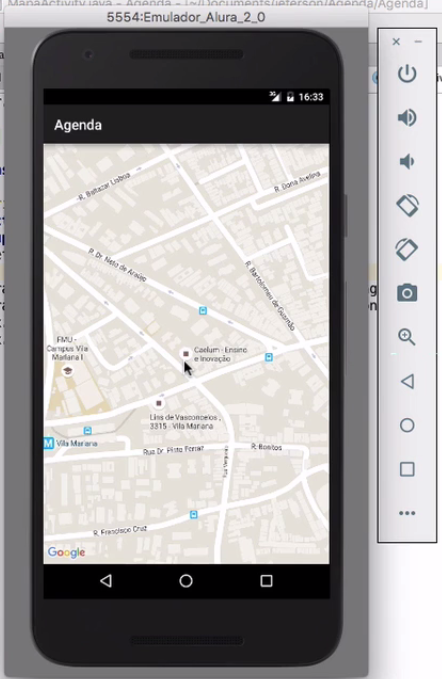
Concluindo as Funcionalidades do Mapa

Para concluirmos as funcionalidades de mapa, falta vermos como trabalhar com GPS no Android.

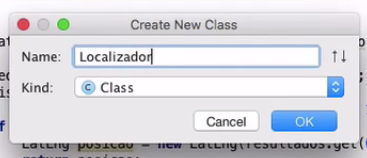
Atualmente, quando clicamos na funcionalidade de mapa, vemos a localização centralizada na Caelum.



Agora, gostaria que o mapa fosse aberto indicando a posição em que o usuário está naquele momento. Por exemplo, se estou na Avenida Paulista, quero que o mapa seja centralizado nesta localização e que fosse sendo reposicionado enquanto eu caminhasse.

Como está e uma funcionalidade da parte de mapa, nós iremos fazer alterações no MapaFragment.java. Nele que manipulamos os marcadores e especificamos a localização de cada um dos alunos.

Queremos que quando o mapa esteja pronto, seja ativada a funcionalidade de GPS e vinculá-los. E todas as vezes em que houver uma mudança de posição, quero que atualize o centro do meu mapa. Para isto, criaremos uma classe que ficará responsável por gerenciar a parte do GPS, que chamaremos de Localizador.



Teremos agora o arquivo Localizador.java, iremos instanciar a nova classe e fazer o set up necessário para conseguirmos consultar o serviço de localização do nosso celular. Vamos começar criando um construtor.

public class Localizador {  
  
 public Localizador() {  
 GoogleApiClient client  
 }  
 }

Para fazer uma conexão com qualquer serviço dentro no nosso celular, usamos a classe GoogleApiClient. Após importá-la, precisaremos de uma instância, usaremos o Builder. Para poder passar como parâmetro um Contexto, teremos que usá-lo também no construtor.

public class Localizador {  
  
 public Localizador(Context context) {  
 GoogleApiClient client = new GoogleApiClient.Builder(context)  
 }  
}

Recebemos o context para instancair o Builder. Dentro, temos a opção de chamar vários métodos. No nosso caso, primeiramente, usaremos o método addApi() para configurar a Api, e acessaremos o LocationsServices.

.AddApi(LocationServices.API)

Em seguida, faremos um build().

.build();

E faremos um connect com o client.

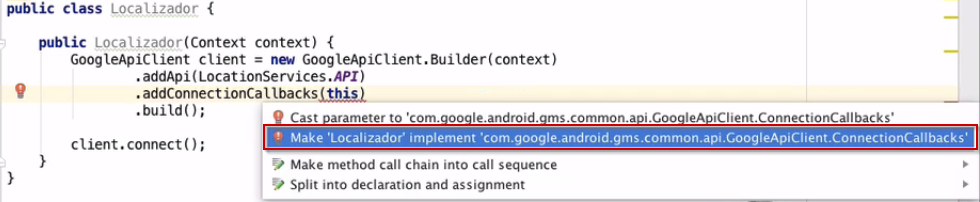
client.connect();

Porém, o connect() não é um método síncrono - a aplicação trava e simplesmente não deixa o usuário efetuar mais nenhuma operação. Ele é **assíncrono** e permite que ocorra a troca de dados das aplicações sem que elas fiquem aguardando o retorno da requisição para dar prosseguimento a suas atividades.

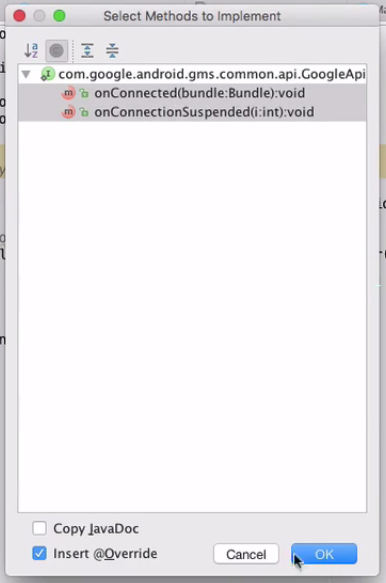
Após dispararmos o connect() e ele conseguir sincronizar com algum serviço, ele precisará avisar para alguém e solicitar o próximo passo. Ou seja, temos um período de espera. Teremos que especificar no Builder quem será avisado quando tentarmos fazer um conexão. Faremos isto incluindo o addConnectionCallbacks(this) - ou seja, usaremos a própria classe.

public class Localizador {  
  
 public Localizador(Context context) {  
 GoogleApiClient client = new GoogleApiClient.Builder(context)  
 .AddApi(LocationServices.API)  
 .addConnectionCallbacks(this)  
 .build();  
  
 client.connect();   
 }  
}

Como precisamos que a classe implemente uma interface específica, selecionaremos this e depois, clicaremos em "Enter". Escolheremos a opção para implementar o addConnectionCallbacks.



Depois, o Android dirá que precisaremos implementar dois métodos da interface.



Com os métodos implementados, o Localizador.java ficará assim:

public class Localizador implements GoogleApiClient.ConnectionCallbacks {  
  
 public Localizador(Context context) {  
 GoogleApiClient client = new GoogleApiClient.Builder(context)  
 .AddApi(LocationServices.API)  
 .addConnectionCallbacks(this)  
 .build();  
  
 client.connect();   
 }  
  
 @Override  
 public void onConnected(@Nullable Bundle bundle) {  
 }  
  
 @Overridepublic void onConnectionSuspended(int i) {  
 }  
}

Agora, o Localizador implementa ConnectionCallbacks, que será a classe que iremos chamar quando conseguirmos fazer a conexão, e depois, especificamos o que iríamos fazer ao conectar (onConnected) ou quando a conexão for suspensa ( onConnectionSuspended).

Quando conectarmos, a aplicação conseguirá se comunicar com o GPS do celular e pedir as atualizações de posição. Então, iremos construir dentro do onConnected(), um objeto que descreva quais são os tipos de dados que queremos receber do GPS do celular. Chamaremos o objeto de request e iremos instanciar o LocationRequest().

@Override  
public void onConnected(@Nullable Bundle bundle) {  
 LocationRequest request = new LocationsRequest();  
  
}

Vamos fazer a seguinte pergunta: é interessante receber informações do celular se estamos parados em um local? Não seria muito útil. Mas quando nos movemos, é interessante receber as atualizações do GPS.

Para informarmos que só queremos receber informações enquanto estivermos nos deslocando, usaremos um parâmetro no request que será o setSmallestDisplacement(). Com ele, especificaremos o deslocamento mínimo para que as posições do GPS precisem ser informadas. No caso, usaremos como parâmetro o valor de 50. Se o usuário se deslocar em um raio de 50 metros, o GPS não precisará enviar informações.

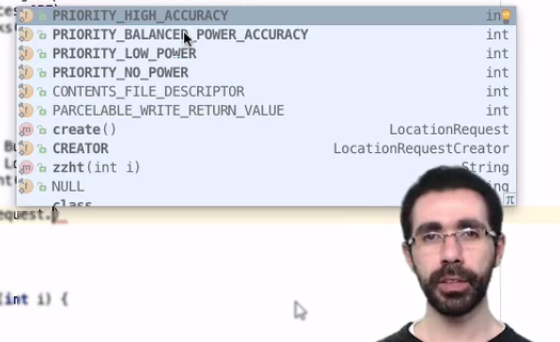
@Override  
public void onConnected(@Nullable Bundle bundle) {  
 LocationRequest request = new LocationsRequest();  
 request.setSmallestDisplacement(50);  
  
}

Agora, imaginaremos a situação em que o usuário esteja se movendo muito rápido, então, o GPS terá que enviar atualizações frequentemente. O ideal é que existisse um intervalo no envio de posição. Usaremos outro parâmetro do request, nós usaremos o setInterval(). Ele está definido em **milisegundos**, logo, se usarmos o valor 1000, será o equivalente a **1 segundo**.

@Override  
public void onConnected(@Nullable Bundle bundle) {  
 LocationRequest request = new LocationsRequest();  
 request.setSmallestDisplacement(50);  
 request.setInterval(1000);  
  
}

O setSmallestDisplacement() e o setInterval() irão funcionar como uma condição **E** (&&). Caso o usuário tenha se deslocado 50 metros **e** se o intervalo for maior que 1 segundo, nós receberemos uma atualização da localização. Se ele ficar parado, não importa quanto tempo passe, não será enviada uma atualização. O último item que conseguimos configurar no request é a **precisão** da localização que queremos receber.

Nós podemos solicitar informações da rede 3G, do Wi-fi, do GPS. Quando escolhemos este último, estamos priorizamos a posição da localização e penalizamos o uso da bateria - quanto maior a precisão, maior seráo gasto de energia. Vamos definir o item usando o setPriority(), que possui uma série de constantes que podem ser utilizadas. Se escolhemos o PRIORITY\_HIGH\_ACCURACY, teremos uma precisão muito alta e será escolhido no seu celular o hardware mais apropriado. Mas existem outras precisões que podem ser utilizadas.



Nós usaremos o PRIORITY\_HIGH\_ACCURACY, porque queremos utilizar o GPS.

request.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);

O nosso request ficou assim:

@Override  
public void onConnected(@Nullable Bundle bundle) {  
 LocationRequest request = new LocationsRequest();  
 request.setSmallestDisplacement(50);  
 request.setInterval(1000);  
 request.setPriority(LocationRequest.PRIORITY\_HIGH\_ACCURACY);  
  
 }

Agora, só precisamos passar o request para o LocationServices. Para isto, colocaremos dentro desta classe uma API chamará FusedLocationApi, que terá o método requestLocationUpdates().

LocationServices.FusedLocationApi.requestLocationUpdates()

Para conseguir pegar as requesties e mandá-las para o LocationServices precisaremos da conexão estabelecida anteriormente, representada pelo client. Qual é o GoogleApiClient está sendo utilizado para se conectar ao serviço? Podemos responder que é o client - que foi passado anteriormente como uma variável local. Antes, teremos que transformar o client em um atributo da nossa classe. Vamos retirar a definição (GoogleApiClient) e vamos criar um *field* para client.

public class Localizador implements GoogleApiClient.ConnectionCallbacks {  
  
 private final GoogleApiClient client;  
  
 public Localizador(Context context) {  
 client = new GoogleApiClient>Builder(context)  
 .addApi(LocationServices.API)  
 .addConnectionCallbacks(this)  
 .build();  
  
 client.connect();  
 }

Depois, poderemos usar o client no requestLocationUpdates. Como segundo parâmetro, passaremos o request.

LocationServices.FusedLocationApi.requestLocationUpdates(client, request)

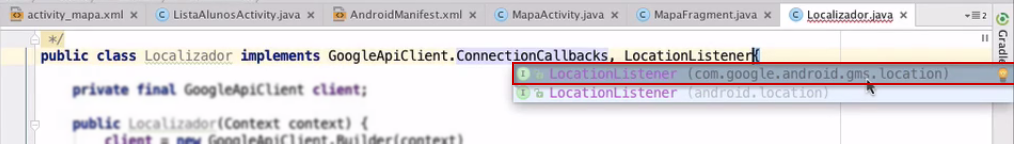
Nós configuramos o GPS, precisaremos informar como terceiro parâmetro quem será o responsável por tratar a atualização da posição. No caso, será a própria classe, por isso, usaremos o this.

LocationServices.FusedLocationApi.requestLocationUpdates(client, request, this);

Temos a conexão, os dados da requisição, e quem irá tratar os dados. O terceiro parâmetro pede uma interface chamada LocationListener. Teremos que implementá-la mais acima.

public class Localizador implements GoogleApiClient.ConnectionCallbacks, LocationListenerr

**Atenção:** Existem duas LocationListener, no Android. Nós queremos implementar é com.google.android.gms.location.



Em seguida, implementaremos o método onLocationChanged() da interface. Ele irá passar um objeto do tipo Location.

@Override  
public void onLocationChanged(Location location) {  
  
}

Neste ponto, quando a localização foi alterada, precisaremos modificar a posição do nosso mapa. Teremos que criar uma referência para este, dentro do onLocationChanged. Faremos isto, acrescentando a referência no construtor.

private final GoogleApiClient client;  
  
public Localizador(Context context, GoogleMap mapa) {  
 client = new GoogleApiClient>Builder(context)  
 .addApi(LocationServices.API)  
 .addConnectionCallbacks(this)  
 .build();  
  
 client.connect();  
  
 this.mapa = mapa;  
}

Depois, criaremos um atributo para mapa.

private final GoogleApiClient client;  
private final GoogleMap mapa;  
  
public Localizador(Context context, GoogleMap mapa) {  
 client = new GoogleApiClient>Builder(context)  
 .addApi(LocationServices.API)  
 .addConnectionCallbacks(this)  
 .build();  
  
 client.connect();  
  
 this.mapa = mapa;  
}

Com a referência do mapa pronta, chamaremos o moveCamera() no onLocationChanged. Com o cameraUpdate, informaremos a nova posição da câmera do mapa. Iremos construir uma variável com o CameraUpdateFactory.

@Override  
public void onLocationChanged(Location location) {  
 CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLng();  
 mapa.moveCamera(cameraUpdate);  
  
}

O método construtor NewLatLng() irá especificar apenas a posição recebida como parâmetro.

Depois, criaremos o LatLng() e iremos instânciar um novo LatLng(). Usaremos como primeiro parâmetro a latitude, e o segundo será a longitude.

@Override  
public void onLocationChanged(Location location) {  
 LatLng coordenada = new LatLng(location.getLatitude(), location.getLongitude());  
 CameraUpdate cameraUpdate = CameraUpdateFactory.newLatLng(coordenada);  
 mapa.moveCamera(cameraUpdate);  
  
}

Passamos o coordenada para o CameraUpdateFactory.

Observe que a seguinte linha estará sinalizada com vermelho:

LocationServices.FusedLocationApi.requestLocationUpdates(client, request, this);

A sinalização indica que precisamos atualizar algumas permissões na aplicação para trabalhar com o GPS. Uma delas nós já adicionamos no AndroidManifest.java, a permissão ACCESS\_FINE\_LOCATION e que serve para requisitar a localização do usuário. Precisaremos adicionar a ACCESS\_COARSE\_LOCATION.

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

As duas são utilizadas para podermos usar os dispositivos que identificam a localização do usuário.

Precisaremos adicionar outra que irá requisitar a permissão do usuário. No caso, precisamos pedir ao usuário que ele autorize que a sua localização seja identificada. Como este não é o nosso foco, nós iremos compilar o código com a linha sinalizada no Localizador.java. Como está aqui, o usuário dará a permissão manualmente. Nos [exercícios](https://cursos.alura.com.br/course/android-studio-iii-fragments-maps-gps/section/3/task/11), você terá que explicar como fazer isto da forma correta.

Terminamos nossa classe Localizador, e queremos que ela seja vinculado ao MapaFragment. No MapaFragment.java, iremos instanciar o new Localizador, após criarmos o mapa e colocarmos as marcações.

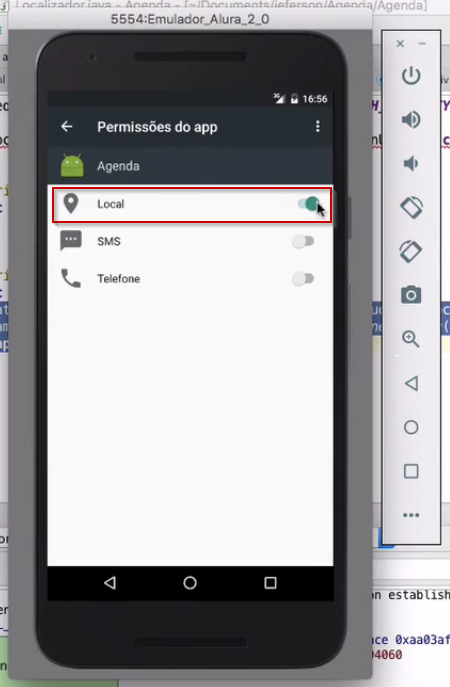
new Localizador(getContext(), googleMap);

O trecho do código ficará assim:

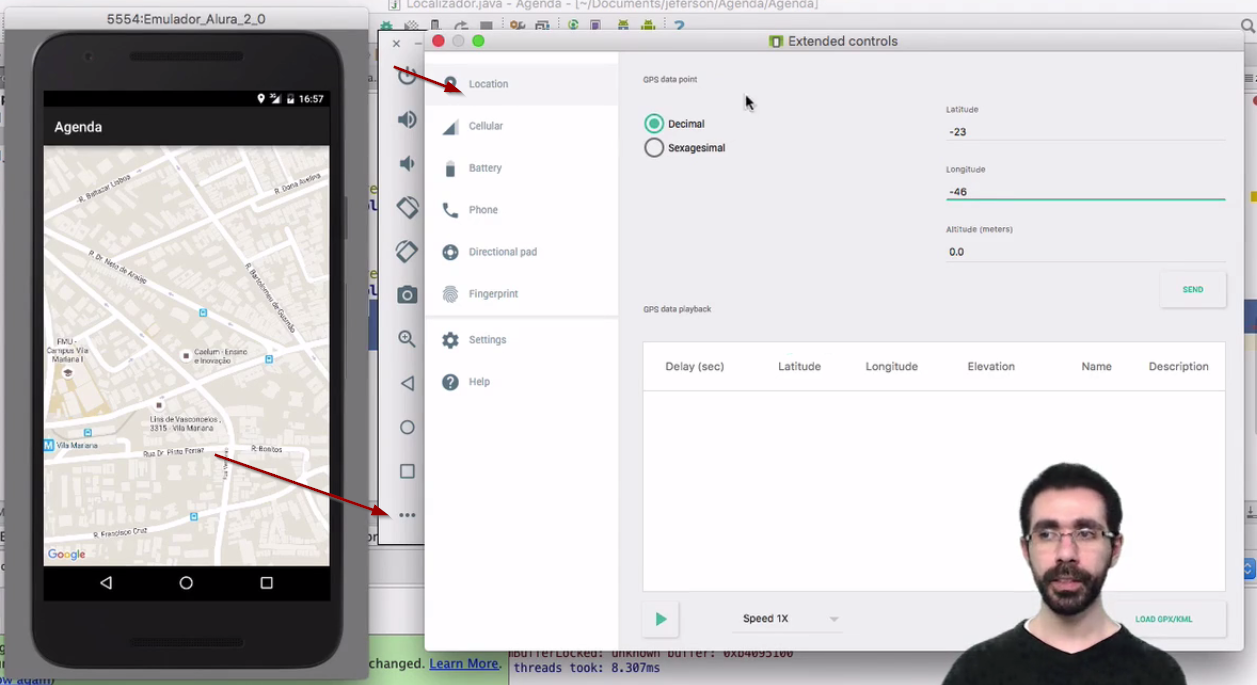
@Override  
public void onMapReady(GoogleMap googleMap) {  
 LatLng posicaoDaEscola = pegaCoordenadaDoEndereco("Rua Vergueiro 3185, Vila, Sao Paulo");  
 if (posicaoDaEscola != null) {  
 CameraUpdate update = CameraUpdateFactory.newLatLngZoom(posicaoDaEscola, 17);  
 googleMap.moveCamera(update);  
}  
  
AlunoDAO alunoDAO = new AlunoDAO(getContet());  
for (Aluno aluno : alunoDAO.buscaAlunos()) {  
 LatLng coordenada = pegaCoordenadaDoEndereco(aluno.getEndereco());  
 if (coordenada != null) {  
 MarkerOptions marcador = new MarkerOptions();  
 marcador.position(coordenada);  
 marcador.title(aluno.getNome());  
 marcador.snippet(String.valueOf(aluno.getNota()));  
 googleMap.addMarker(marcador);  
  
  
 }  
}  
alunoDAO.close();  
  
new Localizador(getContext(), googleMap);

Quando damos um new Localizador, rodaremos o construtor client, no qual configuramos a API e pedimos para conectar. Após a conexão, será feita a requisição para o GPS, a partir daí, de forma assíncrona, começaremos a receber as informações no método onLocationChanged. Sempre que isto acontecer, o nosso mapa será centralizado em uma nova posição.

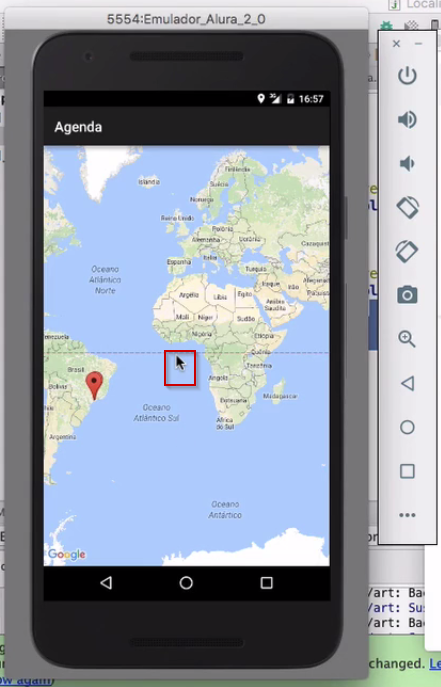
Agora, vamos testar nossa aplicação no emulador. Primeiramente, daremos a permissão para o uso do GPS manualmente. Iremos acessar as permissões do app e acionaremos a de "Local".



Depois, abriremos a aplicação da Agenda. Acessaremos a parte de mapa e para simular a mudança de localização, clicaremos no ícone de reticências (...) no meu à direita e selecionaremos "Location".



Vamos usar o valor 0 para latitude e a longitude. O nosso mapa será posicionado no oceano.



Ao realizarmos novos testes com diferentes valores de latitude e longitude, o mapa irá se mover. Se estivéssemos com o dispositivo na mão, caminhando pela rua, a cada 50 metros e após 1 segundo, o mapa seria atualizado automaticamente.

Nós vimos como trabalhar com o GPS, mas usando o método onLocationChanged poderia ser feito outras melhorias. Por exemplo, com as posições, em vez de atualizar o mapa, poderíamos salvar as informações de tempo e posição, no banco de dados. Depois, poderíamos plotar no mapa todos os lugares por onde o usuário passou. Poderíamos mostrar o percurso como nas aplicações de corrida.

Você pode usar a sua criatividade e utilizar de diferentes formas o GPS.

Com isto, concluímos o terceiro curso de Android. Parabéns por ter chegado até aqui! Agora, é colocar a mão na massa, exercitar bastante, e aproveitar tudo o que foi visto no curso.