore.text.toString().toInt())  
    setResult(RESULT_OK, ret)  
    super.finish()  
}
```



## COMUNICAÇÃO IMPLÍCITA ENTRE ACTIVITIES





## Comunicação Implícita entre Activities

---

Alguns tipos de serviços e suas URIs correspondentes são:

Serviço	Descrição	URL
Intent.ACTION_VIEW	Visualização conteúdo	http://www.fiap.com.br geo:0,0?z=19 content://contacts/people
Intent.ACTION_CALL	Chamada telefônica	tel://(+55)1133858010
Intent.ACTION_DIAL	Discagem número	tel://(+55)1133858010
Intent.ACTION_EDIT	Edição de conteúdo	content://contacts/people/1

## Comunicação Implícita entre Activities

---

Exemplos:

**Abrir uma página web no navegador:**

```
val i = Intent(Intent.ACTION_VIEW, Uri.parse("http://www.fiap.com.br"))
startActivity(i)
```

**Abrir o WhatsApp:**

```
val sendIntent = Intent()
sendIntent.action = Intent.ACTION_SEND
sendIntent.putExtra(Intent.EXTRA_TEXT, message)
sendIntent.type = "text/plain"
sendIntent.setPackage("com.whatsapp")
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

## Intents Implícitos

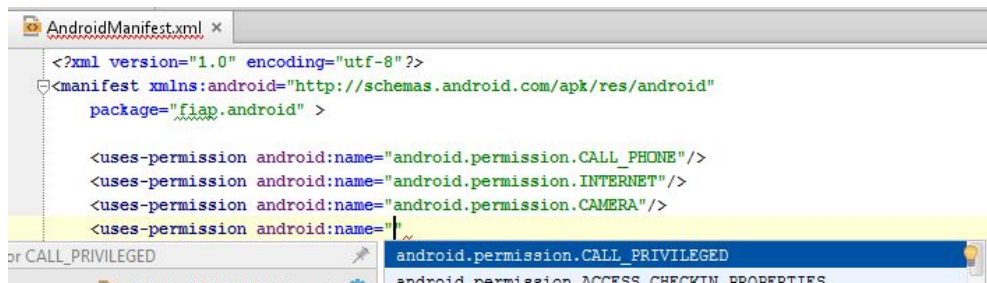
Para executar determinadas ações é necessário que sejam configuradas algumas permissões no AndroidManifest.xml, entre as tags <manifest>...</manifest>;

Exemplo:

```
<uses-permission android:name="android.permission.CALL_PRIVILEGED" />
<uses-permission android:name="android.permission.CALL_PHONE"/>

<uses-permission android:name="android.permission.INTERNET"/>

<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```



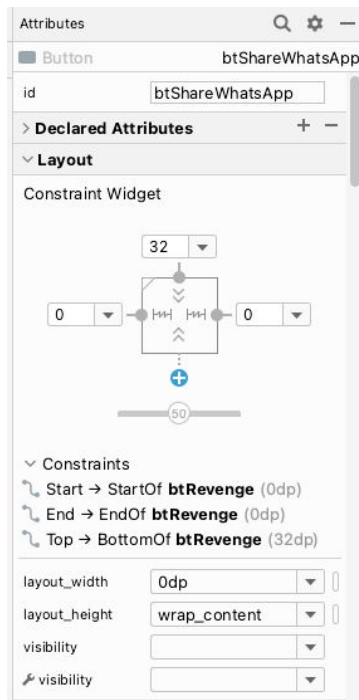


## COMPARTILHANDO COM O WHATSAPP



## COMPARTILHANDO COM O WHATSAPP

Adicione um novo botão no layout **activity\_main.xml**



## COMPARTILHANDO COM O WHATSAPP

---

Adicione o método para compartilhar com o **WhatsApp**:

```
private fun shareWhatsApp () {  
    try{  
        val whatsappIntent = Intent(Intent.ACTION_SEND)  
        whatsappIntent.type = "text/plain"  
        whatsappIntent.setPackage("com.whatsapp")  
  
        val message = getString(R.string.message_to_share,  
            binding.tvPlayerOneName.text,  
            binding.tvPlayerTwoName.text,  
            playerOneScore,  
            playerTwoScore)  
  
        whatsappIntent.putExtra(Intent.EXTRA_TEXT, message)  
        startActivity(whatsappIntent)
```

## COMPARTILHANDO COM O WHATSAPP

---

Adicione o método para compartilhar com o **WhatsApp**:

```
} catch (e: ActivityNotFoundException) {  
    //Toast.makeText(this, "WhatsApp não instalado",  
    Toast.LENGTH_LONG).show()  
    val appPackageName = "com.whatsapp"  
    try {  
        startActivity(Intent( Intent.ACTION_VIEW,  
Uri.parse("market://details?id= $appPackageName")))  
        } catch (anfe: android.content.ActivityNotFoundException) {  
            startActivity(Intent( Intent.ACTION_VIEW,  
Uri.parse("https://play.google.com/store/apps/details?id= $appPackageN  
ame")))  
        }  
  
    }  
}
```

## COMPARTILHANDO COM O WHATSAPP

---

Implemente a chamada no clique do botão dentro da função **setUpListeners()**

```
binding.btShareWhatsApp.setOnClickListener {  
    shareWhatsApp()  
}
```





## EXERCÍCIO I



## Exercício

Altere o layout do aplicativo para que ele fique conforme as imagens abaixo:



## Adicionando as cores do aplicativo

---

Abra o arquivo **colors.xml** e adicione as seguintes cores:

```
<?xml version="1.0" encoding="utf-8" ?>
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>

    <color name="background_home">#FF0053</color>
    <color name="background_away">#16FAE8</color>

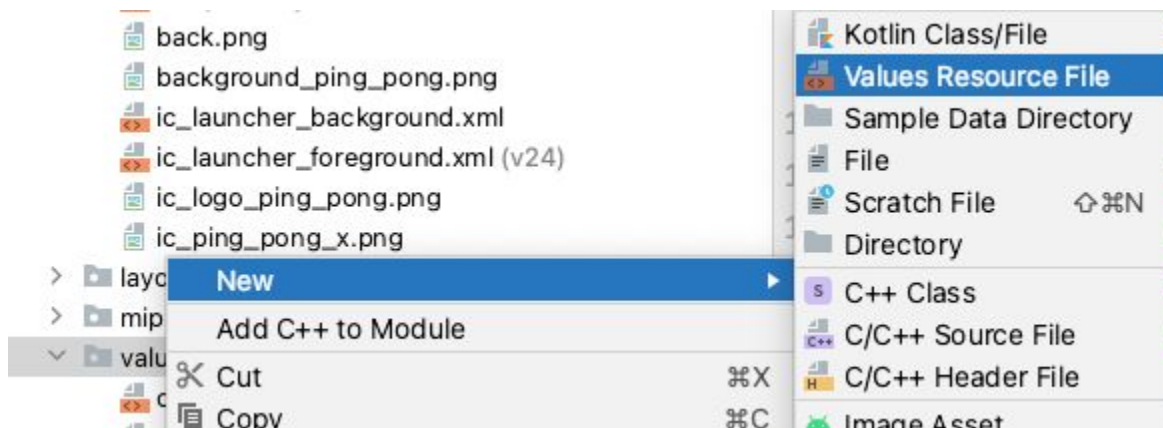
    <color name="title_text_color">#959595</color>

    <color name="input_text_color">#FFFFFF</color>
    <color name="input_hint_text_color">#EEEEEE</color>

    <color name="button_text_color">#FFF</color>
</resources>
```

## Criando o arquivo dimens.xml

Dentro da pasta **res** → **values** crie um arquivo chamado **dimens.xml**



File name:	<input type="text" value="dimens"/>
Root element:	<input type="text" value="resources"/>

## Criando o arquivo dimens.xml

---

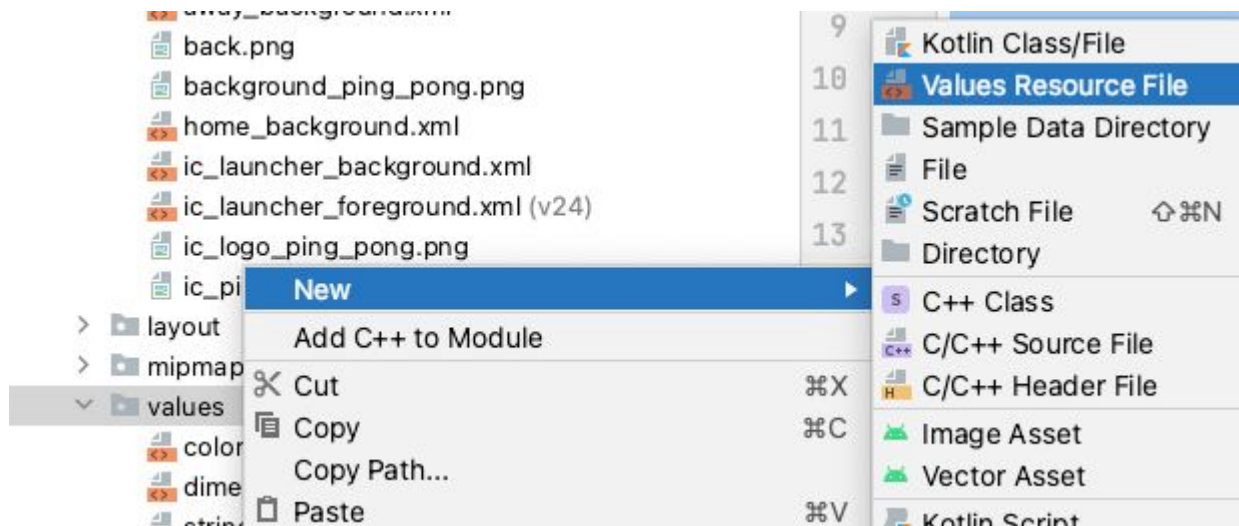
Adicione as seguintes dimensões:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <dimen name="radius_input">4dp</dimen>

    <dimen name="padding_horizontal_input">8dp</dimen>
    <dimen name="padding_vertical_input">16dp</dimen>
</resources>
```

## Criando o styles

Dentro da pasta **values** crie um arquivo chamado **styles.xml**



File name: styles

Root element: resources

## Criando o styles

---

Adicione o seguinte código ao **styles.xml**

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="FullScreen" parent="AppTheme">
        <item name="windowActionBar">false</item>
        <item name="windowNoTitle">true</item>
        <item name="android:windowFullscreen">true</item>
    </style>
</resources>
```

## Deletando o styles atual

---

Remove a os themes do aplicativo selecionando-o e clicando em **delete**





## Aplicando o styles

---

Abra o arquivo **AndroidManifest.xml** e aplique o tema **FullScreen**

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/FullScreen">
```

## CRIANDO MAIS STYLES

---

Abra o arquivo **styles.xml** e adicione o seguinte código:

```
<style name="title">
    <item name="android:textSize">28sp</item>
    <item name="android:textColor">@color/title_text_color</item>
</style>

<style name="subtitle">
    <item name="android:textSize">22sp</item>
    <item name="android:textColor">@color/title_text_color</item>
    <item name="android:textStyle">bold</item>
</style>

<style name="description">
    <item name="android:textSize">20sp</item>
    <item name="android:textColor">@color/title_text_color</item>
</style>
```

## CRIANDO MAIS STYLES

---

Abra o arquivo **activity\_player.xml** e adicione o seguinte código (mantenha os demais):

```
<TextView
    style="@style/title"
    android:text="Ping Pong X"

/>

<TextView
    style="@style/subtitle"
    android:text="Começar a partida"

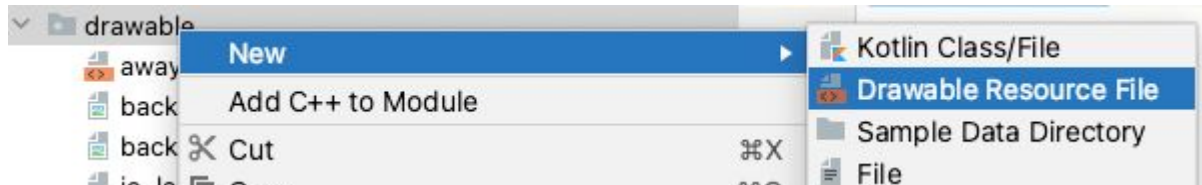
/>

<TextView
    style="@style/description"
    android:text="Preencha os campos"

/>
```

## CRIANDO O HOME BACKGROUND

Dentro da pasta **res** → **drawable** crie um arquivo chamado **home\_background.xml**



File name:

Root element:

## CRIANDO O HOME BACKGROUND

---

Adicione o seguinte código ao arquivo **home\_background.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">

    <solid
        android:color="@color/background_home"/>

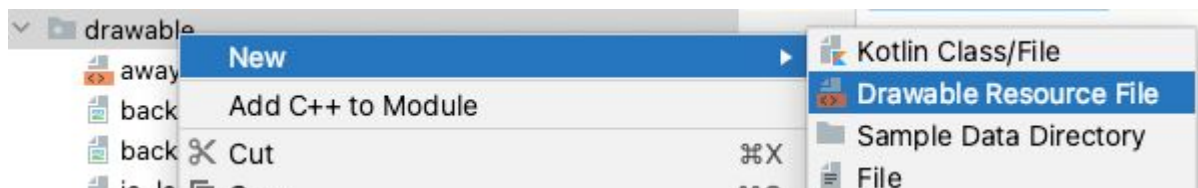
    <corners
        android:radius="@dimen/radius_input"/>

    <padding
        android:left="@dimen/padding_horizontal_input"
        android:right="@dimen/padding_horizontal_input"
        android:top="@dimen/padding_vertical_input"
        android:bottom="@dimen/padding_vertical_input"/>

</shape>
```

## CRIANDO O AWAY BACKGROUND

Dentro da pasta **res** → **drawable** crie um arquivo chamado **away\_background.xml**



File name:	<input type="text" value="away_background"/>
Root element:	<input type="text" value="selector"/>

## CRIANDO O AWAY BACKGROUND

---

Adicione o seguinte código ao arquivo **home\_background.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android">

    <solid
        android:color="@color/background_home"/>

    <corners
        android:radius="@dimen/radius_input"/>

    <padding
        android:left="@dimen/padding_horizontal_input"
        android:right="@dimen/padding_horizontal_input"
        android:top="@dimen/padding_vertical_input"
        android:bottom="@dimen/padding_vertical_input"/>

</shape>
```

## CRIANDO OS STYLES DOS INPUTS

---

Abra o arquivo **styles.xml** e adicione o seguinte código:

```
<style name="home_input">
    <item name="android:background">@drawable/home_background</item>
    <item name="android:textColor">@color/input_text_color</item>
    <item
name="android:textColorHint">@color/input_hint_text_color</item>
</style>

<style name="away_input">
    <item name="android:background">@drawable/away_background</item>
    <item name="android:textColor">@color/input_text_color</item>
    <item
name="android:textColorHint">@color/input_hint_text_color</item>
</style>
```



## APLICANDO OS STYLES DOS INPUTS

---

Abra o arquivo **activity\_player.xml** e adicione o seguinte código. Adicione o style e mantenha as demais configurações já criadas:

```
<EditText  
    style="@style/home_input"  
    //...  
>
```

```
<EditText  
    style="@style/away_input"  
    //...  
>
```

## CRIANDO O BOTÃO

---

Crie um novo arquivo dentro da pasta **drawable** chamado **custom\_button.xml**



File name:

# CRIANDO O BOTÃO

---

Adicione o seguinte código ao **custom\_button.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android" >

    <!--<item android:drawable="@drawable/home_background"
        android:state_enabled="true" android:state_pressed="true"
    />-->

    <item android:state_pressed="true">
        <shape>
            <solid android:color="#E5FF00"/>

            <corners android:radius="4dp"/>

            <padding android:left="8dp"
                android:right="8dp"
                android:top="12dp"
                android:bottom="12dp"/>
        </shape>
    </item>
```

## CRIANDO O BOTÃO

---

Adicione o seguinte código ao **custom\_button.xml**:

```
<item>
    <shape>
        <gradient
            android:startColor="#E5FF00"
            android:endColor="#51571B"
            android:angle="270"/>

        <corners android:radius="4dp"/>

        <padding android:left="8dp"
            android:right="8dp"
            android:top="12dp"
            android:bottom="12dp"/>
    </shape>
</item>

</selector>
```

# CRIANDO O BOTÃO

---

Abra o arquivo **styles.xml** e adicione o seguinte código:

```
<style name="custom_button">
    <item name="android:background">@drawable/custom_button</item>
    <item name="android:textColor">@color/button_text_color</item>
</style>
```

## APLICANDO O ESTILO AO BOTÃO

---

Abra o arquivo **activity\_player.xml** e aplique o estilo do botão.

```
<Button
    style="@style/custom_button"
    android:id="@+id/btStart"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:text="Começar"
    app:layout_constraintBottom_toTopOf="@+id/textView4"
    app:layout_constraintEnd_toEndOf="@+id/textView4"
    app:layout_constraintStart_toStartOf="@+id/textView4" />
```

## EXERCICIO: MAIN ACTIVITY

Agora é com você criar os botões e formatações necessárias para criar a tela do jogo conforme à imagem ao lado:





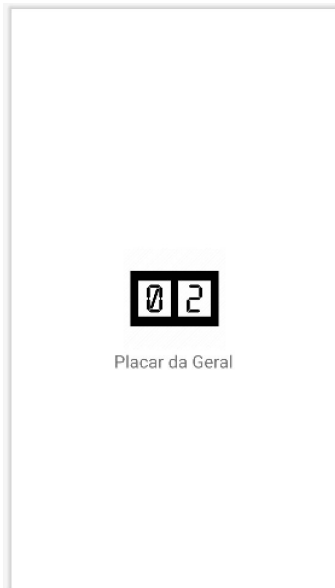
## EXERCÍCIO II



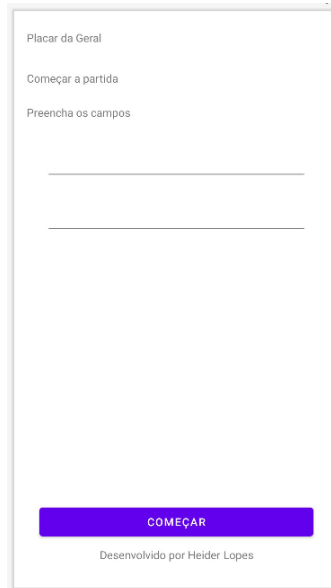


## Exercício II

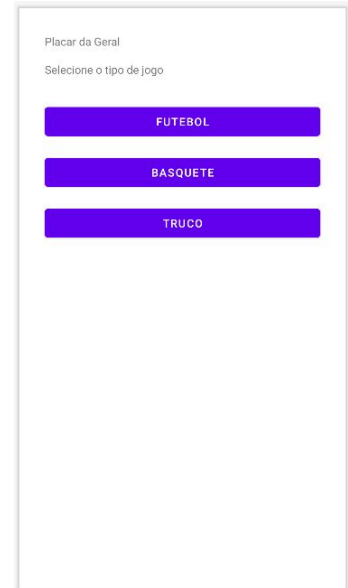
Crie um novo aplicativo chamado **Placar da Geral**. Este aplicativo deverá constar:



SplashActivity



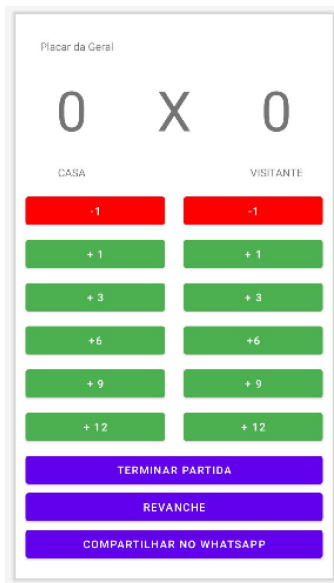
PlayerActivity



MenuActivity

## Exercício II

Crie um novo aplicativo chamado **Placar da Geral**. Este aplicativo deverá constar:



TrucoScoreActivity



BasketScoreActivity



FootballScoreActivity

## Exercício II

---

Implemente os códigos referentes às activities criadas

Customize o layout para dar uma melhor aparência e usabilidade para o usuário:

# OBRIGADO



/heider.lopes



/in/heider-lopes-a06b2869/

FIAP

Copyright © 2019 | Professor (a) Heider Lopes

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

FIAP