	FTCE Faculdade de Tecnologia e Ciências Exatas

USJT - 2019 – Programação para Web e Dispositivos Móveis

Professores: Anderson e Bossini

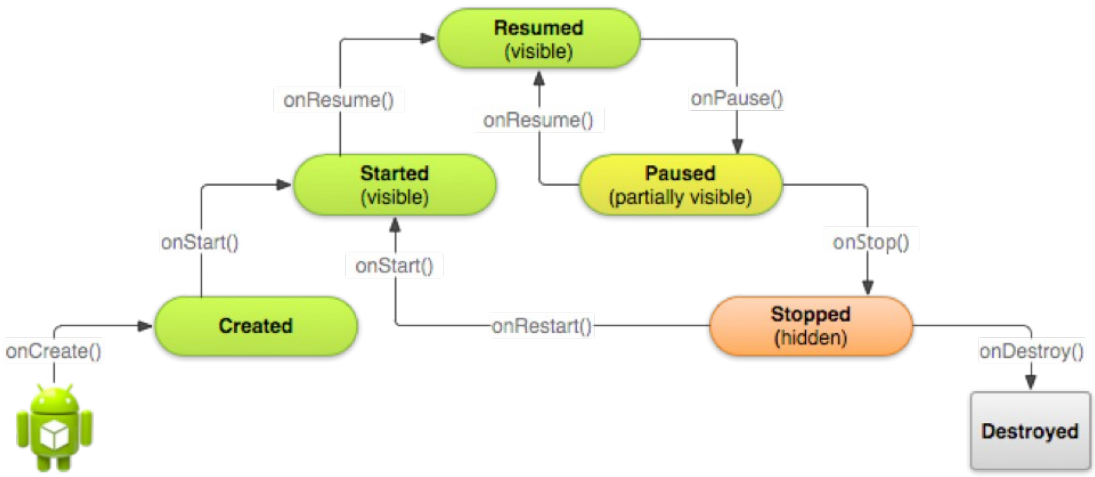
Aula: 02
Assunto: Activity - Ciclo de Vida

Conceitos básicos

Introdução

1. Ao contrário das aplicações Java Standard, as aplicações Android não começam em um método main.
2. Elas são divididas em Activities que são ativadas, carregadas, pausadas, desativadas, etc (ciclo de vida) por meio de métodos de callback, isto é, métodos chamados indiretamente, pois é o sistema operacional que os chama.

Callbacks do Ciclo de Vida



- os callbacks são como uma pirâmide
- o topo é o app rodando e o usuário interagindo com ele
- conforme vai descendo de novo o app vai sendo desligado
- você não precisa implementar todos os métodos, mas é preciso fazer com que a aplicação reaja corretamente se:
  - o usuário recebe uma ligação (o app não capote)
  - não consuma recursos valiosos se não estiver em uso (pausar um vídeo se a aplicação for para background)
  - não perca o progresso do usuário se ele sair e voltar depois

- não capote quando a tela gira (muda a orientação de retrato para paisagem, por exemplo)

Os estados duráveis, isto é, nos quais a aplicação pode ficar por um período grande de tempo são:
- Resumed: a activity está em foreground e pode ser usada
- Paused: está parcialmente encoberta por outra atividade (transparente ou que não usa a tela toda); a activity não recebe nenhum input e não pode executar nenhum código
- Stopped: a atividade não está visível pelo usuário; o estado e as variáveis são retidos, mas ela não pode executar nenhum código
- Os outros estados são temporários; quando a aplicação começa é chamado o onCreate() que rapidamente chama o onStart() que rapidamente chama o onResume()

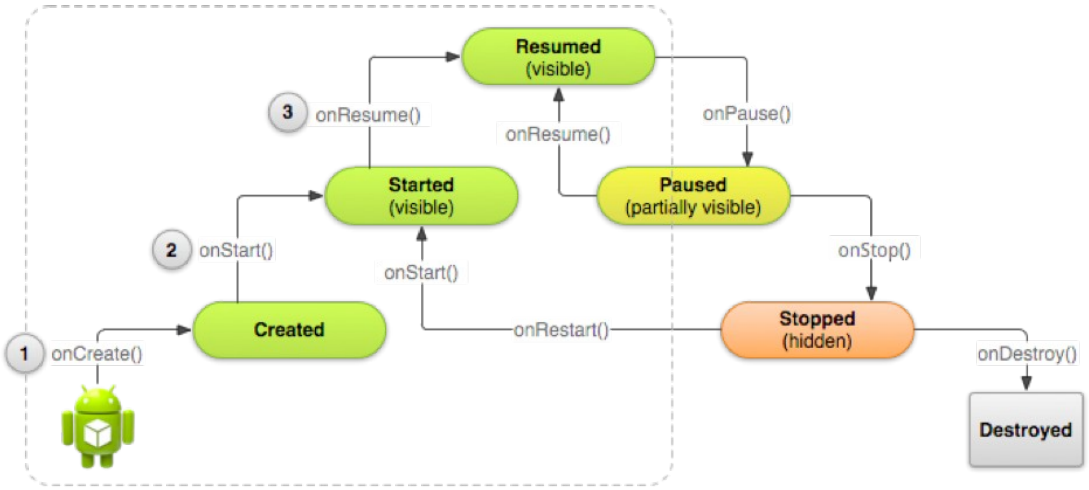
- Você especifica a atividade principal do seu app no AndroidManifest.xml
- Quando o usuário toca no ícone do seu app, esta será a Activity que terá seu método onCreate() chamado.
- Esta activity é o “main”

```
<activity android:name=".MainActivity"
          android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Criando uma nova instância

```
TextView mTextView; // Member variable for text view in the layout

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Set the user interface layout for this Activity
    // The layout file is defined in the project
res/layout/main_activity.xml file
    setContentView(R.layout.main_activity);

    // Initialize member TextView so we can manipulate it later
    mTextView = (TextView) findViewById(R.id.text_message);

    // Make sure we're running on Honeycomb or higher to use ActionBar APIs
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
        // For the main activity, make sure the app icon in the action bar
        // does not behave as a button
        ActionBar actionBar = getActionBar();
        actionBar.setHomeButtonEnabled(false);
    }
}
```

- O onCreate() chama o onStart()
- Tecnicamente a aplicação estará visível no onStart(), mas rapidamente ele já chama o onResume()
- A aplicação fica em Resumed até que alguma coisa aconteça.



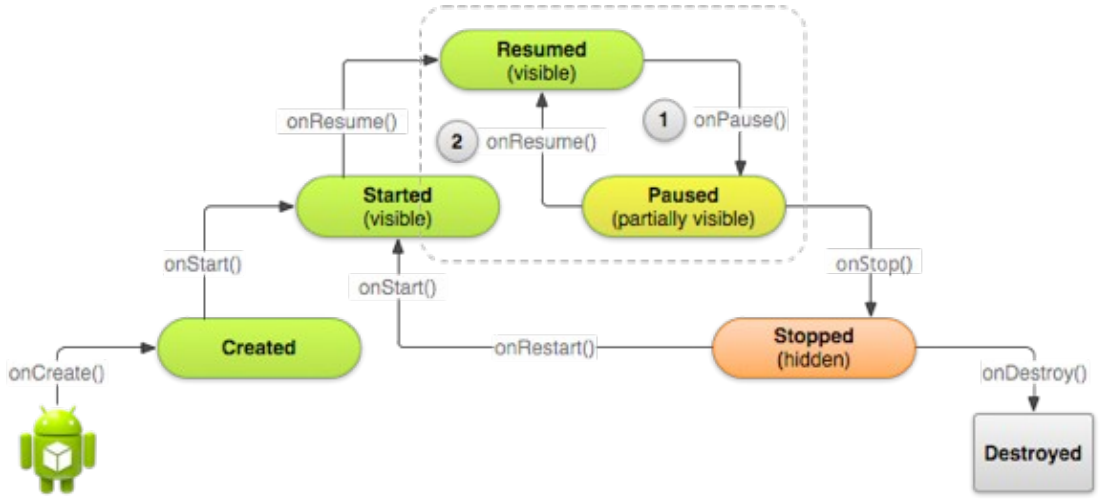
Destruindo uma activity

- O sistema chama este método antes de ser completamente removido da memória.
- Se o sistema tem threads em background esta é a hora de dar kill nelas ou vão virar memory leak.
- O sistema sempre chama o onPause() e o onStop() antes de chegar no onDestroy(), a não ser que você chama finish() no onCreate(); neste caso ele vai para o onDestroy() direto.

```
@Override
public void onDestroy() {
    super.onDestroy();  // Always call the superclass

    // Stop method tracing that the activity started during onCreate()
    android.os.Debug.stopMethodTracing();
```

Pause e Resume

- Se seu app for parcialmente obstruído por outro, o sistema chama onPause().
- Então você pode parar alguma coisa, como um vídeo, ou persistir informações que serão perdidas se o usuário sair do sistema.
- Se o usuário voltar para a atividade o sistema chama onResume()



Paused

- Pare animações que possam consumir CPU
- Dê commit em informações que não estão salvas
- Libere recursos como broadcast receivers, handlers de sensores (com GPS) ou outra coisa que possa gastar bateria enquanto seu app está parado
- Nunca salve informações que o usuário ainda não mandou salvar; somente aquelas que ele espera que sejam autosaved

```
@Override
public void onPause() {
    super.onPause();  // Always call the superclass method first

    // Release the Camera because we don't need it when paused
    // and other activities might need to use it.
    if (mCamera != null) {
        mCamera.release()
        mCamera = null;
    }
}
```

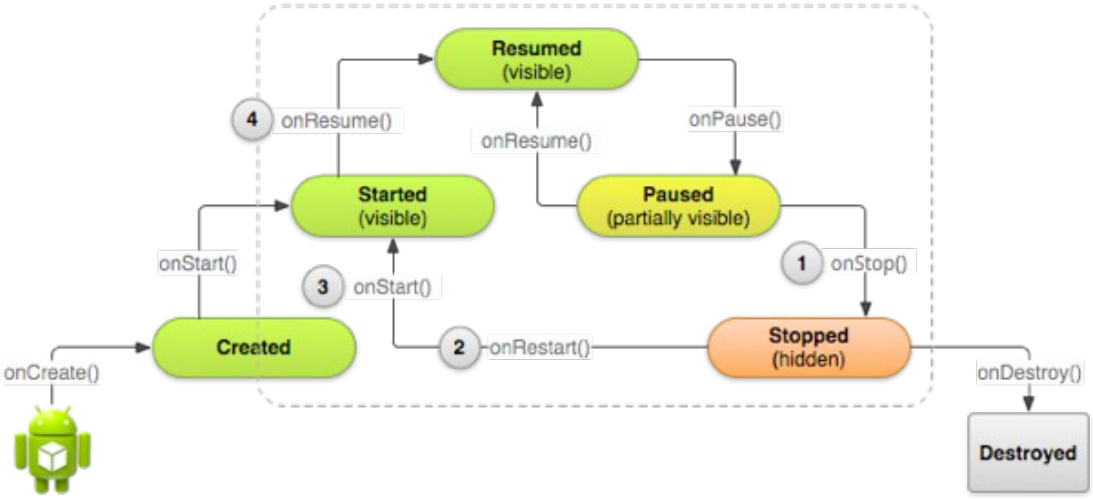
Resumed

- É chamado toda vez que seu sistema sai de background para foreground
- Use para recuperar os recursos que liberou no onPause(), recomeçar animações, etc.

```
@Override
public void onResume() {
    super.onResume();  // Always call the superclass method first

    // Get the Camera instance as the activity achieves full user focus
    if (mCamera == null) {
        initializeCamera(); // Local method to handle camera init
    }
}
```

Stop e Start

- Sua atividade entre em stop quando o usuário abre a janela de apps recentes e muda do seu app para outro; quando o usuário volta para seu app, o sistema chama onRestart()
- O usuário inicia uma nova Activity no seu App. Esta é parada quando a segunda é criada, e chama onRestart quando usuário clica em Back.
- O usuário recebe uma ligação telefônica



Parando a Atividade

- Use onStop() para liberar todos os recursos possíveis
- Apesar de onPause() ter sido chamado antes de onStop(), deixe operações que consomem mais CPU para o onStop(), como gravação em banco de dados (exemplo).

```
@Override
protected void onStop() {
    super.onStop();  // Always call the superclass method first

    // Save the note's current draft, because the activity is stopping
    // and we want to be sure the current note progress isn't lost.
    ContentValues values = new ContentValues();
    values.put(NotePad.Notes.COLUMN_NAME_NOTE, getCurrentNoteText());
    values.put(NotePad.Notes.COLUMN_NAME_TITLE, getCurrentNoteTitle());

    getContentResolver().update(
            mUri,    // The URI for the note to update.
            values,  // The map of column names and new values to apply to
them.
            null,    // No SELECT criteria are used.
            null     // No WHERE columns are used.
            );
}
```

Reiniciando a Atividade

```
@Override
protected void onStart() {
    super.onStart();  // Always call the superclass method first
```

```
    // The activity is either being restarted or started for the first time
    // so this is where we should make sure that GPS is enabled
    LocationManager locationManager =
            (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    boolean gpsEnabled =
locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

    if (!gpsEnabled) {
        // Create a dialog here that requests the user to enable GPS, and
use an intent
        // with the
android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS action
        // to take the user to the Settings screen to enable GPS when they
click "OK"
    }
}

@Override
protected void onRestart() {
    super.onRestart();  // Always call the superclass method first

    // Activity being restarted from stopped state
}
```

- o onStart() é sempre chamado depois do onRestart(), pois ele sempre é chamado quando uma atividade se torna visível;
- Por isso, use o onRestart() para reiniciar recursos que não precisam ser iniciados no onStart()
- Lembre-se que o onStop() mata todos os recursos, então eles precisam ser reinstanciados.

Recriando uma Atividade

- Há poucos cenários nos quais sua atividade é destruída normalmente, quando o usuário aperta o botão Back ou quando a própria Activity chama finish()
- Este comportamento indica para o sistema que a Activity não é mais necessária
- Nos outros casos, nos quais o sistema chama o onDestroy(), ele guarda informações para quando o usuário for voltar para aplicação e o sistema precisar recriá-la.
- O sistema grava as informações necessárias, chamadas instance state, em um Bundle no formato chave-valor.
- Sua atividade será destruída e recriada, por exemplo, quando o dispositivo móvel é girado (rotate). Isso porque haverá mudanças de layout e novos recursos serão carregados.

Restart

- Por default o sistema grava as informações da View no Bundle, como o valor escrito em um EditText
- Mas você pode usar o método onSaveInstanceState() para guardar coisas suas, e o método onRestoreInstanceState() para pegar as coisas de volta

Salvando informações

- Faça a sobrecarga do método e salve suas informações no Bundle
- Sempre chame o onSaveInstance na superclasse depois, caso contrário as informações da View não serão salvas.

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

Recuperando informações

- Você pode fazer isso no onCreate, mas não se esqueça de verificar se o Bundle não é null, pois é isso que vai acontecer se for a primeira vez que a Activity estiver sendo carregada

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new
instance
    }
    ...
```

```
}
```

Recuperando informações

Ou você pode chamar o onRestoreInstanceState() que será chamado depois do onCreate()
Ele só é chamado quando o objeto é restaurado, então o Bundle não será null
Não se esqueça de chamar o método na superclasse antes

```
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

Exemplo

Neste exemplo iremos ilustrar o uso dos métodos do ciclo de vida de uma Activity para ativar e desativar o hardware de GPS conforme a aplicação ganha e perde o foco do usuário.

Passo 1 (Criando o projeto) No Android Studio, crie um novo projeto com os seguintes dados.

Template: Basic Activity
Name: Ciclo de Vida GPS e Mapas
Package Name: br.usjt
Save Location: Mantenha o valor padrão
Language: Java
Minimum API Level: API 23: Android Marshmallow

Mantenha qualquer outro campo com seu valor padrão e clique em Finish.

Passo 2 (Inspecionando os arquivos criados) Note que agora temos dois arquivos .xml para descrever a tela: activity_main.xml e content_main.xml. O primeiro inclui o segundo. No primeiro, definimos um layout principal e o botão flutuante. Ele inclui o segundo, no qual iremos adicionar o conteúdo da tela.

Passo 3 (Declarando os componentes para uso do GPS) O Android possui diferentes APIs para lidar com GPS. Iremos utilizar as classes LocationManager e LocationListener para isso. LocationManager é uma classe que permite lidar com o hardware de GPS. Iremos criar uma instância de LocationListener e sobrescrever um método de interesse, que será chamado quando houver alguma atualização na localização do usuário. Trata-se de uma aplicação do conhecido padrão de projeto Observer. Na classe MainActivity, declare as variáveis de instância mostradas na Listagem 3.1.

Listagem 3.1

```
private LocationManager locationManager;
private LocationListener locationListener;
```

Passo 4 (Inicializando a variável locationManager) LocationManager é um serviço do sistema operacional que podemos obter por meio do método getSystemService. No método onCreate, chame esse método como mostra a Listagem 4.1.

Listagem 4.1

```
locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

Passo 5 (Inicializando a variável locationListener) Agora iremos criar uma instância de LocationListener. Como LocationListener é uma interface, precisaremos criar uma interface que a implementa. Faremos isso por meio de uma classe interna anônima. No método onCreate, escreva a atribuição da Listagem 5.1. Não se esqueça de deixar o Android Studio te ajudar completando o código para você.

Listagem 5.1

```
locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {

    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) {
    }
    @Override
    public void onProviderEnabled(String provider) {
    }
    @Override
    public void onProviderDisabled(String provider) {
    }
};
```

Passo 6 (Ativando o hardware de GPS) Note que até então não ativamos o hardware de GPS. Isso será feito quando registrarmos o locationListener como observador do locationManager. Nesta aplicação iremos ligar o GPS quando ela ganhar o foco do usuário e desligá-lo quando a aplicação perder o foco. Por isso, iremos escolher um par de métodos do ciclo de vida, por exemplo, onStart para ativar e onStop para desativar. Note que ativar o hardware de GPS é uma atividade que requer que o usuário forneça permissão em tempo de execução. Daí o código da Listagem 6.1.

Listagem 6.1

```
@Override
protected void onStart() {
    super.onStart();
    //a permissão já foi dada?
    if (ActivityCompat.checkSelfPermission(
            this, Manifest.permission.ACCESS_FINE_LOCATION) ==
            PackageManager.PERMISSION_GRANTED){
        //somente ativa
        //a localização é obtida via hardware, intervalo de 0
segundos e 0 metros entre atualizações

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);
    }
    else{
        //permissão ainda não foi nada, solicita ao usuário
        //quando o usuário responder, o método
onRequestPermissionsResult vai ser chamado
        ActivityCompat.requestPermissions(this,
                new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, 1001);
    }
}
```

Passo 7 (Adicionando a permissão no AndroidManifest.xml) Além de solicitar a permissão em tempo de execução, é necessário informá-la no arquivo AndroidManifest.xml. Veja a Listagem 7.1. A tag uses-permission fica fora da tag application.

Listagem 7.1

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Passo 8 (Tratando a resposta do usuário) Quando pedimos permissão em tempo de execução ao usuário, um diálogo será exibido e ele bloqueará a execução do programa até que o usuário decida o que deseja. Uma vez que ele interaja com o diálogo, a execução será desbloqueada desencadeando uma chamada ao método onRequestPermissionsResult, o qual possui parâmetros que nos permite identificar qual foi a resposta do usuário. Dependendo da resposta do usuário iremos ativar ou não o hardware de GPS. Veja o método na Listagem 8.1.

Listagem 8.1

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
    if (requestCode == 1001){
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
            //permissão concedida, ativamos o GPS
            if (ActivityCompat.checkSelfPermission(this,
                    Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED){

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);
            }
        }
        else{
            //usuário negou, não ativamos
            Toast.makeText(this, getString(R.string.no_gps_no_app),
Toast.LENGTH_SHORT).show();
        }
    }
}
```

Passo 9 (Criando a string e trocando o valor hard coded) Note que usamos um recurso de string chamado no_gps_no_app sem que ele exista. Precisamos criá-lo. Para isso, abra o arquivo strings.xml (na pasta values) e adicione o código da Listagem 9.1.

Listagem 9.1

```
<string name="no_gps_no_app">Sem o GPS o aplicativo não
funciona</string>
```

Note também que utilizamos o código 1001 para representar a requisição por permissão de uso do hardware de GPS. Isso ocorre pois em um aplicativo podemos pedir diferentes permissões e o Android utiliza um número inteiro para identificá-las e permitir que elas sejam diferenciadas quando o método onRequestPermissionsResult for chamado. É uma péssima prática manter o valor fixo como está no código, o que caracteriza o uso do anti-pattern “números mágicos”. Quando usamos esse anti-pattern, nosso código possui números espalhados que fazem com que a aplicação supostamente funcione mas cujo significado é, em geral, obscuro para quem tenta inspecionar o código. Assim, crie a constante da Listagem 9.2 e substitua as ocorrências do número 1001 por ela.

Listagem 9.2

```
private static final int REQUEST_CODE_GPS = 1001;
```

Nota: O número 1001 não tem nada de especial. Qualquer número inteiro positivo serve, basta não repetir o mesmo número para requisições diferentes.

Passo 10 (Desabilitando o hardware de GPS) A fim de economizar bateria, iremos desabilitar o hardware de GPS quando a aplicação perder o foco do usuário. Para isso, iremos utilizar o método onStop, que é naturalmente o complemento do método onStart. Veja sua implementação na Listagem 10.1.

Listagem 10.1

```
@Override
protected void onStop() {
    super.onStop();
    locationManager.removeUpdates(locationListener);
}
```

Passo 11 (Ajustando o TextView para exibir as coordenadas obtidas) A fim de exibir as coordenadas obtidas pelo GPS, iremos utilizar o TextView já existente no aplicativo. Para isso, faça o seguinte:

11.1 Abra o arquivo content_main.xml e adicione o id coordenadasTextView ao TextView ali existente (**android:id="@+id/locationTextView"**).

11.2 Apague a propriedade text do TextView que exibe atualmente o texto Hello World.

11.3 No arquivo MainActivity.java, adicione a variável de instância da Listagem 11.1.

Listagem 11.1

```
private TextView locationTextView;
```

11.4 No método onCreate, inicialize a variável locationTextView com o código da Listagem 11.2. Lembre-se de que isso só pode ocorrer depois de o método setContentView ser executado. Assim, coloque essa instrução logo depois da chamada ao método setContentView.

Listagem 11.2

```
locationTextView = findViewById(R.id.locationTextView);
```

Passo 12 (Exibindo as coordenadas) Conforme a localização do usuário muda, o método onLocationChanged (do LocationManager, que já criamos) é chamado. Agora iremos implementar esse método de modo que ele pegue as coordenadas latitude e longitude e simplesmente faça com que o locationTextView as exiba. Veja a Listagem 12.1.

Listagem 12.1

```
@Override
public void onLocationChanged(Location location) {
    double lat = location.getLatitude();
    double lon = location.getLongitude();
    locationTextView.setText(String.format("Lat: %f, Long: %f", lat,
lon));
}
```

Passo 13 (Buscando restaurantes por perto) Quando o usuário clicar no botão flutuante, queremos que uma busca por restaurantes seja realizada e um mapa do Google Maps os exiba. Para isso, precisamos:

13.1 Criar duas variáveis de instância para guardar latitude e lontitude. Veja a Listagem 13.1.

Listagem 13.1

```
private double latitudeAtual;
private double longitudeAtual;
```

13.2 Atualizar os valores das novas variáveis no método onLocationChanged, como mostra a Listagem 13.2.

Listagem 13.2

```
@Override
public void onLocationChanged(Location location) {
    double lat = location.getLatitude();
    double lon = location.getLongitude();
    latitudeAtual = lat;
    longitudeAtual = lon;
    locationTextView.setText(String.format("Lat: %f, Long: %f", lat,
lon));
}
```

13.3 Utilizar os valores no método onClick do observer já registrado no botão flutuante. Veja a Listagem 13.3.

Listagem 13.3

```
@Override
public void onClick(View view) {
    Uri gmmIntentUri =
            Uri.parse(String.format("geo:%f,%f?q=restaurantes",
                    latitudeAtual, longitudeAtual));
    Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
    mapIntent.setPackage("com.google.android.apps.maps");
    startActivity(mapIntent);
}
```

Atividade Prática

O projeto desta disciplina será integrado com a disciplina de Arquitetura de Software. Nossa aplicação oferecerá funcionalides de previsão do tempo.

1. Crie um novo aplicativo Android no Android Studio com versão mínima 6.0 (Marshmallow).

2. Repita todo o processo da aula, de modo que seu aplicativo lide adequadamente com o hardware de GPS por meio do ciclo de vida.

3. Suba seu projeto para o GitHub (para essa atividade você irá criar um novo repositório (só dessa vez), separado do repositório do Hello World). Mantenha esse repositório até o final do projeto. Cada nova atualização deverá ser salva neste repositório.

4. Gere uma release e envie um e-mail para o seu professor da seguinte forma:

Assunto: DESWEBMOB – Projeto Previsão do tempo A1
Corpo: Nome completo sem abreviações, RA e Link do repositório

Basta enviar um único e-mail. Porém, semanalmente você deverá gerar as releases em seu repositório pois o Github controla horários e o professor irá verificar isso.