Customização do Mapa

Para customizar o Fragment teremos que lembrar os conceitos de orientação de objetos.

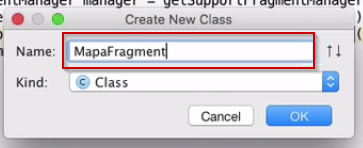
FragmentManager manager = getSupportFragmentManager();  
 FragmentTransaction tx = manager.beginTransaction();  
 tx.replace(R.id.frame\_mapa, new SupportMapFragment());  
 tx.commit();

No arquivo MapaActivity.java, estávamos substituindo o frame\_mapa por uma instância do SupportMapFragment.

tx.replace(R.id.frame\_mapa, new SupportMapFragment());

Agora, o que gostaríamos de fazer é ter todos os comportamentos do Google Maps e adicionar algumas funcionalidades a mais. Não queríamos ter o trabalho de implementar o desenho do mapa, o scroll, o zoom... Quando pensamos em adicionar funcionalidade, na Orientação de Objetos, primeiramente lembraremos de **herança**. Podemos criar uma classe que herda de SupportMapFragment() e estendermos as capacidades dessa classe colocando o que desejamos. É o que faremos adiante.

Criaremos um "Java Class" que chamaremos de MapaFragment.



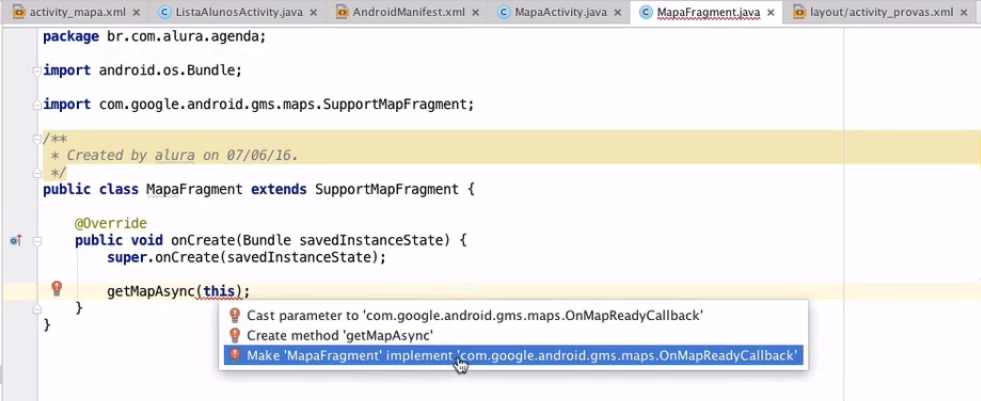
\...  
public class MapaFragment extends SupportMapFragment {  
  
}

O MapaFragment já herdou o SupportMapFragment. Com isto, temos um Fragment igual ao anterior. se voltarmos na aba MapaActivity.java e substituirmos o SupportMapFragment() por MapaFragment(), a aplicação continuará funcionando normalmente.

A vantagem é que o Fragment agora é nosso e por isso, conseguimos criar métodos e instanciar objetos e outras melhorias no comportamento. Faremos isto em seguida.

No arquivo MapaFragment.java, primeiramente iremos manipular o método onCreate. Lembrando que o Fragment terá um ciclo de vida um pouco parecido com a Activity.

Em seguida, chamaremos o método getMapAsync(), que irá preparar uma instância do Google Maps e nos permitirá colocar os marcadores ou outras customizações no mapa. Dentro dos parênteses teremos que informar um objeto que responda ao momento em que o mapa estiver pronto, usaremos o this. Para que ele funcione, precisamos ter um método específico, indicando que o mapa está pronto: o OnMapReadyCallback.



Após implementarmos a interface, ele será adicionado no código:

public class MapaFragment extend SupportMapFragment implements OnMapReadyCallback {  
  
 @Override  
 public void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
  
 getMapAsync(this);  
 }  
  
 @Override  
 public void onMapReady(GoogleMap GoogleMap) {  
  
 }  
}

O método OnMapReadyCallback será chamado assim que o mapa estiver pronto e então, podemos customizá-lo como quisermos.

Primeiramente, gostaríamos de centralizar o mapa na nossa escola - especificamente, no endereço da Caelum. Para isto, utilizaremos o método moveCamera(), que irá pedir o objeto CameraUpdate. Teremos que construí-lo, adicionando um update.

@Override  
public void onMapReady(GoogleMap GoogleMap) {  
 googleMap.moveCamera(update);  
  
}

Solicitaremos ao Android Studio que inicialize a variável, que será do tipo CameraUpdate.

@Override  
public void onMapReady(GoogleMap GoogleMap) {  
 CameraUpdate update;  
 googleMap.moveCamera(update);  
  
}

Depois, usaremos uma Factory. Teremos um objeto especialista em criar CameraUpdate. Adicionaremos o método newLatLngZoom(), com o qual especificaremos uma **latitude**, uma **longitude** e um nível de **zoom**. Teremos que passar um objeto LatLng, float. O LatLngcriaremos manualmente. Mas antes, precisaremos descobrir qual a latitude e a longitude da Caelum. Para transformarmos o endereço em uma coordenada geográfica, precisaremos de uma *geocoding*. Temos uma classe que irá realizar este processo, chamada Geocoder.

@Override  
public void onMapReady(GoogleMap GoogleMap) {  
 Geocoder geocoder = new Geocoder (getContext());  
  
 LatLng posicaoDaEscola = new LatLng()  
 CameraUpdate update= CameraUpdateFactory.newLatLngZoom();  
 googleMap.moveCamera(update);  
  
}

O Geocoder precisará ter acesso aos recursos do sistema, por isso irá pedir um contexto. Como estamos dentro de um Fragment, passamos o getContext() como parâmetro.

A utilidade que daremos para o geocoder é que ele converta o endereço em uma coordenada. Adicionaremos o método getFromLocationName(), que irá pedir como parâmetro o endereço e quantos resultados de busca pela localização nós temos interesse. No caso, em apenas 1 resultado.

geocoder.getFromLocationName("Rua Vergueiro 3185, Vila Mariana, São Paulo", 1);

Depois, iremos introduzir o getFromLocationName em uma variável local, que chamaremos de variável local.

List<Address> resultados =  
 geocoder.getFromLocationName("Rua Vergueiro 3185, Vila Mariana, São Paulo", 1);

Toda vez que ele quiser converter um endereço, precisará fazer uma requisição para o Google, é possível que aconteça algum problema na operação. Neste caso, ele irá pedir para tratarmos uma *Exception* de IO. Por isso, iremos colocar o trecho dentro de um try-catch. No fim, faremos um e.printStackTrace(), caso aconteça algum problema.

@Override  
public void onMapReady(GoogleMap GoogleMap) {  
 try {  
 Geocoder geocoder = new Geocoder (getContext());  
 List<Address> resultados =  
 geocoder.getFromLocationName("Rua Vergueiro 3185, Vila Mariana, São Paulo", 1);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

Teremos que verificar se tivemos um resultado. Especificamos o número de resultado máximo, mas talvez, ele seja igual a 0. Se não for encontrado o endereço, o resultado virá vazio. Vamos usar um if para o caso em que o resultado **não** for vazio, tentaremos extrair as coordenadas. Em seguida, pegaremos o primeiro elemento da lista com o get(0), e chamaremos o método getLongitude().

if (!resultados.isEmpty()) {  
 resultados.get(0).getLongitude()  
}

O código ficou assim:

@Override  
public void onMapReady(GoogleMap GoogleMap) {  
 try {  
 Geocoder geocoder = new Geocoder (getContext());  
 List<Address> resultados =  
 geocoder.getFromLocationName("Rua Vergueiro 3185, Vila Mariana, São Paulo", 1);  
 if (!resultados.isEmpty()) {  
 resultados.get(0).getLongitude()  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }

Para manter a organização do código, vamos recortar o trecho referente à geo coordenada e criaremos um método privado da classe que devolverá um LatLng e que chamaremos de pegaCoordenadaDoEndereco.

private LatLng pegaCoordenadaDoEndereco(String endereco) {  
  
}

Passamos uma *String* endereco. Mais acima, iremos substituir o trecho com endereço, pelo método endereco. Também alteraremos o resultados.

private LatLng pegaCoordenadaDoEndereco(String endereco) {  
 try {  
 Geocoder geocoder = new Geocoder (getContext());  
 List<Address> resultados =  
 geocoder.getFromLocationName(endereco, 1);  
 if (!resultados.isEmpty()) {  
 LatLng posicao = new LatLng(resultados.get(0).getLatitude(), resultados.getLongitude());  
 return posicao;  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }   
  
 return null;  
 }

Se tivermos um resultado válido para a nossa busca, teremos um LatLng e a latitude e longitude pegaremos dos resultados. Depois, ele devolveu a posição. No fim, caso dê algum problema e não tivermos nenhum resultado, o retorno será null.

Agora que temos o método, quando precisarmos pegar a posicaoDaEscola, não iremos mais instanciar LatLng() manualmente. Em vez disso, chamaremos o método pegaCoordenadaDoEndereco e adicionaremos o endereço da Caelum.

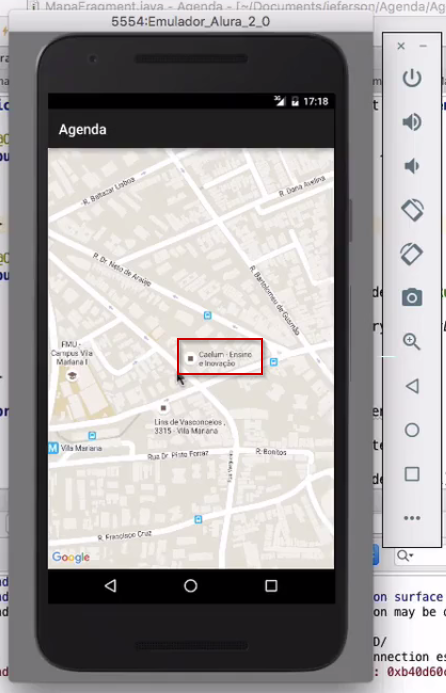
LatLng posicaoDaEscola = pegaCoordenadaDoEndereco("Rua Vergueiro 3185, Vila Mariana, Sao Paulo");

O método fará a busca do geocoding e irá converter em uma coordenada, que será retornada no posicaoDaEscola. Se a posição da escola não for nula, iremos atualizar a câmera e reposicionar o mapa. Como parâmetros do newLatLngZoom() usaremos o posicaoDaEscola e o nível de zoom, por padrão, definiremos como 17. Quanto maior for o valor, mais próximo ficaremos da localização.

Até aqui, o nosso código ficou assim:

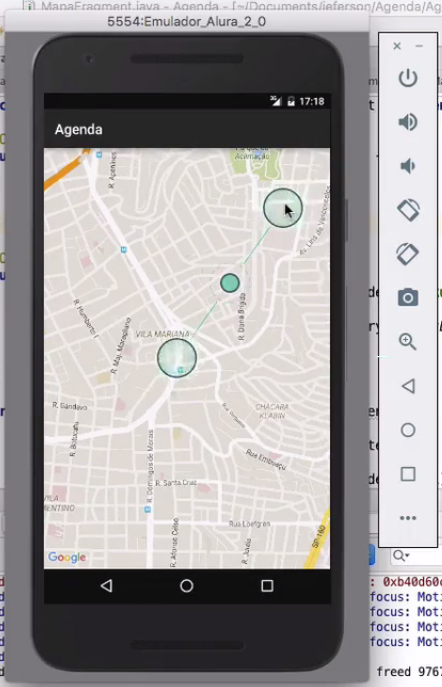
@Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_mapa);  
 }  
  
 @Override  
 public void onMapReady(GoogleMap GoogleMap) {  
  
 LatLng posicaoDaEscola = pegaCoordenadaDoEndereco("Rua Vergueiro 3185, Vila Mariana, Sao Paulo");  
 if (posicaoDaEscola != null) {  
 CameraUpdate update = CameraUpdateEndereco.newLatLngZoom(posicaoDaEscola, 17);  
 gooleMap.moveCamera(update);  
 }  
 }  
  
 private LatLng pegaCoordenadaDoEndereco(String endereco) {  
 try {  
 Geocoder geocoder = new Geocoder (getContext());  
 List<Address> resultados =  
 geocoder.getFromLocationName(endereco, 1);  
 if (!resultados.isEmpty()) {  
 LatLng posicao = new LatLng(resultados.get(0).getLatitude(), resultados.getLongitude());  
 return posicao;  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }   
  
 return null;  
 }

Vamos testar a aplicação, se tudo estiver correto, ela funcionará e estará focada na localização da nossa escola.



Ao entramos na parte de mapas, a app fez a busca que pedimos e o mapa ficou posicionado sobre a Caelum. Nós passamos o endereço e ela conseguiu identificar a latitude e a longitude.

Ainda queremos que apareça no mapa a localização de todos os nossos alunos. Ao fazermos um *zoom out*, deveríamos conseguir ver uma marcação com a localização dos nossos alunos.



Mas não estão aparecendo no momento. Iremos trabalhar com isto mais adiante.

Primeiramente, precisaremos ter os alunos. Para isto, utilizaremos o AlunoDAO que criamos anteriormente. No MapaFragment.java, abaixo do onMapReady(), adicionaremos a variável alunoDAO, vou instanciá-la passando o contexto.

AlunoDAO alunoDAO = new AlunoDAO(getContext());

Com o AlunoDAO, poderemos buscar todos os alunos da nossa agenda e os seus endereços. Em seguida, adicionaremos um laço e para cada aluno que tivermos no alunoDAO.buscaAluno, queremos a coordenada da sua localização.

AlunoDAO alunoDAO = new AlunoDAO(getContext());  
for (Aluno aluno : alunoDAO.buscaAlunos()) {  
 LatLng coordenada = pegaCoordenadaDoEndereco(aluno.getEndereco());  
}

Criamos uma variável do tipo coordenada e usamos o pegaCoordenadaDoEndereco(). Com isto, sabemos onde colocar o pino marcador. Se a coordenadanão for nula, queremos gerar o pino e que ele seja inserido no mapa. Vamos usar um if, dentro do for. Usaremos a classe MarkerOptions, que receberá o nome de marcador. Depois, iremos importá-la.

if (coordenada != null) {  
 MarkerOptions marcador = new MarkerOptions();  
}

Criamos o marcador, agora, precisaremos passar as características: sua posição (coordenada), qual texto será associado (nome do aluno e a nota).

if (coordenada != null) {  
 MarkerOptions marcador = new MarkerOptions();  
 marcador.position(coordenada);  
 marcador.title(aluno.getNome());  
 marcador.snippet(aluno.getNota());  
}

Mas se usarmos o marcador.snippet(aluno.getNota()), o Android vai reclamar, porque o snippet espera por uma *string* e a nota é um Double. Para convertê-lo, adicionaremos String.valueOf().

marcador.snippet(String.valueOf(aluno.getNota()));

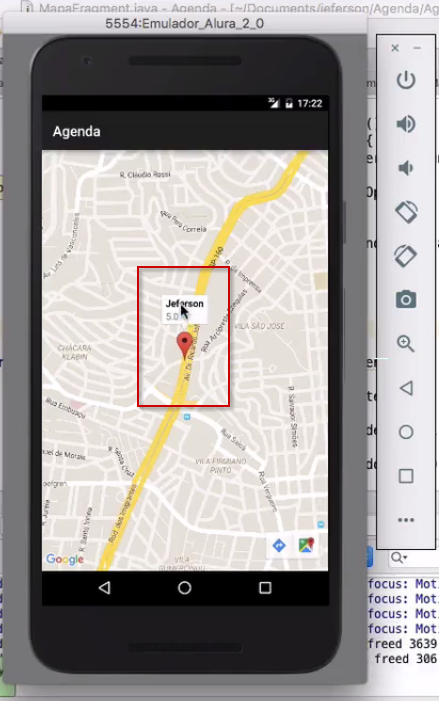
Por enquanto, criamos o pino, mas queremos inseri-lo no mapa. Para isto, informaremos a instância do mapa e adicionaremos o marcador.

googleMap.addMarker(marcador);

Depois, fecharemos o alunoDAO. O laço ficou assim:

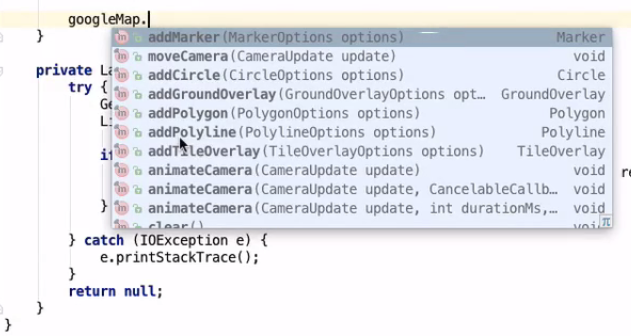
AlunoDAO alunoDAO = new AlunoDAO(getContext());  
for (Aluno aluno : alunoDAO.buscaAlunos()) {  
 LatLng coordenada = pegaCoordenadaDoEndereco(aluno.getEndereco());  
 if (coordenada != null) {  
 MarkerOptions marcador = new MarkerOptions();  
 marcador.position(coordenada);  
 marcador.title(aluno.getNome());  
 marcador.snippet(String.valueOf(aluno.getNota()));  
 googleMap.addMarker(marcador);  
  
  
 }  
}  
alunoDAO.close();

Vamos voltar ao emulador e ver se os pinos com a localização dos três alunos aparecem no mapa.

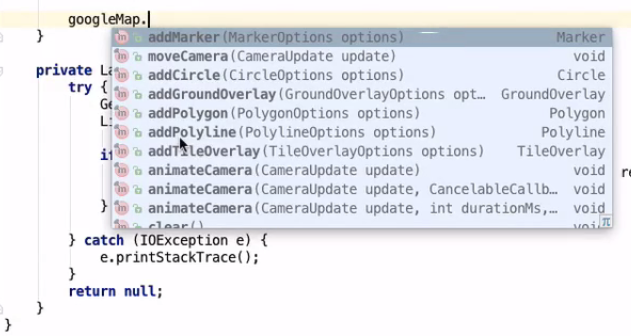


Cada aluno estava localizado em um endereço diferente e os três foram marcados no mapa. Então, conseguimos adicionar os pinos e as informações extras.

Com isto, conseguimos customizar o comportamento do mapa. Poderíamos também utilizar o Listerner. Dentro da instância do Google Maps, poderíamos adicionar círculos, polígonos e polylines.



E Listeners... Quando o usuário clicasse em alguma posição do mapa ou no marcador.



Temos a opção de alterarmos o comportamento do funcionamento do mapa. Ele é bastante expansivo e pode colaborar bastante com a sua aplicação. Aproveite para brincar e testar estas outras opções.

Encerramos a parte de trazer o mapa para nossa aplicação.