

Trabalhando com Mapas

Introdução

Atualmente a API de mapas da google está na versão 2, a grande diferença para sua versão anterior é que ela foi totalmente reescrita para utilizar vetores para suportar visualizações 2D e 3D que são ativados automaticamente dependendo do zoom e inclinação no mapa, assunto que vamos ver ao longo dessa aula, também houve um ganho em desempenho tornando suas interações e animações mais fluidas.

Google Play Services

Antes de começarmos a trabalhar com esse recurso do Android precisamos saber que o Mapas, assim como outros serviços da google, como Drive, Localização, Google Plus entre outros, são distribuídos pelo Google Play Services, que por sua vez tem a finalidade de conectar sua aplicação a um serviço da Google.

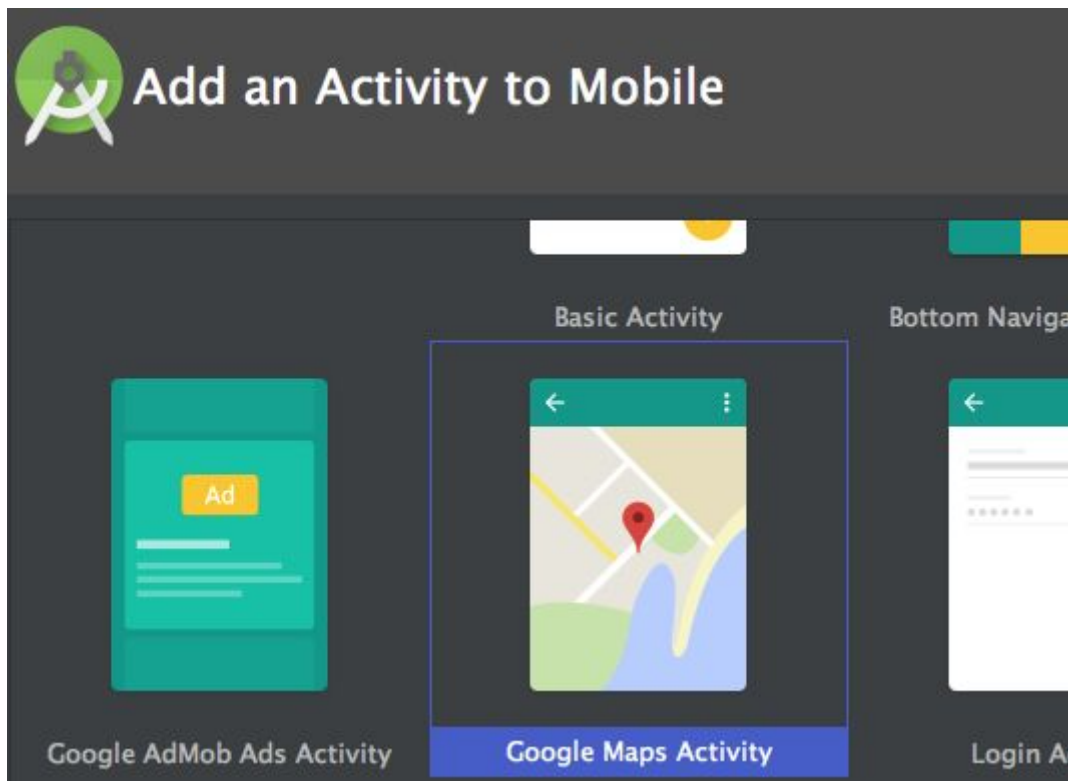
Então precisamos configurar corretamente o Google Play Services em nossa aplicação, para isso, adicione no arquivo app/build.gradle a dependência.

```
compile 'com.google.android.gms:play-services-maps:10.2.1'
```

*Dependência Google Play Services para utilizar o mapas.
Atualmente na versão 10.2.1.*

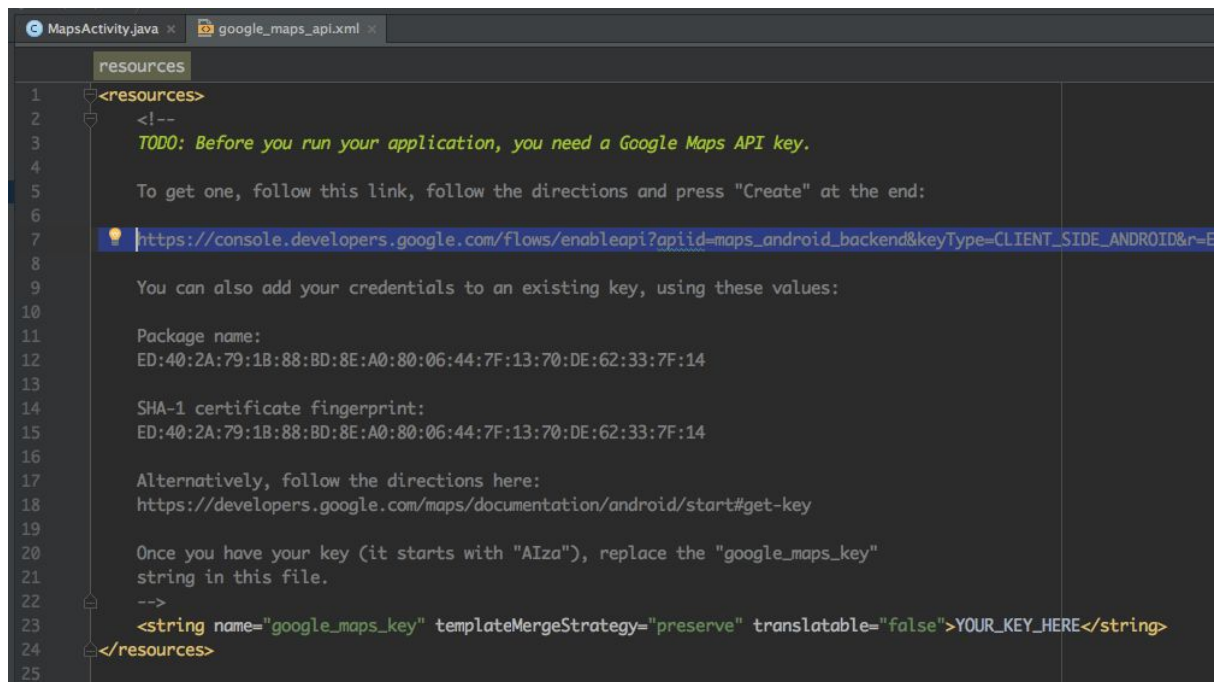
Criando o projeto

Começar um projeto para trabalhar com Mapas hoje é um processo bem simples, pois a própria IDE nos permite iniciar um projeto utilizando o Mapa, então crie um novo projeto com o nome *Trabalhando com Mapas*, escolha a versão de API 15, em seguida escolha a opção “Google Maps Activity” para começar um novo projeto com os recursos necessários para trabalhar com Mapas já pré-configurados, depois é só continuar pressionando “próximo” até criar o projeto.



Escolhendo a opção Google Maps Activity.

Com o projeto criado você vai ver uma tela igual a imagem abaixo, o arquivo “[google_maps_api.xml](#)” é criado automaticamente pelo Android Studio, note que existe um item criado com o nome “[google_maps_key](#)”, vamos ver mais sobre esse arquivo a seguir, basicamente precisamos criar uma chave de acesso para o seu projeto utilizar a API de Mapas do Google Play Services.



```
1 <resources>
2 <!--
3  TODO: Before you run your application, you need a Google Maps API key.
4
5  To get one, follow this link, follow the directions and press "Create" at the end:
6
7  https://console.developers.google.com/flows/enableapi?apiid=maps\_android\_backend&keyType=CLIENT\_SIDE\_ANDROID&r=E
8
9  You can also add your credentials to an existing key, using these values:
10
11  Package name:
12  ED:40:2A:79:1B:88:BD:8E:A0:80:06:44:7F:13:70:DE:62:33:7F:14
13
14  SHA-1 certificate fingerprint:
15  ED:40:2A:79:1B:88:BD:8E:A0:80:06:44:7F:13:70:DE:62:33:7F:14
16
17  Alternatively, follow the directions here:
18  https://developers.google.com/maps/documentation/android/start#get-key
19
20  Once you have your key (it starts with "AIza"), replace the "google_maps_key"
21  string in this file.
22  -->
23  <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">YOUR_KEY_HERE</string>
24 </resources>
25
```

Arquivo google_maps_api.xml criado para configurar uma chave de acesso para sua aplicação utilizar a API de Mapas.

Permissão

Para gerar uma chave de acesso para nosso aplicativo, precisamos acessar o link da linha 7 do arquivo "google_maps_api.xml", como podemos ver nos comentários da imagem anterior.

Esse é um link para o painel de gerenciamento de API, ao acessar pela primeira vez esse link, você deve estar vendo uma tela como a imagem abaixo.

Registre seu aplicativo no serviço Google Maps Android API no Console de API do Google

Com Console de API do Google, você gerencia seu aplicativo e monitora o uso de API.

Selecione um projeto em que seu pedido será registrado

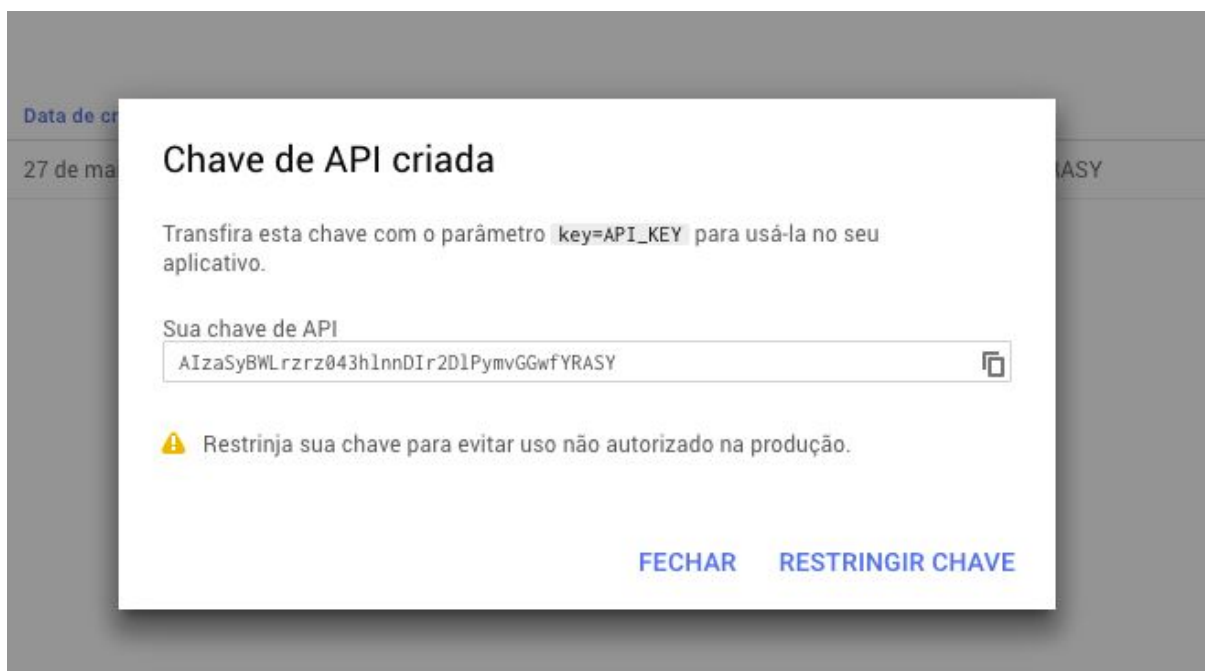
Você pode usar um projeto para gerenciar todos os seus aplicativos ou pode criar um projeto diferente para cada aplicativo.

Criar um projeto

Continuar

Deixe marcado a opção para “criar um novo projeto” e clique no botão “Continuar”, na próxima página clique em “Criar Chave de API” para que sua credencial seja criada.

Depois de finalizar a ativação da API vai ser gerado uma chave de acesso como na imagem abaixo, copie essa chave pois vamos utilizar em nosso projeto, recomendo que clique em “Restringir Chave” para configurar algumas informações para sua Chave de API.



Chave de API criada.

Clicando em “Restringir Chave” você vai ser redirecionado para uma página para configurar sua chave, nessa página você pode colocar um nome para identificar essa Chave, pois podem existir várias chaves de API para um usuário, além de definir qual tipo de conexão pode utilizar nossa Chave, lembre de deixar marcado a opção “Apps para Android”.

Nome

Trabalhando com Mapas API

Restrição de chave

Restringir uma chave é especificar quais websites, endereços IP ou aplicativos podem utilizá-la. [Saiba mais](#)

☐ Nenhum
☐ Referenciadores de HTTP (sites da Web)
☐ Endereços IP (servidores da Web, cron jobs etc.)
☒ Apps para Android
☐ Apps para iOS

Restringir o uso de seus apps para Android (Opcional)

Adicione o nome do seu pacote e a impressão digital do certificado de assinatura SHA-1 para restringir o uso dos seus aplicativos para Android

Saiba mais Receba o nome do pacote do arquivo AndroidManifest.xml. Depois, use o comando a seguir para conseguir a impressão digital:

```
$ keytool -list -v -keystore mystore.keystore
```

Nome do pacote	Impressão digital para certificação SHA-1
com.faus.apps.trabalhandocommap	ED:40:2A:79:1B:88:BD:8E:A0:80:06:44:7F:13:70:DE:62:33:7F:14

+ Adicionar nome do pacote e impressão digital

Nota: pode levar até cinco minutos para que as configurações sejam aplicadas

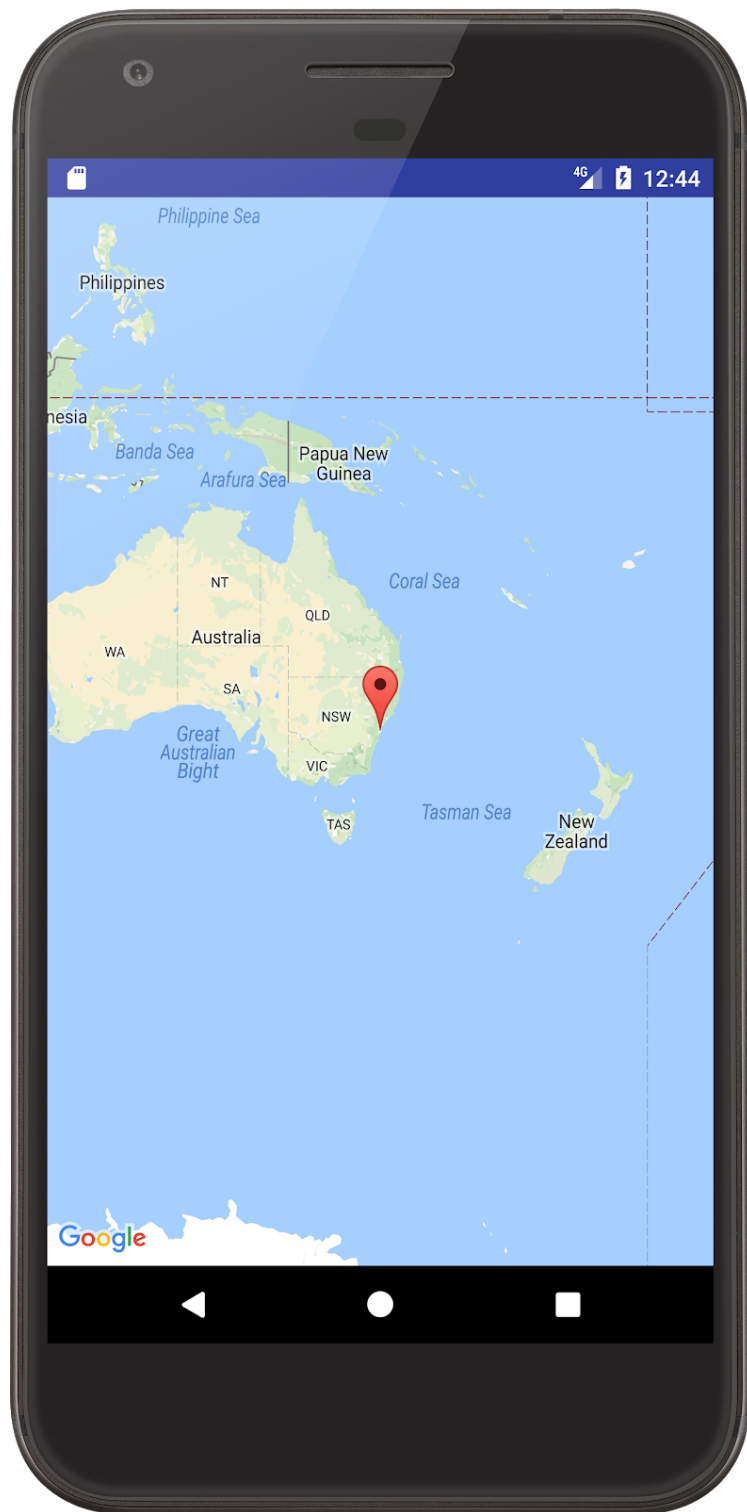
Salvar

Cancelar

Página para configurar informações adicionais sobre sua Chave de API.

Por último devemos voltar ao nosso Android Studio e colar nossa chave de acesso no item `google_maps_key`, assim nosso projeto estará configurado corretamente para rodar o mapa sem nenhum problema, rode seu app para confirmar essa informação.

```
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AlzaSyBWLrzz043hInnDir2DIPymvGGwfYRASy</string>
```



App rodando o Google Mapas.

Entendendo o Projeto

Caso não tenha tido nenhum problema até agora, deverá estar vendo em seu app um mapa com um marcador sobre “Sydney”, esse não é o comportamento padrão do Mapas, em algum momento em seu código foi criado essa estrutura de código pelo Android Studio, precisamos então entender tudo que foi criado pela IDE para darmos continuidade em nosso aplicativo.

Abra o arquivo **MapsActivity.java**, devemos observar três pontos interessantes nos códigos dessa classe, primeiro um objeto criado como variável de classe chamado do tipo **GoogleMap**, um **gerenciador de fragments** e por último um método chamado **onMapReady**, agora vamos falar um pouco sobre cada um deles.

Começando pela classe **GoogleMap**, ela representa o mapa do Google e, por meio dela, podemos controlar visualização do Mapa, nível de zoom, aparência, localização, adicionar marcadores entre outros, além disso ela tem um papel muito importante, pois ela encapsula toda a complexidade de acessar o mapa do Google, tornando o trabalho do desenvolvedor mais simples.

Sobre o **gerenciador de fragments**, podemos notar que estamos buscando um fragment com o id `R.id.map`, *não falamos sobre esse fragment ainda, o que precisamos saber é que ele foi criado dentro do layout `activity_maps.xml`*, logo após estamos chamando o método **getMapASync**, sua função é se conectar com os serviços da Google, sua execução é assíncrona e o resultado desse método vai ser entregue ao método **onMapReady**.

Depois de conectar com o serviço de Mapa da Google o método **onMapReady** é invocado com uma instância de um objeto **GoogleMap** configurado, assim podemos começar a brincar em nosso mapa, podemos notar alguns códigos dentro desse método, como vamos aprender mais sobre esses ao decorrer do curso, recomendo que apague todo esses códigos e cole o código abaixo, pois vamos começar a aprender sobre eles a seguir.

```

mMap = googleMap;

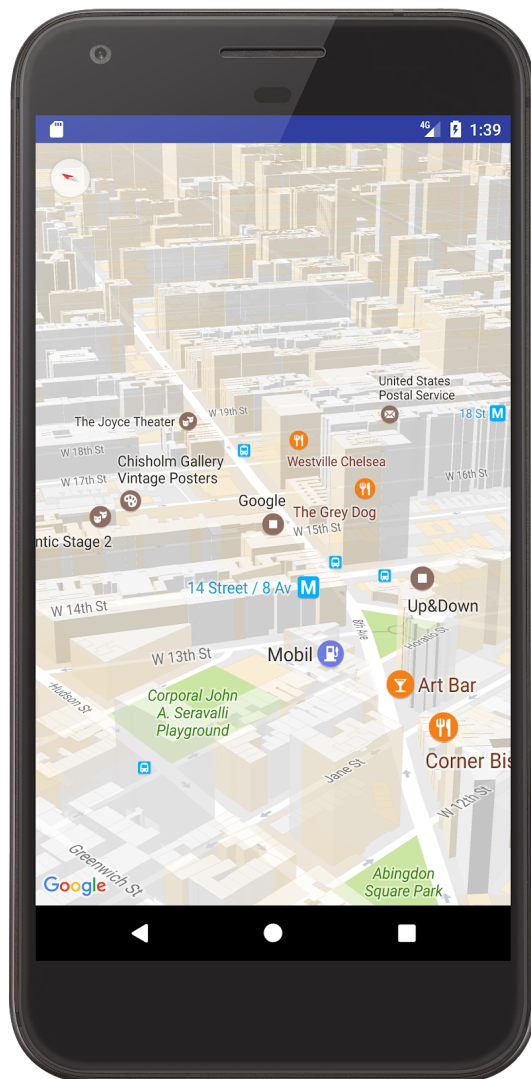
// Localização sede da Google
LatLng google = new LatLng(40.740637, -74.002039);

// Configuração da câmera
final CameraPosition position = new CameraPosition.Builder()
    .target( google )    // Localização
    .bearing( 45 )       // Rotação da câmera
    .tilt( 90 )          // Ângulo em graus
    .zoom( 17 )          // Zoom
    .build();

CameraUpdate update = CameraUpdateFactory.newCameraPosition( position );

mMap.animateCamera( update );

```



Visualização do Mapa com os códigos acima.

Câmera

Depois executar o seu app com os códigos acima, podemos perceber uma diferença muito grande no mapa, todos os códigos vão ser comentados durante essa seção, antes precisamos entender a classe que torna tudo isso possível.

A classe câmera é responsável por apresentar o mapa para o usuário, com ela podemos manipular a exibição do mapa, desde proximidade, inclusão, rotação entre recursos.

- Zoom

- A propriedade zoom é responsável por aproximar o mapa visualização do mapa até o ponto máximo, algumas localidades que já suportam o mapa em seu modo 3D.
- Uma curiosidade sobre esse propriedade é que o mapa pode ser apresentado em 3D depois do valor 17 para o zoom.
- Podemos declarar o zoom para a classe Câmera de diversas maneiras, no exemplo do código acima, estamos usando o método `zoom()`, durante o desenvolvimento dos aplicativos vamos ver outras implementações dessa propriedade no mapa.

- Position

- Uma localização no mapa é definida por dois atributos, latitude e longitude.
- Na api de Mapas temos uma classe para representar uma localização a partir desses atributos, podemos chamar a classe **LatLng** e passar em seu construtor os valores de latitude e longitude, assim temos métodos para recuperar os valores individualmente, além de se implementadas por outras classes da API de Mapas.
- Podemos recuperar um determinado endereço a partir de uma latitude e longitude, como também é possível fazer o processo inverso, durante o desenvolvimento do aplicativo vamos ver essa funcionalidade.
- No exemplo do código estamos criando um objeto **LatLng** para representar a localização da sede da Google.

- Animate

- Podemos mostrar uma localização para o usuário com os métodos da classe GoogleMap, esse processo pode ser feito de duas formas, a mais bruta que o Câmera dá um salto para a próxima localização, sem nenhuma animação, essa opção talvez não seja tão interessante para o usuário.
- Podemos então especificar para que esse processo seja animado, com o método **animateCamera**.

- Tipo do Mapa

- Outra propriedade muito interessante é a possibilidade de alterar o estilo do mapa, temos cinco tipos diferentes de estilo que podem ser alterados pelo método ***setMapType*** da classe ***GoogleMap*** usando as seguintes constantes.

- **GoogleMap.MAP_TYPE_NONE**

- Modo de visualização mais simples do mapa, de forma que nenhuma informação extra é exibida.

- **GoogleMap.MAP_TYPE_NORMAL**

- Modo de visualização **padrão** dos mapas, no qual podemos visualizar as ruas, estradas e rios.

- **GoogleMap.MAP_TYPE_SATELLITE**

- Modo de visualização com os dados do satélite.

- **GoogleMap.MAP_TYPE_HYBRID**

- Modo de visualização com os dados fotográficos do satélite, com os mapas das ruas. É o modo de satélite mais detalhado.

- **GoogleMap.MAP_TYPE_TERRAIN**

- Modo de visualização que exibe os dados topográficos do mapa.

- Rotação

- Um dos parâmetros mais interessantes da classe ***Câmera*** é a possibilidade de definir uma rotação para um determinado ponto.
- Podemos notar que a apresentação do mapa depois de modificar os códigos é diferentes da exibição inicial, isso porque estamos rotacionando o mapa 45 graus à direita.
- Durante o desenvolvimento do aplicativo vamos ver outras maneiras de utilizar essa propriedade .

- Inclinação

- Por último o método ***tilt()*** nos aplica uma inclinação no mapa, dando um efeito de profundidade, em nosso exemplo estamos usando uma profundidade de 90 graus.
- Essa propriedade é bastante utilizada quando estamos mais próximo ao mapa e queremos ter mais informações sobre as localidades próximas.

Marcadores

Os marcadores permitem adicionar figura ao mapa, para marcar os lugares de interesse, a fim de exibir informações importantes e interagir com o usuário. o marcador é representado pela classe **marker** e adicionado ao mapa pelo método **addMarker** da classe **GoogleMap**.

Para criar um **marker** precisamos usar uma classe de configuração chamada **MarkerOptions**, onde podemos configurar diversas propriedades, como cor, localização, título, descrição, entre outros, abaixo um exemplo de como criar um marker.

```
// Criando um objeto do tipo MarkerOptions
final MarkerOptions markerOptions = new MarkerOptions();

// Configurando as propriedades do marker
markerOptions.position( google ) // Localização
    .title("Google Inc.") // Título
    .snippet("Sede da Google"); // Descrição
```

Podemos ainda definir um ícone personalizado, estilizar o balão que exibe o título e a descrição, conhecido por **Info Window** entre outras propriedades, fique tranquilo pois vamos aprender mais sobre esse assunto durante as próximas seções.

Agora precisamos adicionar o marker ao Map, para isso use o método **addMarker** passando o **markerOptions** como parâmetro.

```
// Adicionando marcador ao mapa
mMap.addMarker( markerOptions );
```



*Marker adicionado ao mapa,
com a **info window** ativada.*

Polyline

A classe **Polyline** nos permite desenhar no mapa uma linha ligando dois pontos, duas localizações, é muito utilizada para apresentar uma rota para um usuário, neste exemplo vamos ver sua implementação em nosso aplicativo.

Por enquanto temos apenas uma localização no nosso mapa, então precisamos de outra localização para criar nossa linha, porém vamos começar brincar um pouco com nosso mapa.

Vamos criar uma linha entre a sede da Google e um ponto específico do mapa que clicarmos, mais como fazer isso se ainda não vimos sobre, fique tranquilo, pois esse assunto será abordado nas próximas seções, nesse momento você deve saber apenas que existe eventos que podemos configurar em nosso Mapa, o evento que vamos trabalhar para desenvolver essa funcionalidade em nosso aplicativo é o `setOnMapClickListener`, como no exemplo abaixo, assim toda vez que o mapa for pressionado o método `onMapClick` será invocado e a localização do local será entregue como parâmetro.

```
mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng latLng) {
        Double latitude = latLng.latitude;
        Double longitude = latLng.longitude;

        Toast.makeText(MapsActivity.this, "Latitude: " + latitude + ", Longitude: "+ longitude,
        Toast.LENGTH_SHORT).show();
    }
});
```

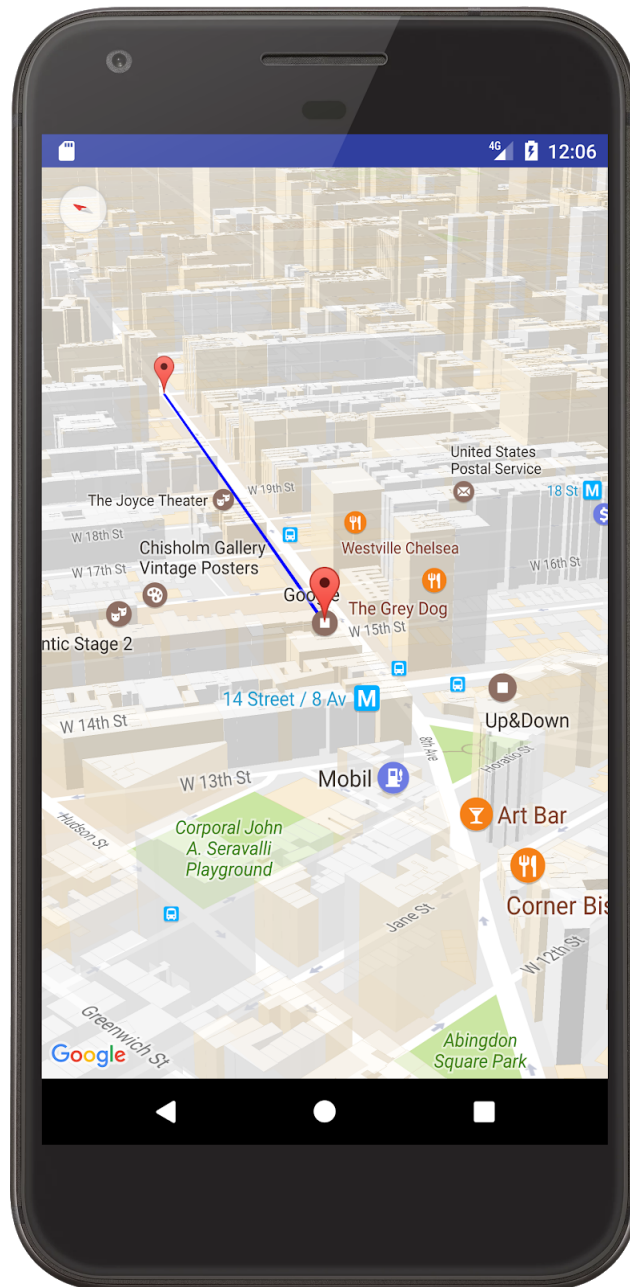
Com isso podemos adicionar um marker nessa localização e em seguida criar nossa linha, pois vamos ter a duas localizações.

```
mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng latLng) {
        // Criar marker no marker
        MarkerOptions options = new MarkerOptions();
        options.position( latLng );

        mMap.addMarker( options );

        // Configurando as propriedades da Linha
        PolylineOptions polylineOptions = new PolylineOptions();
        polylineOptions.add( google );
        polylineOptions.add( latLng );
        polylineOptions.color( Color.BLUE );
    }
});
```

```
}  
});  
  
// Adiciona a linha no mapa  
mMap.addPolyline( polylineOptions );
```



Linha entre duas localizações com a classe Polyline.

Conclusão

Neste artigo aprendemos como começar um aplicativo Android para trabalhar com mapas, desde sua configuração até seus principais recursos. Na próxima seção vamos aprender como trabalhar com a API de Localização em cima de tudo que aprendemos nessa aula, assim vamos conseguir pegar a localização atual do usuário, calcular distâncias entre localizações, traçar rotas, entre outros recursos dessa incrível API, espero que tenha gostado de mais uma aula, até a próxima.