Dispositivos Móveis

Aula 08 - Localização e Mapas

Apresentação

Olá! Seja bem-vindo! Já concluímos nossas aulas sobre Activities e Interfaces Gráficas e podemos assim dizer que já aprendemos os conceitos básicos da programação Android. A partir de agora, conheceremos aspectos mais avançados da plataforma e, com isso, aumentaremos nossas capacidades para desenvolver aplicativos de maior qualidade e de maior utilidade.

Na aula de hoje, estudaremos localização e mapas. Como temos visto em nossas aulas, o acesso às diversas funcionalidades existentes no aparelho é um ponto forte do Android. Mais uma vez, temos a capacidade de trabalhar com funções do aparelho que nos dão uma nova gama de oportunidades. Veremos como é possível adquirir e utilizar a localização geográfica de um usuário, bem como integrar os mapas do Google à nossa aplicação.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Utilizar a geolocalização de um usuário.
- Integrar os mapas do Google à sua aplicação.

Localização

Saber onde o usuário se encontra e poder utilizar essa informação em tempo real é uma funcionalidade muito interessante quando estamos falando em aplicações para dispositivos móveis. Uma enorme gama de aplicações novas surge quando se tem acesso a esse tipo de informação. A capacidade de poder tornar a sua aplicação mais local, utilizando informações da região do usuário, mesmo que seja para exibir anúncios personalizados, é de grande valia e não pode ser deixada de lado. Por esses motivos, a partir de agora estudaremos como utilizar a localização do usuário em uma aplicação Android.

O Android disponibiliza duas maneiras de descobrir a localização do usuário: o GPS e o Android's Network Location Provider. O GPS já é um instrumento conhecido e consolidado para se obter a localização. Como grande parte das pessoas já sabe, o GPS utiliza-se de uma grande rede de satélites para determinar a posição atual do usuário. O grande problema desse tipo de abordagem é que o uso dos satélites não é viável se o dispositivo não estiver em um ambiente aberto, ou seja, temos um problema para detectar posicionamentos em ambientes fechados. Além disso, a comunicação com satélites desse tipo gera um grande consumo de bateria, o que é uma grande preocupação para usuários de dispositivos móveis.

Para auxiliar o GPS, o Android também disponibiliza o Network Location Provider. Esse segundo método se utiliza de informações obtidas a partir das torres de telefonia móvel, bem como das informações conhecidas sobre as Wi-Fi disponíveis para obter, assim, uma localização um pouco mais precisa do usuário. Esse método é mais rápido que a utilização do GPS e também independe se o usuário está em um ambiente externo ou interno. Além disso, utilizar o Network Location Provider consome menos bateria que utilizar o GPS puro. Então, qual método utilizar em sua aplicação? Isso fica a seu critério, seja apenas um deles, ou os dois. Geralmente utilizamos os dois, deixando que o aparelho detecte a localização rapidamente.

Problemas para se Obter a Localização do Usuário

Como você já deve estar imaginando, obter a localização de um usuário em um dispositivo móvel não é uma tarefa simples. Alguns problemas são bem claros quando estamos pensando em adquirir a localização de um usuário. Vamos discutir um pouco alguns deles.

O primeiro caso a se pensar quando estamos requisitando a localização de um usuário é saber quais os recursos de localização aquele usuário terá disponível naquele momento. Por se tratar de um dispositivo móvel, pode ser que um usuário que possui GPS, em um dado momento, esteja em uma área coberta, perdendo grande parte da eficiência desse método. Esse mesmo usuário pode estar fora do alcance de redes Wi-Fi, o que também pode dificultar a obtenção de informações de posicionamento.

Outro ponto importante é que essa disponibilidade dos recursos de localização é alterada a todo o momento, visto que se trata de um dispositivo móvel, que pode estar mudando de localização constantemente. Graças à possibilidade que o dispositivo tem de mudar de posição, é necessário ao desenvolvedor se preocupar com a obtenção de novas posições, que confirmem ou não o movimento do usuário. Isso, se feito de maneira errônea, pode levar a informações pouco precisas, ou mesmo a um gasto excessivo de energia do aparelho.

Por fim, outro problema a se pensar quando estamos lidando com localização é a mudança na precisão dos dados. Um usuário que há 30 segundos estava em um local aberto, com ótimo sinal de GPS, pode agora estar dentro de um shopping, por exemplo, tendo péssimas leituras de posição. Dentro desse shopping, o usuário pode estar em um local com acesso a algumas redes Wi-Fi mapeadas, o que garante uma boa precisão ao outro método de localização, porém, em outros 30 segundos, o usuário já pode ter saído do alcance dessas redes, perdendo mais uma vez a precisão no serviço de localização. Ainda assim, nos próximos 30 segundos, esse usuário já pode ter saído do shopping e voltado a um local aberto, onde a precisão do GPS é bem melhor. Viu quantas mudanças de precisão nos dois sistemas um usuário sofreu em um curto espaço de tempo? Isso também deve ser pensado e levado em conta quando se utiliza serviços de localização.

Esses problemas podem afetar de maneira significativa uma aplicação que depende da localização do usuário. Durante esta aula, buscaremos demonstrar maneiras para evitar cair em qualquer um desses casos e perder na qualidade final da aplicação.

Vídeo 02 - Gerenciador de Localizações

Obtendo a Localização

Para obtermos a localização de um usuário, precisamos requisitar ao Android acesso ao serviço que provê as aplicações com informações sobre a localização do usuário. Esse serviço é fornecido através do LocationManager e é obtido através de uma chamada ao método Context.getSystemService(Context.LOCATION_SERVICE). Através dessa chamada, obtemos uma instância do LocationManager, que é responsável pelas atualizações da localização do usuário. Vejamos na **Listagem 1** como devemos implementar um código para receber a localização do usuário.

O código a seguir deve ser incluído dentro de seu método onCreate(). O Android pode solicitar, por padrão, que sejam feitos testes se as permissões necessárias estão liberadas pelo usuário ou não, através de ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION).

```
LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
LocationListener locationListener = new LocationListener() {

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
    //Código para responder a alterações em um provedor de localização
    }
    @Override
    public void onProviderEnabled(String provider) {
    //Código para responder à habilitação de um provedor de localização
    }
    @Override
    public void onProviderDisabled(String provider) {
    //Código para responder à desabilitação de um provedor de localização
    }
    @Override
    public void onLocationChanged(Location location) {
    //Código para utilizar a nova localização.
    }
};
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0,
locationListener);
```

Listagem 1 - Utilizando o LocationManager

Os métodos são bastante explicativos e os comentários ajudam a definir a utilidade de cada um. O destaque vai para o método onLocationChanged, que é o responsável por notificar a aplicação de uma mudança na localização que tenha sido detectada pelo Manager.

Outro ponto importante a se notar são as chamadas à função requestLocationUpdates. Essa função é a responsável por pegar uma instância do LocationManager e indicar qual (ou quais, como no exemplo) serviços de localização aquela instância ouvirá. O primeiro parâmetro indica qual será esse serviço. No caso da utilização de ambos os serviços, como no exemplo, basta fazer duas chamadas ao método e ele se registrará em ambos os provedores de localização. O segundo parâmetro indica qual é o tempo que deve haver entre a requisição atual e a anterior para tornar um resultado notificável. Já o terceiro parâmetro indica a diferença de distância que há entre notificações, ignorando também as que ficarem abaixo da estipulada. Em casos como o do exemplo, nos quais ambos os parâmetros

são configurados para zero, as atualizações devem ocorrer independente da distância e na maior taxa possível. Já o último parâmetro é o LocationListener responsável por prover os *callbacks* necessários.

Como qualquer outra funcionalidade que você utiliza do Android, antes de compilar a aplicação, é necessário adicionar ao Manifest a permissão necessária para aquela funcionalidade. No caso da utilização da localização do usuário, temos duas permissões que podem ser adicionadas, dependendo de qual provedor de localização a sua aplicação vai utilizar. No caso da utilização dos dois, basta adicionar a permissão de localização fina, que já inclui as duas. As permissões são:

android.permission.ACCESS_COARSE_LOCATION utilizada quando o provedor utilizado na aplicação é apenas o provedor de Network Location;

android.permission.ACCESS_FINE_LOCATION, utilizado quando a aplicação vai acessar o GPS e também no caso da aplicação utilizar os dois provedores de localização.

Agora que já vimos o básico de como adicionar localização à nossa aplicação, vamos ver brevemente algumas soluções conhecidas para os problemas citados na seção: **"Problemas para se obter a localização do usuário"**.

Otimizando Aplicações com Acesso à Localização

Ao desenvolver aplicações que se utilizam da localização do usuário, podemos enfrentar diversos problemas, como os citados na subseção anterior. Esses problemas, apesar de serem consideravelmente complicados, possuem soluções plausíveis e de fácil implementação. Basta que tenhamos um plano para desenvolvê-las. Comecemos com o fluxo básico para obtenção da localização do usuário.

Após iniciar a aplicação, o primeiro aspecto a se perceber está relacionado à inicialização dos serviços de localização. Esse registro não precisa ser feito logo na inicialização, mas deve ser feito antes de qualquer chamada relacionada à localização. Escolha quais deverão ser os provedores utilizados e registre o *Listener* para responder a eventos que aconteçam neles. Esse passo é simples e foi discutido na subseção anterior.

Em seguida, é importante manter uma melhor estimativa do local possível. Isso pode ser feito através de critérios de tempo e precisão. Por exemplo, se uma informação de localização for mais nova que a outra e, porém, vier com uma precisão menor, pode valer a pena manter a informação antiga como sendo a válida, já que tinha uma precisão melhor. No entanto, caso a informação que tenha a melhor precisão comece a ficar muito antiga, é melhor arriscar e passar a mostrar baseado na menos precisa, pois isso ajuda a acompanhar o usuário, caso esteja se movendo. Não há uma métrica exata para isso e deve partir do desenvolvedor definir quanto é velho demais e quanto é uma perda de precisão considerável.

Depois de ter obtido a informação com precisão e em tempo adequado, deve-se parar imediatamente o serviço de localização, pois, como foi dito anteriormente, esses serviços são extremamente custosos no que diz respeito ao consumo de bateria. Após parar esse serviço, a última localização obtida deverá ser utilizada para futuras referências, até que haja a necessidade novamente de se obter uma nova localização. Também é importante usar essa última localização como base, caso seja necessário iniciar o serviço novamente. Como leva um tempo para os serviços iniciarem novamente, utilizar o último valor que havia sido adquirido, caso tenha sido adquirido há um tempo relativamente pequeno, pode acelerar a inicialização geral da aplicação.

Agora que já vimos o fluxo por completo, vamos discutir um pouco mais partes específicas.

- *Quando começar a buscar a localização?*

  Esse é um ponto importante em uma aplicação que usa a localização. Nunca é demais ressaltar a importância de só manter as atualizações de localização ativas quando forem necessárias, a fim de economizar bateria. Porém, lembre-se que à medida que o tempo passa, as requisições costumam ficar mais precisas, então, é importante através de testes, tentativas e erros, encontrar o balanço entre precisão e tempo ativo dos provedores de localização. De qualquer maneira, a busca deve ser inicializada juntamente com a aplicação, ou quando os dados forem necessários pela primeira vez e se manter ativa pelo tempo que for necessário apenas.

- *Utilizando a última posição conhecida*

  Como já dissemos, o tempo que leva até o serviço de localização começar a mandar informações para o Listener pode ser grande demais para que o usuário aguarde. Para contornar esse problema, o LocationManager tem o método *getLastKnownLocation,* que é responsável por retornar rapidamente a última posição conhecida do usuário. Utilizando esse artifício é possível inicializar sua aplicação baseada nessa posição enquanto as atualizações não começam a chegar.

- *Decidindo quando parar de receber a localização*

  Aqui, mais uma vez, enfrentamos o mesmo problema relacionado à bateria. Se mantivermos a recepção de dados do provedor de localização ativa por muito tempo, faremos com que a bateria acabe muito rapidamente, o que não irá agradar nenhum usuário. O problema aqui é um pouco menor, pois é mais fácil saber quando é possível parar o serviço do que quando começá-lo adequadamente. Para parar o serviço de localização no seu aplicativo, basta tirar o registro do Listener, utilizando o método do LocationManager *removeUpdates,* passando o seu Listener.

Estratégias para Economizar Bateria

Existem algumas estratégias conhecidas para economizar bateria sem prejudicar muito a precisão de sua aplicação. Vejamos agora quais são essas estratégias.

1. *Reduza o tempo ligado ao provedor de localização:* Uma menor janela de tempo conectado aos provedores significa, necessariamente, uma redução no consumo de bateria gerado pelo seu aplicativo. Se esforce para calibrar isso da melhor maneira.

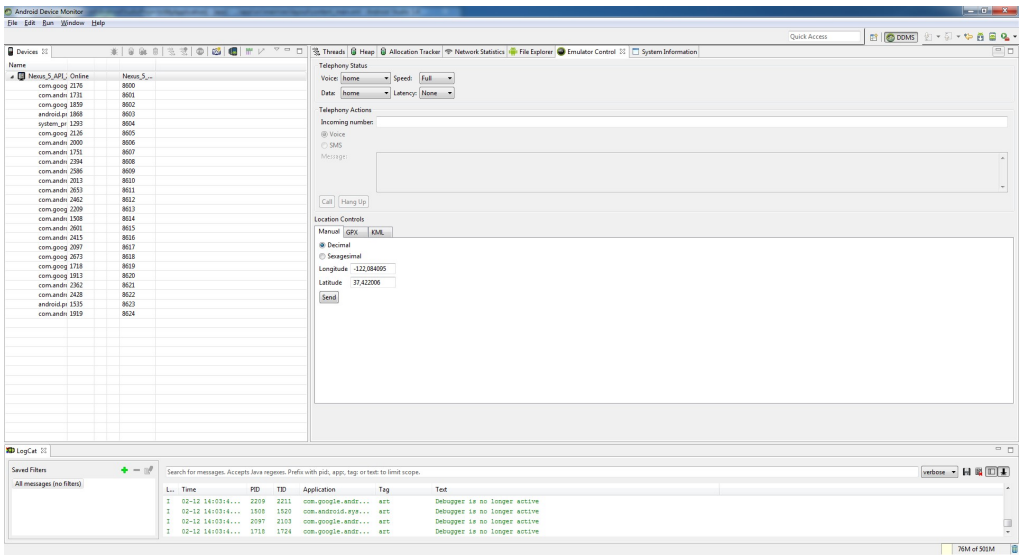
2. *Diminua o número de atualizações retornadas pelos provedores:* Ao registrar um Listener a um provedor, como vimos anteriormente, é possível definir o tempo que leva para uma atualização ser reportada ao Listener. Aumentar esse tempo pode diminuir o consumo de bateria e principalmente de tráfego de rede gerado pela aplicação.

3. *Restrinja os provedores:* Como já vimos, é possível utilizar somente uma das duas opções que o Android provê para conseguirmos acesso à localização de um usuário. Se uma aplicação vai usar apenas um dos dois provedores, então limite-a, fazendo com que ela não fique pesquisando qual dos dois é o melhor. Por exemplo, uma aplicação sobre caminhadas na mata não vai se beneficiar muito de redes Wi-Fi, correto?. Mas, se há a possibilidade de utilizar os dois tipos de provedores, então deixe-os ativos e o próprio Android irá cuidar do consumo dos recursos, escolhendo qual dos dois é o melhor no momento.

Simulando Localizações no Android Studio

Para testar aplicativos que utilizam localização no emulador, precisamos de uma maneira de simular a localização do aparelho, já que a maior parte dos computadores não possui GPS e mesmo os que possuem não o têm integrado ao emulador do Android. Para isso, o Android Studio nos fornece uma tela específica, como podemos ver na **Figura 1**. Essa tela pode ser acessada através do menu **Tools → Android → Android Device Monitor**.

Figura 01 - Android Device Monitor.



Através dessa tela, é possível definir posições geográficas específicas e então configurar o dispositivo para agir como se estivesse posicionado no local indicado pela simulação. Também é possível gerar arquivos do tipo GPX para simular caminhos por diversos pontos geográficos. Tudo isso pode ser feito facilmente, de maneira visual, pelo menu mostrado na **Figura 1**.

É importante ressaltar também que para que essa tela funcione corretamente é necessário que haja um emulador ativo no seu sistema. Perceba também que por vezes o Android Studio gera um erro ao tentar abrir essa tela. Isso é causado por falta de permissões para o acesso ao emulador e pode ser corrigido executando o Android Studio como administrador (clicando com o botão direito no ícone e escolhendo a opção "Executar como Administrador". Obviamente, é necessário ter permissões de administrador na máquina utilizada para isso.

Com isso, terminamos nossos estudos sobre localização. O próximo passo é estudar como o Google nos disponibiliza o acesso a sua API de Mapas no desenvolvimento de aplicações Android.

Atividade 01

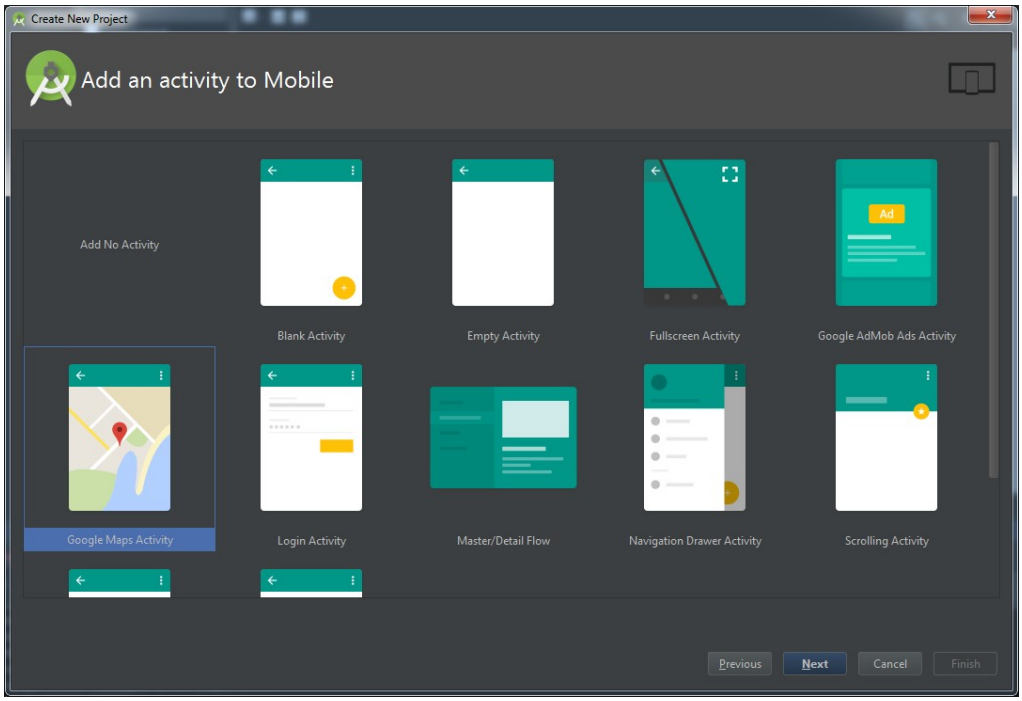
1. Crie uma Activity que contenha um botão capaz de ligar ou desligar a atualização de posição do usuário, quando clicado. Mostre a posição atual em uma TextView acima do botão.

Mapas

A utilização de mapas em aplicações Android mudou recentemente, com a adição da plataforma Google Play Services e a migração de interface de desenvolvimento do Eclipse para o Android Studio. As mudanças vieram facilitar a utilização de mapas e agora veremos, passo a passo, como criar um novo projeto e adicionar a ele mapas a partir da própria API do Google.

O primeiro passo é a criação de um novo projeto no Android Studio. Perceba que, durante a criação das Activities, é possível escolher a opção Google Maps Activity, como visto na **Figura 2**.

Figura 02 - Criação de um novo projeto selecionando a opção Google Maps Activity.



Selecionada essa Activity, podemos avançar e utilizar os valores padrão para os campos da tela seguinte, finalizando então a criação de nosso novo projeto.

Após criado o projeto, o Android Studio já irá, por padrão, abrir um arquivo chamado google_maps_api.xml. Esse arquivo contém o caminho necessário para conseguir uma chave do Google Maps API. Essa chave é necessária para que identifiquemos a nossa aplicação ao conectar no serviço de mapas do Google. Ela é única para sua aplicação e pode ser obtida gratuitamente seguindo as instruções que estão descritas nesse arquivo. Caso o arquivo não abra automaticamente, ele pode ser encontrado na pasta Res -> Values para que seja aberto manualmente.

A maneira mais simples de se obter a chave é utilizar o link que está descrito no arquivo google_maps_api.xml e seguir as instruções lá descritas, criando um novo projeto e, em seguida, a chave. O link deve ser algo parecido com:

```
https://console.developers.google.com/flows/enableapi?
apiid=maps_android_backend&keyType=CLIENT_SIDE_
ANDROID&r=A0:D1:D2:A0:AC:B2:B5:4A:31:87:79:C5:33:
23:22:A1:29:CB:15:3D%3Bbr.ufrn.imd.myapplication
```

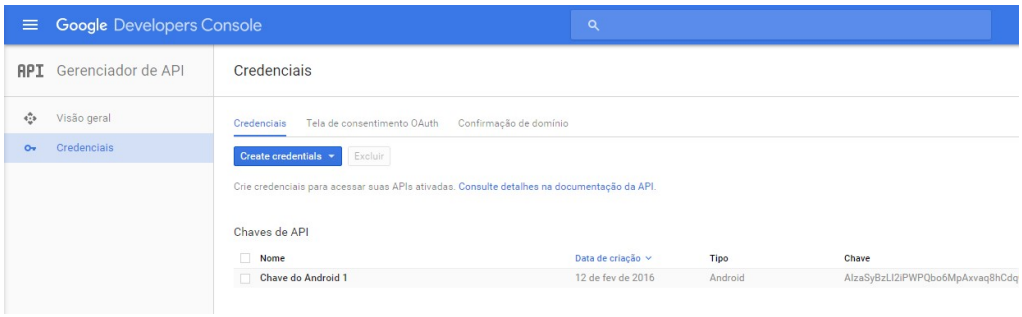
Ao acessar o link você deve utilizar a sua conta Google e então registrar a sua aplicação no Google Developers Console, através da criação de um novo projeto (também é possível selecionar um já existente), como pode ser visto na **Figura 3**.

Figura 03 - Criação de um novo projeto no Google Developers Console para obter a chave do Google Maps API.

Após clicar em Continuar, seu projeto será criado e você estará apto então a receber credenciais de diversas APIs do Google. Clique em "Acessar credenciais" para prosseguir com a obtenção da chave.

A próxima tela surgirá com a opção de criar chave de API do Android já selecionada e, se você seguiu para ela pelo link, como indicado anteriormente, com os campos já preenchidos para garantir a criação de sua chave da maneira correta, com a certificação adequada. Caso a impressão digital não esteja preenchida, ela pode ser encontrada no mesmo arquivo que o link havia sido encontrado, o google_maps_api.xml em sua aplicação no Android Studio. Com os campos preenchidos, clique em "Criar" para receber sua chave de APIs do Android. Ela ficará listada como uma das credenciais de seu projeto, como visto na **Figura 4**.

Figura 04 - Credenciais criadas para o projeto criado no Google Developers Console.



Após ter acesso à sua chave, copie-a e substitua, no arquivo google_maps_api.xml do seu projeto no Android Studio, a string lá contida. A **Listagem 2** mostra como deve ficar o código do seu arquivo após a adição da chave.

```
<resources>
<string name="google_maps_key" translatable="false" templateMergeStrategy="preserve">
AIzaSyBzLI2iPWPQbo6MpAxvaq8hCdq
</string>
</resources>
```

Listagem 2 – Código do arquivo google_maps_api.xml com a chave substituída de maneira adequada.

Com a chave substituída adequadamente no arquivo google_maps_api.xml, podemos então testar a nossa aplicação, uma vez que a Activity inicial que adicionamos ao nosso projeto já é a Activity de mapas, sugerida pelo Android como template.

Configurando o Emulador

Quando vamos utilizar mapas em nossa aplicação, também precisamos modificar os dispositivos virtuais envolvidos. É necessário ter um emulador configurado com o alvo para o Google APIs e em uma versão igual ou superior a API 17. Para fazermos isso, devemos acessar o AVD Manager, pelo Android Studio, e então criar um AVD novo ou editar um antigo para que tenha como alvo o Google APIs da versão atual.

Uma vez configurado o emulador para ter como alvo o Google APIs apropriado para sua versão de aplicação, basta executá-lo normalmente. A partir desse momento, é possível utilizar aplicações que incluam acesso aos mapas sem

problemas. Lembre-se também de abrir a tela de Emulator Control, da **Figura 1**, para simular corretamente as localizações, caso seja necessário.

Referenciando a Utilização dos Mapas no AndroidManifest

Para utilizar a biblioteca externa de mapas em sua aplicação, é necessário declarar isso no Android Manifest. Caso você tenha criado o projeto utilizando o template de Map Activity, o Android Studio já se encarrega de fazer essa inclusão para você. A **Listagem 3** mostra a linha que deve ser adicionada ao Manifest, dentro da tag de aplicação.

```
1 <meta-data
2 android:name="com.google.android.geo.API_KEY"
3 android:value="@string/google_maps_key" />
```

Listagem 3 – Indicando a utilização da biblioteca de mapas no Manifest

Com isso, temos um mapa funcional, capaz de se movimentar e trabalhar com zoom. Além disso, agora já sabemos como podemos criar camadas sobre o mapa para que possamos indicar pontos ou até mesmo criar um marcador, como mostrado na Activity padrão do template criado. Assim, encerramos a nossa aula de número oito!

Atividade 02

---

1. Crie uma Activity contendo um mapa, obtenha uma chave válida para que o mapa possa ser exibido corretamente e então exiba-o de um modo que seja possível movimentá-lo e lidar com o zoom.

2. Crie um ícone no mapa, na posição atual do usuário, obtida através da solução da Atividade 1.

Leitura Complementar

LOCATION and Maps. Disponível em: <http://developer.android.com/guide/topics/location/index.html>. Acesso em: 12 fev. 2016.

GOOGLE MAPS APIs. Disponível em: <https://developers.google.com/maps/documentation/android-api/start>. Acesso em: 12 fev. 2016.

Resumo

Nesta aula, aprendemos a lidar com a localização do usuário. Estudamos a melhor maneira de obtê-la tendo em vista o alto custo energético dessa funcionalidade. Além disso, aprendemos técnicas de aceleração do processo e como escolher o melhor dos métodos que o Android disponibiliza aos desenvolvedores. Na segunda parte da aula, aprendemos a lidar com mapas. Vimos que o processo de criação de Activities com Mapas foi muito facilitado pelo Android Studio, necessitando apenas algum esforço para conseguir a chave para acessar a API. Vimos também que o Emulador deve ser reconfigurado para executar aplicações com mapas e como simular localizações nele. Sem dúvidas, uma aula muito interessante! Até a próxima!

Autoavaliação

1. Quais são os dois principais métodos para se obter a localização do usuário no Android? Quando cada um deve ser utilizado?
2. Que declarações devem existir no AndroidManifest.xml para que uma aplicação que desenha em um mapa as N últimas posições de um usuário possa ser executada?

Referências

ANDROID Developers. 2015. Disponível em: <https://developer.android.com/>. Acesso em: 20 mai. 2015.

DIMARZIO, J. **Android**: A Programmer's Guide. New York:McGraw-Hill, 2008. ISBN 9780071599887. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 3 ago. 2012.

LECHETA, Ricardo R. **Google android**: aprenda a criar aplicaçoes. 2. ed. São Paulo: Novatec, 2010.

LECHETA, Ricardo R. **Google android para tablets**. São Paulo: Novatec, 2012.

MEIER, R. **Professional Android 2 Application Development**. New Jersey: John Wiley & Sons, 2010. ISBN 9780470874516. Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC>. Acesso em: 3 ago. 2012.