



Conteúdo Exclusivo

Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

Seu melhor Email

Cadastrar email



Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gráts conteúdos Android sem precedentes! ! Você receberá um email de confirmação. Somente depois de confirma-lo é que eu poderei lhe enviar os conteúdos semanais exclusivos. Os artigos em PDF são entregues somente para os inscritos na lista.

Seu melhor Email

Cadastrar email



Desenvolvedor Kotlin Android - Bibliotecas para o dia a dia

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

Porque e Como Utilizar Vetores no Android

Vinícius Thiengo

⚡ (911)

Curtir 245

Compartilhar

Você já iniciou no dev Kotlin Android? Neste novo título eu lhe guio passo a passo no desenvolvimento Android com a mais nova linguagem oficial deste SO.

Acessar livro (com + R\$194,00 em bônus)

Buscar - Google Search



Go-ahead

"Você tem de refletir sobre as grandes coisas enquanto está fazendo as pequenas coisas, para que todas as pequenas coisas sigam na direção certa."

Alvin Toffler

Kotlin Android



Título Desenvolvedor Kotlin Android - Bibliotecas para o dia a dia

Categorias Android, Kotlin

Autor Vinícius Thiengo

Edição 1^a

Capítulos 19

Páginas 1035

[Acessar Livro](#)

Treinamento Oficial



Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email

Cursos Android, Páginas Profissionais de Aplicativos

CategoriaAndroid

InstrutorVinícius Thiengo

NívelTodos os níveis

Vídeo aulas186

PlataformaUdemy

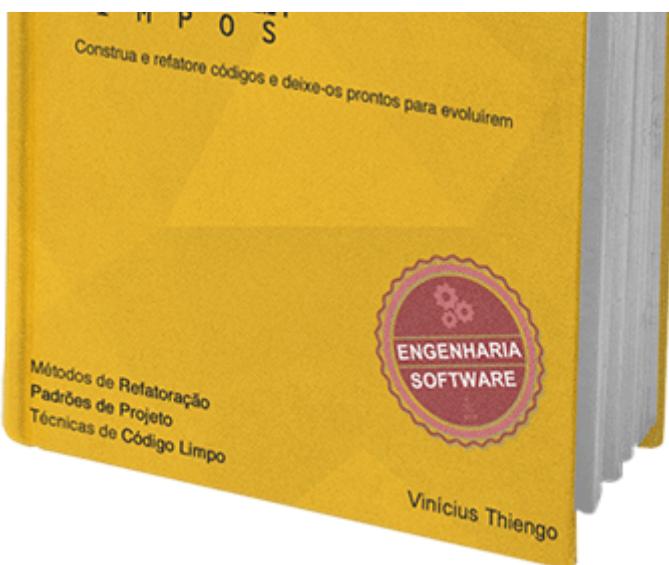
Acessar Curso

Receitas Android





Título Receitas Para Desenvolvedores Android
Categoria Desenvolvimento Android
Autor Vinícius Thiengo
Edição 1^a
Ano 2017
Capítulos 20
Páginas 936

[Acessar Livro](#)

Título Refatorando Para Programas Limpos

Categoria Engenharia de Software

Autor Vinícius Thiengo

Edição 2^a

Capítulos 46

Páginas 599



[Acessar Livro](#)[curso gratuito no Blog.](#)

Conteúdo Exclusivo

Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

Seu melhor Email

Cadastrar email

Curso Android, gratuito </>

Tudo bem?

Primeiro, pegue um café ☕ ☐. Pois o conteúdo é muito [importante para desenvolvedores Android](#) e está bem completo.

Neste artigo vamos passo a passo destrinchar, com um olhar de desenvolvedor, a API de drawables vetoriais estáticos no Android.

API que se utilizada de maneira consciente vai somente trazer ganhos a todo o projeto de aplicativo, independente do domínio de problema.

Um informe importante:

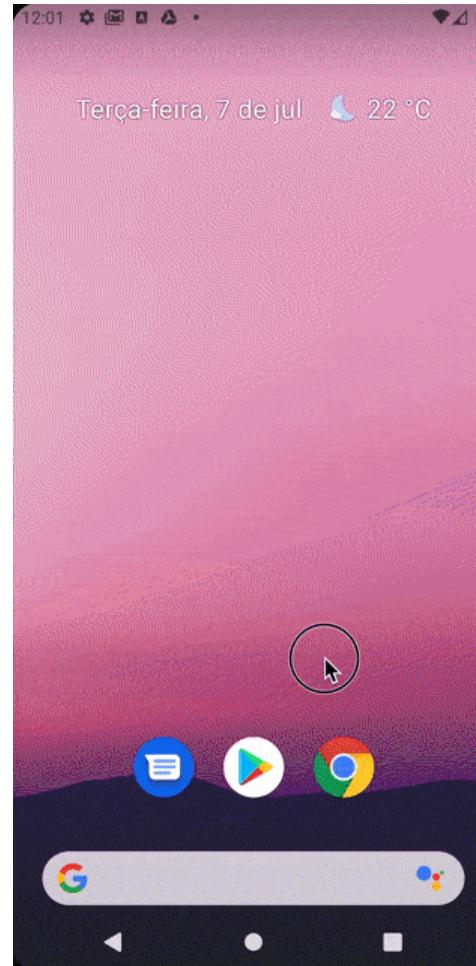
Não deixe de dar um olhar especial à seção em que abordaremos a API de suporte de vetores... pois neste ponto do conteúdo vou lhe explicar, passo a passo, o porquê de evita-la.

O artigo está dividido em duas partes:

Primeiro vamos ao estudo completo das imagens vetoriais no Android;

e Então vamos a um projeto de exemplo. Projeto que simula um app real de acampamento de verão.





Ao longo do artigo você também tem disponível os [slides](#) e as [vídeo aulas](#), caso prefira alguns destes formatos. De qualquer forma, não deixe de também ler o conteúdo por completo.

Antes de prosseguir, não esqueça de se **inscrever na lista de e-mails** do Blog para receber os novos conteúdos Android em primeira mão e também em versão PDF (lembrando que os PDFs dos artigos são liberados somente aos inscritos da lista e-mails... e ela é gratuita).

A seguir os tópicos abordados neste "artigo aula":

[O porquê das imagens vetoriais](#);

[Imagens vetoriais no Android](#):

[Quando utilizar e quando não utilizar](#):

[O processo de renderização](#):

[Fique atento ao tamanho do arquivo](#).

[Estrutura básica \(o necessário\)](#);

[Suporte ao SVG e ao PSD](#);



Referência em código:

O comum: carregamento direto no XML de layout:



= true;

Projeto Android:

Protótipo estático;

Iniciando um novo aplicativo;

Configurações Gradle;

Strings de sistema;

Configurações AndroidManifest;

Ícones de abertura do app.

Configurações de estilo:

Cores;

Dimensões;

Tema.

Ícones de sistema:

Como assim: "(...) não padronização das densidades de telas";

Origem e download;

Ícones de topo;

Ícones do menu "opções de itinerário";

Ícone de item selecionado.

Classes de domínio;

Pacote de dados;

Pacote adapter;

Layout de item;



A sombra no CardView;

ViewHolder



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.

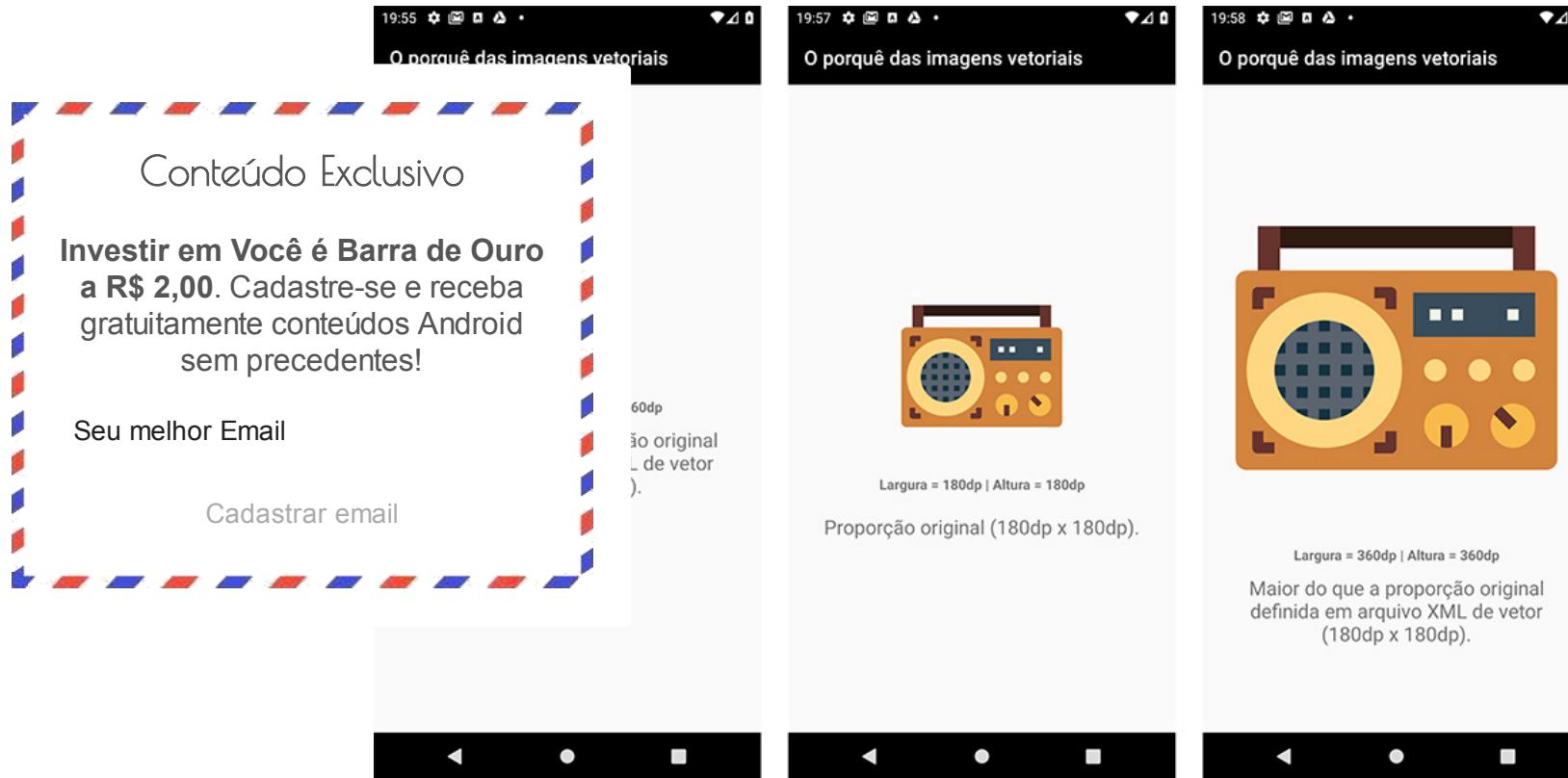
Mais precisamente em 1999, com a possibilidade de uso de vetores também em páginas Web.

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".

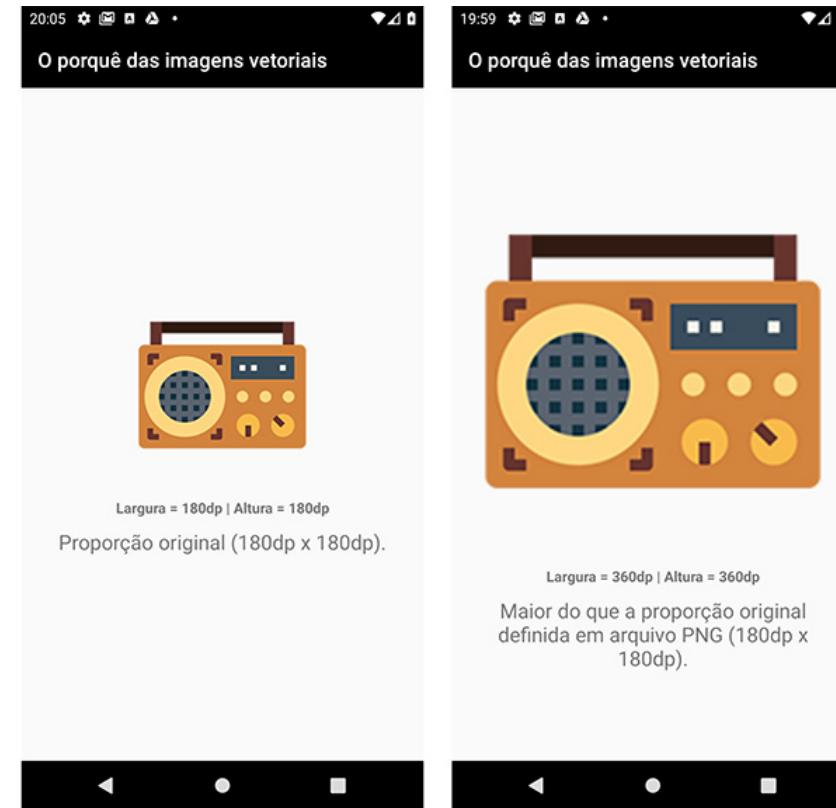
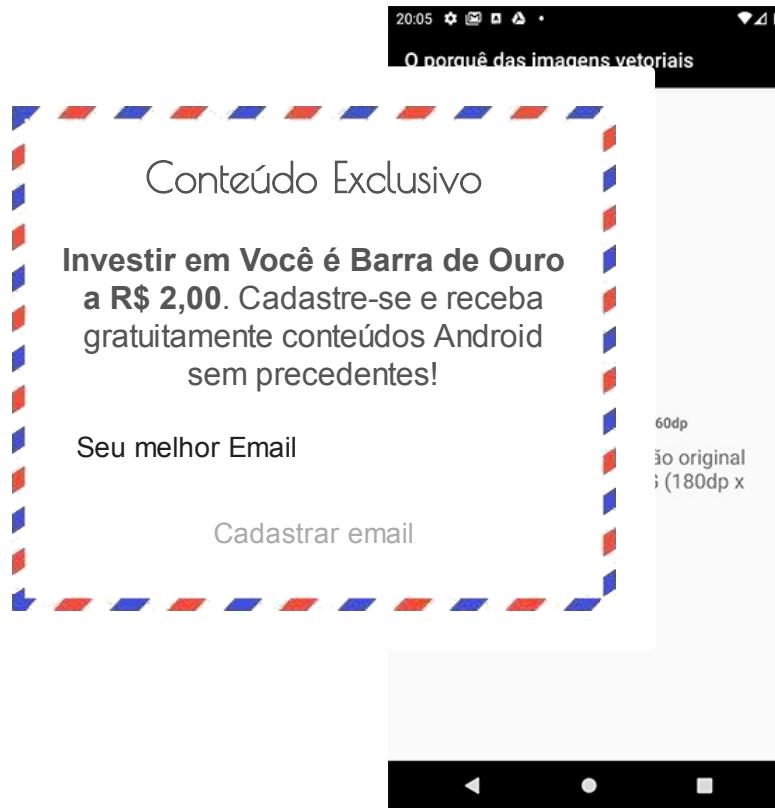
Então quando levando em conta os termos visuais a olhos humanos, acaba ficando "tudo ok" utilizar imagens rasterizadas em proporções menores do que as dimensões originais.

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

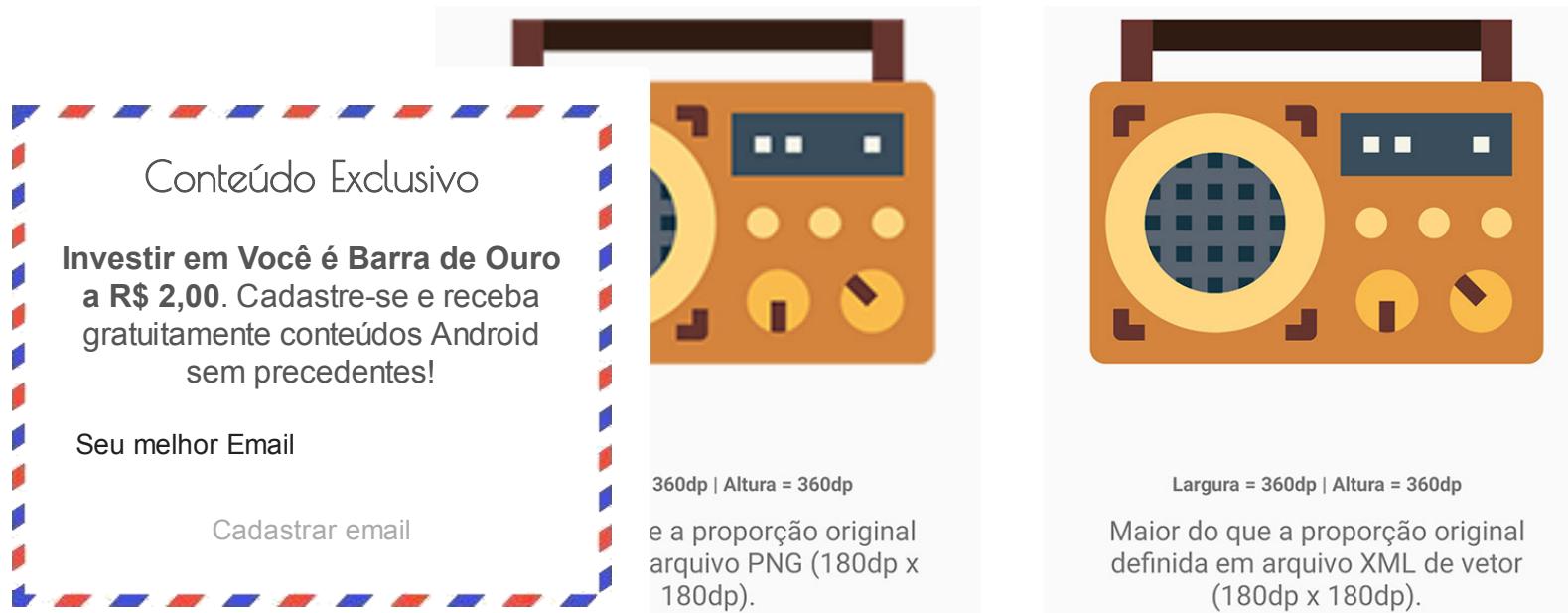
... neste caso é nítido a qualquer um que estas imagens perdem em qualidade. Isso por sofrerem com a "falta de informação".

Ou seja, pixels extras que faltam à imagem.





Mesmo que pouco perceptível, a terceira imagem 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.



As imagens rasterizadas são baseadas em:

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

Já as imagens vetoriais 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 com o artigo □.

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:



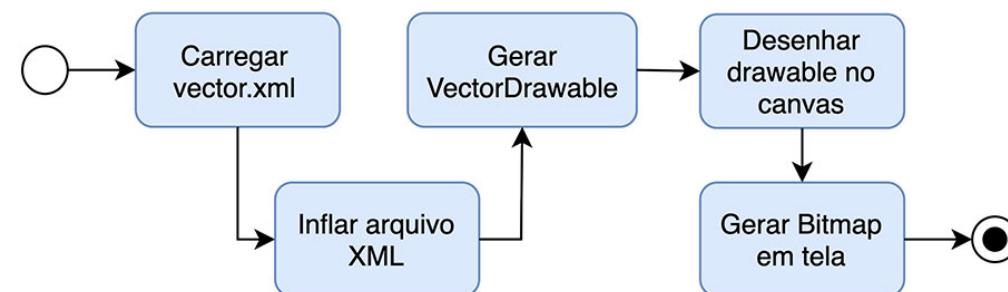
XML de vetor.
São de extrema importância a um desenvolvedor Android, independente do nível dele. ["ícone de sistema"](#) onde o suporte de vetores Android é completo.

de sem necessidade de ao menos quatro arquivos de imagens no Android. As versões drawable em **mdpi**,
licativos deve ser moderado quando saímos do contexto "ícones de sistema".
r imagens vetoriais quando:
iores que 200dp x 200dp.
o de imagens rasterizadas seja em disparado uma melhor escolha.
ais.
ender "como funciona" e "como utilizar" o antigo e ainda válido modelo de [imagens rasterizadas em folders](#)

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



um **OutOfMemoryException**.

é o "tamanho do arquivo" vetorial.

Isso, pois eles devem entender que essa limitação é óbvia e você desenvolvedor deve literalmente não cometer

espeita as dimensões máximas de 200dp x 200dp recomendadas na documentação oficial...

er utilizado, pois será melhor escolha do que uma imagem rasterizada.

nportar arquivos SVG e arquivos PSD (já já falaremos mais sobre estes dois formatos).

is vetoriais com características e detalhes a nível de imagens rasterizadas (por consequência a estrutura XML



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> e **<path>**. E mais atributos além dos que apresentarei aqui.
drawables vetoriais no desenvolvimento de aplicativos Android e conhecendo os contextos onde esses

cê precisa, como desenvolvedor Android, saber sobre vetores no Android.
editorial apresentada acima:

<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>**.

Independentemente 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



atualizar em arquivo de vetor... isso nas raras vezes que você precisar trabalhar diretamente na estrutura do

dá suporte à versões do Android abaixo da API 21, então o recomendado é que o valor de **tint** seja *hard-coded*, ele deve ser colocado na "unha".

Accent, por exemplo. Utilize o hexadecimal da cor direto como valor (*hard-coded*), #FF0000 (vermelho). [Studio](#) junto ao plugin do Gradle, assim que o app é compilado, gera versões rasterizadas de cada drawable (você terá uma explicação detalhada sobre o porquê disso).

geradas que vão ser utilizadas nas versões do Android abaixo da API 21.

res não são colocadas *hard-coded*, pode haver problemas na geração dessas imagens rasterizadas...

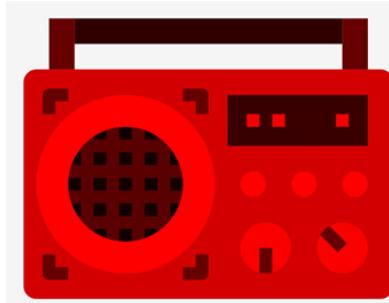
ativo, em versões do Android abaixo da API 21, totalmente desalinhado com o esperado.

a versões do Android abaixo da API 21, então para todos os atributos de arquivos vetoriais forneça valores S.

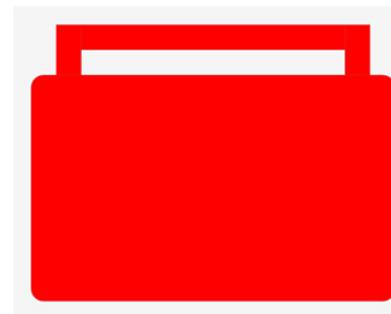
android:tint será aplicado. O valor padrão é **src_in**.



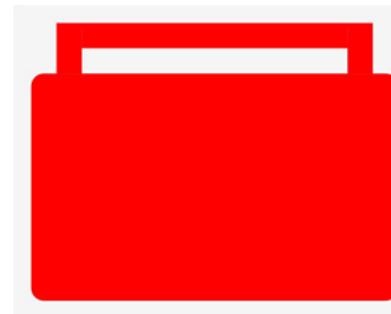
↳ **multiply**:



↳ screen:



↳ src_in;



↳ src_over:





android:tint com a cor vermelha (#FF0000).

Definido de maneira explícita, então o valor em **android:tintMode** é ignorado.

erem desenhados.

il total ou toda a configuração de caminho da imagem vetorial.

› para um vetor estático:

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 desse **<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... isará em termos de estrutura XML de um arquivo vetorial.



"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**:

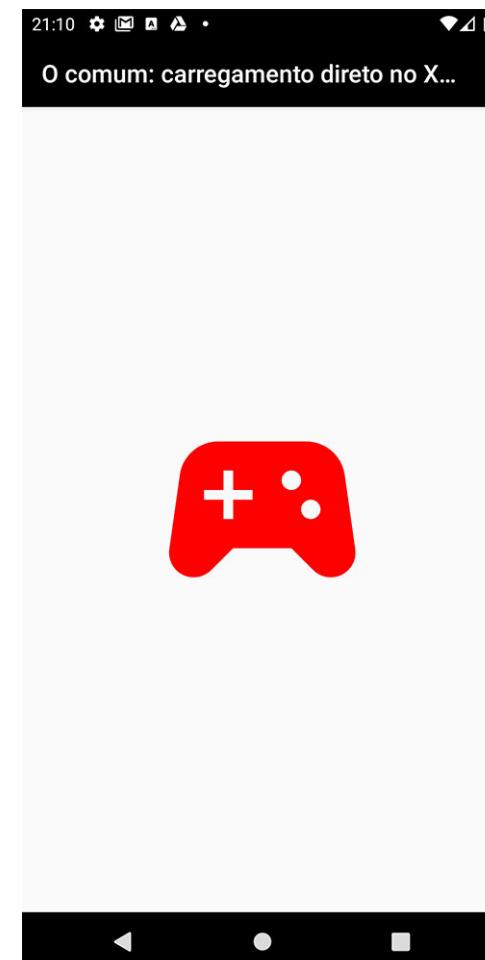
```
<vector
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="24dp"
    android:height="24dp"
    android:tint="#FF0000"
    android:viewportWidth="24"
    android:viewportHeight="24">

    <path
        android:fillColor="@android:color/white"
        android:pathData="M21.58,16.09l-1.09,-7.66C20.21,6.46 18.52,5 16.53,5H7.47C5.48,5 3.79,6.46 3.51,8.43l-1.09,7.66C2.2,17.63 3.39,19 4.94,19h0c0.68,0
    </vector>
```

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



:tor"
orts" />



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.

item_width e **android:layout_height** sobrescrevem respectivamente as definições **android:width** e



lregamento de imagem vetorial via código dinâmico.

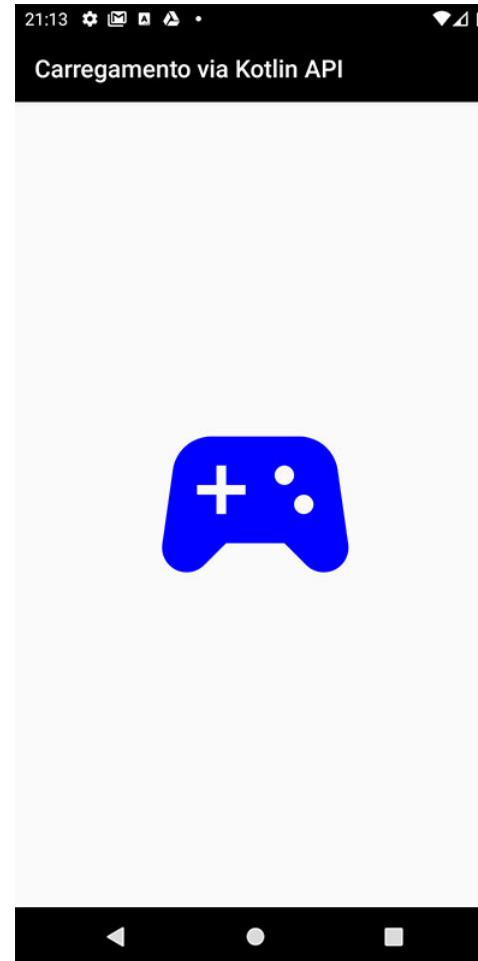
Do anterior, poderíamos ter em código Kotlin o seguinte:

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.
código dinâmico para criar um vetor do zero...
vetores da documentação oficial Android. Confesso que nem mesmo na comunidade Android eu

Conteúdo Exclusivo

Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

Seu melhor Email

Cadastrar email

detalhes com vetores, então você estará seguindo mais como um designer do que como um *developer*. Esse

ais foi adicionado a partir do Android 21, Lollipop.
etores mesmo em versões do Android anteriores à API 21 é preciso o uso das APIs de suporte.
API 14, Ice Cream Sandwich (será isso verdade?! *We will see*).

licativo, ou **build.gradle (Module: app)**, a definição correta para trabalho com as API de suporte.

```
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.

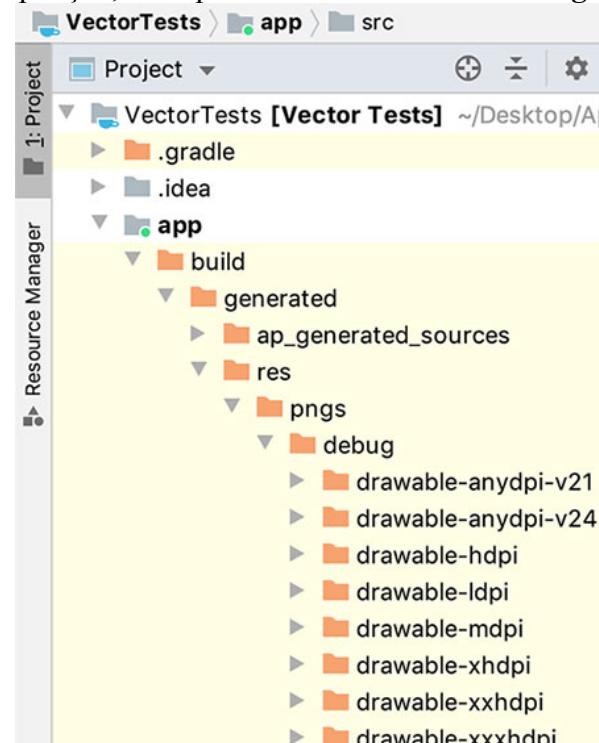


Porém tem um grande problema:

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



ido em aparelhos com o Android abaixo da API 21 haverá exceções e o app fechará de maneira abrupta. nculo de um vetor ao um [TextView](#), por exemplo, com o simples atributo `android:drawableStart`. go dinâmico, criar um objeto **Drawable** do vetor, utilizando o **ResourceCompat**, por exemplo. do [TextView](#) via `drawableStart`. suficiente para "remover muito" da vantagem do trabalho com vetores ao invés de imagens rasterizadas: mático". n, Java, PHP, Python, ...) somente os algoritmos de domínio de problema. estático (XML, [HTML](#), CSS, ...). **seSupportLibrary = true** em projeto, como vou dar suporte a versões do Android abaixo do Lollipop? stá com uma versão mais atual) foi colocado no IDE uma ferramenta que junto ao plugin do Gradle faz com asterizadas. porte a versões do Android abaixo da API 21. Então é gerada seis versões de cada imagem vetorial presente i 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;

Conteúdo Exclusivo

Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

Seu melhor Email

Cadastrar email

izadas diretamente. Você deve continuar atualizando somente os vetores. Na compilação as novas versões *or do que quando não utilizando somente imagens rasterizadas, certo?*

do Android abaixo da API 21 e ele também faz uso de drawables vetoriais. Seu APK final ficará com alguns poucos bytes a mais.

iente a prejudicar o aplicativo.

ganho na manutenção do projeto continua sendo o que faz os pontos prós serem muito melhores do que os

xo da API 21, utiliza as imagens rasterizadas ao invés de vetores.

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 esta seção:

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

E aqui nós vamos seguir sem ela nos dois projetos do conteúdo □... 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"
```

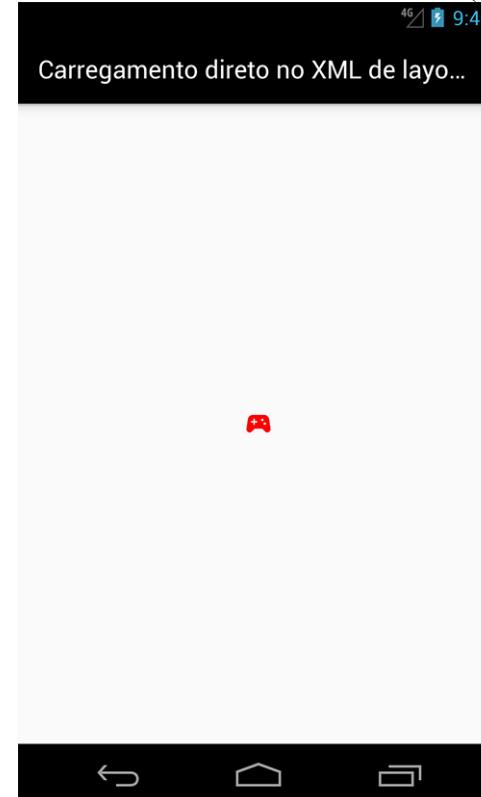


```
        android:contentDescription="@string/simple_vector"  
        android:src="@drawable/ic_baseline_sports_esports" />
```



ciso colocar na raiz do layout (de preferência no **ViewRoot** raiz) a [Android.com/apk/res-auto](#)".

do IDE acesse "Build" e aione "Clear Projetc" para assim remover as imagens rasterizadas geradas na
ior em um emulador com o Android API 16 (Jelly Bean), temos:

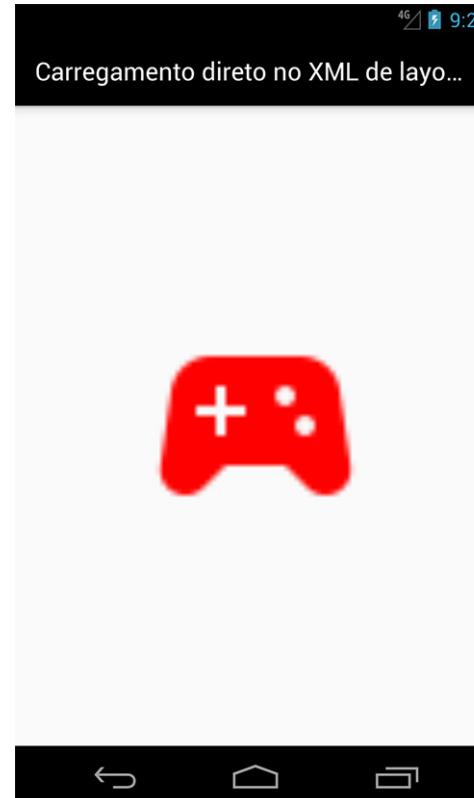


Você deve estar se questionando:

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 pixelagem.

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`.

As configurações corretas para que em versões do Android abaixo da API 21 o layout seja apresentado



[Kotlin API](#) também é válido aqui.
ble em componente visual é muito similar:

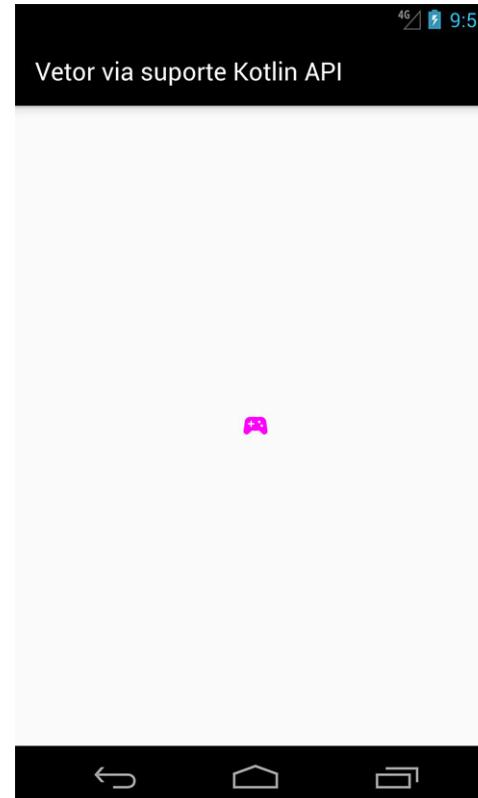
Desta vez utilizamos o método `setImageResource()` para carregamento do drawable.

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:

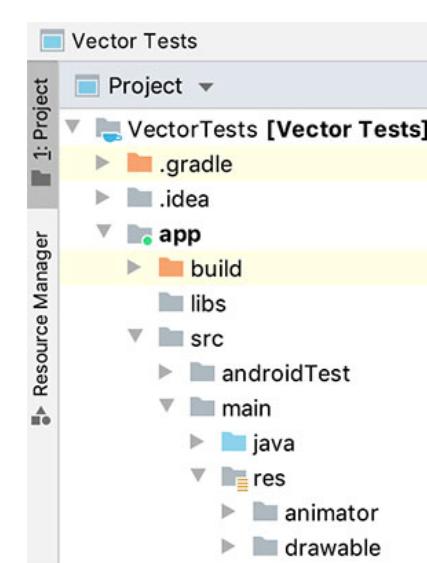
```
...
iv_vector_sample.setImageDrawable(
    ResourcesCompat.getDrawable(
        resources,
        R.drawable.ic_baseline_sports_esports,
        null
    )
)
...
```

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

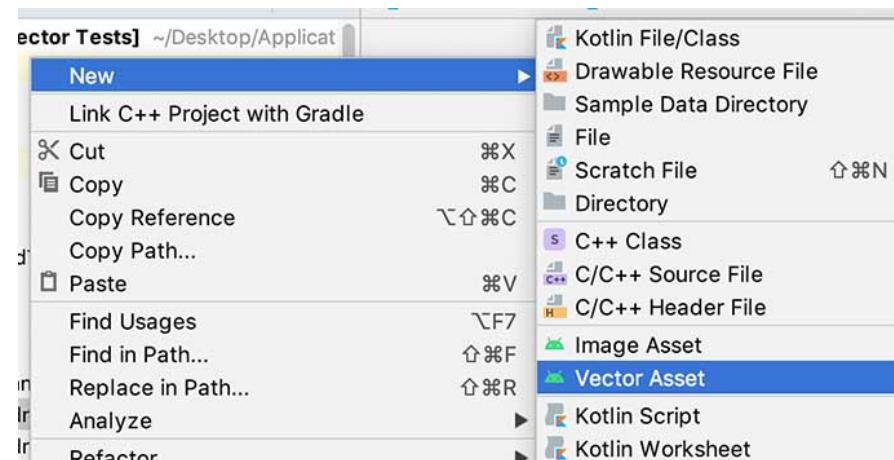
Primeiro vamos a um passo a passo sobre como obter ícones já disponíveis na ferramenta.



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

Clique em "New";

Logo depois em "Vector Asset".

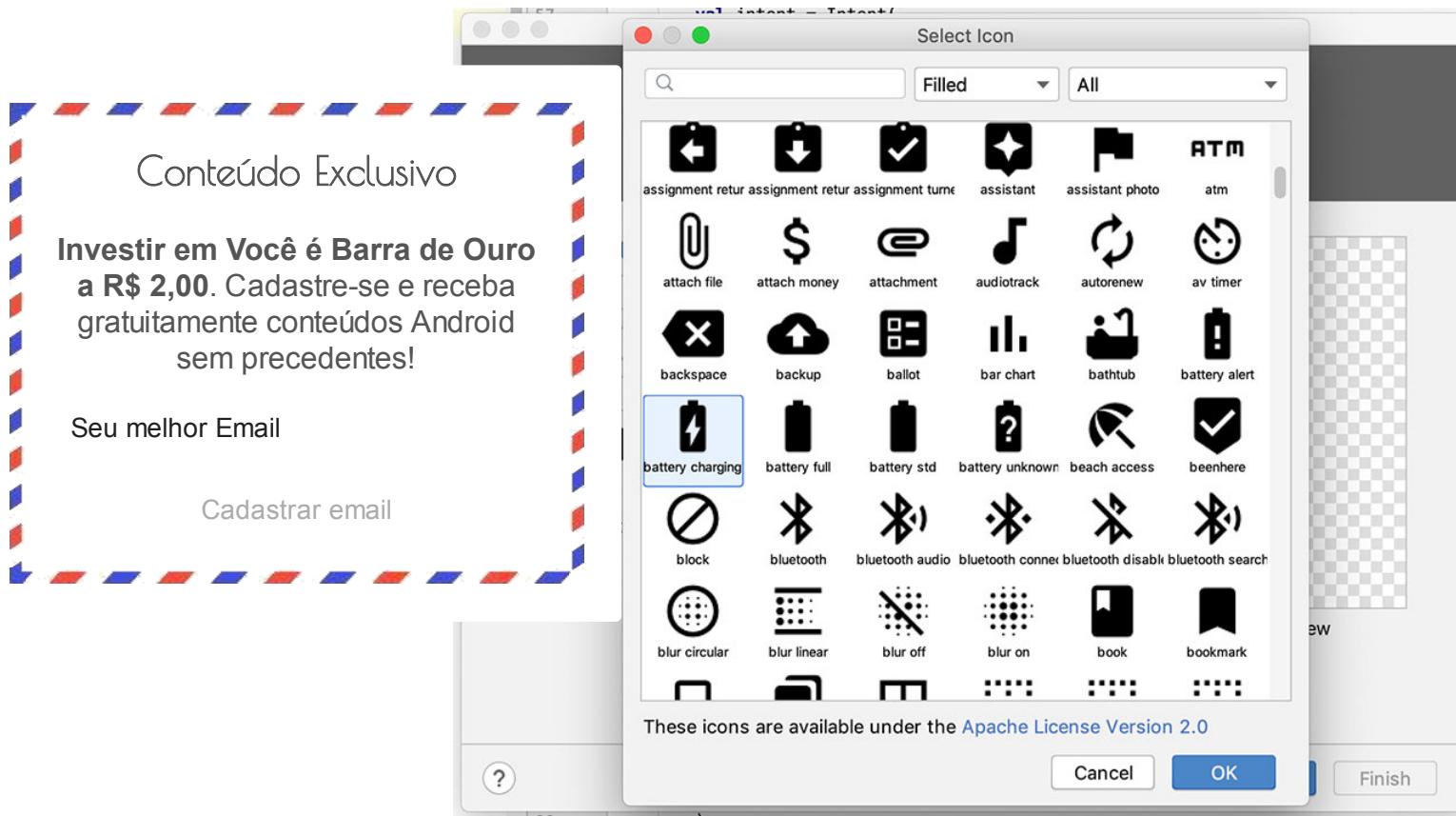


Com a caixa de diálogo do Vector Asset Studio aberta:

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 atualizá-los novamente. Então é isso.

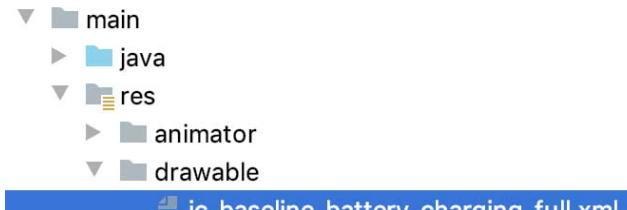
Em questão de poucos minutos é possível ter ao menos toda a configuração de ícones de sistema de um aplicativo Android.

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**:





completa) neste exemplo:

res/android"

```
<vector>
    <path data="M15.6,17.4c-0.6,0-1.34,-0.6 1.34,-1.33V5.33C17.4,6 16.4,4 15.6
    </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:

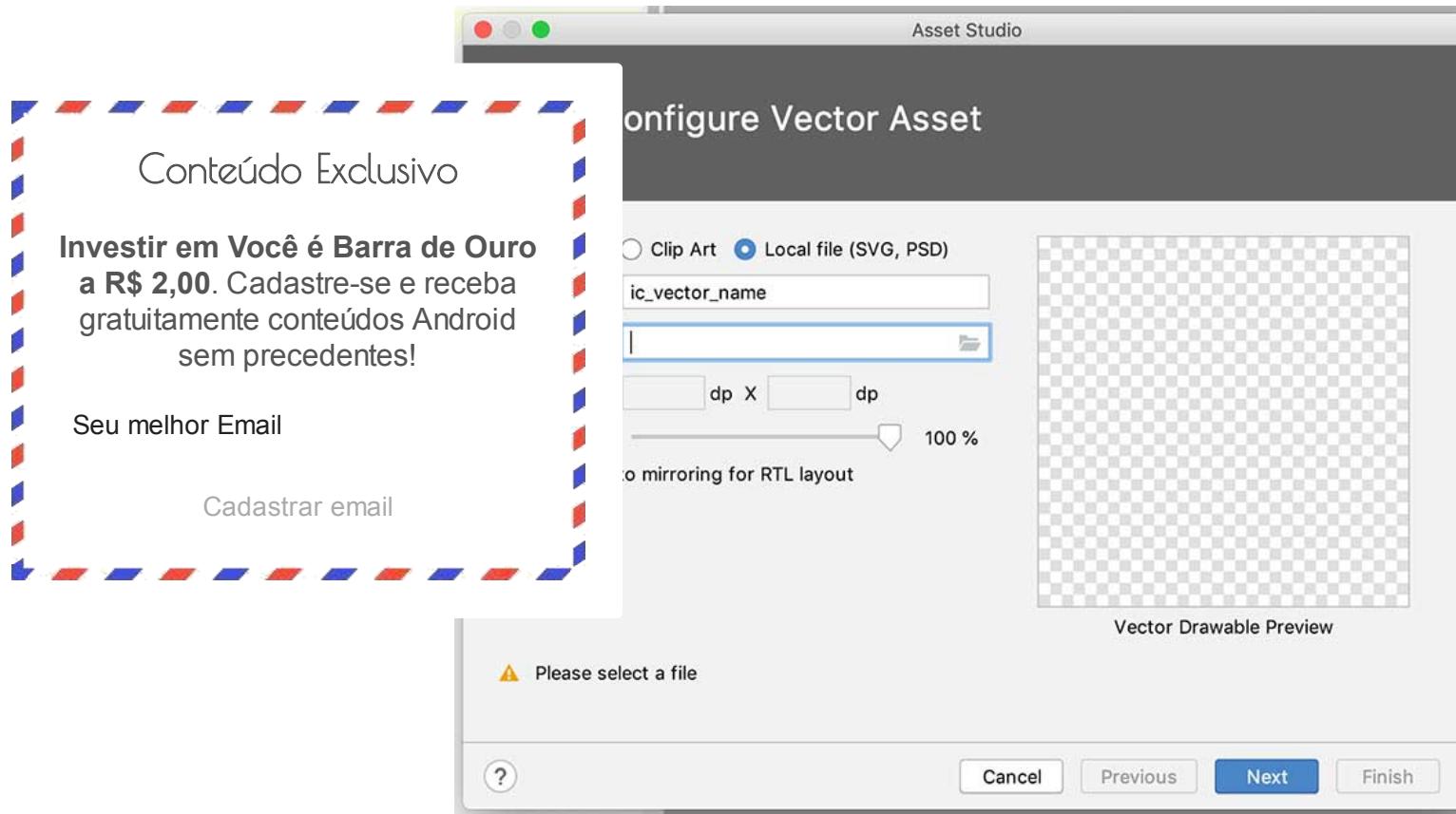
Clique em "New";

Logo depois em "Vector Asset".

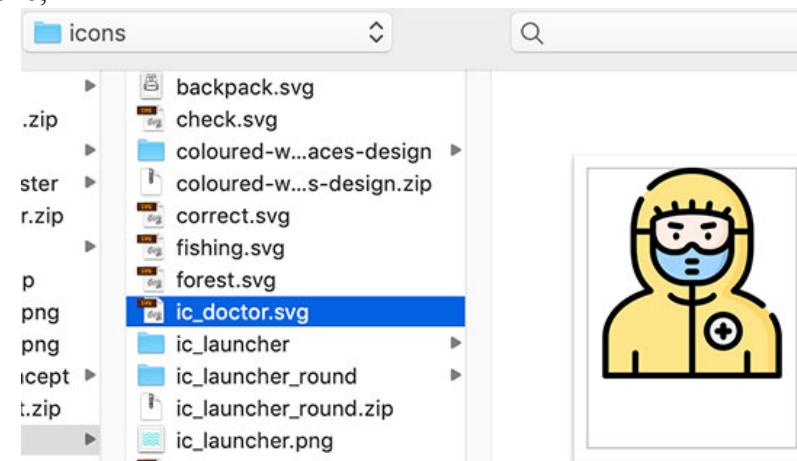
Com a caixa de diálogo do Vector Asset Studio aberta:

Selezione "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 m todas as possíveis variantes de compilação.

ser modificados somente depois que o vetor já tiver sido escolhido. Caso contrário será necessário atualiza-

de vetores é tão simples quanto o uso de ícones já presentes na ferramenta.

Vector Asset Studio, aqui os vetores selecionados também deverão ficar no folder /res/drawable. e de médico):

res/android"

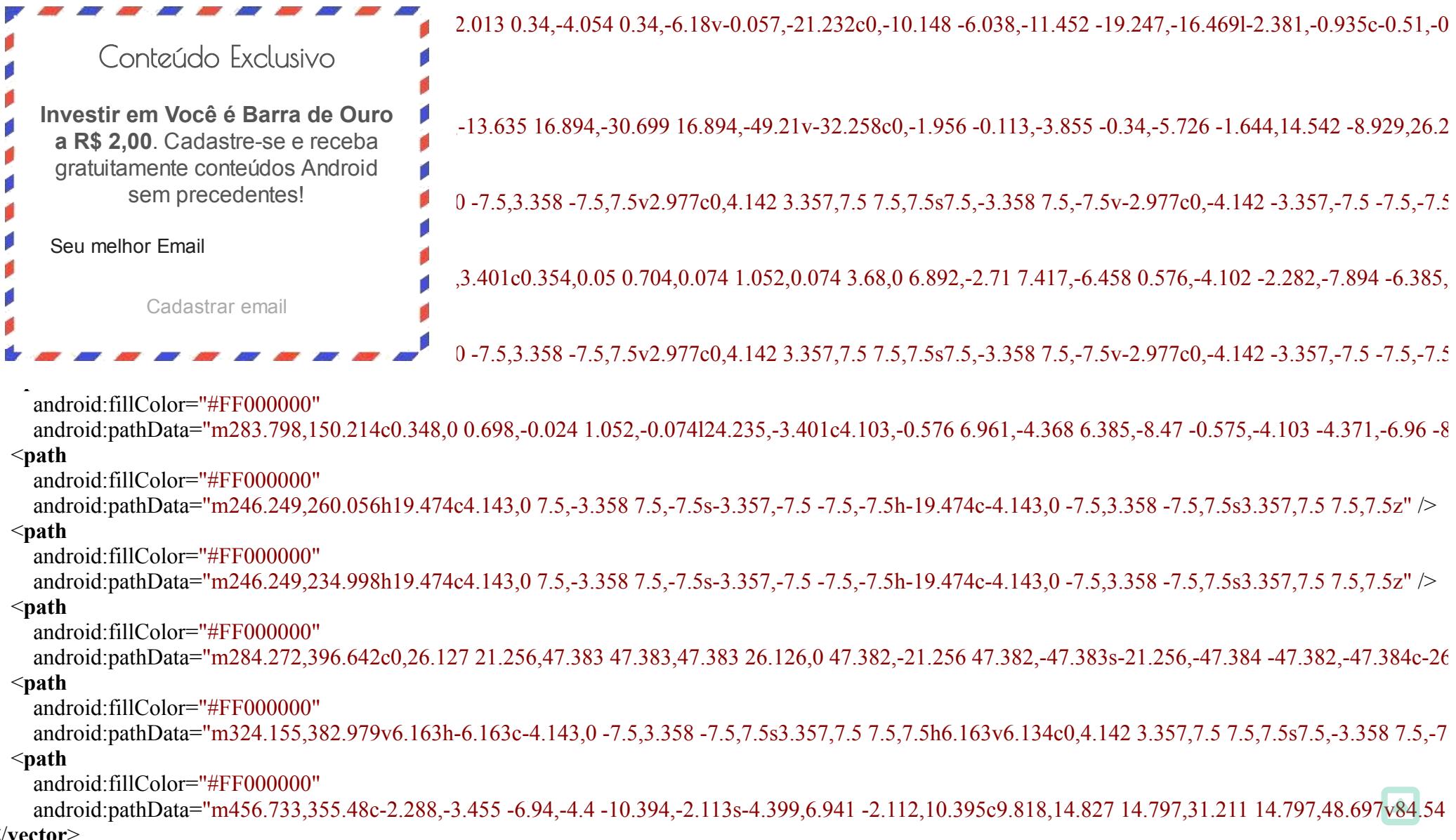


```

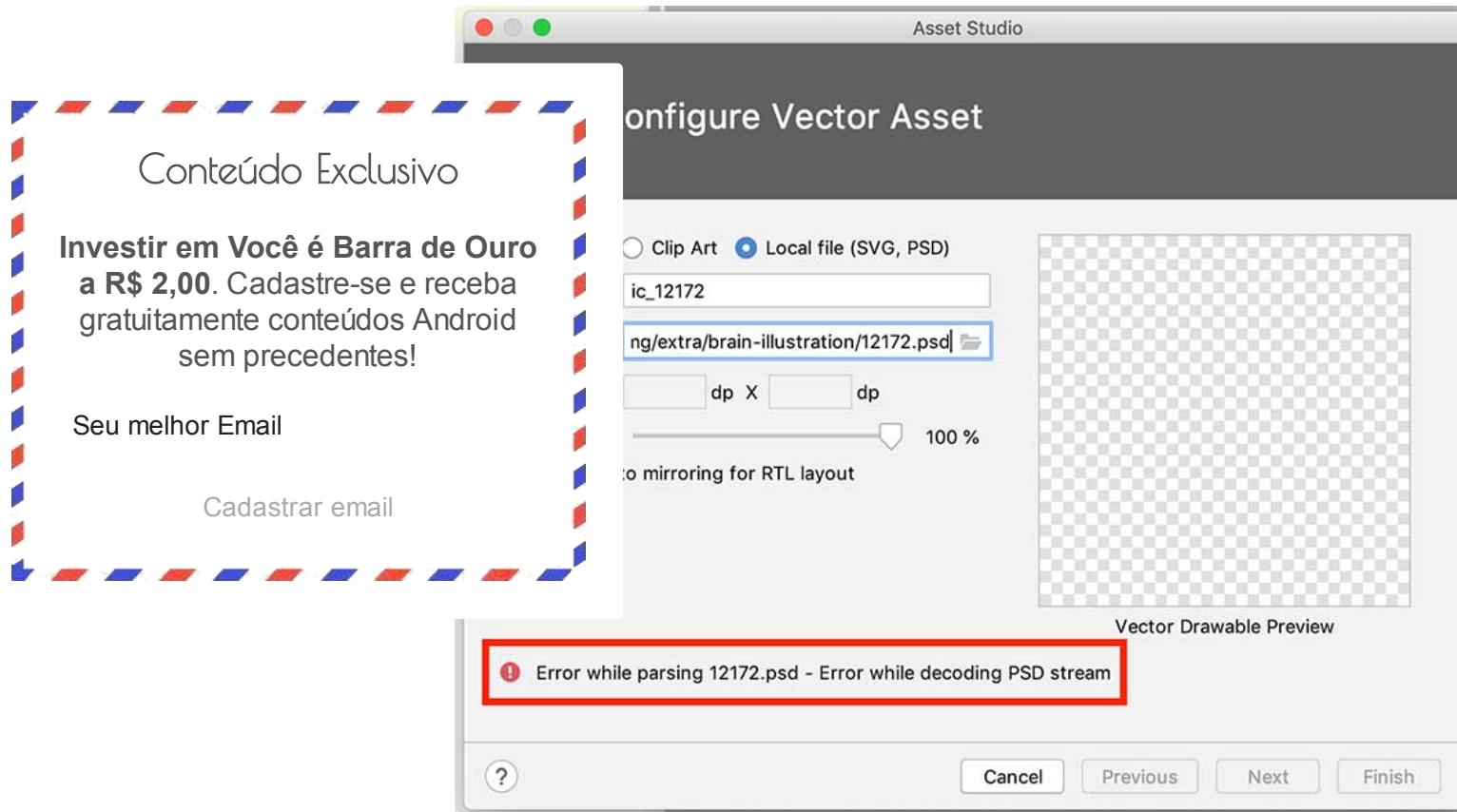
android:type="evenOdd"
    android:pathData="m351.299,154.987v0.057c0,2.126 -0.113,4.167 -0.34,6.18 0.227,1.871 0.34,3.77 0.34,5.726v32.258c0,18.51 -6.321,35.575 -16.894,49.21
<path
    android:fillColor="#a1d4ff"
    android:fillType="evenOdd"
    android:pathData="m161.07,161.053c-0.255,1.928 -0.368,3.883 -0.368,5.896v32.258c0,20.154 7.483,38.608 19.785,52.781 14.796,17.036 36.623,27.836 60.0
<path
    android:fillColor="#fdd76f"
    android:fillType="evenOdd"
    android:pathData="m398.182,190.364v-35.376h-26.305v34.498c0,53.717 -22.308,82.12 -40.62,108.255 -2.211,3.146 -7.937,13.181 6.151,17.065 46.771,11.0
<path
    android:fillColor="#fdd76f"
    android:fillType="evenOdd"
    android:pathData="m398.182,154.335c0,-92.211 -63.919,-146.835 -142.182,-146.835 -2.551,0 -5.074,0.057 -7.568,0.17 69.476,8.05 123.446,61.058 123.446,61.058
<path
    android:fillColor="#ffeee6"
    android:fillType="evenOdd"
    android:pathData="m294.55,109.122c-12.245,-1.446 -25.398,-2.183 -38.55,-2.183s-26.305,0.737 -38.522,2.183c-13.181,1.559 -25.256,3.969 -35.12,7.228 -35.12,7.228
<path
    android:fillColor="#eff6ff"
    android:fillType="evenOdd"
    android:pathData="m331.655,356.758c-22.025,0 -39.883,17.858 -39.883,39.884 0,22.025 17.858,39.884 39.883,39.884s39.883,-17.858 39.883,-39.884c-0.0

```

```
<path
    android:fillColor="#ffdfcf"
```



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;

[Freepick](#) - 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 artigo 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.



íble ou a API de suporte `AnimatedVectorDrawableCompat`.
etores já neste conteúdo.

ndo em mente que somente a parte de vetores estáticos já é grande o suficiente para um único e longo artigo
no contexto "ícones de sistema", são os vetores que realmente fazem diferença em projeto, diferença para a
robusta quanto a [Lottie API](#), por exemplo, que permite animações mais complexas em arquivos mais leves.
n artigo aula futuro aqui do Blog, não se preocupe com isso.
e está sendo apresentado neste conteúdo e também conhecer ou a [API ObjectAnimator](#) ou a API AnimatorSet
ações de vetores no Android não precisará aguardar esse novo conteúdo.

itular alguns dos pontos negativos.
que fazem com que vetores não sejam sempre uma melhor solução frente a imagens rasterizadas:
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 dúvida 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.

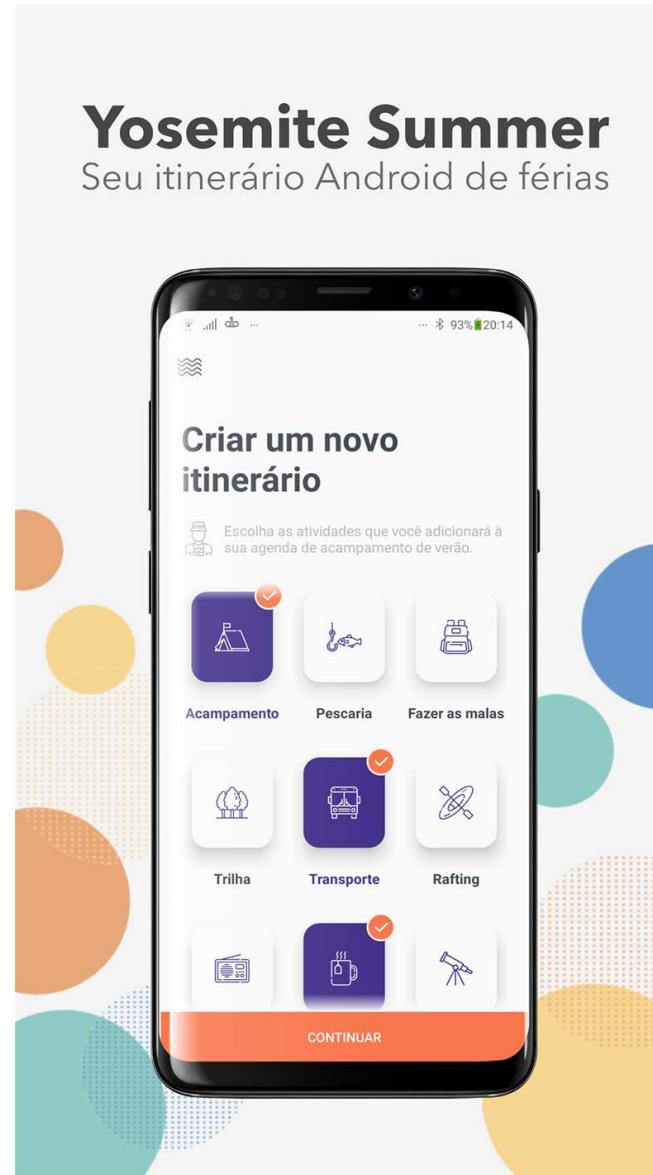
Repositório do projeto

O projeto Android que foi utilizado para rodar os algoritmos apresentados até este ponto do artigo...

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

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

Slides



Nosso objetivo será trocar os ícones rasterizados, PNG, por ícones de vetores. Consequentemente tendo como ganho ao menos:
Menor arquivo APK;

E o principal → facilidade em manutenções futuras na parte de design do projeto.

Todo o projeto de exemplo será desenvolvido em duas partes:

Primeiro vamos a configuração total de projeto, porém utilizando imagens rasterizadas;

Depois apenas vamos trocar os tipos de ícones de sistema para vetores... e realizar testes.

Você pode acessar toda a configuração do projeto no GitHub público dele em:

<https://github.com/viniciusthiengo/yosemite-summer-camp>.

De qualquer forma, recomendo que você estude todo o projeto pelo artigo para entender cada um dos detalhes abordados.



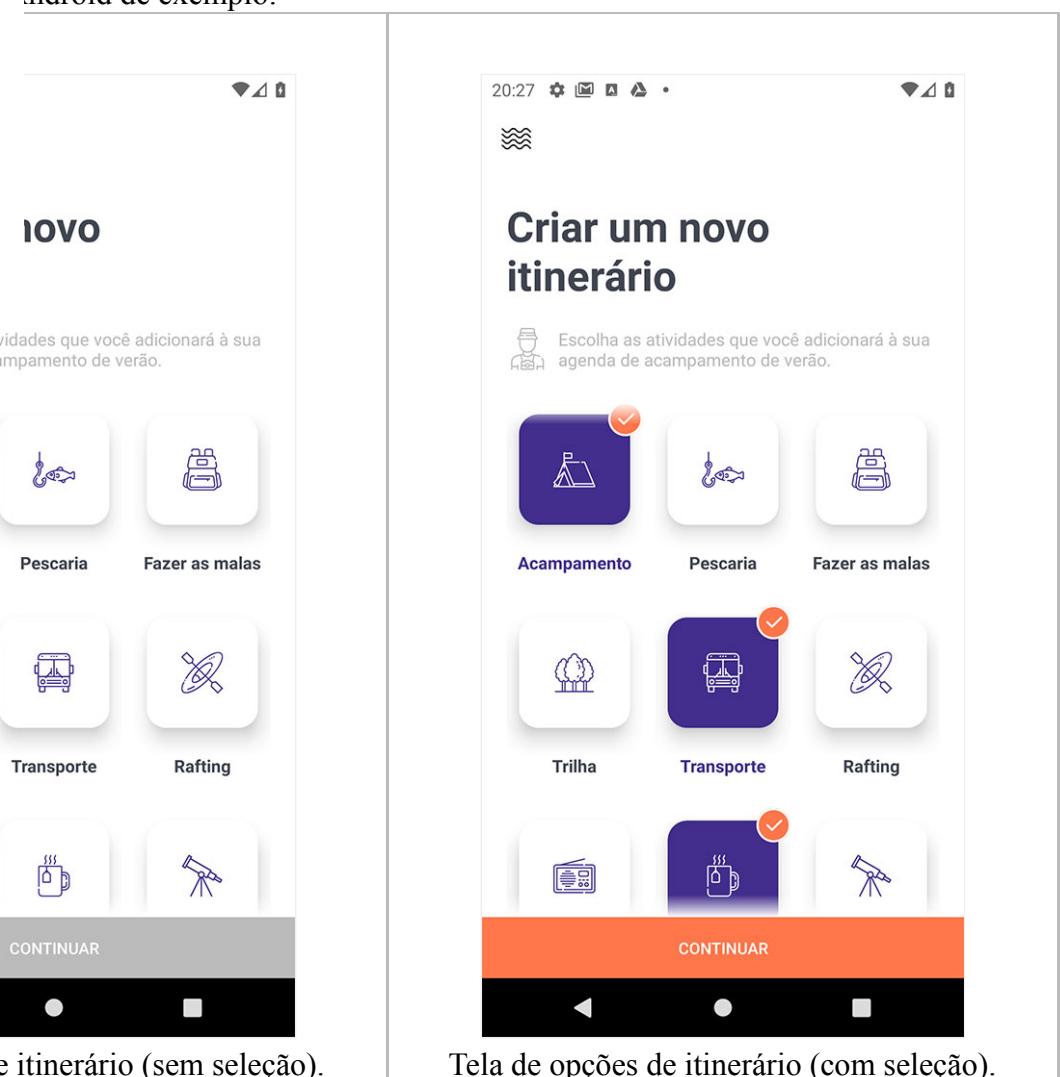
nte do [lado estratégico de um projeto](#) de aplicativo Android.

am ser feitas antes mesmo da primeira linha de código ser colocada em IDE.

inúmeros artigos do Blog, a fase de codificação (lado tático do projeto) é facilitada consideravelmente.

dúvidas em fase de desenvolvimento sejam praticamente nulas.

ndroid de exemplo:



Tela de opções de itinerário (sem seleção).

Tela de opções de itinerário (com seleção).

Iniciando um novo aplicativo

<https://www.thiengo.com.br/porque-e-como-utilizar-vetores-no-android>

Em seu ambiente de desenvolvimento, sua instalação do Android Studio, inicie um novo projeto Kotlin Android como a seguir:

Nome da aplicação: Yosemite Summer Camp



→ dos aparelhos Android em mercado sendo atendidos;

definidos por padrão.

→ projeto, ou **build.gradle (Project:YosemiteSummerCamp)**:

```
    "maven { url "https://kotlin.bintray.com/kotlin-android"
      name "ktor-gradle-plugin"
      authentication "Basic realm = \"ktor-gradle-plugin\""
      credentials {
        username "ktor-gradle-plugin"
        password "$kotlin_version"
      }
    }
```

```
allprojects {
  repositories {
    google()
    jcenter()
  }
}
```

```
task clean(type: Delete) {
  delete rootProject.buildDir
}
```

E então as configurações do nosso Gradle Nível de Aplicativo, ou **build.gradle (Module: app)**:

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'
```

```
android {
  compileSdkVersion 30
```

A seguir você tem a versão em slides de todo o conteúdo de Vetores Android que foi apresentado até este ponto do artigo.

É exatamente o mesmo conteúdo:

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email

Porque e como Utilizar Vetores no **Android**

Thiengo.com

1 of 79

Projeto Android

Como projeto de exemplo, para facilitar o entendimento de como e quando aplicar vetores estáticos em projeto...

... esse exemplo será parte de um aplicativo Android de acampamento de verão, onde a tela de itinerário (um [grid menu](#)) estará disponível para o usuário poder construir o roteiro dele em *camping*.

buildToolsVersion "30.0.0"



```
targetCompatibility JavaVersion.VERSION_1_8
}
kotlinOptions {
    jvmTarget = '1.8'
}
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])

    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"

    implementation 'androidx.core:core-ktx:1.3.0'
    implementation 'androidx.appcompat:appcompat:1.1.0'

    implementation 'com.google.android.material:material:1.1.0'
}
```

Se no tempo que você estiver consumindo este artigo já houver versões mais atuais das bibliotecas, plugins e Android API referenciadas acima. Então utilize as versões mais atuais, pois o projeto deverá continuar executando sem problemas.

Strings de sistema

A seguir o arquivo de Strings estáticas de nosso projeto, mais precisamente o arquivo **/res/values/strings.xml**:



```
</string>

<string name="item_selected">
    Ícone item selecionado
</string>

<string name="button_continue">
    Continuar
</string>
</resources>
```

Lembrando da importância deste arquivo principalmente para a manutenção do projeto.

Se a String é estática, não sofre atualizações por parte do usuário, então ela deve estar no arquivo de **Strings**.

Isso pensando também no momento de internacionalização do aplicativo, onde outros idiomas serão necessários.

Configurações AndroidManifest

Abaixo a configuração completa do AndroidManifest.xml de nosso projeto Android de exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="thiengo.com.br.yosemitesummercamp">
```

```
<application
    android:allowBackup="true"
```



```
und"
```

```
onBar"
```

```
:ion.MAIN" />
category.LAUNCHER" />
```

```
</activity>
</application>
<manifest>
```

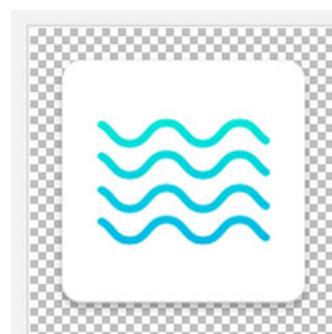
Como a atividade **ItineraryActivity** é basicamente uma tela de [menu vertical](#), onde alguns componentes visuais são apresentáveis somente na vertical...
... como isso já está definido desde o protótipo estático, então achei prudente permitir o uso dessa atividade somente no modo *portrait*, colocando em **<activity>** o atributo **android:screenOrientation="portrait"**.

Mas é aquilo, para travas de tela em *portrait* ou em *landscape* é aconselhável que você tenha boas explicações.

Pois travando a tela você estará interferindo diretamente na experiência do usuário (*user experience - UX*).

Ícones de abertura do app

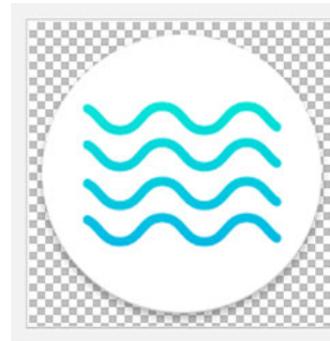
Ainda é preciso colocar em projeto os ícones de abertura de aplicativo. No caso os ícones que aparecem na tela de apps do aparelho Android.
Primeiro o ícone em versão retangular, **ic_launcher.png**:



[/res/mipmap-mdpi/ic_launcher.png;](#)
[/res/mipmap-hdpi/ic_launcher.png;](#)



.png:



[/res/mipmap-xhdpi/ic_launcher_round.png;](#)
[/res/mipmap-xxhdpi/ic_launcher_round.png;](#)
[/res/mipmap-xxxhdpi/ic_launcher_round.png.](#)

Aqui não trabalharemos com ícones adaptativos de aplicativo. Mesmo sabendo que eles também podem fazer uso de drawables vetoriais. Ícones adaptativos vão ser assunto de um artigo aula futuro.

Configurações de estilo

Agora os arquivos responsáveis pelo [estilo](#), [tema](#), do projeto Yosemite Android.

Cores

Primeiro o arquivo de definição de cores, mais precisamente o arquivo [/res/values/colors.xml](#):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Branca. -->
    <color name="colorPrimary">#FFFFFF</color>

    <!-- Branca. -->
    <color name="colorPrimaryDark">#FFFFFF</color>

    <!-- Laranja. -->
    <color name="colorAccent">#FF774B</color>

    <!-- Cinza escuro. -->
```



```
<color name="colorDarkGrey">#414651</color>
```



'color>

or>

arquivo /res/values/dimnes.xml:

n>

que é repetida inúmeras vezes em projeto. Repetida no mesmo contexto e em lugares distintos:

```
/res/layout/menu_item.xml,
/res/drawable/cv_background_normal.xml;
e /res/drawable/cv_background_selected.xml.
```

Continue com a leitura. Logo logo chegaremos nos arquivos XML acima.

Tema

Por fim os arquivos de estilo, tema.

Primeiro o arquivo de definição de estilo comum a todas as versões do Android que podem executar o aplicativo.

Mais precisamente o arquivo /res/values/styles.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- Tema base do aplicativo. -->
    <style
        name="AppTheme"
        parent="Theme.AppCompat.Light.DarkActionBar">

        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.NoActionBar">
```

```
<item name="windowActionBar">false</item>
<item name="windowNoTitle">true</item>
```



Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

!_ Seu melhor Email

O tema abaixo foi criado para que fosse possível mudar
a cor do ícone acima à esquerda na barra de topo.

Jetpack Compose vai acabar com essa "dor de cabeça".

```
-->
<style
    name="AppTheme.ToolbarTheme"
    parent="@style/ThemeOverlay.AppCompat.ActionBar">

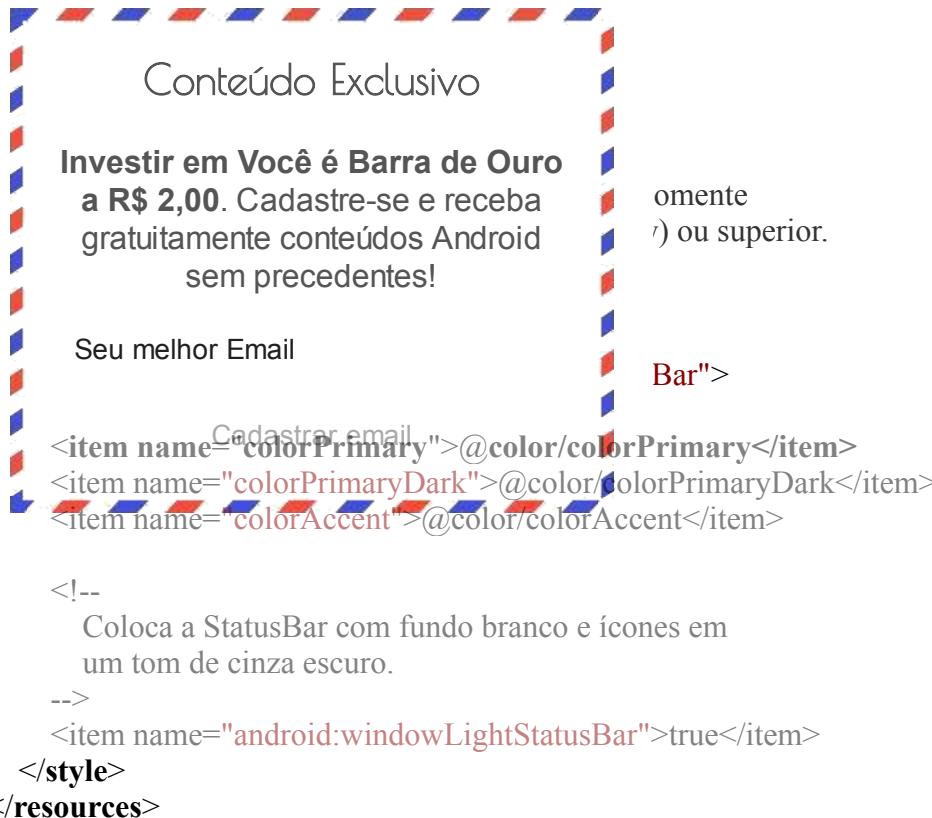
    <item name="colorControlNormal">@color/colorDarkGrey</item>
</style>

<!--
    O tema abaixo foi criado para as Views do layout que
    têm a função de colocar efeito de fade no framework
    de lista (aqui o RecyclerView).
-->
<style name="AppTheme.Shadow">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">18dp</item>
    <item name="android:background">@drawable/white_gradient_fade</item>
</style>
</resources>
```

Não se preocupe agora com o entendimento do arquivo **white_gradient_fade.xml** que está definido no tema **AppTheme.Shadow**. Logo vamos à construção dele, assim que chegarmos ao layout de item de menu.



Agora vamos a versão de **styles.xml** que sobrescreve o que é necessário no tema para um melhor *fit* do aplicativo em aparelhos Android [a partir da versão 23, Marshmallow](#)



Ícones de sistema

Com o protótipo estático já definido, na fase estratégica do projeto, é possível colocar inúmeros recursos em projeto Android antes mesmo da primeira linha de código dinâmico.

E um desses recursos é todo o pacote de ícones de sistema.

E nesta primeira parte do projeto nós estaremos utilizando somente imagens rasterizadas, como informado logo no início dele.

Sendo assim, devido à não padronização das densidades de telas de aparelhos Android...

... devido a isso, ao invés de ter cada imagem rasterizada em quatro folders drawable, teremos na verdade cinco versões de cada imagem.

São elas:

```
/res/drawable-mdpi;
/res/drawable-hdpi;
/res/drawable-xhdpi;
/res/drawable-xxhdpi;
/res/drawable-xxxhdpi.
```



Assim é certo, enquanto não temos o trabalho com vetores, que nosso aplicativo não sofrerá com o problema de pixalagem.

Como assim: "(/) não padronização das densidades de telas"



o Android está explícito que: Android e que a densidade de tela pode estar entre as classes de DPIs padrões definidos em sistema. A densidade de tela dele esteja entre **xhdpi** e **xxhdpi**, mas não exatamente em nenhuma das duas classes de telas, pois assim não há possibilidade de pixalagem e consequentemente quebra de design. e imagens rasterizadas: gente em qualquer densidade de pixels em tela. Mesmo as telas de densidades não padrões. Es gratutas deles) do Flaticon. e depois convertidos cada um nas cinco versões DPIs necessárias utilizando a ferramenta [Generic icon](#) qui o link para download dos ícones de sistema, pois eles permanecerão por tempo limitado em projeto. Es para download dos ícones vetoriais. o GitHub já será o projeto final, sem imagens rasterizadas.

Ícones de topo

Seguindo o protótipo estático, temos na barra de topo o ícone **ic_yosemite.png**, de 24dp x 24dp:



Logo abaixo do principal rótulo do aplicativo temos o ícone **ic_tourist.png**, de 34dp x 34dp:



Posteriormente neste artigo aula você vai entender o porquê do ícone acima ser o único que não é na cor cinza clara (#BBBBBB).

Ícones do menu "opções de itinerário"

Sim, são exatos 12 ícones em 5 versões cada um. Um total de 60 imagens rasterizadas 😊. Todos com densidade em 34dp x 34dp. O primeiro é o ícone Acampamento, **ic_camp.png**:



O segundo é o ícone Pescaria, **ic_fishing.png**:



O sexto é o ícone Rafting, **ic_rafting.png**:



O sétimo é o ícone Rádio, **ic_radio.png**:

O oitavo é o ícone Café, **ic_coffee.png**:

O nono é o ícone Telescópio, **ic_telescope.png**:





board.png:



Ícone de item selecionado

Ainda tem o ícone que aparece em tela para cada item que foi selecionado pelo usuário:



Rapel

Este é o ícone **ic_selected.png**, de 28dp x 28dp:



Classes de domínio

Teremos duas classes de domínio para representar respectivamente:

Um item de menu;

e O status do item de menu em projeto.
Uma dessas classes na verdade é um enum



[habilidades melhor divididas](#) e não há perda na leitura de código.
s do projeto) coloque o enum **MenuItemStatus** como a seguir:
main

nuItem:
main

```
var icon: Int,
var isSelected: MenuItemStatus = MenuItemStatus.NOT_SELECTED )
```

Pacote de dados

Como estamos trabalhando com apenas parte de um aplicativo real, então vamos utilizar dados simulados (*mock data*).
Nossa "[base de dados](#)" estará no pacote **/data** (crie este pacote na raiz de classes do projeto).

Essa persistência será a classe **Menu**, como a seguir:

```
class Menu {
    companion object{
        const val NUMBER_COLUMNS = 3
        fun getItems()
            = listOf(
                MenuItem(
                    label = "Acampamento",
                    icon = R.drawable.ic_camp
                ),
                MenuItem(
                    label = "Pescaria",
                    icon = R.drawable.ic_fishing
                ),
            )
    }
}
```



```
MenuItem(  
    label = "Fazer as malas"
```

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email

```
        icon = R.drawable.ic_radio  
,  
MenuItem(  
    label = "Café",  
    icon = R.drawable.ic_coffee  
,  
MenuItem(  
    label = "Telescópio",  
    icon = R.drawable.ic_telescope  
,  
MenuItem(  
    label = "Rapel",  
    icon = R.drawable.ic_rapel  
,  
MenuItem(  
    label = "Cart",  
    icon = R.drawable.ic_cart  
,  
MenuItem(  
    label = "Surf",  
    icon = R.drawable.ic_surfboard
```



pois facilita a leitura de todo o algoritmo.
tilizados neste projeto de exemplo é porque o código nativo do método está em Java, linguagem que não dá

pacote adapter

Neste projeto vamos trabalhar com um pacote adapter.

É prudente separar a classe adaptadora de itens da classe ViewHolder, isso quando o código começa a ficar grande.

E sim, enquanto o Jetpack Compose não é liberado ao menos em versão beta (depende da época em que você estiver consumindo este conteúdo) nós vamos manter o uso da antiga e conhecida Toolkit UI via XML.

Aliás, já vai se despedindo do [RecyclerView](#) e de outros populares [frameworks de lista](#).

Pois ao menos a API AdapterList já está pronta no Jetpack Compose e o *boilerplate code* (incluindo toda a configuração do ViewHolder) em pouco tempo não mais fará parte de nossa vida como profissional desenvolvedor de aplicativos Android.

É isso... vamos aos códigos.

Layout de item

O layout de item tem algumas **Views** aninhadas. Isso soa ruim quando temos a possibilidade de dezenas de itens em lista.

Mas aqui teremos somente 12, então não haverá riscos eminentes de um **OutOfMemoryException**.

Segue o layout XML /res/layout/menu_item.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="5dp"
    android:layout_marginRight="5dp"
    android:layout_marginBottom="25dp"
```



```
    android:orientation="vertical">
```



```
        android:id="@+id/iv_icon"
        android:layout_width="34dp"
        android:layout_height="34dp"
        android:layout_gravity="center"
        android:layout_marginTop="28dp"
        android:layout_marginBottom="28dp"
        android:tint="@color/colorDarkPurple" />
    </androidx.cardview.widget.CardView>
```

```
<View
    android:id="@+id/vv_gradient"
    style="@style/AppTheme.Shadow"
    android:layout_width="match_parent"
    android:layout_height="41dp"
    android:layout_gravity="top"
    android:elevation="13dp" />
```

```
<ImageView
    android:id="@+id/iv_selected"
    android:layout_width="28dp"
    android:layout_height="28dp"
```

```
    android:layout_gravity="top|end"
    android:contentDescription="@string/item_selected"
```



on é utilizado em:

```
<View
    android:id="@+id/vv_gradient"
    ...
    android:elevation="13dp" />
```

```
<ImageView
    android:id="@+id/iv_selected"
    ...
    android:elevation="14dp" />
```

Para que estas visualizações fiquem sobre o [CardView](#).

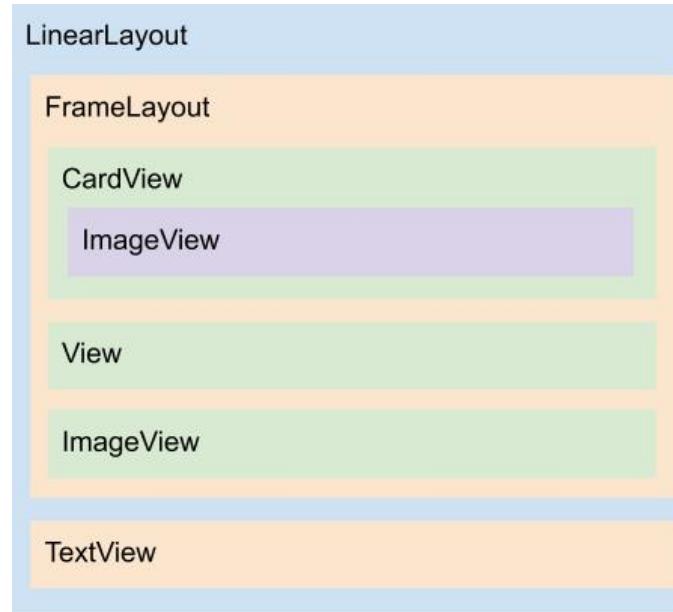
Caso contrário, a partir do Android 21, o uso do atributo **app:cardElevation** no **CardView** com qualquer valor maior do que zero faria com que a regra de empilhamento de **Views** do [FrameLayout](#) fosse ignorada e assim o **Cardview** ficaria em cima das outras visualizações...

... mesmo ele sendo a primeira visualização filha dentro do **ViewRoot, FrameLayout**.

Note que o uso de **app:cardElevation** foi necessário, pois é este atributo que coloca a sombra no **CardView** como definido em protótipo estático.

A seguir o diagrama do layout anterior:





Ainda em `/res/layout/menu_item.xml`, mais precisamente na `View` com função de eliminar parte da sombra do card. Nessa `View` temos o estilo `AppTheme.Shadow` sendo referenciado via atributo `style`. A seguir vamos recapitular este estilo:

```

...
<style name="AppTheme.Shadow">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">18dp</item>
    <item name="android:background">@drawable/white_gradient_fade</item>
</style>
...
    
```

Visualizando o tema acima, temos o background `white_gradient_fade.xml`. Background definido em `/res/drawable/white_gradient_fade.xml` com a seguinte estrutura:

```

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

<gradient
    android:type="linear"
    android:angle="270"
    android:startColor="#fff"
    
```

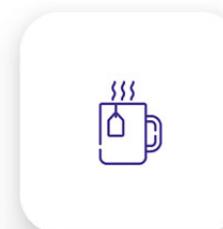
```
        android:endColor="@android:color/transparent" />
</shape>
```



nada de maneira estratégica em layout, conseguimos uma visualização suave do item:



Café



Café

A sombra na parte de cima do card ficaria mais intensa, fugindo do que foi definido em protótipo.

ViewHolder

Agora o código do ViewHolder, dentro do pacote **/adapter** (crie este pacote na raiz de classes do projeto).

Aqui o ViewHoder terá o rótulo **MenuViewHolder**:

```
package thiengo.com.br.yosemitesumercamp.adapter
```

```
import android.graphics.Color
import android.graphics.PorterDuff
import android.view.View
import android.widget.ImageView
import android.widget.TextView
import androidx.cardview.widget.CardView
import androidx.core.content.res.ResourcesCompat
import androidx.recyclerview.widget.RecyclerView
```

```
import thiengo.com.br.yosemitessumercamp.R
import thiengo.com.br.yosemitessumercamp.domain.MenuItem
n.MenuItemStatus
```

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email

```
>)->Unit,
temView ), View.OnClickListener {
```

```
cvIcon = itemView.findViewById( R.id.cv_icon )
ivSelected = itemView.findViewById( R.id.iv_selected )
vvGradient = itemView.findViewById( R.id.vv_gradient )
ivIcon = itemView.findViewById( R.id.iv_icon )
tvLabel = itemView.findViewById( R.id.tv_label )
}
```

```
fun setModel( item: MenuItem ) {
    tvLabel.text = item.label
    ivIcon.setImageResource( item.icon )
    ivIcon.contentDescription = item.label
    setStyle( item = item )
}
```

```
private fun setStyle( item: MenuItem ){
    /*
     * Até a última declaração de variável mutável (var)
     * tem toda a definição de estilo de um
     * "item não selecionado" (isSelected == false) que
     * é o valor inicial de cada item de itinerário em
```

```
* tela.
```

```
*/
```



```
_background_normal
```

```
or(
```

```
TED ){
```

```
1
```

```
_background_selected
```

```
ivSelectedVisibility = View.VISIBLE  
vvGradientVisibility = View.INVISIBLE  
ivIconColor = Color.WHITE  
tvLabelColor = R.color.colorDarkPurple
```

```
}
```

```
cvIcon.setBackgroundResource( cvBackgroundResource )  
ivSelected.visibility = ivSelectedVisibility  
vvGradient.visibility = vvGradientVisibility
```

```
/*  
 * A invocação setColorFilter( ivIconColor, PorterDuff.Mode.SRC_IN )  
 * informa que somente os pixels não transparentes da  
 * imagem (dentro do ImageView) é que devem receber a  
 * cor definida em ivIconColor.  
 *  
 * A constante PorterDuff.Mode.SRC_IN é responsável pela  
 * regra de negócio "somente os pixels não transparentes (...)"  
 */  
ivIcon.setColorFilter( ivIconColor, PorterDuff.Mode.SRC_IN )
```



```
tvLabel.setTextColor(  
    ResourcesCompat.getColor(  
        context,  
        R.color.  
        colorPrimaryDark  
    )
```

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
 a R\$ 2,00. Cadastre-se e receba
 gratuitamente conteúdos Android
 sem precedentes!

Seu melhor Email

Cadastrar email

car, no
adequados

emos abreviar
mos utilizar
ition]".

```
*/  
with( adapter.items[ adapterPosition ]){  
  
    this.isSelected = when( this.isSelected ){  
        MenuItemStatus.SELECTED -> MenuItemStatus.NOT_SELECTED  
        else -> MenuItemStatus.SELECTED  
    }  
}  
adapter.notifyItemChanged( adapterPosition )  
  
changeButtonStatusCallback( adapter.items )  
}  
}
```

Parece que o código é complexo, mas se você notar bem é muito código "b-aba". Só é grande.

Os arquivos de background do CardView

Ainda no código da classe **MenuViewHolder**.

Mais precisamente nos dois possíveis valores de background do **CardView** de item, valores de recurso que são colocados na variável **cvBackgroundResource** do método **setStyle()**...

... esses arquivos de background...

... primeiro o **/res/drawable/cv_background_normal.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
```



res/android"

`radius" />`

xml:

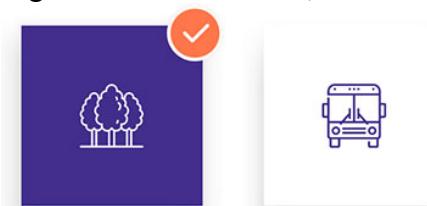
res/android"

`radius" />`

`/>`

`</shape>`

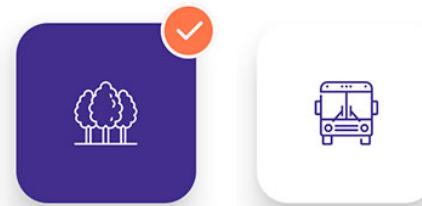
Se esses backgrounds não forem utilizados dessa forma, incluindo a definição de **android:radius** (curvatura das pontas da figura retangular)...
... se isso não ocorrer e somente houver a mudança da cor de background do CardView, então o resultado será:



Trilha

Transporte

Com a alternância dos backgrounds que têm também a definição do atributo radius, temos um resultado como definido em protótipo estático:



Trilha

Transporte

É isso mesmo:

Sem esta estratégia de background o valor do atributo **app:cardCornerRadius** do CardView é ignorado.



Use adaptadora **MenuAdapter** como a seguir:

pter

n.MenuItem

z>)->Unit)

.Recycler view.Adapter<ivmenu viewholder>U {

override fun **onCreateViewHolder**(

parent: ViewGroup,
viewType: Int) : MenuViewHolder {

val layout = LayoutInflater

.from(context)
.inflate(
R.layout.menu_item,
parent,
false

)

return MenuViewHolder(

adapter = **this**,
changeButtonStatusCallback = changeButtonStatusCallback,
itemView = layout

)

override fun **onBindViewHolder**(

```
holder: MenuViewHolder,  
position: Int ) {
```



Parte de nosso projeto Android de exemplo.

veja o código fonte /res/layout/content_itinerary.xml:

```
res/android"  
xmlns:app="http://schemas.android.com/apk/res-auto"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:orientation="vertical"  
app:layout_behavior="@string/appbar_scrolling_view_behavior">
```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginStart="20dp"  
    android:layout_marginTop="20dp"  
    android:layout_marginEnd="20dp"  
    android:layout_marginBottom="20dp"  
    android:text="@string/new_itinerary"  
    android:textColor="@color/colorDarkGrey"  
    android:textSize="34sp"  
    android:textStyle="bold" />
```

```
<TextView  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"
```

```
    android:layout_marginStart="20dp"
    android:layout_marginEnd="20dp"
```



```
    android:layout_height="match_parent"
    android:paddingStart="15dp"
    android:paddingLeft="15dp"
    android:paddingEnd="15dp"
    android:paddingRight="15dp" />
```

<!--

A View abaixo faz o papel de "fade" ao topo do FrameLayout, para que os itens em RecyclerView tenham a aparência de que estão "saindo ou entrando aos poucos" em tela e não de maneira abrupta.

-->

<View

```
    style="@style/AppTheme.Shadow"
    android:layout_gravity="top" />
```

<!--

A View abaixo faz o mesmo papel de "fade de lista" da View acima.



Porém devido aos atributos android:rotation="180"
e android:layout_gravity="bottom" o fade é no



```
    android:paddingBottom="16dp"
    android:text="@string/button_continue"
    android:textAllCaps="true"
    android:textColor="@android:color/white"
    android:textSize="13sp"
    android:textStyle="normal" />
</LinearLayout>
```

A seguir o diagrama do layout anterior:





Imagen embutida ao texto

Ainda no layout **content_itinerary.xml**, temos o uso de uma imagem embutida ao texto.

Tudo isso consumindo somente uma **View**, mais precisamente um **TextView**:

```
...
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="20dp"
    android:layout_marginEnd="20dp"
    android:layout_marginBottom="25dp"
    android:drawableStart="@drawable/ic_tourist"
    android:drawableLeft="@drawable/ic_tourist"
    android:drawablePadding="8dp"
    android:drawableTint="@color/colorMediumGrey"
    android:text="@string/activity_schedule"
```



```
    android:textColor="@color/colorMediumGrey" />
```



Escolha as atividades que você adicionará à sua agenda de acampamento de verão.

emente a partir do Android API 23, Marshmallow.

lo projeto ser a API 16; e devido também ao uso de uma imagem rasterizada em **android:drawableStart** e

ta (aqui, cinza médio, **#BBBBBB**).

xo da API 23, o design não será fiel ao que foi definido em protótipo estático.

e cor, nós somente mudaríamos o **android:tint** no arquivo de vetor.

s novamente o efeito de *fade*.

ão de [layout de item de menu](#).

ie os itens saiam ou entrem na tela de maneira abrupta.

ntém o **RecyclerView**:

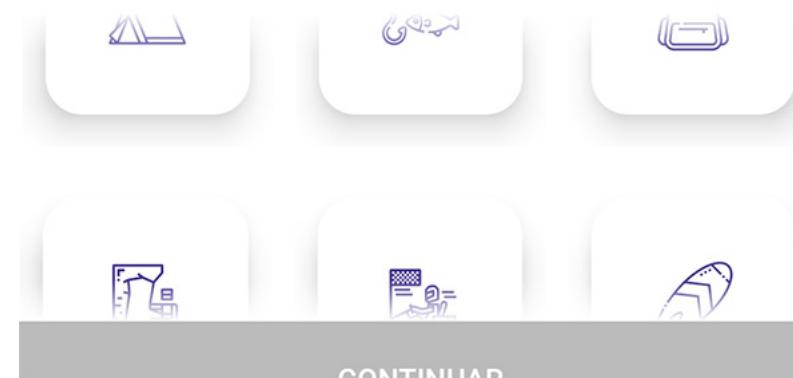
```
<FrameLayout
...
<androidx.recyclerview.widget.RecyclerView
...
...
<View
    style="@style/AppTheme.Shadow"
    android:layout_gravity="top" />
...
<View
    style="@style/AppTheme.Shadow"
    android:layout_gravity="bottom"
    android:rotation="180" />
</FrameLayout>
...
```

Sem estas **Views** o layout ficaria:

→ Topo de lista:



→ Fundo de lista:



Layout principal

Agora o layout principal, mais precisamente o layout /res/layout/activity_itinerary.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
    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="@android:color/white"
```

```
tools:context=".ItineraryActivity">>
```



topo

0.

Layout

"erlay"

"eme.ToolbarTheme"
on que está

A definição app:navigationIcon="@drawable/ic_yosemite"
coloca em barra de topo, mais precisamente no lado
esquerdo dela, o ícone personalizado do aplicativo.

Exatamente como definido no protótipo estático.

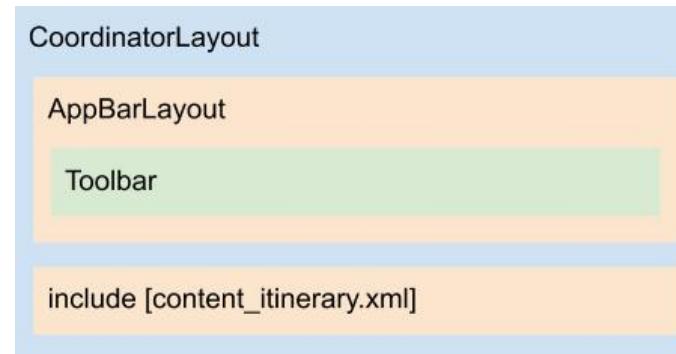
```
-->
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:theme="@style/AppTheme.ToolbarTheme"
    app:navigationIcon="@drawable/ic_yosemite"
    app:popupTheme="@style/AppTheme.PopupOverlay" />
</com.google.android.material.appbar.AppBarLayout>

<include layout="@layout/content_itinerary" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```



A seguir o diagrama do layout principal:



ItineraryActivity:

```

import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.res.ResourcesCompat
import androidx.recyclerview.widget.GridLayoutManager
import kotlinx.android.synthetic.main.activity_itinerary.*
import kotlinx.android.synthetic.main.content_itinerary.*
import thiengo.com.br.yosemitessummercamp.adapter.MenuAdapter
import thiengo.com.br.yosemitessummercamp.data.Menu
import thiengo.com.br.yosemitessummercamp.domain.MenuItem
import thiengo.com.br.yosemitessummercamp.domain.MenuItemStatus

class ItineraryActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_itinerary)
        setSupportActionBar(toolbar)

        /*
         * HackCode para colocar um ícone no canto superior
         * esquerdo da tela (lado esquerdo da Toolbar).
         */
        supportActionBar?.setDisplayHomeAsUpEnabled(true);
    }
}
  
```

```
supportActionBar?.setDisplayHomeAsUpEnabled( true );
```



```
        items -> changeButtonStatus( items )
    }
}

/*
 * Método responsável por atualizar o status de clique e o
 * background do botão "CONTINUE".
 *
 * Se houver ao menos um item selecionado, então o botão
 * fica como "disponível para clique", com o background
 * laranja (bt_orange_ripple) aplicado a ele.
 */
private fun changeButtonStatus( items: List<MenuItem> ){

    var isEnabled = false
    var backgroundId = R.color.colorMediumGrey

    /*
     * O método any() precisa encontrar somente um item que
     * retorne true para o predíicato
    
```

* "it.isSelected == MenuItemStatus.SELECTED" que assim
 * ele para com a execução e retorna true para a variável



D

não é possível
erada caso

```
backgroundId = if( Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP ){
    R.drawable.bt_orange_ripple
}
else{
    R.color.colorAccent
}
bt_continue.isEnabled = isEnabled
bt_continue.background = ResourcesCompat.getDrawable(
    resources,
    backgroundId,
    null
)
}
```



E para simplificar...

... segue o código do arquivo **/res/drawable/bt_orange_ripple.xml** que é utilizado como possível background de botão no método **changeButtonStatus()**:

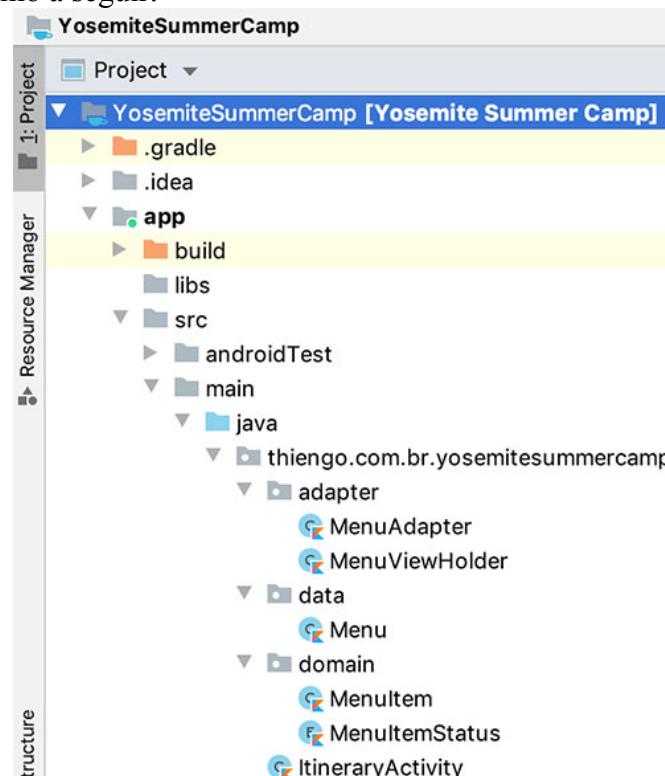
```
<?xml version="1.0" encoding="utf-8"?>
```



res/android"

/>

:como a seguir:



structure

Com isso podemos citar, novamente, alguns problemas nesta primeira versão de app Android.

Arquivo executável e manutenção

Rodando o "Optimize Imports". Logo depois acionando o "Clean Project" e por fim executando o projeto.

Temos um executável com o tamanho de **3.126.712 bytes** (3.1 MB).

Ou seja, ao menos para essa única tela de projeto não tem problema algum em manter todos os ícones como imagens rasterizadas.

Ok! Mas e a manutenção do aplicativo?



"pções" em menu itinerário? Deixando eles maiores, pois teremos duas colunas e não mais três colunas. Imagens 😊. Isso levando em consideração somente uma tela de todo o aplicativo e somente as imagens

os uma melhora considerável quando utilizando, para ícones de sistema, drawables vetoriais ao invés de

ícone no mesmo local que as imagens rasterizadas. No Flaticon, existem opções rasterizadas.

SVG em projeto utilizando o Vector Asset Studio. No caso importar cada imagem com a configuração correta. O projeto dá suporte a partir da API 16 do Android e que com a não utilização da configuração

es rasterizadas sendo criadas com base no que está definido em cada um dos XMLs estruturais deles. Download de cada vetor (já convertido) que será utilizado em projeto.

Coloque todos os vetores XML baixados no folder **/res/drawable**:

[/res/drawable/ic_yosemite](#) para o ícone de barra de topo;

[/res/drawable/ic_tourist](#) para o ícone de subtítulo do app;

[/res/drawable/ic_camp](#) para a opção Acampamento;

[/res/drawable/ic_fishing](#) para a opção Pescaria;

[/res/drawable/ic_packing](#) para a opção Fazer as malas;

[/res/drawable/ic_forest](#) para a opção Trilha;

[/res/drawable/ic_transport](#) para a opção Transporte;

[/res/drawable/ic_rafting](#) para a opção Rafting;

[/res/drawable/ic_radio](#) para a opção Rádio;

[/res/drawable/ic_coffee](#) para a opção Café;

[/res/drawable/ic_telescope](#) para a opção Telescópio;

[/res/drawable/ic_rapel](#) para a opção Rapel;

[/res/drawable/ic_cart](#) para a opção Cart;

[/res/drawable/ic_surfboard](#) para a opção Surf;

[/res/drawable/ic_selected](#) para o ícone que representa uma opção de itinerário selecionada.

Com todos os vetores em projeto, você seguramente pode remover por completo os folders a seguir. Pois neste aplicativo somente tem nesses folders os ícones de sistema:

[/res/drawable-mdpi](#);

[/res/drawable-hdpi](#);

[/res/drawable-xhdpi](#);

/res/drawable-xxhdpi;
/res/drawable-xxxhdpi



etores utilizando as APIs de **Drawable** e de recursos (**Int**), não precisamos mudar absolutamente nada em

:t";
lique em "Rebuild Project".
relho real de testes:

14:21 ☰ 📲 📺 ⏱



Criar um novo itinerário



Escolha as atividades que você adicionará à sua agenda de acampamento de verão.



Acampamento



Pescaria



Fazer as malas



Trilha



Transporte



Rafting



CONTINUAR



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

Ao menos a sua renderização será "saudável"



Ter concluído o conteúdo. Isso diz muito sobre o seu comprometimento como profissional desenvolvedor ▼

uer artigo de desenvolvimento.

io Android, deixe logo abaixo nos comentários.

·mails para também garantir a versão em PDF de cada novo "artigo mini-curso".

[Documentação oficial Kotlin - Enums](#)

[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](#)

O executável final tem **3.117.254 bytes** (3.1 MB) 🎉 😊.

É isso mesmo, muito próximo da quando utilizando somente imagens rasterizadas.

, volte a seção [O problema do vectorDrawables.useSupportLibrary = true.](#)

versões Android necessária a um engenheiro desenvolvedor mobile.

Blog para receber em primeira mão conteúdos Android exclusivos e também em suas versões PDF (liberadas

[Thiengo](#).

arte do artigo, pode ser acessado no seguinte GitHub:

[.camp.](#)

oid...

ima, sendo atualizado para uso de drawables vetoriais ao invés de drawables rasterizados:

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

Conteúdo Exclusivo

Investir em Você é Barra de Ouro a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

Seu melhor Email

Cadastrar email



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.



Canvas (GUI)

Vector graphics

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email

7a (Novatec - 2012)

Cadastre-se e receba gráts conteúdos Android sem precedentes!

Cadastrar email

Relacionado



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



[Como Impulsionar o App Android - Compartilhamento Nativo](#)



[Annotation Span Para Estilização de Texto no Android](#)





Clube dos Geeks

Conteúdo Exclusivo

Investir em Você é Barra de Ouro
a R\$ 2,00. Cadastre-se e receba
gratuitamente conteúdos Android
sem precedentes!

Seu melhor Email

Cadastrar email



Registre-se e se mantenha atualizado:

Email

Registrar

© 2013 - 2020 Thiengo [Calopsita] - Todos os direitos reservados
Desenvolvido por Thiengo [Calopsita]





Protótipo do ProjetoAndroid

[iwable](#) - [imagem vetorial](#) - [imagem rasterizada](#) - [svg](#) - [psd](#) - [apk leve](#) - [fácil manutenção](#)

Conteúdo Exclusivo

Curtir 245

Compartilhar

... em Você é Barra de Ouro

a R\$ 2,00. Cadastre-se e receba gratuitamente conteúdos Android sem precedentes!

0 comentários

Seu melhor Email

Classificar por [Mais antigos](#)

Adicione um comentário...

Plugin de comentários do Facebook

Comentários Blog

Para código / script, coloque entre [code] e [/code] para receber marcação específica.

*Nome

*Email (não será publicado)

Web Site

Comentário

Enviar



[Mario Verde Games](#)



[Central do Inglês](#)

