**Aula XIV e XVI de Desenvolvimento de Aplicativo I**

**Revisão de Layout**

**Continuação do projeto Calculadora**

**Classe R - Android**

**Método findViewById();**

**Listener de Eventos**

**Métodos de Eventos**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Fontes:**

[Criar seu primeiro app | Desenvolvedores Android](https://developer.android.com/training/basics/firstapp?hl=pt-br)

[Criar uma IU simples | Desenvolvedores Android](https://developer.android.com/training/basics/firstapp/building-ui?hl=pt-br)

[Tutorial de Android Studio: Introdução completa ao Android Studio](https://www.devmedia.com.br/tutorial-de-android-studio/34003)

[Visão geral dos recursos de aplicativo | Desenvolvedores Android](https://developer.android.com/guide/topics/resources/providing-resources?hl=pt-br)

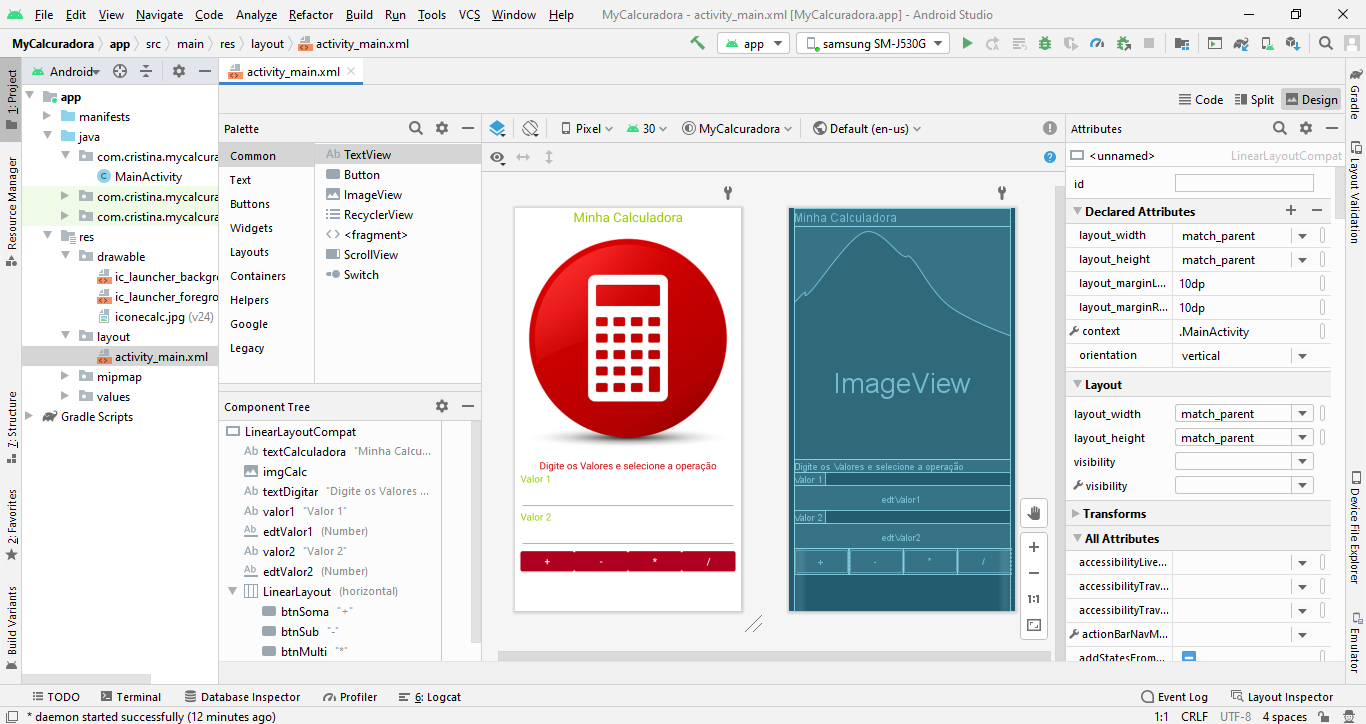
[Manipulando Views com Android](https://tableless.com.br/manipulando-views-com-android/)

<https://developer.android.com/guide/topics/ui/ui-events?hl=pt-br#:~:text=Um%20listener%20de%20eventos%20%C3%A9,com%20o%20item%20na%20IU>.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

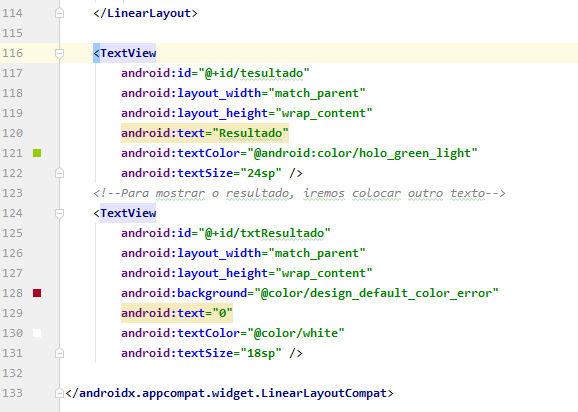
**Continuação do projeto Calculadora**

Vamos dar continuidade ao projeto da calculadora, na aula de ontem, finalizamos o nosso tutorial da seguinte forma:

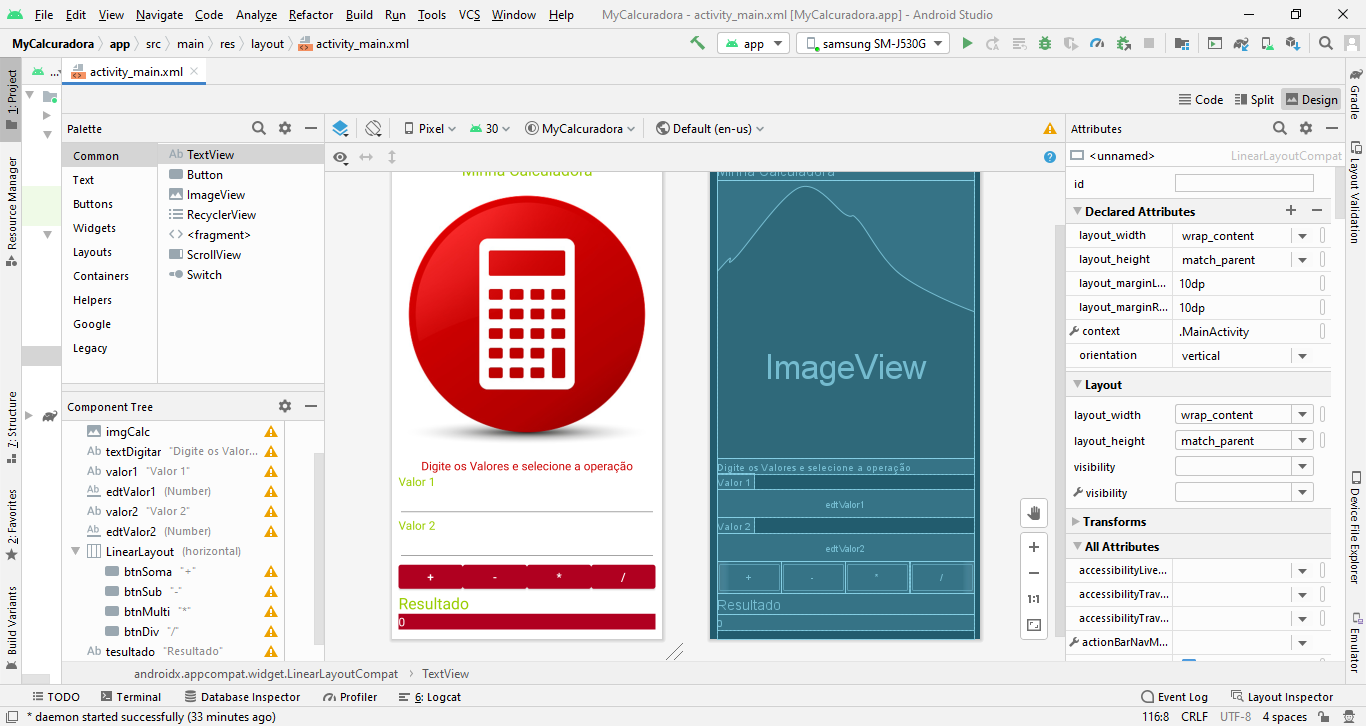


Observação: o tutorial está em cor diferente do documento anterior porque estes prints foram feitos em aula e este app ficou mais legal que o original ….. ;)

Na próxima etapa iremos inserir mais um campo de texto para o resultado e para facilitar a nossa vida, iremos fazer via código de acordo com a imagem abaixo! Observem os nomes, eles serão muito importantes.



**Visualização do nosso App!**

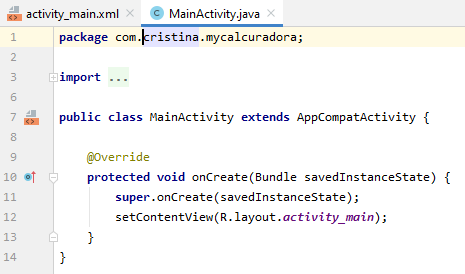


Ainda é possível colocar **margin\_top** ou **bottom** para configurar o distanciamento dos componentes inseridos no layout do app, ou ainda: **zerar todos os atributos layout\_height e configurar o layout\_weight por valores, tipo 2.4, 3.5 …. desta forma estaremos configurando as alturas pela proporção, o que é aconselhável para controle de tamanho de imagens.**

**Classe R - Android**

**Afinal, qual é a classe R no Android?**

No Android Studio, quando nós criamos um projeto usando uma empty activity, o AS gera o seguinte código!



**A classe R** **é uma classe de mapeamento e/ou setamento que cria objetos de acesso para todos os componentes/objetos do nosso layout no app.** Não é porque criamos os componentes no layout que não precisaremos criar no Java também! Nesta criação é construído um vínculo entre o elemento construído no design e mapeado em nós no xml e o Java, portanto, **a classe R é responsável por criar uma “interface de controle” ou comunicação entre as estruturas de um projeto Android - os arquivos xml e Java!**

**Classe R** contém as definições de todos os recursos de um pacote de aplicativo específico . Ele está no namespace do pacote do aplicativo.

Geralmente, há duas R classes com as quais trabalhamos:

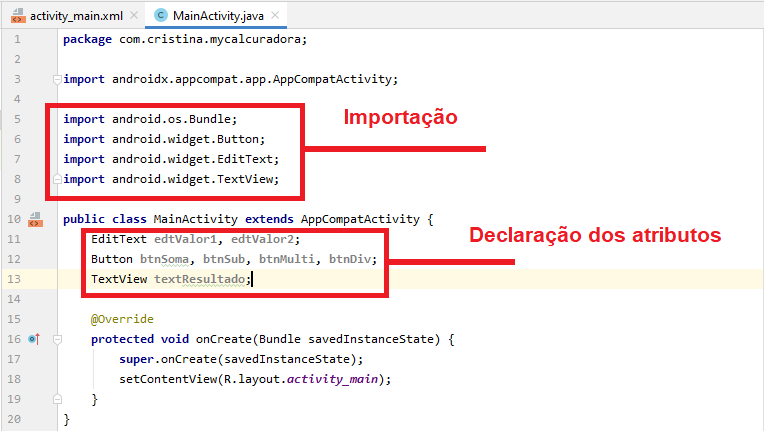
1. **Os recursos da estrutura em android.Re**
2. **Seu próprio namespace**

**Vamos para a classe R!**

Alternamos para o modo Code e vamos acessar o arquivo **MainActivity.java**

**1ª Parte!**

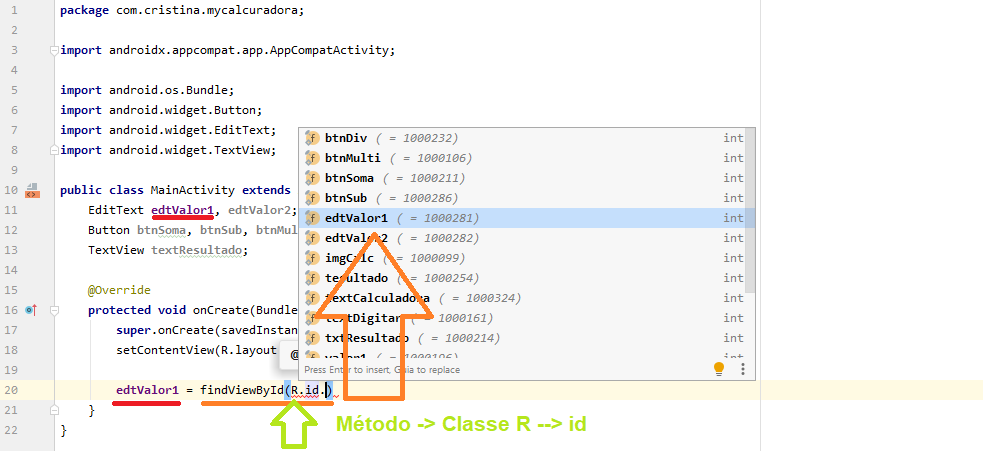
Declarando os atributos Java e importando as classes destes dados.



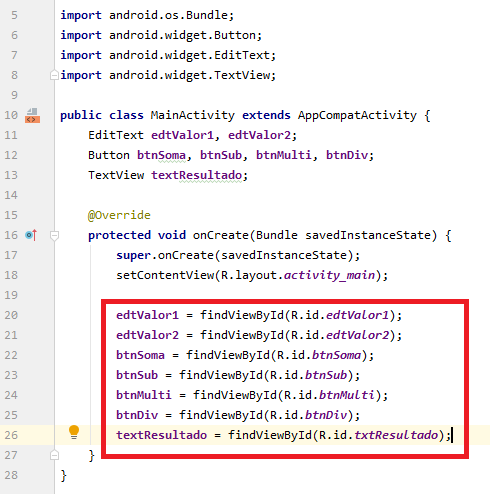
**Agora o mapeamento dos objetos criados no xml e no java!**

**2ª Parte!**

Pois então! Todo o componente criado no Java precisa ser mapeado para o xml que é o nosso layout, por isso a importância de definirmos os **“IDs”** de forma correta e adequada ao que cada elemento significa na aplicação. Vamos usar o método **findViewById();** Este método tem a função de achar a view do nosso layout e associar (mapear) ao objeto Java criado!



Vamos repetir para cada objeto criado no Java! Observe que as cores dos objetos criados em Java mudaram de cor após o mapeamento! Desta forma, após este mapeamento, a aplicação saberá que quando o componente de layout for “chamado” por uma ação do usuário, ele tem a referência do objeto Java especificado.



**3ª Parte!**

**Configuração dos Listener de Eventos e os Métodos de Eventos.**

**Listener** define sempre quando algum componente está **"à espera" da ação que é o evento promovido pelo usuário que está manipulando o app que desenvolvemos.**

**Os métodos de eventos** podem se resumir a:

**onClick()**

De **View.OnClickListener.** É chamado quando o usuário toca no item (no modo de toque) ou foca no item com as teclas de navegação ou o trackball e pressiona a tecla "Enter" adequada ou pressiona o trackball.

**onLongClick()**

De **View.OnLongClickListener**. É chamado quando o usuário mantém o item pressionado (no modo de toque) ou foca no item com as teclas de navegação ou o trackball e mantém pressionada a tecla "Enter" adequada ou mantém o trackball pressionado (por um segundo).

**onFocusChange()**

De **View.OnFocusChangeListener.** É chamado quando o usuário navega para ou do item usando as teclas de navegação ou o trackball.

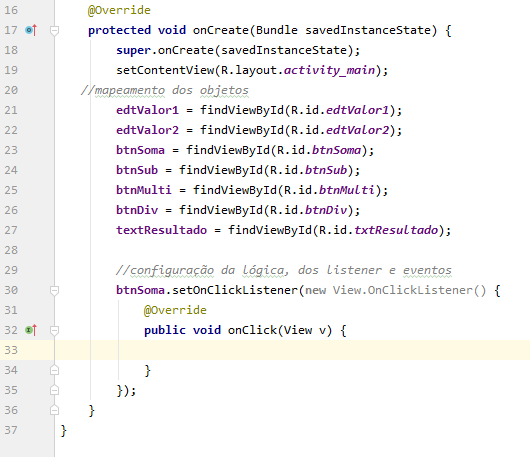
**onKey()**

De **View.OnKeyListener.** É chamado quando o usuário está com foco no item e pressiona ou solta uma tecla de hardware no dispositivo.

**onTouch()**

De **View.OnTouchListener.** É chamado quando o usuário realiza uma ação qualificada como um evento de toque, incluindo o pressionamento, a liberação ou qualquer outro gesto de movimento na tela (dentro dos limites do item).

(**fonte:** <https://developer.android.com/guide/topics/ui/ui-events?hl=pt-br#:~:text=Um%20listener%20de%20eventos%20%C3%A9,com%20o%20item%20na%20IU>. )



Na imagem acima temos a configuração do setOnClickListener(). Vamos repetir a mesma codificação para os demais componentes, de acordo com a imagem abaixo.

Você irá utilizar este bloco quando você usar algum método que lance uma CheckedException e quando você quer dar algum tratamento a exception.

Um bloco try é chamado de bloco protegido porque, caso ocorra algum problema com os comandos dentro do bloco, a execução desviará para os blocos catch correspondentes.

Necessitamos usar try, porque estamos fazendo operação de conversão, é uma maneira mais robusta de tratar possíveis erros no momento da conversão, por exemplo, não é possível converter um caractere ? por um número, porém como a entrada de dados é liberada o usuário final poderá digitar algo inadequado, resultando em erro e quebra da execução do programa por falha, com o try podemos evitar esta queda brusca e então tratar o erro da melhor forma.

Fonte: [Try/Catch: Blocos Try/Catch no Java](https://www.devmedia.com.br/blocos-try-catch/7339)