



Desenvolvedor Android  
**Algoritmos em Kotlin**

**Android Studio IDE**  
Projeto no GitHub

Vinícius **Thiengo**

# Porque e Como Utilizar Vetores no **Android**

[Thiengo.com.br](http://Thiengo.com.br)

# O porquê das imagens vetoriais

Primeiro é importante saber que imagens vetoriais não foram criadas devido ao Android.

Elas são bem mais antigas e a popularidade delas em desenvolvimento de software, principalmente Web (aqui no formato SVG - *Scalable Vector Graphics*), se iniciou no final da década de 90.

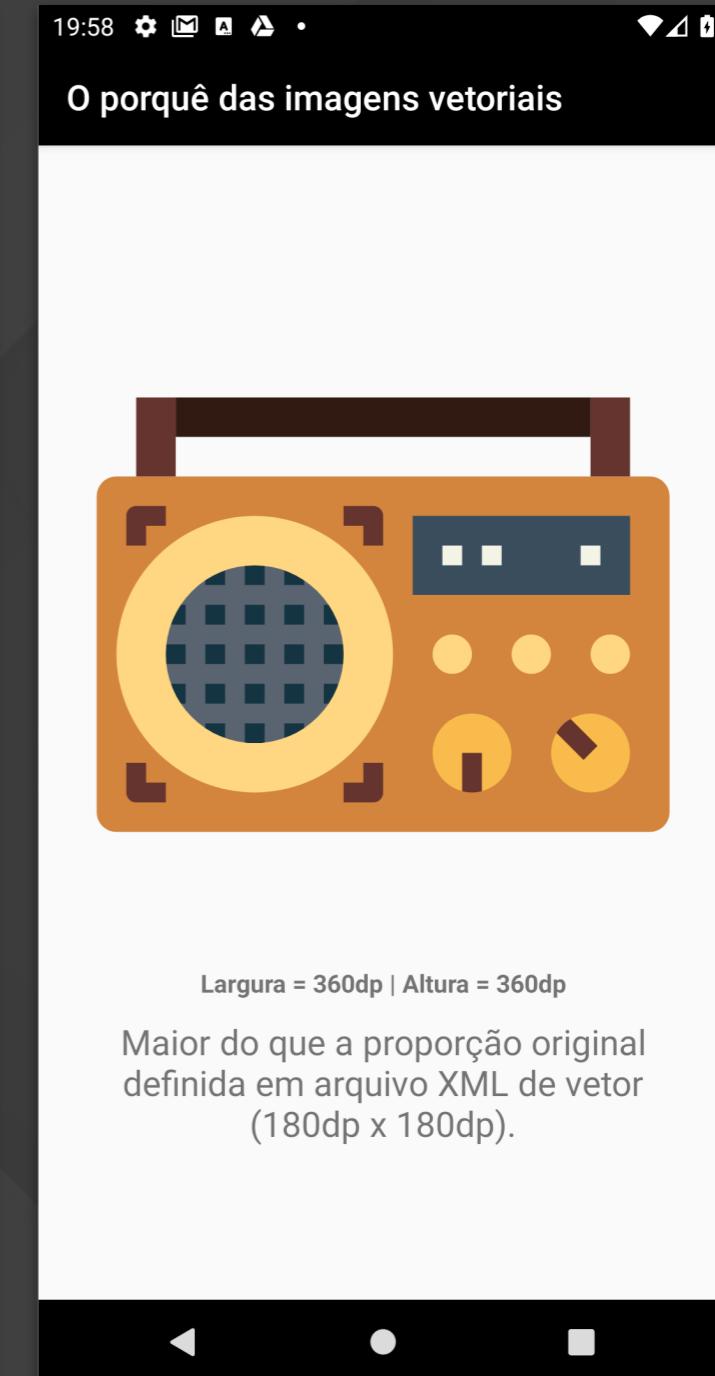
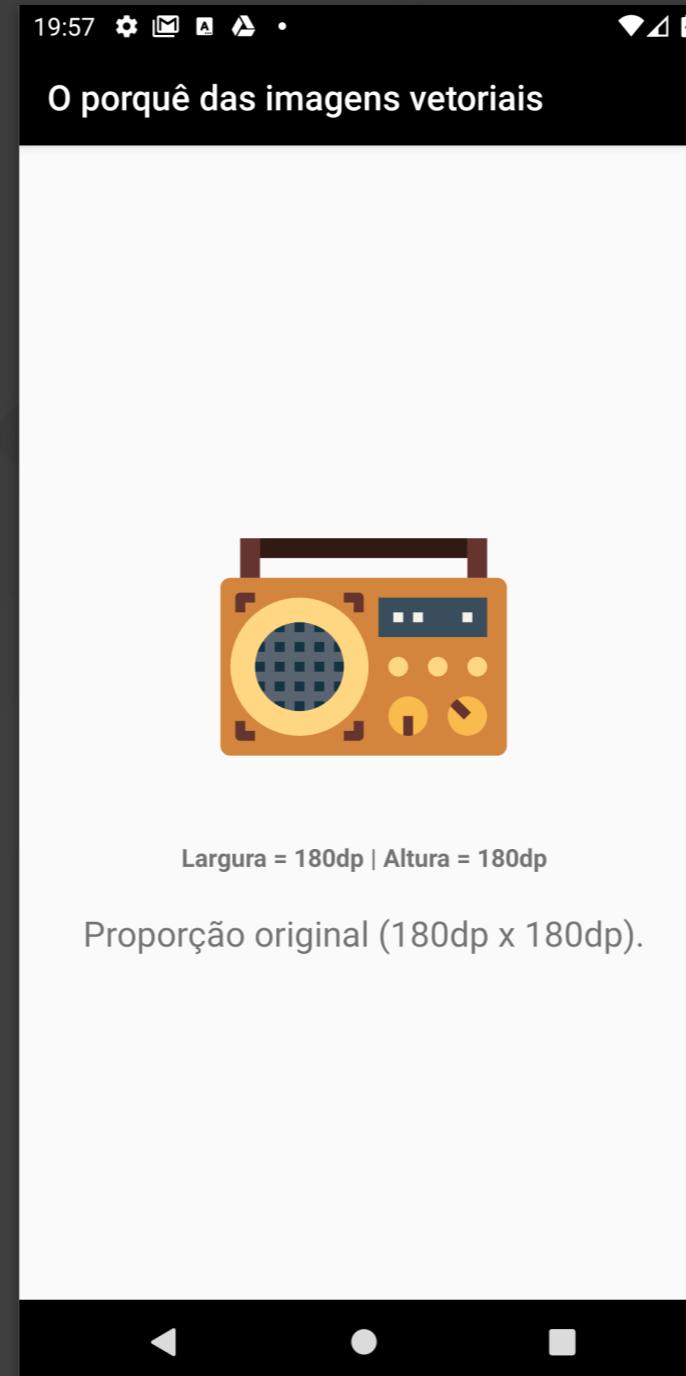
Mais precisamente em 1999, com a possibilidade de uso de vetores também em páginas Web.

O livro "Fundamentos da SVG" de Maurício Samy Silva (o Maujor) apresenta bem a história das imagens vetoriais. Vale a leitura se você estiver com um tempo extra.

A real "força" (popularidade) das imagens vetoriais vem principalmente devido à capacidade de escalabilidade sem perda de qualidade.

Ou seja, uma mesma imagem vetorial pode ser utilizada em diferentes tamanhos que mesmo assim a qualidade de visualização será a mesma.





Algo que contrasta muito quando comparando imagens vetoriais às imagens rasterizadas (JPEG, PNG, GIF, MPEG4, ...). Imagens rasterizadas que são os conhecidos bitmap (mapa de bits).



As *raster images* quando escaladas para abaixo das proporções originais... os olhos humanos praticamente não detectam problemas na visualização, pois estas imagens têm "perda de informação".

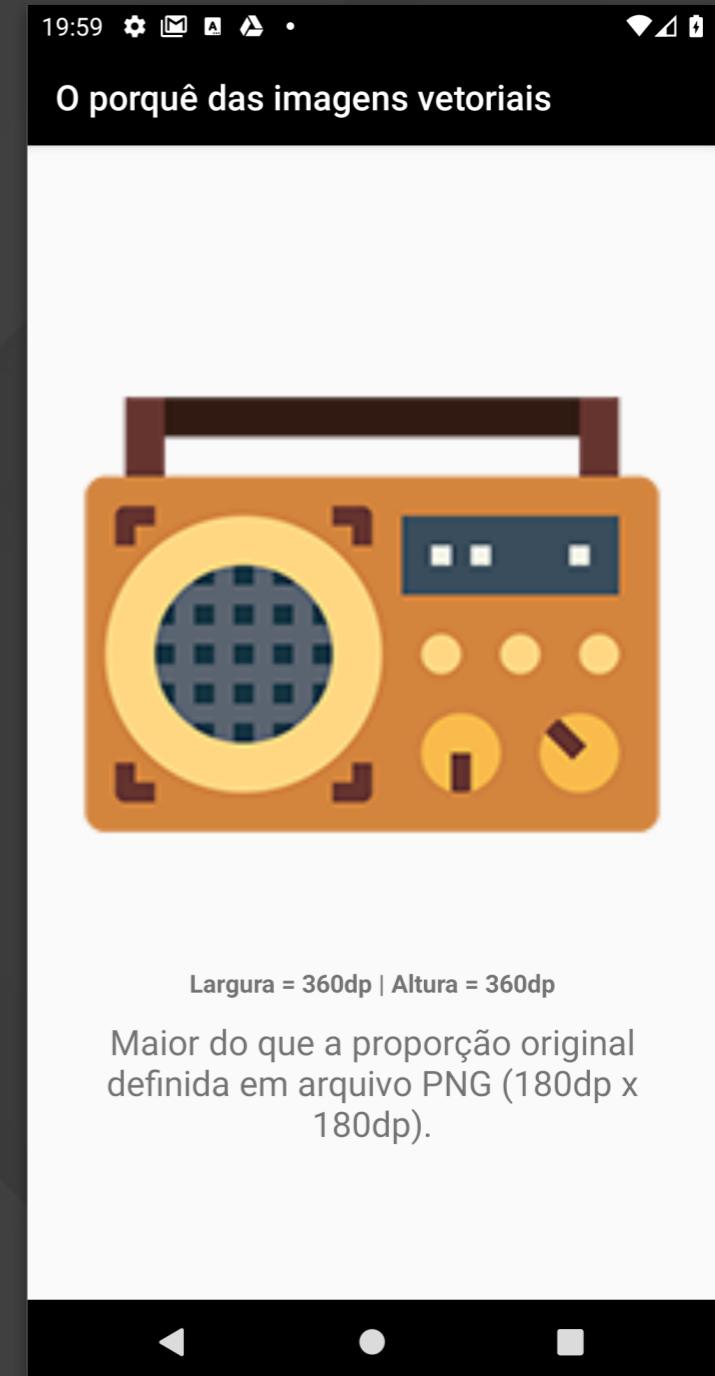
Então quando levando em conta os termos visuais a olhos humanos, acaba ficando "tudo ok" utilizar imagens rasterizadas em proporções menores do que as dimensões originais.

Porém quando as imagens rasterizadas são colocadas em proporções maiores do que as originais...

... neste caso é nítido a qualquer um que estas imagens perdem em qualidade. Isso por sofrerem com a "falta de informação".

Ou seja, pixels extras que faltam à imagem.





Mesmo que pouco perceptível, a terceira imagem de rádio da figura acima (com arquivo PNG - bitmap) sofre de pixelagem (distorção por falta de pixels).



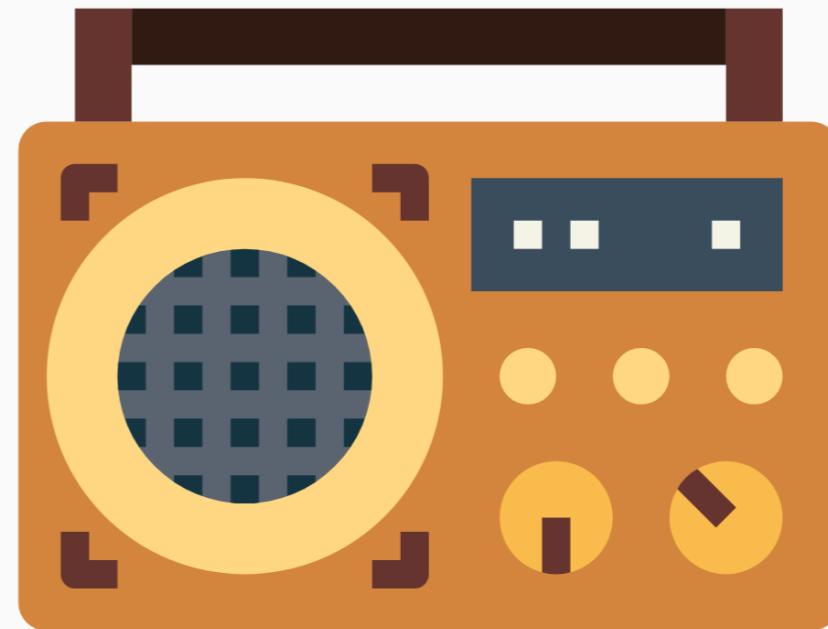
Pois essa imagem está sendo carregada em proporções maiores do que a versão original.

A terceira imagem da primeira figura desta seção (com arquivo vetorial) não sofre do mesmo problema. Porque por ser vetorial, independente da proporção original definida em arquivo, ela continua com a mesma qualidade.



Largura = 360dp | Altura = 360dp

Maior do que a proporção original  
definida em arquivo PNG (180dp x  
180dp).



Largura = 360dp | Altura = 360dp

Maior do que a proporção original  
definida em arquivo XML de vetor  
(180dp x 180dp).



Imagens vetoriais não sofrem com pixelagem, pois elas são baseadas em:

Uso de pontos e conexões para desenho de formas (linhas, curvas, polígonos e outros) em um plano cartesiano.

Já as imagens rasterizadas são baseadas em:

Informações de pixel. Onde cada pixel sabe somente a cor que deve ter. Ou seja, não foram feitos para trabalhar a escalabilidade ou replicação de pixels.

Parece ótimo o mundo somente com imagens vetoriais, certo?

Mas não se engane.

Há alguns contextos comuns no dia a dia do desenvolvedor Android onde as imagens vetoriais não são nem de perto uma boa escolha.

Vamos falar mais sobre isso no decorrer do conteúdo, então continue lendo 😊.

# Imagens vetoriais no Android

O Google, mais precisamente no Google I/O 2014, liberou as primeiras APIs Android para trabalho com imagens vetoriais.

As APIs principais são as seguintes:

- **VectorDrawable**;
- e **AnimatedVectorDrawable**.

A liberação veio junto ao anúncio do Android Lollipop (API 21).

A seguir as APIs de suporte, para versões do Android abaixo da API 21:

- **VectorDrawableCompat**;
- e **AnimatedVectorDrawableCompat**.

Se você já é um convededor de vetores, principalmente se você tem experiência com interface com o usuário no mundo Web...

... se você é este tipo de profissional, então fique ciente que a biblioteca de vetores no Android tem certas limitações.



Na verdade essa biblioteca roda algumas características do SVG mobile.

Isso, pois como o Android é o sistema operacional de inúmeros aparelhos de diferentes marcas, foi identificado que vários *devices* não seriam capazes de apresentar em tela um vetor com todas as possibilidades de um SVG.

Na verdade até hoje há navegador Web que não dá 100% de suporte a imagens vetoriais.

A seguir o exemplo de duas características SVG que não são suportadas pelo conjunto de APIs de vetores do Android:

- Não é possível vincular arquivos [JavaScript](#);
- e Não é possível vincular imagens rasterizadas ao XML de vetor.

De qualquer forma, as APIs de vetores Android ainda são de extrema importância a um desenvolvedor Android, independente do nível dele.

Principalmente por causa de algo chamado "[ícones de sistema](#)" onde o suporte de vetores Android é completo.

# Quando utilizar e quando não utilizar

Sim. Nem tudo são flores.

Apesar da promessa incrível em torno da escalabilidade sem necessidade de ao menos quatro arquivos de imagens no Android. As versões drawable em mdpi, hdpi, xhdpi e xxhdpi.

Apesar disso o trabalho com imagens vetoriais em aplicativos deve ser moderado quando saímos do contexto "ícones de sistema".

Segundo a documentação oficial nós devemos utilizar imagens vetoriais quando:

- For para ícones de sistema;
- ou Quando a imagem não tiver dimensões em tela maiores que 200dp x 200dp.

Em qualquer outra situação é bem provável que o uso de imagens rasterizadas seja em disparado uma melhor escolha.

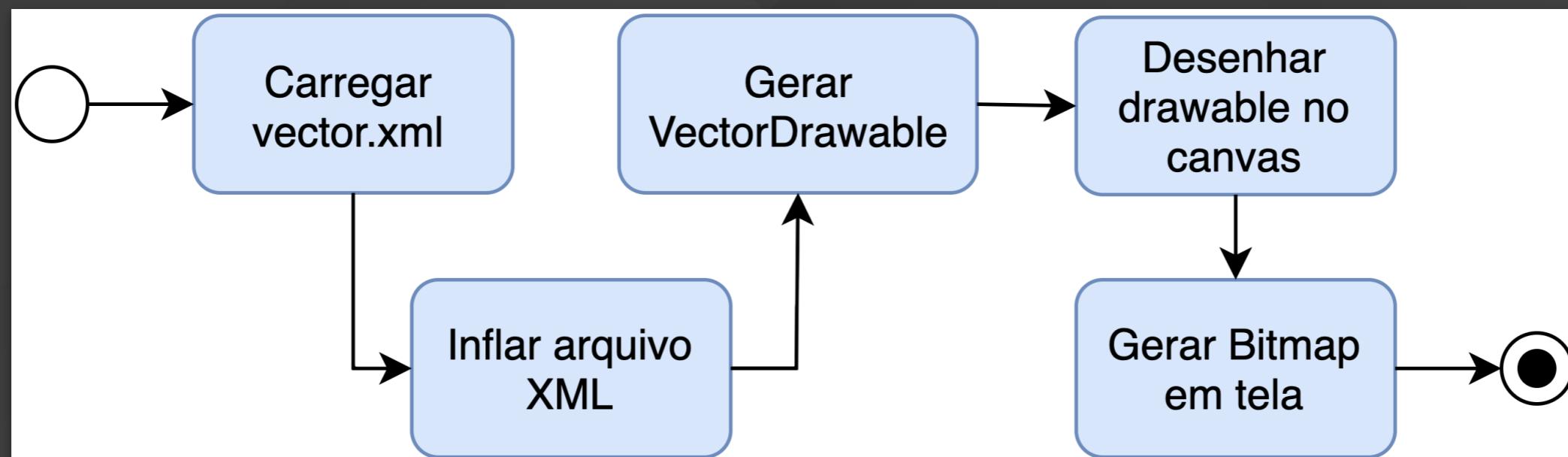
Isso devido ao peso da renderização de imagens vetoriais.

Ou seja, não deixe de também investir tempo para entender "como funciona" e "como utilizar" o antigo e ainda válido modelo de [imagens rasterizadas em folders drawable](#).

# O processo de renderização

O processo de renderização de vetores no Android é mais pesado do que o processo de renderização de imagens rasterizadas.

Veja a seguir o fluxo deste processo:



Enquanto uma imagem rasterizada passa por apenas três passos:

- Carregar arquivo de imagem;
- Desenhar drawable no canvas;
- Gerar Bitmap em tela.

A imagem vetorial passa por cinco:

- Carregar arquivo de vetor;
- Inflar estrutura XML de vetor;
- Gerar **VectorDrawable**;
- Desenhar drawable no canvas;
- Gerar Bitmap em tela.

Devido a isso é importante respeitar as regras de "quando utilizar uma imagem vetorial", caso contrário será nítido ao usuário do aplicativo que ao menos a renderização está lenta...

... isso sem levar em conta a grande possibilidade de um **OutOfMemoryException**.

# Fique atento ao tamanho do arquivo

Um ponto muito importante a levar em consideração é o "tamanho do arquivo" vetorial.

E certamente a documentação oficial não menciona isso, pois eles devem entender que essa limitação é óbvia e você desenvolvedor deve literalmente não cometer esse erro:

Achar que somente porque a ilustração vetorial respeita as dimensões máximas de 200dp x 200dp recomendadas na documentação oficial...

... achar que devido a isso qualquer vetor pode ser utilizado, pois será melhor escolha do que uma imagem rasterizada.

Quando falamos em vetores, no Android é possível importar arquivos SVG e arquivos PSD (já já falaremos mais sobre estes dois formatos).

E no caso dos arquivos PSD é comum termos imagens vetoriais com características e detalhes a nível de imagens rasterizadas (por consequência a estrutura XML desses vetores é bem mais complexa).



Veja a imagem a seguir:



Seria uma "beleza" uma imagem assim, vetorial, em nosso projeto. Sem necessidade de replicação em outros arquivos drawable, certo?

Não seria!



A versão JPG dessa imagem, sem nenhum tratamento de compressão de bits, tem 8,5 MB de tamanho.

E o processo de renderização, como comentado anteriormente, é mais simples devido a ser uma imagem rasterizada.

E eu não estou levando em consideração que se a imagem for colocada nos exatos tamanhos necessários em projeto...

... muito provavelmente todas as versões juntas dessa imagem não vão atingir nem mesmo um 1 MB em APK.

Pois bem.

A versão PSD dessa imagem tem o tamanho de... pasme... 157,1 MB 😱.

Ou seja, mesmo que a [View](#), na qual a versão vetorial da imagem vai ser carregada, tenha proporção 15dp x 15dp...

... mesmo assim um XML vetorial de 157,1 MB primeiro terá que ser colocado em projeto e depois renderizado em tela.

Resumo:

Se o arquivo vetorial chegou na casa dos megabytes (MB), então é muito provável que o trabalho com imagens rasterizadas seja uma melhor opção.

# Estrutura básica (o necessário)

Com uma visão de desenvolvedor (e não de design), a estrutura XML de um vetor Android que realmente é útil conhecer é a seguinte:

```
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    ...>

    <path
        ...
        .../>
    ...

</vector>
```

Bem simples, certo?

Já lhe adianto que há algumas outras tags além de **<vector>** e **<path>**. E mais atributos além dos que apresentarei aqui.



Porém, com um certo tempo de experiência no uso de drawables vetoriais no desenvolvimento de aplicativos Android e conhecendo os contextos onde esses drawables realmente acrescentam valor ao projeto...

... com isso eu seguramente lhe afirmo:

O que será apresentado aqui é 99,99% do que você precisa, como desenvolvedor Android, saber sobre vetores no Android.

Sendo assim, vamos às tags e atributos da estrutura vetorial apresentada anteriormente:

### <vector>

Tag raiz, onde entram todas as outras tags de configuração de desenho (ou animação) vetorial.

Alguns importantes atributos de toda a estrutura também são definidos nesta tag.

A seguir os importantes atributos de <vector> para um vetor estático:



## → **android:viewportWidth:**

Define a largura da ViewPort. A ViewPort é basicamente a área virtual (*canvas*) na qual o vetor será desenhado por completo.

Para facilitar o entendimento: é possível ter mais de uma imagem vetorial em um mesmo *canvas*.

O recomendado é que não se defina um tipo de unidade aqui (**dp**, **cm**, **px**, ...). Assim a unidade utilizada será a mesma definida em **android:width** e **android:height**.

## → **android:viewportHeight:**

Define a altura da ViewPort.

E como no atributo anterior... o recomendado é que não se defina um tipo de unidade aqui (**dp**, **cm**, **px**, ...).

## → **android:width:**

Define a largura exata da imagem vetorial. Aceita qualquer tipo de unidade, mas no contexto Android o normal é o trabalho com **dp**.

## → **android:height:**

Define a altura exata da imagem vetorial. Também aceita qualquer tipo de unidade, mas no contexto Android o normal é o trabalho com **dp**.



## → android:tint:

Por padrão define a cor que será aplicada em toda a parte preenchida da imagem vetorial. Isso, pois o valor padrão de **android:tintMode** é **src\_in**.

Note que independente das cores de preenchimento (**android:fillColor**) e borda (**android:strokeColor**) definidas nas tags filhas de **<vector>**.

Independente disso, se **android:tint** for utilizado a cor definida nele é que será aplicada em todo o vetor. As cores em tags filhas de **<vector>** serão ignoradas.

**tint** é o atributo que você mais vai atualizar em arquivo de vetor... isso nas raras vezes que você precisar trabalhar diretamente na estrutura do arquivo de vetor.

Por fim, se seu aplicativo também dá suporte à versões do Android abaixo da API 21, então o recomendado é que o valor de **tint** seja *hard-coded*. Ou seja, sem uso de referência, ele deve ser colocado na "unha".

Ao invés de utilizar **@color/colorAccent**, por exemplo. Utilize o hexadecimal da cor direto como valor (*hard-coded*), **#FF0000** (vermelho).

Isso é necessário, pois o Android Studio junto ao plugin do Gradle, assim que o app é compilado, gera versões rasterizadas de cada drawable vetorial (mais para frente no conteúdo você terá uma explicação detalhada sobre o porquê disso).

E serão essas imagens rasterizadas geradas que vão ser utilizadas nas versões do Android abaixo da API 21.

E por algum motivo, quando as cores não são colocadas *hard-coded*, pode haver problemas na geração dessas imagens rasterizadas...

... podendo deixar o design do aplicativo, em versões do Android abaixo da API 21, totalmente desalinhado com o esperado.

Dica: se seu aplicativo dá suporte a versões do Android abaixo da API 21, então para todos os atributos de arquivos vetoriais forneça valores *hard-coded*. Não utilize referências.



## → android:tintMode:

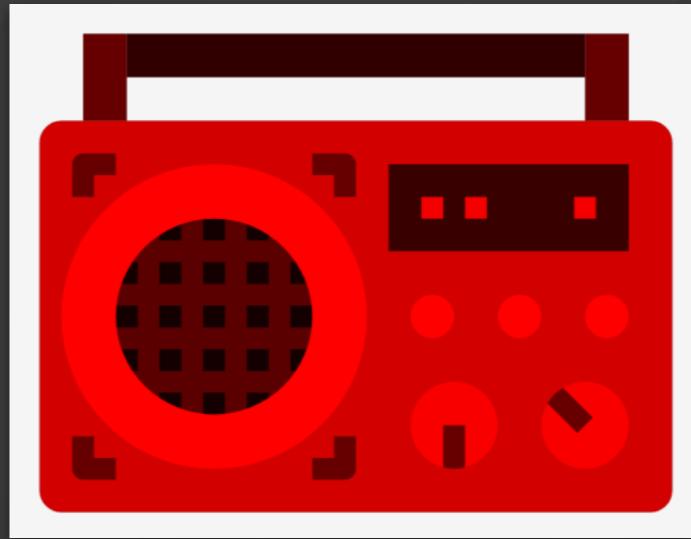
Define o modo como o valor de `android:tint` será aplicado. O valor padrão é `src_in`.

Os valores possíveis são:

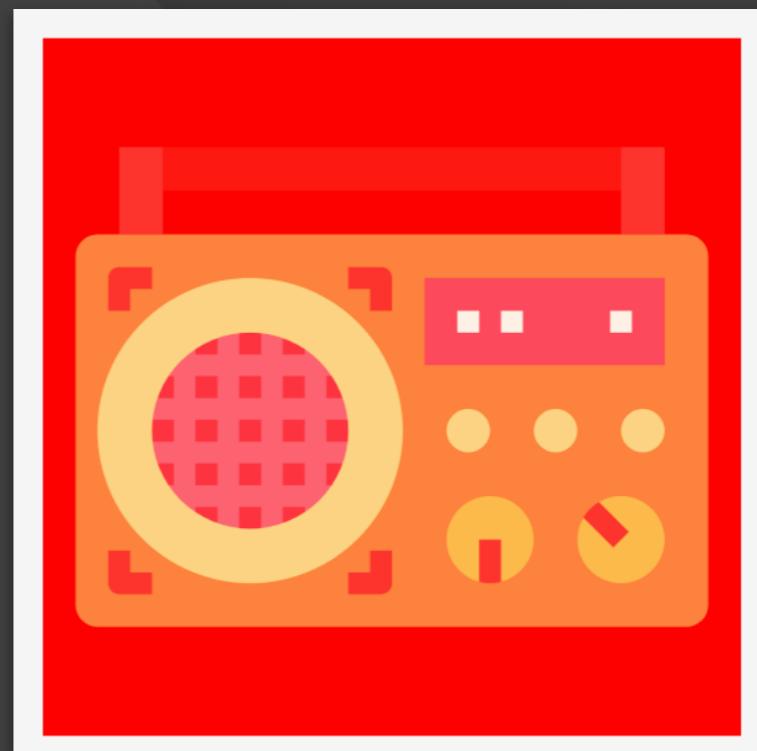
↪ `add`:



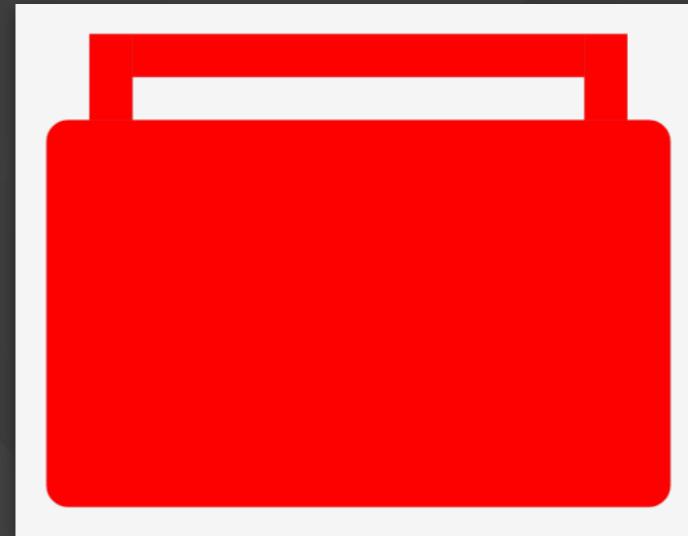
↳ multiply:



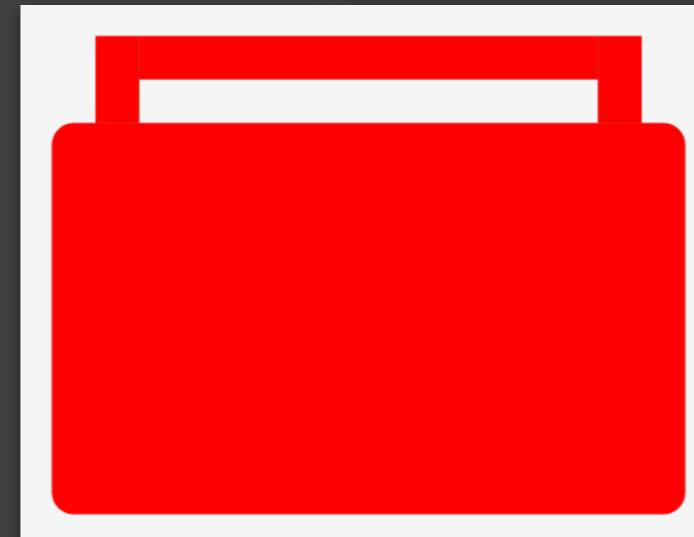
↳ screen:



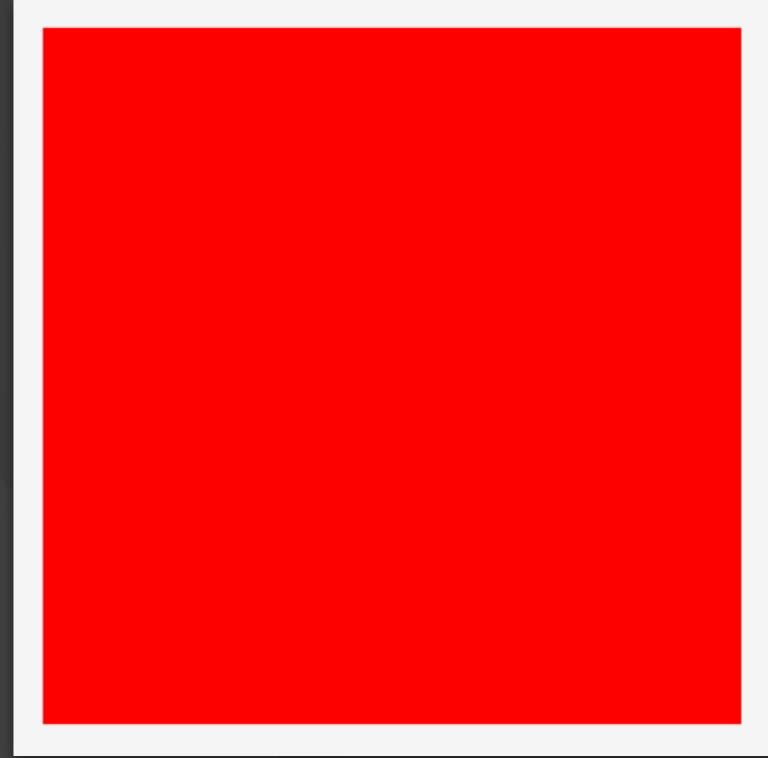
↪ **src\_atop:**



↪ **src\_in:**



↪ **src\_over**:



Nas imagens acima utilizamos o **android:tint** com a cor vermelha (#FF0000).

Note que se **android:tint** não for definido de maneira explícita, então o valor em **android:tintMode** é ignorado.



## <path>

Tag responsável por definir os caminhos a serem desenhados.

Pode conter apenas parte do desenho vetorial total ou toda a configuração de caminho da imagem vetorial.

A seguir os importantes atributos de <path> para um vetor estático:

### → android:pathData:

Define os dados do caminho. O tipo de desenho (linha, curva, círculo, ...) e as coordenadas de cada desenho no plano.

Os possíveis valores aqui são exatamente os mesmos possíveis no atributo **d** de um arquivo SVG.

Um conselho: não se preocupe com os possíveis valores deste atributo, pois como desenvolvedor Android será pouco provável que você tenha que fazer algo aqui na “unha”.



## → android:fillColor:

Define a cor de preenchimento do path, caminho.

Se seu aplicativo Android dá suporte abaixo da API 21 (Android Lollipop). Então, como informado em **android:tint**, não é recomendado utilizar referência a cores como valor de **fillColor**.

Referência como **@color/colorAccent**.

No caso de trabalho com a API de suporte o recomendado é que a cor seja definida de maneira *hard-coded*. Exemplo: **#FF0000** (vermelho).

Note que se não houver definição de cor de preenchimento (**android:fillColor**) quando também não há definição de largura e cor de borda (**android:strokeWidth** e **android:strokeColor**).

Então nada do **<path>** é desenhado em tela, mesmo que o atributo **android:tint** tenha sido definido em **<vector>** e o atributo **android:pathData** de **<path>** tenha valores válidos.



## → **android:strokeColor:**

Define a cor usada para desenhar o contorno do caminho definido em **android:pathData**.

Como em **android:fillColor**...

... se seu aplicativo Android dá suporte abaixo da API 21, então não é recomendado utilizar referência a cores aqui.

A cor deverá ser definida de maneira *hard-coded*. Exemplo: **#0000FF** (azul escuro).

## → **android:strokeWidth:**

Define a largura da borda. Os valores são em ponto flutuante. O padrão é **0.0**.



Como falei anteriormente: existem mais tags e mais atributos.

Mas com um olhar de desenvolvedor Android que tem que utilizar imagens vetoriais e tendo em mente os contextos onde elas são realmente necessárias...

... sabendo disso, você viu acima 99,99% do que precisará em termos de estrutura XML de um arquivo vetorial.

# Suporte ao SVG e ao PSD

A API de vetores do Android dá suporte a somente dois tipos de formatos de vetores:

- SVG (*Scalable Vector Graphics*);
- e PSD (*Photoshop Document*).

Na verdade o suporte é parcial. O suficiente para que imagens estáticas ou animações possam ser utilizadas em qualquer instalação Android.

Digo, qualquer instalação a partir do Android 14 (Ice Cream Sandwich), isso quando utilizando a API de suporte.

Ainda vamos falar um pouco mais sobre o Vector Asset Studio, mas já lhe adianto que é com essa ferramenta do Android Studio que podemos adicionar arquivos SVG ou PSD em projetos Android.

Ou seja, o popular "copiar e colar" que é comum com imagens rasterizadas não funciona com arquivos **.svg** e **.psd**.

Tem que haver a importação da maneira correta, pois há todo um processo de conversão de **.svg** (ou **.psd**) para **.xml**.

# Referência em código

Enfim podemos partir para a codificação.

Acredite, apesar de parecer algo complexo devido à quantidade de explicações que tivemos até chegarmos aqui.

Apesar disso o trabalho com vetores é muito simples.

O mais comum é o carregamento direto em XML, principalmente quando levando em consideração que o contexto comum de uso de vetores é o contexto de "ícones de sistema".

# O comum: carregamento direto no XML de layout

O trabalho com vetores em XML tem as exatas mesmas características quando trabalhando com imagens rasterizadas.

Ou seja, a referência à imagem vetorial pode ocorrer em qualquer **View** capaz de carregar imagens ([ImageView](#), [ImageButton](#), [FloatingActionButton](#), ...).

Abaixo o código XML de uma simples imagem vetorial (controle de vídeo game) de rótulo `ic_baseline_sports_esports.xml`:

```
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:tint="#FF0000"
    android:viewportWidth="24"
    android:viewportHeight="24">

    <path
        android:fillColor="@android:color/white"
        android:pathData="M21.58,..."/>
</vector>
```

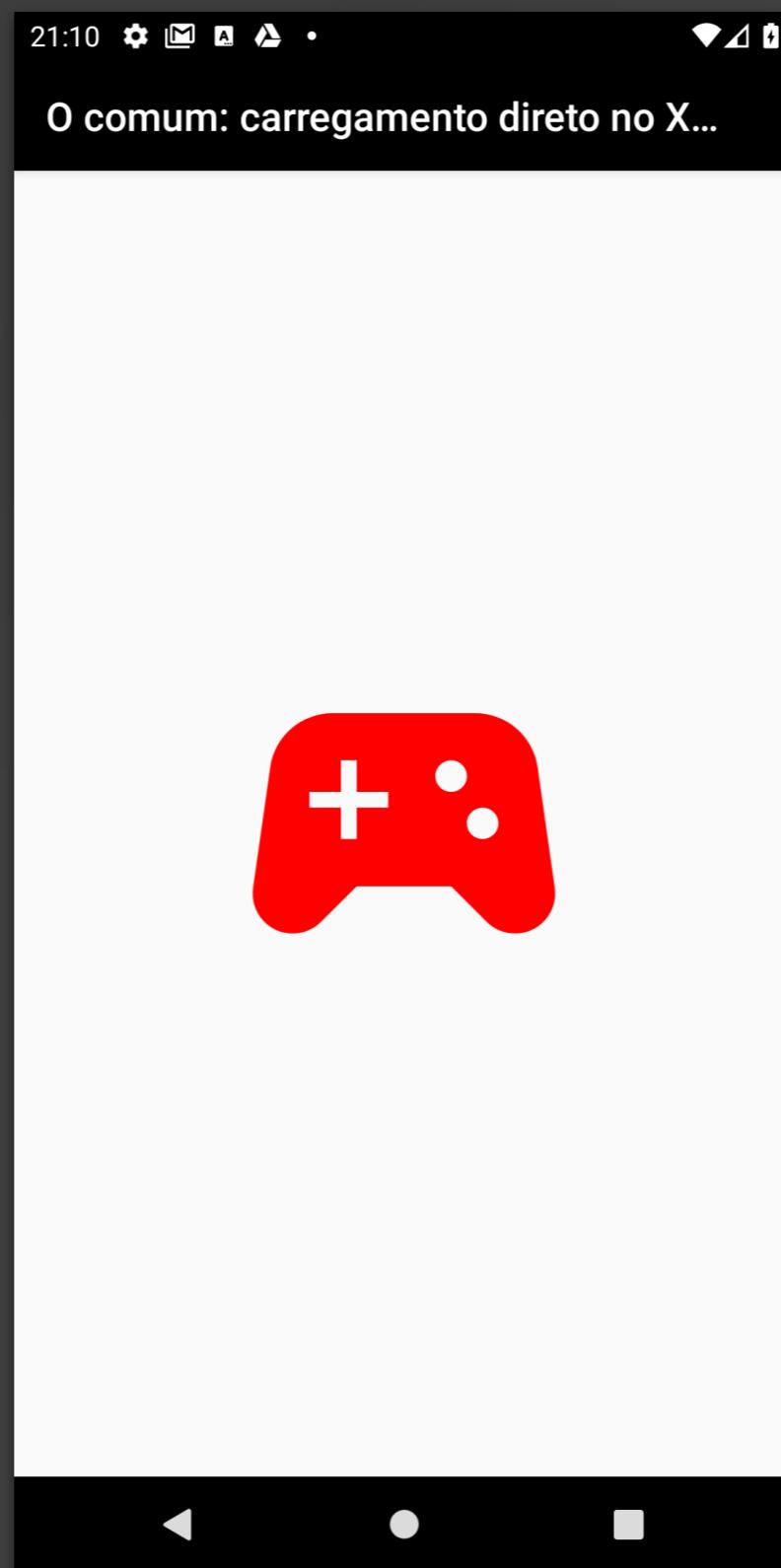


Então o trecho do layout XML que carrega o vetor anterior:

```
...
<ImageView
    android:id="@+id/iv_vector_sample"
    android:layout_width="190dp"
    android:layout_height="190dp"
    android:layout_gravity="center"
    android:contentDescription="@string/simple_vector"
    android:src="@drawable/ic_baseline_sports_esports" />
...
...
```



Como resultado, temos:



Note que se as definições na **View** que está carregando o vetor entrarem em conflito com as definições em arquivo XML de vetor.

Então as definições em **View** é que serão levadas em consideração pelo sistema.

As definições conflitantes presentes no arquivo de vetor serão ignoradas.

No **ImageView** anterior, as definições **android:layout\_width** e **android:layout\_height** sobrescrevem respectivamente as definições **android:width** e **android:height** do arquivo de vetor.

# Carregamento via Kotlin API

Apesar de ser pouco comum, também é possível o carregamento de imagem vetorial via código dinâmico.

Partindo do mesmo XML de vetor carregado na seção anterior, poderíamos ter em código Kotlin o seguinte:

```
...
    iv_vector_sample.setImageDrawable(
        resources.getDrawable(
            R.drawable.ic_baseline_sports_esports,
            null
        )
    )

    iv_vector_sample.setColorFilter(
        Color.BLUE,
        PorterDuff.Mode.SRC_IN
    )
...

```

Como aqui não precisamos trabalhar com algum tema em específico além do tema padrão definido em projeto (definido em `/res/values/styles.xml`). Devido a isso o segundo argumento de `getDrawable()` pode ser seguramente o valor `null`.



Como resultado, temos:



Note que no código Kotlin anterior nós também atualizamos a cor do vetor:

```
...  
iv_vector_sample.setColorFilter(  
    Color.BLUE,  
    PorterDuff.Mode.SRC_IN  
)  
...
```

Fiz isso mais para mostrar que a atualização da imagem vetorial em componente visual é como faríamos se a imagem carregada fosse uma rasterizada.

Lembrando que quando a imagem já está no componente visual, sendo ela vetorial ou não, essa imagem é um Bitmap. Lembre dos passos de renderização discutidos em [O processo de renderização](#).

*Thiengo, imaginei que nesta seção utilizariamos **VectorDrawable** para criarmos um vetor na "unha", via código dinâmico. O que houve?*

Como falei já em alguns pontos até aqui:

Nosso olhar para o trabalho com vetores no Android será um olhar de desenvolvedor.



Querer trabalhar com as APIs Android de vetores em código dinâmico para criar um vetor do zero...

... isso não é apresentado nem mesmo nas páginas de vetores da documentação oficial Android.

Confesso que nem mesmo na comunidade Android eu encontrei algoritmos mostrando como fazer isso.

Mas provavelmente deve ser possível 😐.

Porém volto a afirmar que trabalhando neste nível de detalhes com vetores, então você estará seguindo mais como um designer do que como um *developer*.

Esse não é o nosso objetivo aqui.

# API de suporte

O conjunto de APIs para imagens e animações vetoriais foi adicionado a partir do Android 21, Lollipop.

Sendo assim, para que seja possível o trabalho com vetores mesmo em versões do Android anteriores à API 21 é preciso o uso das APIs de suporte.

APIs que permitem que o suporte inicie do Android API 14, Ice Cream Sandwich (será isso verdade?! *We will see*).

# Gradle Nível de Aplicativo

O primeiro passo é configurar no Gradle Nível de Aplicativo, ou **build.gradle (Module: app)**, a definição correta para trabalho com as API de suporte.

Segue:

```
android {  
    ...  
  
    defaultConfig {  
        ...  
  
        /*  
         * A definição abaixo é necessária para que a API de suporte  
         * de vetores do Android seja utilizada.  
         */  
        vectorDrawables.useSupportLibrary = true  
    }  
    ...  
}
```

Sincronize o projeto.



A partir deste ponto as APIs **VectorDrawable** e **AnimatedVectorDrawable** deixam de ser utilizadas e as APIs de suporte, **VectorDrawableCompat** e **AnimatedVectorDrawableCompat**, passam a ser usadas em projeto.

Mas será que este é o único e melhor caminho para suporte a vetores no Android?

## O problema do `vectorDrawables.useSupportLibrary = true`

Utilizando em projeto a definição **`vectorDrawables.useSupportLibrary = true`** faz com que as APIs de suporte de vetores passem a ser utilizadas e assim nenhuma imagem rasterizada extra é gerada para as versões do Android anteriores a API 21.

Porém tem um grande problema:

Com a definição de **`vectorDrawables.useSupportLibrary = true`** em projeto não mais é possível referenciar arquivos vetoriais como recurso (`Int`). Somente como objetos **Drawable**.

Caso contrário, quando o aplicativo estiver rodando em aparelhos com o Android abaixo da API 21 haverá exceções e o app fechará de maneira abrupta.

Ou seja, "esquece" a possibilidade do simples vinculo de um vetor ao um [`TextView`](#), por exemplo, com o simples atributo **`android:drawableStart`**.

Será preciso acessar o objeto **`TextView`** em código dinâmico, criar um objeto **Drawable** do vetor, utilizando o **`ResourceCompat`**, por exemplo.

E ai sim vincular o objeto **Drawable** como parte do **`TextView`** via **`drawableStart`**.



Do que foi explicado anteriormente somente um ponto já é o suficiente para "remover muito" da vantagem do trabalho com vetores ao invés de imagens rasterizadas:

*"Será preciso acessar o objeto (...) em código dinâmico"* 😞.

Veja bem... busque deixar em código dinâmico (Kotlin, Java, PHP, Python, ...) somente os algoritmos de domínio de problema.

Conteúdo estático deve ficar em arquivo de conteúdo estático (XML, [HTML](#), CSS, ...).

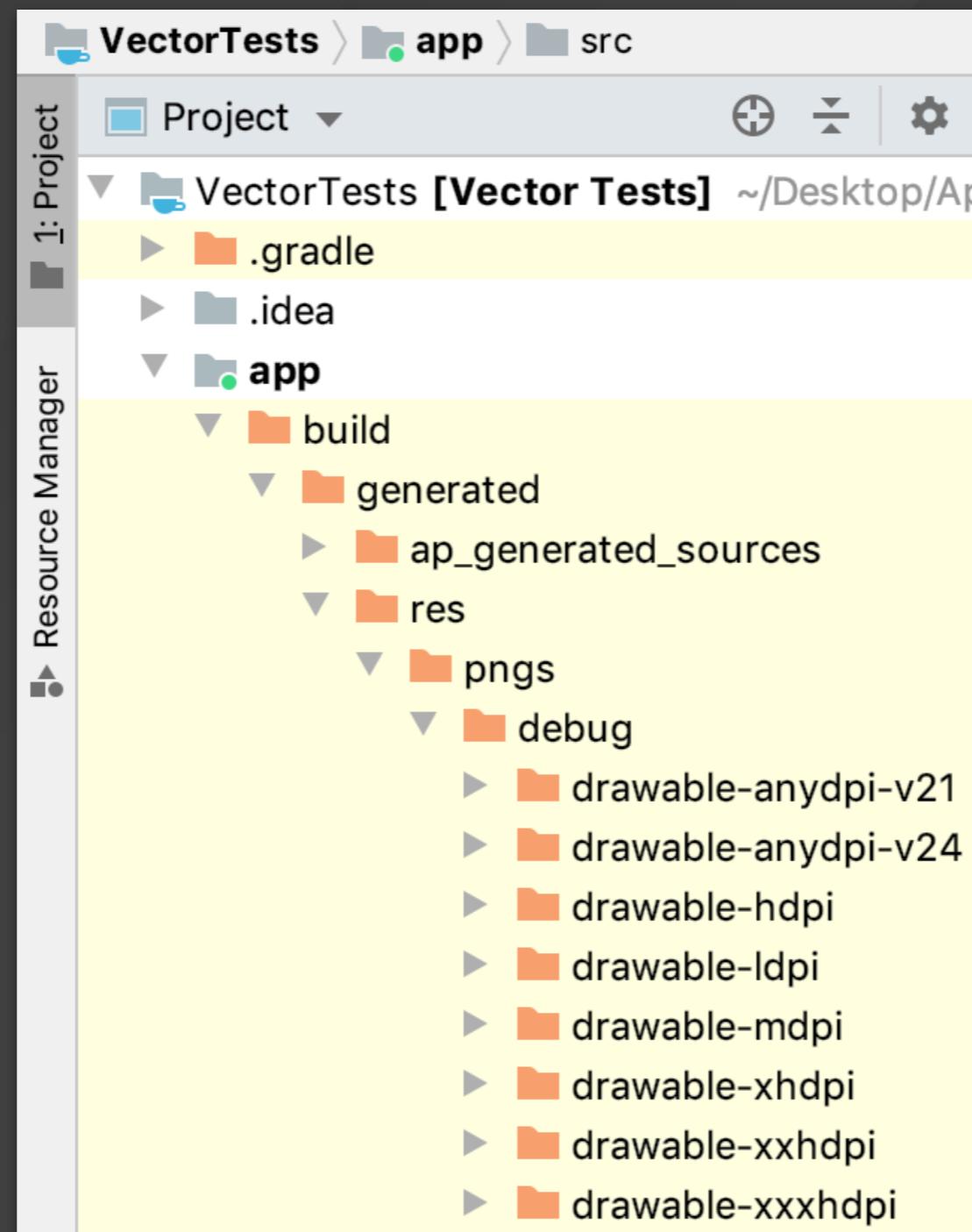
*Ok, Thiengo. E se eu não definir **vectorDrawables.useSupportLibrary = true** em projeto, como vou dar suporte a versões do Android abaixo do Lollipop?*

A partir do Android Studio 1.5 (certamente você já está com uma versão mais atual) foi colocado no IDE uma ferramenta que junto ao plugin do Gradle faz com que imagens vetoriais tenham suas próprias versões rasterizadas.

Ou seja, assim que o projeto é compilado, se ele dá suporte a versões do Android abaixo da API 21. Então é gerada seis versões de cada imagem vetorial presente em projeto.



Essas versões ficam no folder de arquivos gerados em compilação, mais precisamente o folder **/build/generated/res/pngs**:



Cada drawable vetorial tem uma nova imagem rasterizada sendo gerada para cada um dos folders a seguir:

- **/drawable-ldpi;**
- **/drawable-mdpi;**
- **/drawable-hdpi;**
- **/drawable-xhdpi;**
- **/drawable-xxhdpi;**
- e **/drawable-xxxhdpi.**

Essas versões rasterizadas não devem nunca ser atualizadas diretamente. Você deve continuar atualizando somente os vetores. Na compilação as novas versões rasterizadas de cada vetor serão todas geradas.

*Thiengo, desta forma o arquivo APK final ficará maior do que quando não utilizando somente imagens rasterizadas, certo?*

Sim, você está certo.

Se o seu aplicativo também oferece suporte a versões do Android abaixo da API 21 e ele também faz uso de drawables vetoriais.

Então devido a geração de imagens rasterizadas o arquivo APK final ficará com alguns poucos bytes a mais.



Mas acredite:

Esses bytes a mais não são nada que venha realmente a prejudicar o aplicativo.

Pois o tamanho extra tende a ser inofensivo e o ganho na manutenção do projeto continua sendo o que faz os pontos prós serem muito melhores do que os pontos contra.

Note que o sistema Android, quando em versões abaixo da API 21, utiliza as imagens rasterizadas ao invés de vetores.

Por isso que não devemos nos preocupar em informar isso em algum lugar do código quando **vectorDrawables.useSupportLibrary = true** não for definido.

O próprio sistema saberá escolher a versão correta de cada imagem para a versão de Android que estiver executando o aplicativo.



Para fechar está seção:

Nos meus próprios projetos eu não utilizo a configuração  
**vectorDrawables.useSupportLibrary = true**.

E aqui nós vamos seguir sem ela  ... e tudo funcionará como esperado.

Outro ponto que quase esqueci de mencionar:

Sem a definição de **vectorDrawables.useSupportLibrary = true** o suporte se inicia no Android 7 (Eclair) e não somente a partir do Android 14.

# Carregamento direto no XML de layout

Aqui as regras de negócio já comentadas na seção [O comum: carregamento direto no XML de layout](#) são as mesmas.

A única diferença é que se você tiver optado por prosseguir com os exemplos com a configuração **vectorDrawables.useSupportLibrary = true** definida em projeto.

Então o seu código XML deverá utilizar o atributo **app:srcCompat** ao invés de **android:src**.

O arquivo XML de vetor apresentado na seção [O comum: carregamento direto no XML de layout](#) continua intacto, sem alterações. Aqui utilizaremos ele como referência.

Sendo assim vou lhe poupar de ter que ver todo aquele arquivo XML novamente.



Como aqui não estamos com a configuração **vectorDrawables.useSupportLibrary = true** ativa, nosso código XML de carregamento de vetor terá atualização somente na altura e largura do **ImageView**:

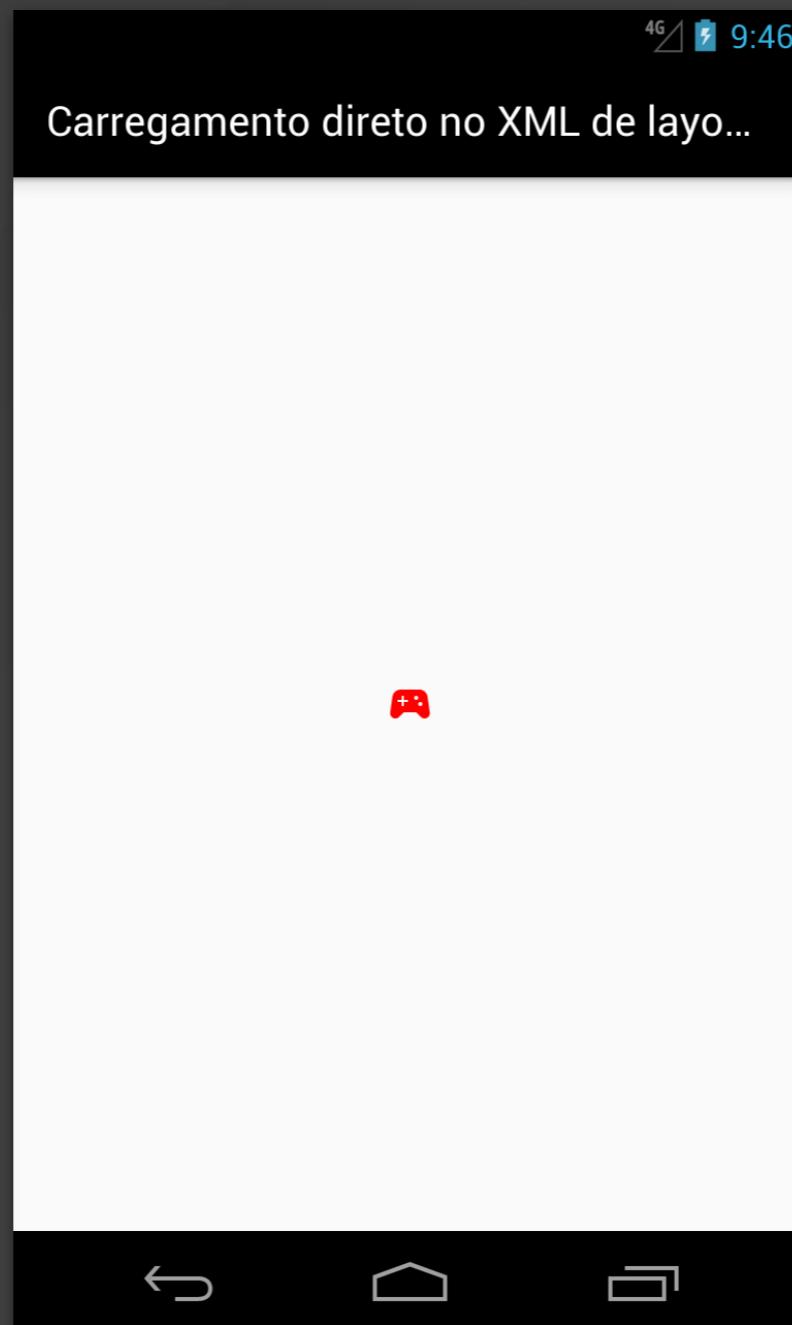
```
...
<ImageView
    android:id="@+id/iv_vector_sample_support"
    android:layout_width="24dp"
    android:layout_height="24dp"
    android:layout_gravity="center"
    android:contentDescription="@string/simple_vector"
    android:src="@drawable/ic_baseline_sports_esports" />
...
...
```

Note que se você for utilizar o **app:srcCompat** é preciso colocar na raiz do layout (de preferência no **ViewRoot** raiz) a seguinte referência **xmlns:app="http://schemas.android.com/apk/res-auto"**.

Antes de executar o código anterior, no menu de topo do IDE acesse "Build" e aione "Clear Project" para assim remover as imagens rasterizadas geradas na última compilação.



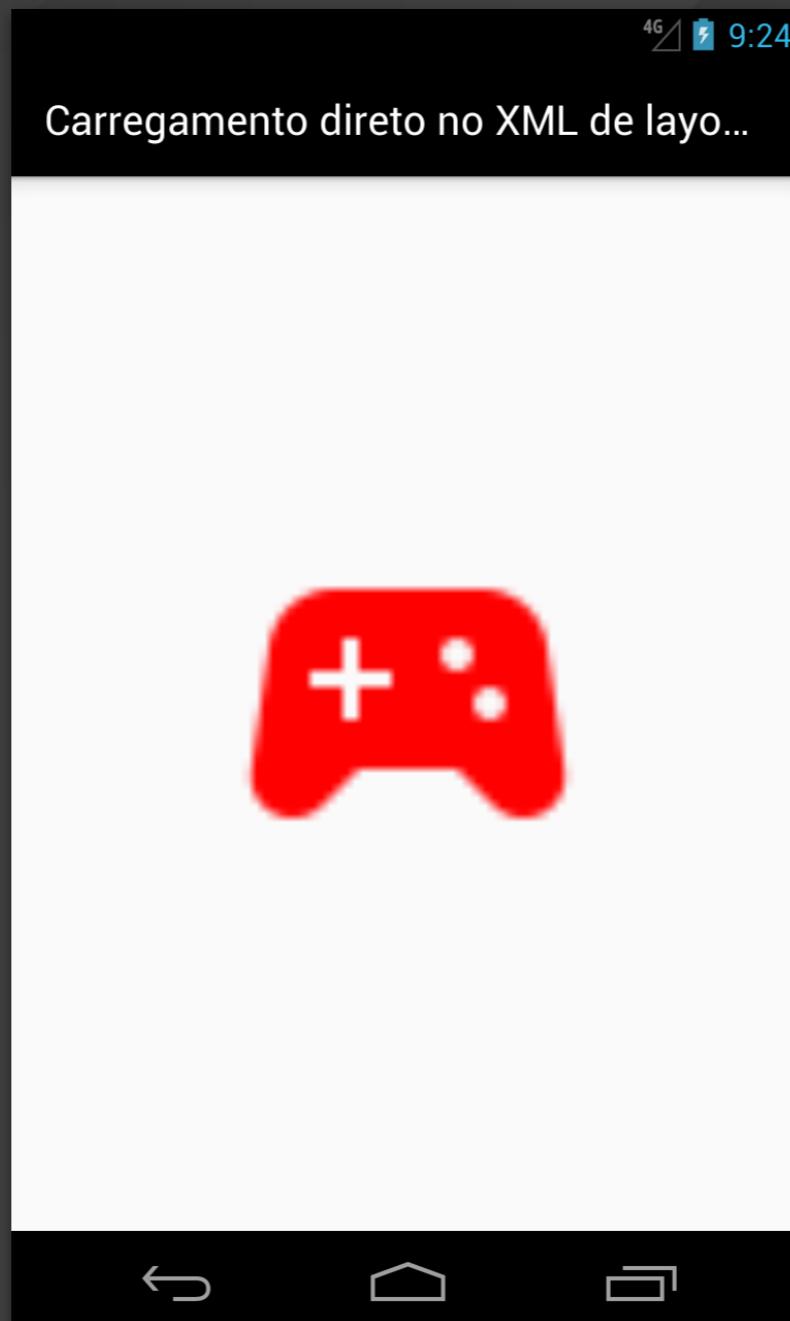
Executando o projeto de exemplo com o código anterior em um emulador com o Android API 16 (Jelly Bean), temos:



Você deve estar se questionando:

*Por que foi necessário mudar as dimensões do ImageView de 190dp para 24dp?*

Nós poderíamos manter os 190dp, porém o resultado seria uma imagem pixelada:



Isso, pois as imagens rasterizadas que são geradas para cada vetor em projeto...  
... essas imagens são geradas com base nas configurações definidas em arquivo XML de cada vetor.

O vetor do exemplo desta seção tem as dimensões 24dp x 24dp:

```
...
<vector
  ...
  android:width="24dp"
  android:height="24dp"
  ...>
...
```



Se quiséssemos manter o tamanho de 190dp no **ImageView**, mesmo para versões do Android abaixo da API 21, e ainda sim não sofrer com o problema de pixalagem.

Teríamos então que atualizar os tamanhos no arquivo de vetor, de 24dp para 190dp.

Então fica a dica:

Se o seu aplicativo for dar suporte também para versões do Android abaixo da API 21 e você inteligentemente não acionar a configuração **vectorDrawables.useSupportLibrary = true**.

Então é necessário colocar nos arquivos de vetores as configurações corretas para que em versões do Android abaixo da API 21 o layout seja apresentado como esperado.

## Vetor via suporte Kotlin API

Tudo que já foi discutido na seção [Carregamento via Kotlin API](#) também é válido aqui.

E o código de carregamento (e atualização) de drawable em componente visual é muito similar:

```
...
iv_vector_sample.setImageResource(
    R.drawable.ic_baseline_sports_esports
)

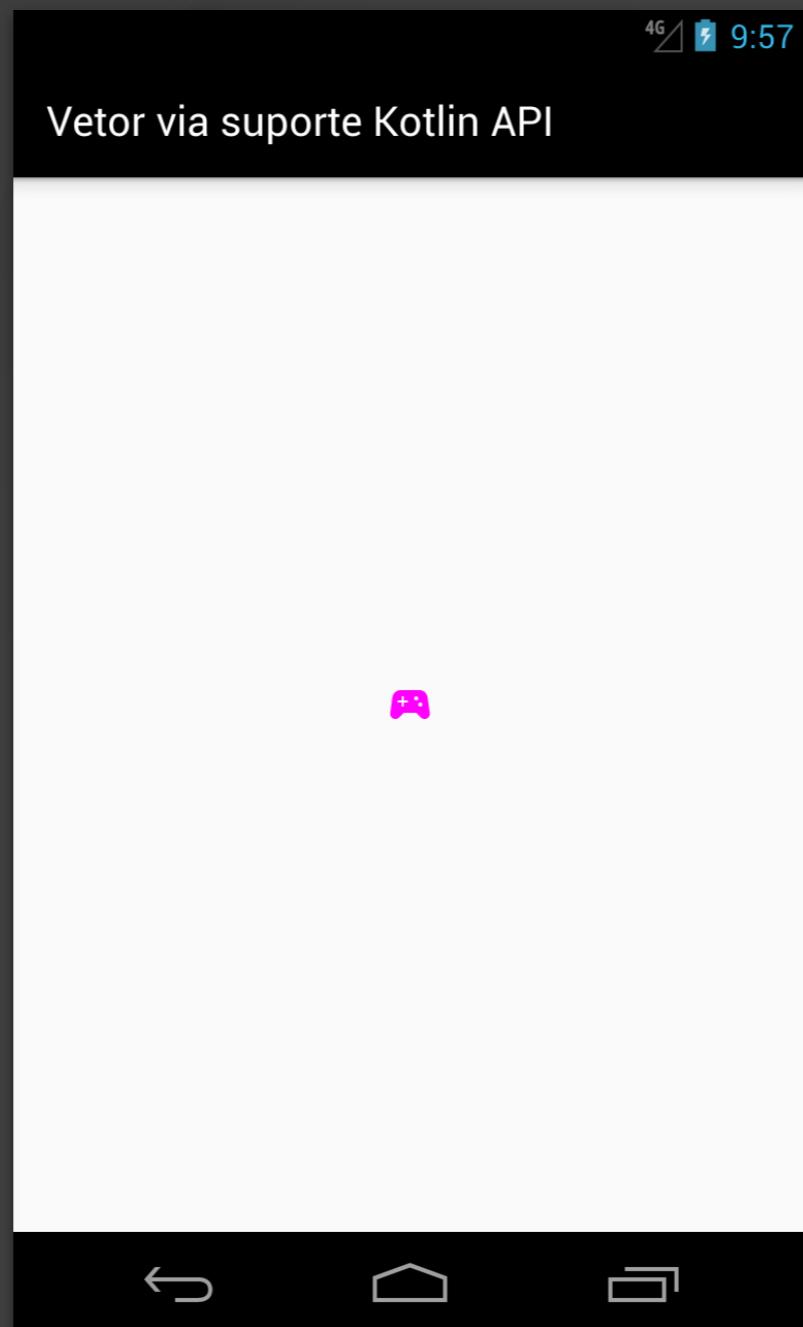
iv_vector_sample.setColorFilter(
    Color.MAGENTA,
    PorterDuff.Mode.SRC_IN
)
...
```

Desta vez utilizamos o método **setImageResource()** para carregamento do drawable. Lembrando que quando o app estiver rodando em uma versão do Android abaixo da API 21 os drawables carregados serão os drawables rasterizados que foram gerados em tempo de compilação.

Sendo assim as regras de configuração em arquivo de vetor discutidas na seção anterior, essas regras também são válidas aqui para que o layout do aplicativo seja apresentado como definido em protótipo estático.



Como resultado do código anterior, temos:



Devido ao suporte a versões do Android anteriores à API 21 é possível que você queira carregar o drawable com o uso de **ResourcesCompat**.

Você pode fazer isso sem receios.

Segue um exemplo:

```
...
    iv_vector_sample.setImageDrawable(
        ResourcesCompat.getDrawable(
            resources,
            R.drawable.ic_baseline_sports_esports,
            null
        )
    ...
}
```

# Vector Asset Studio

O Vector Asset Studio é a ferramenta responsável por facilitar consideravelmente o vida do desenvolvedor Android que precisa trabalhar com vetores.

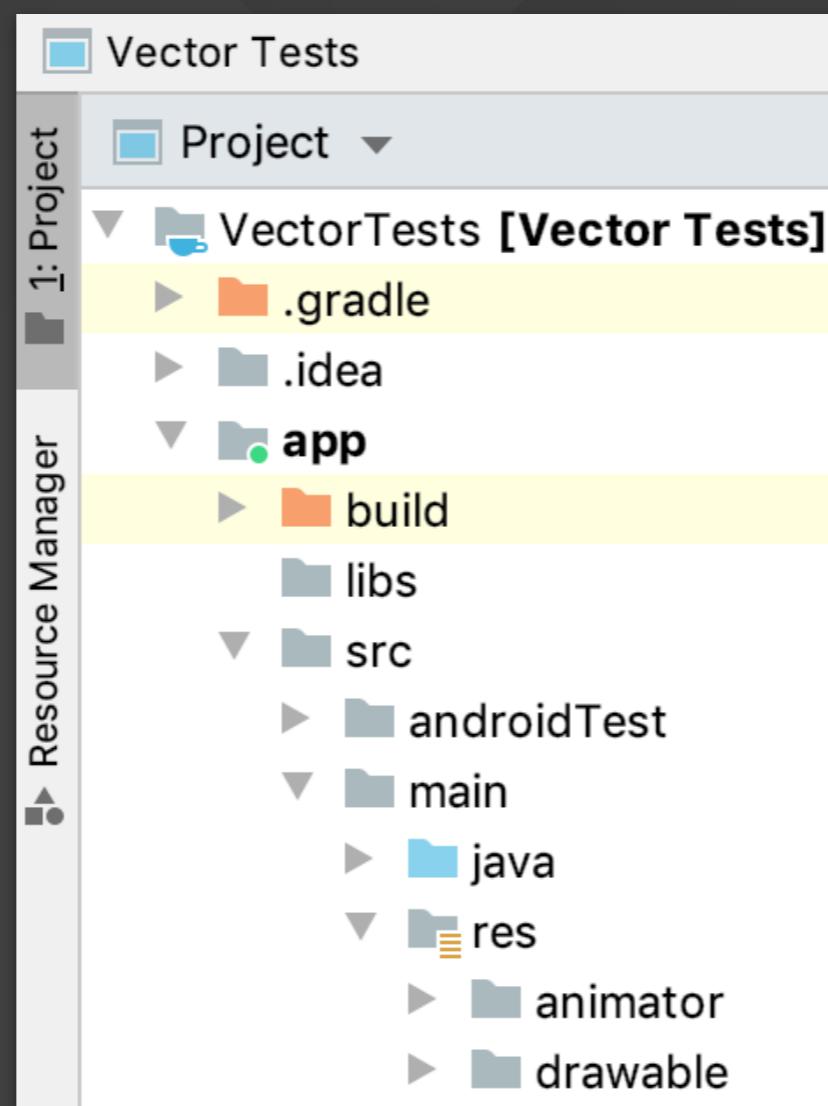
Primeiro porque temos à nossa disposição um *set* com centenas de ícones dos mais variados estilos e prontos para serem incorporados ao projeto.

Segundo porque toda a necessidade de conversão de arquivos SVG ou PSD externos é delegada à essa ferramenta.

# Ícones internos de sistema

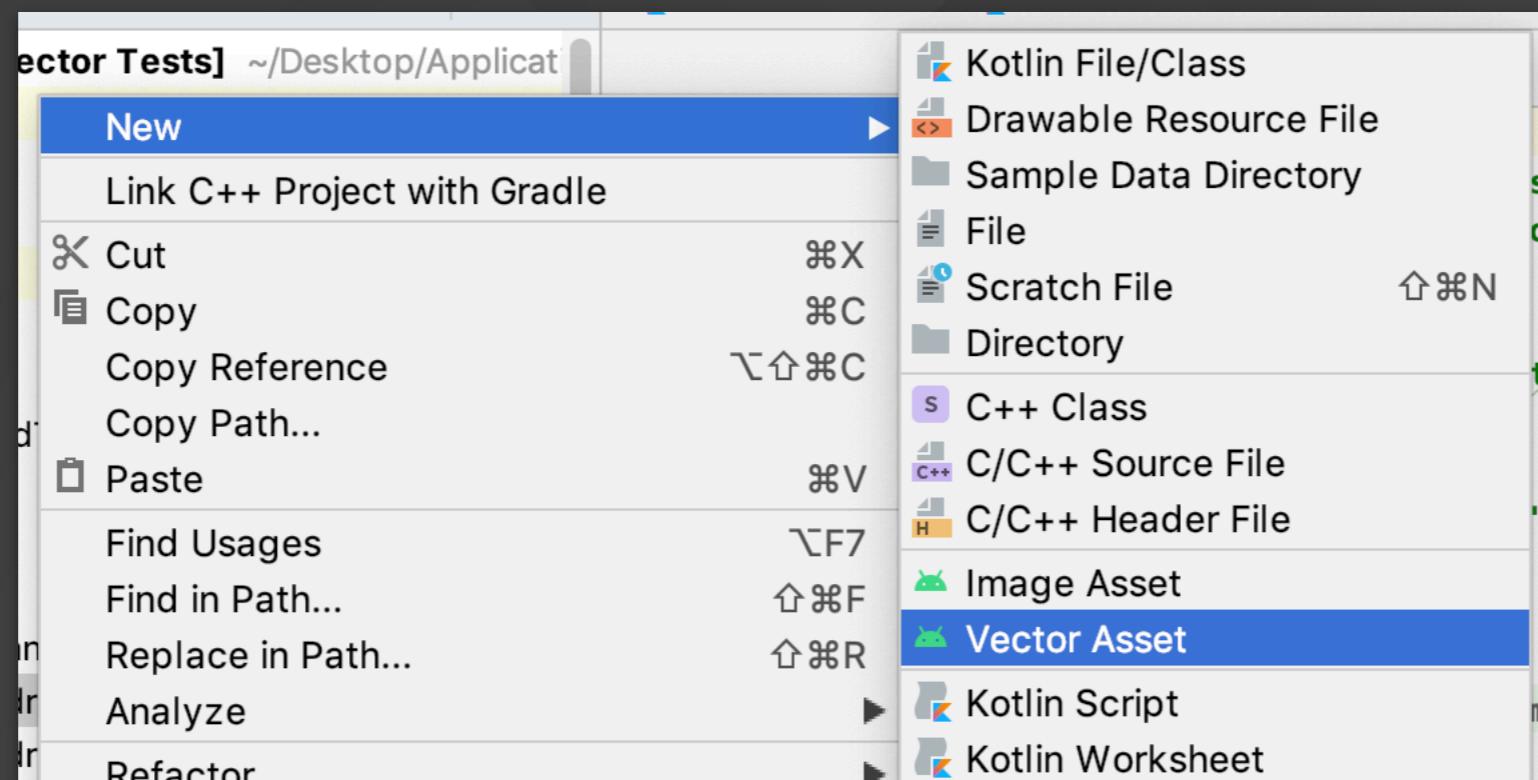
Primeiro vamos a um passo a passo sobre como obter ícones já disponíveis na ferramenta.

Com o projeto (em visualização "Project") expandido...



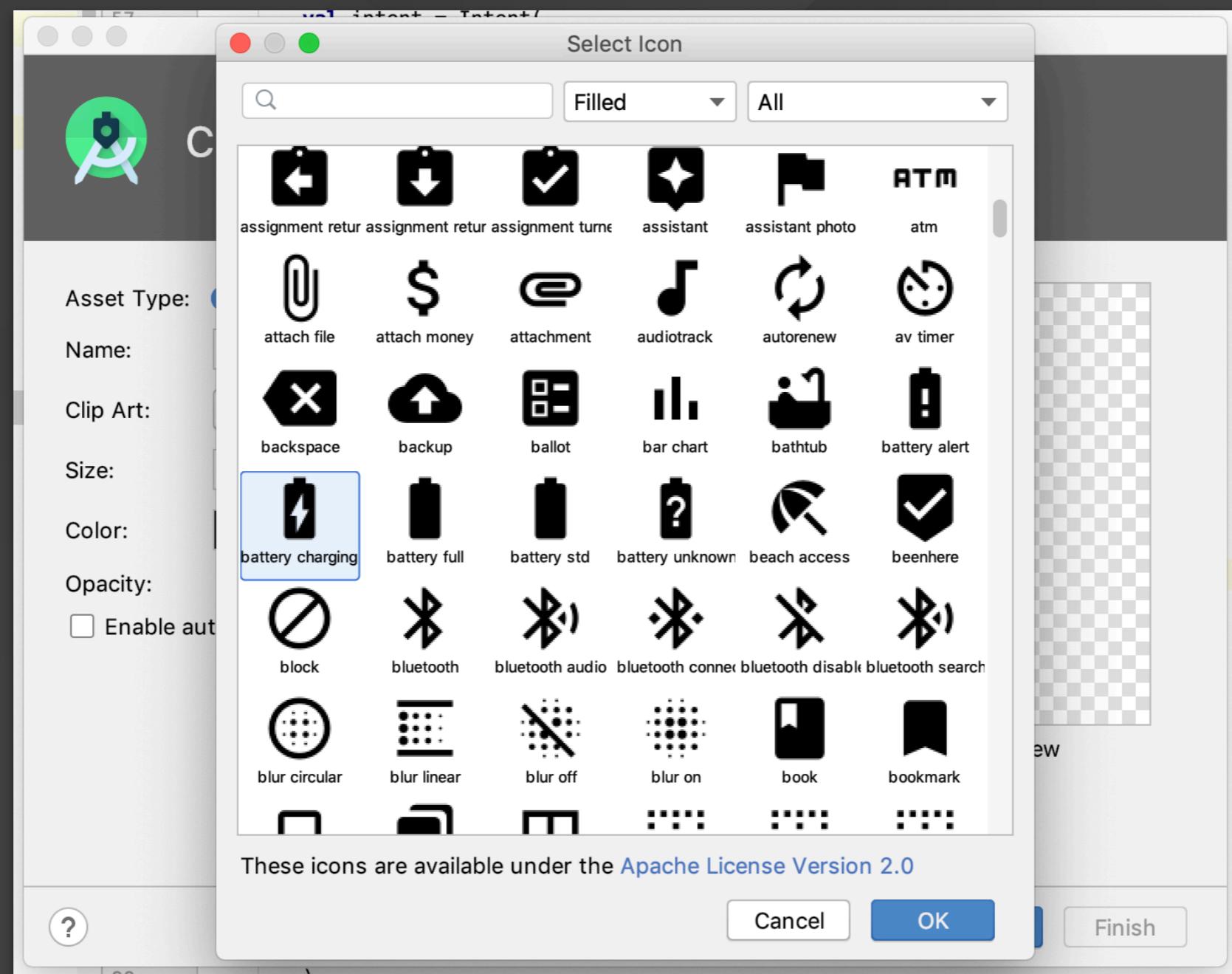
... clique com o botão direito do mouse sobre o folder /drawable e então:

- Clique em "New";
- Logo depois em "Vector Asset".



Com a caixa de diálogo do Vector Asset Studio aberta:

- Mantenha o "Asset Type" em "Clip Art";
- Agora clique no pequeno ícone do campo "Clip Art" e selecione o ícone desejado (aplicando ou não algum filtro na barra de topo);



- Em "Name" defina o rótulo que seja mais conveniente ao tipo de ícone que foi escolhido caso o nome padrão não seja algo útil para a leitura e entendimento de seu projeto. Manter **ic\_** no início do rótulo de um ícone é uma convenção importante adotada pela comunidade Android;
- Os campos "Size", "Color" e "Opacity" são intuitivos e fáceis de serem trabalhados;
- O campo "Enable auto mirroring for RTL layout" deve ser marcado se você for criar uma versão de app que atende a um público que lê da direita para a esquerda;
- Clique em "Next";
- Por fim, na próxima caixa de diálogo, selecione o conjunto de origem de recurso. Onde ficará o drawable vetorial. Neste caso recomendo que você mantenha "Res directory" em **/main**, pois o recurso em **/main** faz com que ele esteja disponível em todas as possíveis variantes de compilação.

Note que os campos "Name" e "Size" devem ser modificados somente depois que o ícone já foi escolhido. Caso contrário será necessário atualiza-los novamente.

Então é isso.

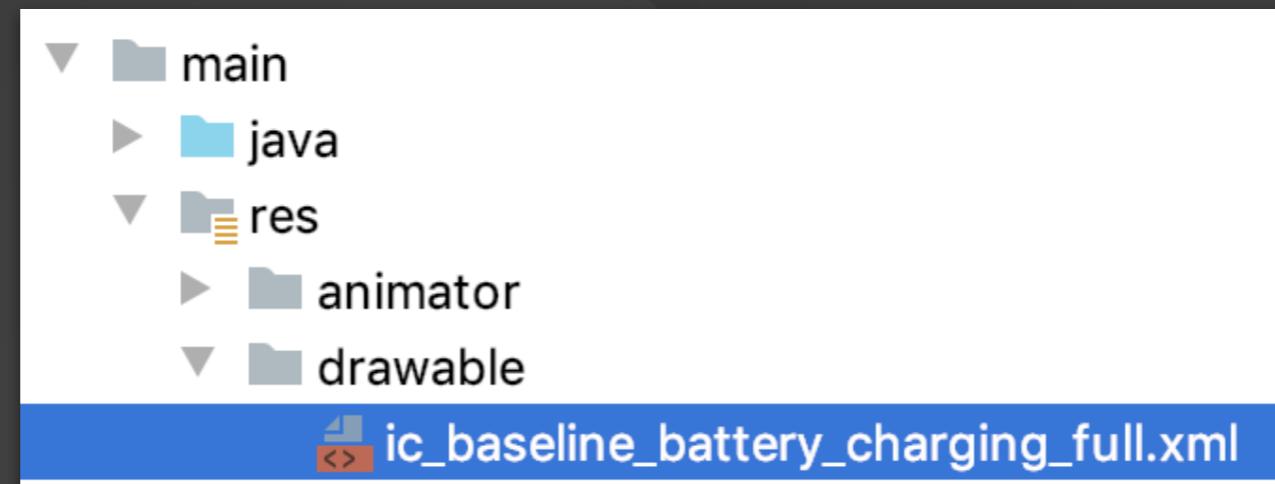
Em questão de poucos minutos é possível ter ao menos toda a configuração de ícones de sistema de um aplicativo Android.



É aquilo, reforçando aqui:

Para ícones de sistema, não tem para onde ir. O uso de imagens vetoriais é um *must* e não um *should*.

Os ícones deverão ficar no folder **/res/drawable**:



A seguir o XML do ícone que selecionamos (bateria completa) neste exemplo:

```
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
    android:viewportHeight="24">

    <path
        android:fillColor="@android:color/black"
        android:pathData="M15.67,4H14V2h-4..." />
</vector>
```

Note que até o momento da construção deste artigo não era incomum o valor definido em "Color", no Vector Asset Studio, não ser respeitado no XML do ícone vetorial escolhido.

Isso devido ao uso de **android:tint="?attr/colorControlNormal"**.

Basta abrir o XML do vetor e remover o **android:tint** ou colocar o valor de cor esperado nele.

# Importação de arquivos externos (SVG e PSD)

Com necessidade de utilizar ícones ou ilustrações externas, a importação de vetor deve ser via Vector Asset Studio.

Pois a conversão é feita somente por esta ferramenta. Digo, quando é possível realizar a conversão.

Pois caso o vetor externo esteja utilizando características não atendidas pela API de vetores do Android...

... neste caso o vetor não é importado.

Vamos novamente ao passo a passo.

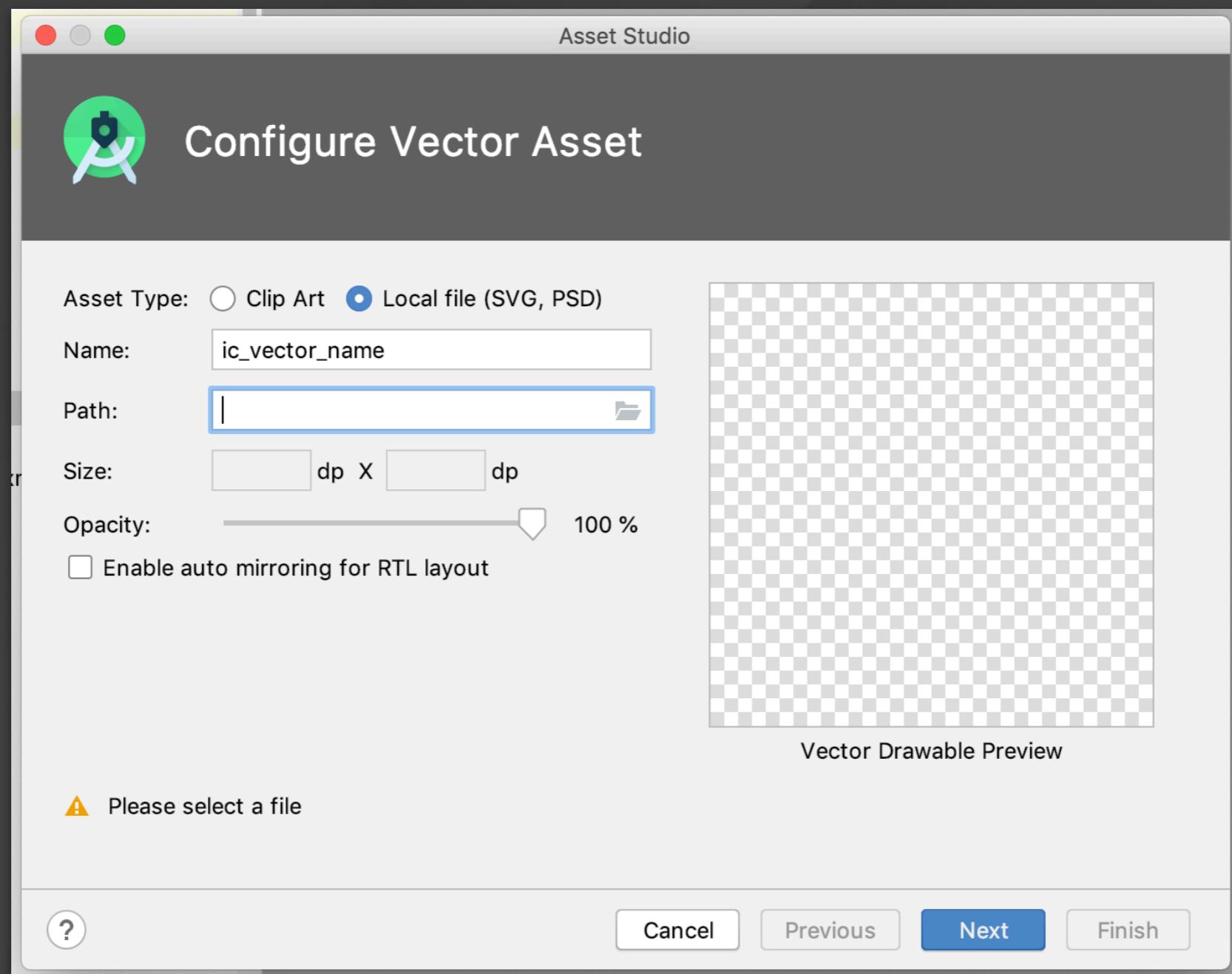
Com o projeto expandido, clique com o botão direito do mouse sobre o folder drawable e então:

- Clique em "New";
- Logo depois em "Vector Asset".

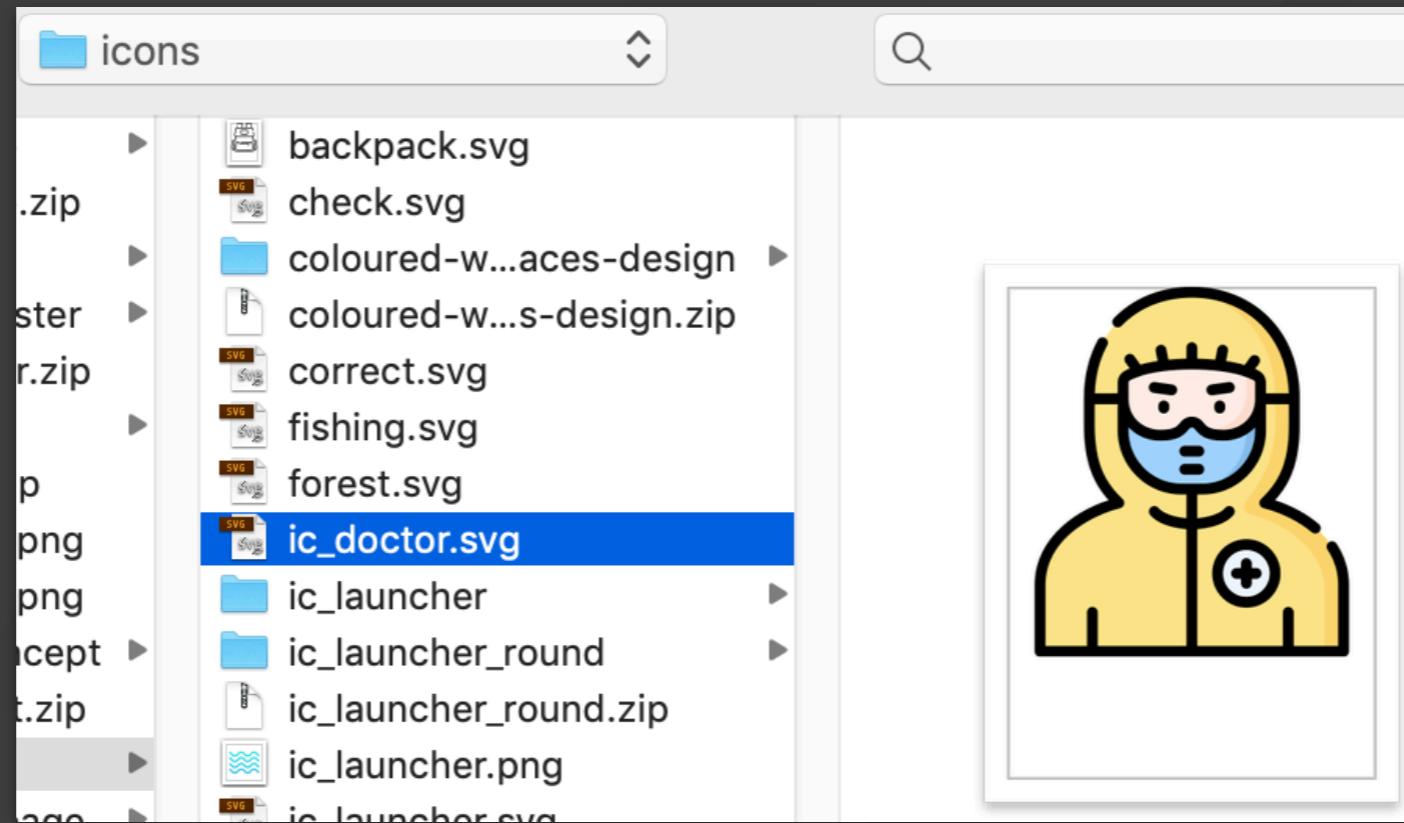


Com a caixa de diálogo do Vector Asset Studio aberta:

- Selecione "Local File (SVG, PSD)" em "Asset Type";
- Em "Path" clique em "Browse" (o ícone de pasta);



- Navegue até o local aonde está o vetor e o selecione;



- Para os campos "Name", "Size", "Color", "Opacity" e "Enable auto mirroring for RTL layout" as regras de negócio são exatamente as mesmas já apresentadas na seção anterior;
- Clique em "Next";
- Para a próxima caixa de diálogo as regras são também as mesmas já discutidas na seção anterior. Sempre que possível mantenha "Res directory" em **/main**, pois o recurso em **/main** faz com que ele esteja disponível em todas as possíveis variantes de compilação.



Não esqueça que os campos "Name" e "Size" devem ser modificados somente depois que o vetor já tiver sido escolhido. Caso contrário será necessário atualiza-los novamente.

Então é isso. A importação de arquivos SVG ou PSD de vetores é tão simples quanto o uso de ícones já presentes na ferramenta.

Como ocorre com ícones de sistema selecionados via Vector Asset Studio, aqui os vetores selecionados também deverão ficar no folder **/res/drawable**.

A seguir o XML (resumido) de vetor que selecionamos (um ícone de médico):

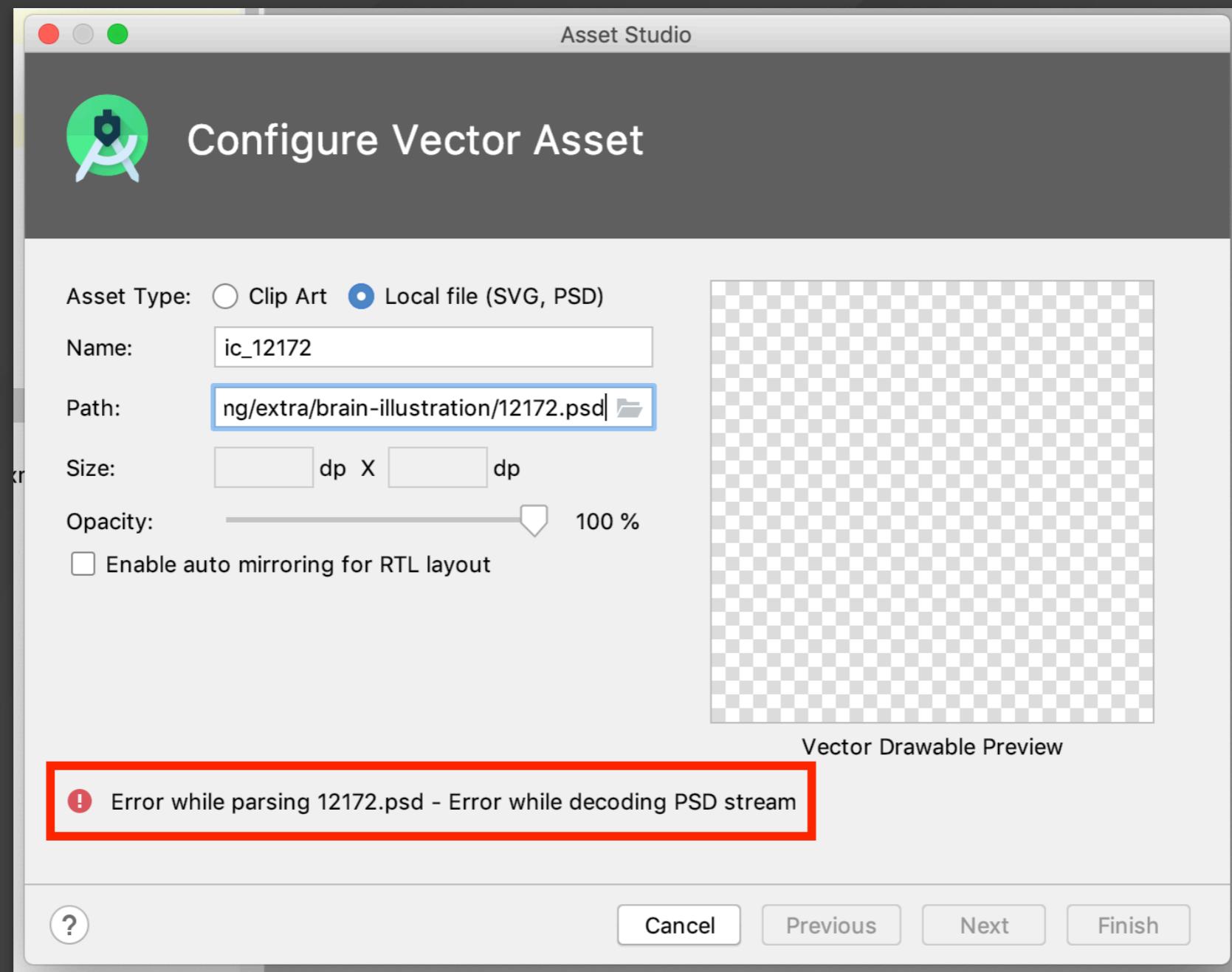
```
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="512"
    android:viewportHeight="512">

    <path
        android:fillColor="#fde588"
        android:fillType="evenOdd"
        android:pathData="..." />

    ...
    <path
        android:fillColor="#FF000000"
        android:pathData="..." />
</vector>
```



Caso não seja possível importar um arquivo, a ferramenta Vector Asset Studio apresenta uma mensagem como a em destaque a seguir:



# Excelentes repositórios de arquivos vetoriais

Para ícones e ilustrações das mais variadas, existem inúmeros repositórios online que também têm recursos gratuitos.

A seguir listo alguns que certamente vão lhe ajudar a colocar melhores recursos visuais, vetores, em seus projetos Android sem que você tenha que ser um expert em design:

- [Flaticon](#) - excelente para encontrar os mais variados ícones SVG;
- [Fleepick](#) - excelente para encontrar não somente ilustrações SVG, mas também PSD.

Faça cadastro em todos que você for utilizar. Caso contrário o limite de downloads gratuitos é muito baixo e logo é alcançado.

Sinta-se convidado a colocar na [área de comentários](#) deste conjunto de "slides aula" os sites de vetores que você conhece e não foram listados aqui.

# Carregamento via API de imagens

Note que, mesmo não sendo mencionado de maneira explícita em documentação oficial. Mesmo com esse "não mencionamento", os drawables vetoriais tendem a ser uma opção para gráficos locais no aplicativo Android.

Então se você projeta carregar, em seu app, gráficos visuais de uma fonte externa junto a alguma API de carregamento remoto de imagens ([Picasso](#) ou [Universal Image Loader](#), por exemplo)...

... se você planeja isso, então trabalhe com imagens rasterizadas.

Pois é muito provável que essas APIs de carregamento remoto de imagens não sejam capazes de carregar e renderizar imagens vetoriais em tela.

Ao menos a documentação oficial das principais APIs de carregamento de imagens não dão a opção para carregamento de vetores XML.

# E as animações Android com vetores?

Sim. É possível criar animações com a API de vetores Android.

Mais precisamente com a API `AnimatedVectorDrawable` ou a API de suporte `AnimatedVectorDrawableCompat`.

Porém eu achei prudente não falar de animações de vetores já neste conteúdo.

Motivos?

- Primeiro porque todo o conteúdo ficaria exaustivo, tendo em mente que somente a parte de vetores estáticos já é grande o suficiente para um único e longo conjunto de slides aula;
- Segundo porque os vetores estáticos, principalmente no contexto "ícones de sistema", são os vetores que realmente fazem diferença em projeto, diferença para a melhora dele;
- Terceiro porque a API de vetores animados não é tão robusta quanto a [Lottie API](#), por exemplo, que permite animações mais complexas em arquivos mais leves.



As animações em vetores serão sim o conteúdo de um conjunto de slides aula futuro do Blog, não se preocupe com isso.

Mas já lhe adianto que se você entender bem tudo que está sendo apresentado neste conteúdo e também conhecer ou a [API ObjectAnimator](#) ou a API AnimatorSet (ambas nativas Android)...

... conhecendo isso, então o seu trabalho com animações de vetores no Android não precisará aguardar esse novo conteúdo.

# Pontos negativos

Mesmo que já informado anteriormente, vamos recapitular alguns dos pontos negativos.

Na verdade vamos recapitular algumas das barreiras que fazem com que vetores não sejam sempre uma melhor solução frente a imagens rasterizadas:

- A renderização em tela é mais pesada do que quando comparada à renderização de imagens rasterizadas;
- Quando as proporções são maiores do que 200dp x 200dp o consumo de memória disponível e o tempo de renderização faz com que vetores não sejam nem de perto uma boa opção;
- Imagens com muitos detalhes, como fotos de pessoas, por exemplo. Essas imagens vetoriais têm um tamanho em bytes muito maior do que quando comparadas às suas versões rasterizadas;
- Quando atendendo à versões do Android abaixo da API 21, então para cada imagem vetorial é criada seis novas versões rasterizadas (**ldpi**, **mdpi**, **hdpi**, **xhdpi**, **xxhdpi** e **xxxhdpi**). Essas versões rasterizadas é que serão utilizadas em versões abaixo do Android Lollipop. Isso, mesmo que pouco, aumenta o tamanho final do APK.

# Pontos positivos

E assim os pontos positivos.

Quando realmente vale a pena (se torna um *must* e não um *should*) o uso de vetores estáticos:

- Para o contexto "ícones de sistema" o uso de vetores é sem sombra de dúvidas a melhor opção. Lembrando que este contexto é comum a qualquer aplicativo, pois todos têm ícones de sistema;
- É necessário somente um arquivo que pode ser utilizado com eficiência e eficácia em qualquer configuração de tela;
- Devido à necessidade de somente um arquivo de vetor para cada imagem, a manutenção do projeto fica bem mais simples;
- Os arquivos de vetores costumam ter uma fração do tamanho em bytes do conjunto de ao menos quatro imagens rasterizadas (**mdpi**, **hdpi**, **xhdpi** e **xxhdpi**) que seriam utilizadas para cada imagem não vetorial presente em projeto.

# Repositório do projeto

O projeto Android que foi utilizado para rodar os algoritmos apresentados aqui...

... esse projeto pode ser acessado no repositório público a seguir:

- <https://github.com/viniciusthiengo/vector-tests>.

# Conclusão

Precisando de ilustrações e, principalmente, ícones de sistema?

Não titubeie! Ao menos para ícones de sistema você certamente terá que utilizar drawables vetoriais.

Não por causa apenas da possível diminuição em bytes do APK final, mas principalmente devido a facilidade de atualização de ícones em projeto.

Quanto às ilustrações:

Para aquelas que não têm muitos detalhes, que em bytes são menores do que suas versões rasterizadas e que não serão carregadas em proporções maiores do que 200dp x 200dp...

... para essas, mantenha o uso de vetores.

Caso contrário, siga com imagens rasterizadas, cinco versões para cada imagem.

Ao menos a sua renderização será "saudável".



Antes de finalizar, primeiro quero lhe parabenizar por ter concluído o conteúdo. Isso diz muito sobre o seu comprometimento como profissional desenvolvedor .

Acredite, são poucos que chegam até o final de qualquer artigo ou conjunto de slides de desenvolvimento.

Então é isso.

Caso você tenha dúvidas ou sugestões sobre vetores no Android, deixe nos [comentários](#).

Curtiu o conteúdo? Não esqueça de compartilhá-lo.

E, por fim, não deixe de **se inscrever na  lista de emails** para também garantir a versão em PDF de cada novo "artigo mini-curso".

Abraço.

Att,  
Vinícius Thiengo.

# Fontes

Conteúdo completo, em texto e em vídeo, no artigo do link a seguir:

[Porque e Como Utilizar Vetores no Android.](#)

Sites e livros:

[Visão geral de drawables vetoriais](#)

[Documentação oficial Android - VectorDrawable](#)

[Documentação oficial Android - Drawable](#)

[Documentação oficial Android - colorControlNormal](#)

[Animar gráficos drawable](#)

[Documentação oficial Android - PorterDuff.Mode](#)

[Documentação oficial Kotlin - Enum](#)

[DevBytes: Android Vector Graphics](#)

[Draw Me a Rainbow: Advanced VectorDrawable Rendering \(Android Dev Summit '18\)](#)

[Documentação oficial Mozilla Developer - Paths SVG](#)

[Understanding Android's vector image format: VectorDrawable](#)

[Draw a Path: Rendering Android VectorDrawables](#)

[Vector drawable is the best practices for Android development](#)

[Android vectorDrawables.useSupportLibrary = true is stopping app - Resposta de Vipul Asri e Community](#)

[Unit of viewportWidth and viewportHeight in case VectorDrawable - Resposta de Bryan Herbst](#)

[Remove android.widget.Toolbar shadow - Resposta de Ferdous Ahamed](#)

[How to change color of Toolbar back button in Android? - Resposta Rajen Raiyarela](#)

[Display Back Arrow on Toolbar - Resposta de MrEngineer13 e Brais Gabin](#)

[Simple Android grid example using RecyclerView with GridLayoutManager \(like the old GridView\) - Resposta de Suragch](#)

[Android studio automatically open's documentation view - Resposta de Clocker](#)

[Android: How to change the ActionBar "Home" Icon to be something other than the app icon? - Resposta de Joe](#)

[How to make gradient background in android - Resposta de Suragch](#)

[How to set tint for an image view programmatically in android? - Resposta de Hardik e Vadim Kotov](#)

[layout\\_margin in CardView is not working properly - Resposta de zonda e pumpkinpie65](#)

[android layout: This tag and its children can be replaced by one <TextView/> and a compound drawable - Resposta de NPike](#)

[Android statusbar icons color - Resposta de Ritesh](#)

[How to set VectorDrawable as an image for ImageView programmatically - Resposta de Pramod Baggoli e Farbod Salamat-Zadeh](#)

[Add ripple effect to my button with button background color? - Resposta de Pei](#)

[Canvas \(GUI\)](#)

[Vector graphics](#)

[PSD File Format](#)

[Data Binding Para Vínculo de Dados na UI Android](#)

[Fundamentos da SVG, 1ª edição, Maurício Samy Silva \(Novatec - 2012\)](#)

# Para estudo

- Treinamento oficial:
  - [Prototipagem Profissional de Aplicativos Android.](#)
- Meus livros:
  - [Desenvolvedor Kotlin Android - Bibliotecas para o dia a dia;](#)
  - [Receitas Para Desenvolvedores Android;](#)
  - [Refatorando Para Programas Limpos.](#)
- Redes:
  - [GitHub](#);
  - [Udemy](#);
  - [LinkedIn](#);
  - [YouTube](#);
  - [Facebook](#);
  - [Twitter](#);
- [Blog App.](#)



Desenvolvedor Android  
**Algoritmos em Kotlin**

**Android Studio IDE**  
Projeto no GitHub

Vinícius **Thiengo**

# Porque e Como Utilizar Vetores no **Android**

thiengocalopsita@gmail.com

**Thiengo.com.br**