

# Google Maps em aplicação

## Integrando o Google Maps com aplicativo

O Google Maps é um serviço de visualização de mapas e imagens disponibilizado pela Google no qual podemos navegar por diversas partes do mundo por meio da internet de forma interativa, rápida e eficaz. Por ser um serviço gratuito, é possível utilizá-lo em nossas aplicações através da Google Maps API, que é responsável por realizar a integração com aplicações Web.

E quanto à utilização do Google Maps em aplicações para dispositivos móveis? Veremos que integrar o serviço Google Maps

com a tecnologia Java ME é tão simples quanto integrá-lo em outras aplicações.

Este é um cenário de uso muito interessante devido à mobilidade dos dispositivos Java ME, o que cria oportunidades para diversos tipos de aplicação.

### Um pouco do Google Maps

O Google Maps foi lançado e disponibilizado gratuitamente em 2005 aos usuários da Internet para acesso a mapas de países, estados, cidades, bairros, avenidas e ruas. Tudo através de um site contendo uma interface simples, capaz de realizar buscas precisas com um ótimo desempenho. O

usuário pode solicitar a visualização de vários tipos de mapas, como: mapas normais, que retornam um diagrama das ruas de uma cidade (**Figura 1**); mapas de satélite, que retornam imagens de satélite de uma determinada região do globo (**Figura 2**); mapas híbridos, que retornam uma combinação de mapas normais com mapas de satélite (**Figura 3**) e os mapas de terreno que exibe o mapa de ruas enfatizando o relevo do

## Resumo DevMan

### De que se trata o artigo:

O artigo aborda a construção de um aplicativo capaz de realizar buscas e visualizações de mapas de países, cidades, ruas entre outros, através da integração do Google Maps com nossas aplicações móveis utilizando a tecnologia J2ME.

### Para que serve:

Fornecer aos desenvolvedores uma maneira simples de construir aplicações capazes de utilizar os serviços de busca de mapas disponibilizados gratuitamente pela Google Maps.

### Em que situação o tema é útil:

Utilizando o Google Maps API podemos construir uma aplicação capaz de realizar buscas por endereços, mostrando o mapa de como localizar, por exemplo, o endereço de uma empresa, de um hotel ou hospital em qualquer lugar do mundo.

### Google Maps em aplicações móveis:

A integração do Google Maps com nossas aplicações móveis é realizada através de uma conexão ao Google Static Maps API. Para isto, solicitamos via HTTP a geração de uma imagem de mapa estática que representa uma determinada localidade. Observe a **Listagem 1** onde chamamos o método `getGeocode()`, presente na **Listagem 2**, para buscar os geocódigos do local por serem necessários na requisição de uma imagem. Logo após é chamado o método `getImage()`, para criação e exibição da imagem na tela do dispositivo. O tratamento de movimentação de mapa são realizados pelos métodos `movPositivo()` e `movNegativo()`. Desta forma, conseguimos construir uma aplicação capaz de exibir mapas de cidades com a possibilidade de navegar de forma intuitiva por todo o mapa que é exibido na tela do dispositivo.

### Nota do Devman

Veja o que você vai aprender adicionalmente neste artigo:

- Como obter a chave para utilizar os serviços da Google Maps API;
- O que são os geocódigos;
- Entendendo a conexão com o servidor;
- O conceito de Markers.



# ões móveis

## s Java ME

Aprenda como criar uma aplicação de solicitação de mapas utilizando o serviço Google Maps

JOSIAS PAES

local (Figura 4). Também existem outros projetos da Google no qual podemos observar imagens do terreno e relevo da Lua e do planeta Marte (Figura 5).

A Google Maps API foi lançada pouco tempo depois. Consiste basicamente de um conjunto de classes JavaScript que proporciona aos desenvolvedores acesso aos serviços disponibilizados pelo Google Maps. Através dela podemos construir aplicativos que realizem consultas por endereços e utilizem funções de *zoom* e de movimentação de mapa.

Para solicitar um mapa de uma determinada localidade, o usuário deve apenas inserir o local desejado para que o serviço retorne o mapa requerido.

Observe a Figura 6. Ela representa em alto nível o fluxo de uma pesquisa no Google Maps:

1. O usuário solicita a busca do mapa de uma cidade;

2. O serviço recupera as informações prestadas pelo usuário e a partir delas é gerada a resposta;
3. O serviço retorna ao usuário o mapa do local solicitado.

### Integrando

Em uma aplicação Web que utilize os serviços prestados pela API, faz-se necessário realizar uma conexão ao servidor. Esta conexão é feita através de uma solicitação HTTP via URL <http://www.google.com/jsapi?key=suachave>. Perceba que no fim da URL de conexão deve ser passado o parâmetro chamado *key*, no qual deve receber a chave de utilização ou API-Chave da Google Maps API, como pode ser observado no código a seguir.

```
<script type="text/javascript" src="http://www.google.com/jsapi?key=SUACHAVE"></script>  
<script type="text/javascript"></script>
```



### Nota do Devman

**API-Chave:** Para que seja possível realizar pesquisas de mapas utilizando a API faz-se necessário obter uma chave de utilização. Esta chave é única para cada usuário e é restritiva quanto ao seu uso. Consiste de uma string que deve ser utilizada sempre que se incluir a API em suas aplicações. Para obter a chave de utilização acesse o site oficial do Google Maps API (<http://code.google.com/apis/maps/>) e clique em *Sign up for a Google Maps API key*.

Não seria diferente a maneira de obter os serviços da API para serem utilizados em aplicações móveis. Ou seja, também é necessário realizar uma conexão ao servidor para obter acesso aos serviços. Entretanto, é necessário considerar que a Google Maps API requer um ambiente Ja-



Figura 1. Imagem de um mapa normal ou roadmap.



Figura 3. Imagem de um mapa híbrido.



Figura 2. Imagem de um mapa de satélite.

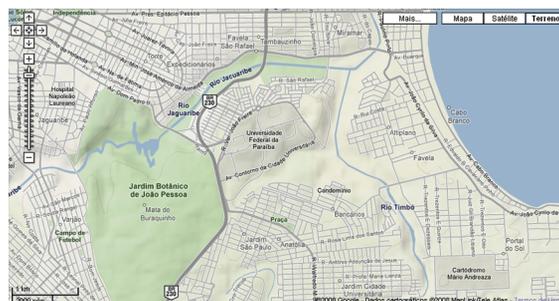


Figura 4. Imagem de um mapa de relevo.

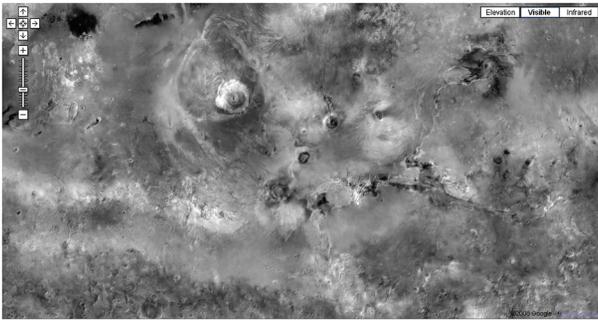


Figura 5. Imagem de satélite do planeta Marte.

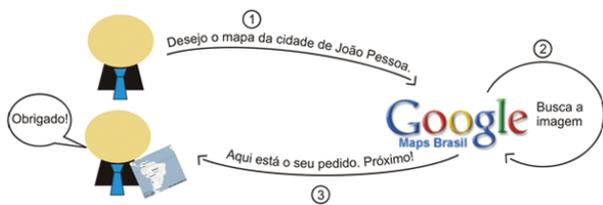


Figura 6. Esquema de uma solicitação de um mapa ao Google Maps.

vaScript completo para podermos utilizar as classes e métodos oferecidos. Por este motivo, não é recomendado o uso desta API na construção de aplicações para dispositivos móveis.

✔ *Mesmo nos dispositivos mais recentes/avançados que contam com browsers completos, incluindo suporte a JavaScript, o runtime de JavaScript é tipicamente "amarrado" ao browser, sendo difícil (e na melhor hipótese, não-portável) acioná-lo a partir de uma aplicação não-web, seja Java ME ou nativa. Já na plataforma Java SE 6, seria simples utilizar a API **javax.script** e o Rhino (interpretador de JavaScript escrito em Java).*

Então, como iremos integrar o Google Maps com as nossas aplicações? Para isso, foi disponibilizada a Google Static Maps API, que, ao receber uma solicitação HTTP via URL retorna imagens estáticas (GIF, JPG e PNG) que podem ser facilmente manipuladas em aplicações móveis. Ao integrarmos nossa aplicação com esta API não será utilizada nenhuma classe ou método para obtenção ou manipulação de imagens de mapas, ao invés disso, apenas realizará solicitações HTTP para receber como resposta uma imagem em que podemos tratar e então exibi-la na tela do dispositivo.

Parâmetros	Descrição	Requisito
center	Define o centro do mapa através dos geocódigos.	Necessário na ausência de markers
zoom	Define o nível de zoom do mapa ou nível de ampliação do mapa.	Necessário na ausência de markers
size	Define as dimensões retangulares da imagem do mapa.	Obrigatório
format	Define o formato da imagem resultante.	Opcional (default = GIF)
maptype	Define o modelo de mapa a construir.	Opcional (default = roadmap)
markers	Define o conjunto de um ou mais marcadores que sevem para marcar um determinado local, possibilitando a inserção de imagens no mapa.	Opcional
path	Define o caminho entre dois ou mais pontos.	Opcional
key	Identifica a API-chave para o domínio que realizou a solicitação.	Obrigatório

Tabela 1. Os parâmetros de URL mais importantes.



## Nota do Devman

**Conexão ao servidor:** O método `conexaoHttp()` é chamado várias vezes durante a execução do aplicativo. Note que as solicitações dos geocódigos e de imagens ao servidor são realizadas por URLs diferentes. Uma delas é para obtermos os geocódigos de uma cidade. Os atributos que devem estar presentes para esta solicitação são: o atributo **q** que recebe o endereço da cidade na qual queremos obter os geocódigos e a **key** que recebe sua chave de acesso aos serviços.

`http://maps.google.com/maps/geo?q=local&key=suachave`

A outra URL é utilizada para obter as imagens de mapas. Os atributos que devem estar presentes são os atributos presentes na Tabela 1 e a **key** que recebe sua chave de acesso aos serviços.

`http://maps.google.com/staticmap?parametros&key=suachave`



## Nota do Devman

**Geocódigos:** Os Geocódigos são simplesmente a latitude e longitude de uma determinada localidade, utilizados para mapear pontos geográficos para geração da imagem do mapa.

Para cada pedido realizado ao servidor, é possível especificar a localização do mapa, o tamanho da imagem, o nível de *zoom*, o tipo de mapa e vários outros atributos para geração da imagem. Cada um dos parâmetros contidos na Tabela 1 pode ser adicionado na URL de conexão para solicitação de uma imagem apropriada ao tipo de aplicação que será construída. Através da URL a seguir é solicitada uma imagem com o centro definido na cidade de João Pessoa (utilizando os geocódigos da cidade definidos no atributo **center**), seu formato será do tipo BMP, o zoom no mapa será de nível 8, seu tamanho de 225x250 pixels e a imagem do mapa será do tipo satélite.

`http://maps.google.com/staticmap?center=-7.11532,-34.861051&format=bmp&zoom=8&size=225x250&maptype=satellite&key=SUAKEY;`

Tendo isto, vamos iniciar a construção da nossa aplicação.

### Utilizando o serviço em nossa aplicação

Usaremos como exemplo de aplicação um sistema capaz de localizar determinada cidade, no qual o usuário poderá selecionar o tipo de mapa que deseja visualizar e utilizar as funções de *zoom* e movimentação de mapa.

Veja o Quadro "Tecnologias utilizadas" para saber como montar seu ambiente de desenvolvimento para execução do aplicativo.

### Listagem 1. Código da classe MidMap.java.

```
package com.maps;
(...)
public class MidMap extends MIDlet implements ActionListener{
    private Display display;
    private fmMain, fmMapa;
    (...)
    private GoogleMaps gMap;
    private int zoom = 8;
    private double lon, lat;
    private String tipoMapa;
    private String msg = "Insira aqui o tutorial";

    public MidMap() {
        display.init(this);
        // Criação e inicialização dos componentes do LWUIT
        //utilizados
        fmMap = new Form("Mapa"){
            public void keyPressed(int keyCode) {
                int game = Display.getInstance().
                    getGameAction(keyCode);
                double sub = 0.5d;
                switch (game) {
                    case Display.GAME_FIRE:
                        if(++zoom<=17)
                            getImagem(lon,lat,zoom,tipoMapa);
                        else
                            --zoom;
                        break;
                    case Display.GAME_UP:
                        getImagem(lon,lat=movPositivo(zoom,lat),zoom,
                            tipoMapa);
                        break;
                    case Display.GAME_DOWN:
                        getImagem(lon,lat=movNegativo(zoom,lat),zoom,
                            tipoMapa);
                        break;
                    case Display.GAME_LEFT:
                        getImagem(lon=movNegativo(zoom,lon),lat,zoom,
                            tipoMapa);
                        break;
                    case Display.GAME_RIGHT:
                        getImagem(lon=movPositivo(zoom,lon),lat,zoom,
                            tipoMapa);
                        break;
                }
                super.keyPressed(keyCode);
            }
        };
        fmMap.setCommandListener(this);
    }

    protected void startApp() throws
        MIDletStateChangeException {
        fmMain.show();
    }
    (...)
    public double movPositivo(int zoom, double coord){
        double dif1 = 2.0d;
        double dif2 = 0.5d;
        if(zoom>0 && zoom<=4){
            coord+=dif1;
        }else if(zoom>=5 && zoom<=8){
            coord+=dif1/3;
        }
        (...)
        }else{
            coord+=dif2/248;
        }
        return coord;
    }

    public double movNegativo(int zoom, double coord){
        double dif1 = 2.0d;
        double dif2 = 0.5d;
        if(zoom>0 && zoom<=4){
            coord-=dif1;
        }else if(zoom>=5 && zoom<=8){
            coord-=dif1/3;
        }
        (...)
        }else{
            coord-=dif2/270;
        }
        return coord;
    }

    public void getImagem(double longit, double latit, int
        zoom, String tipo){
        Image map=null;
        try {
            if(tipo.equals("Normal")){
                map = gMap.getImagemEstatica(225, 250, longit,
                    latit, zoom, "bmp", "roadmap");
            }else if(tipo.equals("Satelite")){
                map = gMap.getImagemEstatica(225, 250, longit,
                    latit, zoom, "bmp", "satellite");
            }else if(tipo.equals("Hibrido")){
                map = gMap.getImagemEstatica(225, 250, longit,
                    latit, zoom, "bmp", "hybrid");
            }
            // Tratamento de inserção e exibição da imagem no
            formulário
        }catch (Exception e) {
        }
    }

    public void ajuda(String msg){
        Dialog d1Info = new Dialog("Comandos");
        // Tratamento do componente Dialog
        d1Info.show();
    }

    public void actionPerformed(ActionEvent ae) {
        if(ae.getCommand()==cmPesquisar){
            gMap= new GoogleMaps("SUA CHAVE");
            try {
                if(tfCidade.getText().equals("") &&
                    tfPais.getText().equals("")){
                    ajuda("Digite o nome da cidade ou país que
                        deseja pesquisar");
                }else{
                    double[] geocodes = gMap.getGeocode(tfCidade.
                        getText()+"", "+tfPais.getText()");
                    lat = geocodes[0];
                    lon = geocodes[1];
                    tipoMapa = (String) cbTipo.getSelectedItem();
                    getImagem(lon,lat,zoom,tipoMapa);
                    ajuda(msg);
                    fmMap.show();
                }
            }catch (Exception e1) {
            }
        }else if (ae.getCommand()==cmOut) {
            getImagem(lon,lat,--zoom,tipoMapa);
        }else if (ae.getCommand()==cmVoltar){
            zoom=8;
            fmMain.show();
        }else if(ae.getCommand()==cmSair){
            notifyDestroyed();
        }
    }
}
```

**Listagem 2.** Código da classe `GoogleMaps.java`.

```
package com.maps;
(...)
public class GoogleMaps{
    private String apiKey = null;

    public GoogleMaps(String apiKey){
        this.apiKey = apiKey;
    }
    (...)
    private static double[] split(String s){
        Vector aux = new Vector();
        int index = 0;
        int prox = 0;
        while((index = s.indexOf('.', prox)) >= 0){
            aux.addElement(s.substring(prox, index));
            prox = index + 1;
        }
        aux.addElement(s.substring(prox));
        double[] data = new double[aux.size()];
        for (int i = 0; i < aux.size(); i++){
            data[i] = Double.parseDouble((String) aux.elementAt(i));
        }
        return data;
    }

    public double[] getGeocode(String address) throws Exception{
        byte[] res = conexaoHttp(getUrl(address));
        String resString = new String(res);
        double[] data = split(resString);
        if(data[0]!=200){
            int errorCode = (int) data[0];
            throw new Exception("Google Maps Exception: " +
                getGeocodeError(errorCode));
        }else{
            return new double[]{data[2], data[3]};
        }
    }

    public Image getImagemEstatica(int width, int height, double lon, double lat,
        int zoom, String format, String tipo) throws Exception{
        byte[] imageData = conexaoHttp(getMapa(width, height, lon, lat, zoom,
            format, tipo));
        return Image.createImage(imageData, 0, imageData.length);
    }

    public String getGeocodeError(int errorCode){
        switch(errorCode){
            case 400:
                return "Solicitação inválida";
            case 500:
                return "Erro de servidor";
            (...)
            default:
                return "Erro genérico";
        }
    }

    public byte[] conexaoHttp(String url) throws Exception{
        HttpURLConnection hc = null;
        InputStream is = null;
        byte[] byteBuffer = null;
        try{
            hc = (HttpURLConnection) Connector.open(url);
            hc.setRequestMethod(HttpURLConnection.GET);
            int ch;
            is = hc.openInputStream();
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            while ((ch = is.read()) != -1)
                bos.write(ch);
            byteBuffer = bos.toByteArray();
            bos.close();
        }catch(Exception e){
        }
        finally{
            try{
                if(is != null)
                    is.close();
                if(hc != null)
                    hc.close();
            }catch(Exception e2){
            }
        }
        return byteBuffer;
    }
}
```

## Tecnologias utilizadas

Utilizaremos a tecnologia Java ME em nossa aplicação. Para isso, é necessário realizar o *download* do *WirelessToolkit* da Sun em <http://java.sun.com/products/sjwtoolkit/> e instalá-lo. Note que alguns IDEs ou (plug-ins de Java ME para IDE) podem já conter o WTK; é o caso, por exemplo, do NetBeans.

Outra tecnologia que vamos utilizar é a API LWUIT. O *download* da API pode ser realizado em <https://lwuit.dev.java.net/>.

A IDE de desenvolvimento Java fica a critério do desenvolvedor. Independente da IDE utilizada, siga os seguintes passos:

- Crie um novo projeto de MIDlet Java ME;
- Adicione ao *path* do projeto o arquivo *lwuit.jar*;
- Importe as classes **MidMap (Listagem 1)** e **GoogleMaps (Listagem 2)**.

Esta aplicação foi testada em uma máquina com Windows XP Professional SP3, Eclipse 3.3.1, EclipseME, WTK 2.5.2 e LWUIT. Os arquivos estão disponíveis na página de download da revista.

Inicialmente construímos a interface gráfica da nossa aplicação utilizando a API LWUIT. Esta API proporciona a nós desenvolvedores a possibilidade de criar GUIs arrojadas para potencializar nossas aplicações. (Para mais informações sobre a LWUIT, ver "LWUIT: 'Swing' para Java ME", Edição 60). Então, ao executar a classe **MidMap (Listagem 1)**, todos os componentes visuais serão instanciados e exibidos na tela do dispositivo, como pode ser observado na **Figura 7**. Os componentes utilizados são:

- Formulários (**Form**) – espécie de *container* para inserção de outros componentes;
- Rótulos (**Label**) – utilizados para exibir informações ao usuário;
- Campos de texto (**TextField**) – utilizados para receber informações inseridas por um usuário;
- Caixas de combinação (**ComboBox**) – utilizados para combinar muitas informações em um pequeno componente onde o usuário pode escolher qual irá utilizar.
- Caixas de diálogo (**Dialog**) – componente de alerta para prestar informações ao usuário.



Figura 7. Iniciando a aplicação.



Figura 8. Inserindo uma nova pesquisa.



Figura 9. Mostrando os comandos de utilização do aplicativo.



Figura 10. Resultado da pesquisa.



Figura 11. Utilizando o atributo path para traçar rotas.

Com o aplicativo em execução, o usuário pode inserir os dados da cidade que deseja pesquisar, selecionar o tipo de mapa que deseja visualizar e então clicar no comando *Pesquisar* (Figura 8). O método *ajuda()* (Listagem 1) será chamado para instruir o usuário sobre a utilização das funções oferecidas pelo aplicativo (observe a Figura 9).

Para realizar a solicitação de uma imagem, faz-se necessário obter os *geocódigos* do local. Para isto, o método *getGeocode()* da classe *GoogleMaps* (Listagem 2) é chamado. Note que neste método são realizadas chamadas de alguns métodos auxiliares, sendo um deles o *conexaoHttp()*, que é o responsável por enviar solicitações via HTTP ao servidor; nesta chamada, estão sendo requeridos os códigos geográficos do local especificado. Também são chamados os métodos *split()* para buscar os valores dos geocódigos contidos na resposta dada pelo servidor (nesta resposta também estão contidos o código do serviço e outras informações. Por conta disto, faz-se necessário tratar a resposta e buscar apenas os valores dos geocódigos) e o *getGeocodeError()* para tratar os possíveis códigos de erro que podem ser retornados na solicitação dos geocódigos.

Logo após é chamado o método *getImagem()* (Listagem 1). Este utiliza outros métodos da classe *GoogleMaps* (Listagem 2)

responsáveis pela solicitação e geração da imagem. Como as imagens são criadas baseadas no tipo de mapa que o usuário escolheu, na invocação de *getImagemEstatica()* é passada a solicitação de criação de uma imagem de acordo com o tipo de mapa informado pelo usuário. Ainda em *getImagem()*, adicionamos a imagem criada em um *container* (componente da API LWUIT) para exibição na tela do dispositivo, como pode ser observado na Figura 10.

Resumindo, devemos obter os geocódigos do local e utilizá-los como argumentos na chamada do método de criação de imagens (*getImagem()*).

### Adicionando as funcionalidades do aplicativo

O método *getImagemEstatica()* da classe *GoogleMaps* (Listagem 2), recebe como argumentos os atributos de configuração para requisição de uma imagem (apresentados na Tabela 1). Será através destes argumentos, especificamente os geocódigos e o *zoom*, que iremos adicionar ao aplicativo as funcionalidades de *zoom* e movimentação de mapa.

```
public Image getImagemEstatica(int width,
    int height,
    double lng, double lat, int zoom,
    String format){
    // cria e retorna uma imagem
}
```

Para cada movimentação ou uso do *zoom* é gerada uma nova imagem. Cada

imagem gerada é inserida no componente *fmMapa* que é um formulário pertencente à API LWUIT. Este componente, bem como vários outros da API, oferecem o método *keyPressed()* (Listagem 1) para tratamento de eventos gerados ao pressionar uma tecla do dispositivo. Neste método, tratamos os eventos da seguinte maneira:

- **Teclas direcionais** – usadas para movimentar o mapa para cima, para baixo, esquerda e direita;
- **Tecla direcional central ou comandos de menu** – usadas para efetuar *zoom in* ou *zoom out* na imagem.

Os métodos *movPositivo()* e *movNegativo()* (Listagem 1) retornam um valor que representa o novo ponto de movimento no mapa. Para isto, são realizados cálculos para o movimento do mapa de acordo com o nível de *zoom* definido. Os valores retornados são passados ao método *getImagem()* para gerar a nova imagem.

```
getImagem(lon, lat = movPositivo(zoom, lat),
    zoom, tipoMapa);
getImagem(lon = movNegativo(zoom, lon), lat,
    zoom, tipoMapa);
```

Ao finalizar estas definições, temos uma aplicação capaz de realizar buscas de mapas de uma determinada localidade com a possibilidade de interagir de forma intuitiva com as imagens geradas de acordo com as limitações dos dispositivos móveis.

### Listagem 3. Método para criação da URL com o atributo path.

```
public String getMapaRota(int width, int height, double lng, double lat, int zoom,
    String format, String type){
    String path = "rgb: 0xfffff, weight: 5 | -7.11532, -34.861051 | -7.11353, -34.859265";
    return "http://maps.google.com/staticmap?center=" + lat + "," + lng + "&format=" +
        format + "&zoom=" + zoom + "&size=" + width + "x" + height + "&maptypes=" + type + "&path=" + path + "&key=" +
        apiKey;
}
```



## Nota do Devman

**Markers:** Atributo utilizado em uma URL para definir o conjunto de um ou mais marcadores para apontar no mapa um determinado local. Nos marcadores podemos adicionar imagens e textos sobre o local marcado.

## Desafio

Com o Google Static Maps API também é possível criar uma aplicação que seja capaz de traçar rotas entre dois ou mais pontos do mapa definidos pelo usuário, ou seja, o usuário pode criar suas próprias rotas de destino, como pode ser observado na **Figura 11**.

Para adicionar esta função ao seu aplicativo é bem simples, basta lembrar que a requisição de imagens através do Google Static Maps API sempre é realizada através de uma requisição HTTP via URL. Para isto, é preciso montar uma URL que contenha o atributo responsável por criar um caminho ou rota a partir de pontos no mapa, este atributo é o **path** (**Tabela 1**). Como o tracejado da rota é realizado a partir de pontos específicos do mapa, é preciso ter um pouco de cuidado com a inserção destes pontos, já que a ligação entre eles é feita pela ordem em que são especificados. Para utilizar o **path** é necessário conhecer os seus atributos, que são:

- **rgb** – especifica um valor de cor hexadecimal (0xffffff) de 24 bits para o tracejado da rota com uma transparência fixa de 50%;
- **rgba** – especifica um valor de cor hexadecimal (0xffffffff) de 32 bits para o tracejado da rota. Ao fim do valor são acrescentados mais dois caracteres para especificar o nível de transparência do tracejado da rota;
- **weight** – especifica a espessura da linha traçada entre os pontos.

O atributo **path** deve ser do tipo String, onde, primeiramente deve ser definida

a cor e a espessura da rota, logo após os pontos do mapa devem ser inseridos e cada um deles deve ser separado pelo caractere *pipe* (`|`). Ao fim, adicione o atributo **path** na URL junto com todos os outros atributos necessários (observe a **Listagem 3**).

Tendo isto, crie uma funcionalidade capaz de traçar rotas cada vez que o usuário utilizar a função de movimento, mostrando assim, todo o caminho que ele percorreu no mapa. Observe o método **keyPressed()** (**Listagem 1**) e perceba que a cada movimento de mapa são gerados novos valores através das chamadas aos métodos **movPositivo()** e **movNegativo()** para geração da nova imagem. Utilize estes novos valores como sendo os pontos do mapa para criar as rotas.

## Conclusões

Neste artigo, aprendemos como integrar o Google Maps em aplicações para dispositivos móveis. Foi apresentada uma pequena comparação para mostrar as diferenças de utilização do Google Maps em aplicações Web com aplicações móveis. Nesta comparação podemos concluir que a principal diferença entre elas é que, em aplicações móveis para solicitações de imagens de mapas, não são utilizadas nenhuma das classes ou métodos que devem estar presentes na integração do Google Maps com aplicações Web.

Também foi construído um exemplo de aplicação com o objetivo de apresentar como é realizada a integração da API para solicitação de imagens de mapas. Em todo o exemplo foram construídas duas classes: **MidMap** (**Listagem 1**) que tem como responsabilidade construir toda a interface gráfica de utilização para o usuário, receber os dados para solicitações de imagens e tratar as funções que o aplicativo incorpora; e **GoogleMaps** (**Listagem 2**) responsável por realizar as conexões com o servidor, tratar as respostas e gerar as imagens solicitadas pelo usuário.

Nossa aplicação construída é bem simples, mas nela ainda é possível adicionar outras funções como, por exemplo, a criação de **markers** (**Tabela 1**) em determinados locais do mapa e inserir imagens para esses locais, tornando assim, uma aplicação mais poderosa e personalizada.

<http://code.google.com/apis/maps/documentation/index.html>

Google Maps API.

<http://code.google.com/apis/maps/documentation/staticmaps/>

Google Static Maps API.

[http://wiki.forum.nokia.com/index.php/J2ME\\_Google\\_Maps\\_API#Get\\_your\\_own\\_Google\\_Maps\\_API\\_Key](http://wiki.forum.nokia.com/index.php/J2ME_Google_Maps_API#Get_your_own_Google_Maps_API_Key)

Forum Nokia.

<http://googlegeodevelopers.blogspot.com/>

Blog Oficial do Google Maps API.

<https://lwuit.dev.java.net/>

LWUIT.



**Josias Paes**

[josiaspaesjr@gmail.com](mailto:josiaspaesjr@gmail.com)

possui grau de Bacharelado em Ciências da Computação pelo Centro Universitário de João Pessoa.

Tem experiência na área de desenvolvimento com ênfase para dispositivos móveis. cursando atualmente especialização no Desenvolvimento de Dispositivos Móveis.

## Dê seu feedback sobre esta edição!

A Java Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

[www.devmedia.com.br/javamagazine/feedback](http://www.devmedia.com.br/javamagazine/feedback)



## A revista do desenvolvedor Web e Wireless

Acesse já! [www.devmedia.com.br/webmobile/pagina.asp](http://www.devmedia.com.br/webmobile/pagina.asp)