PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Ciclo de Vida de uma Aplicação Android

Professor: Danilo Giacobo

OBJETIVOS DA AULA

- Entender o ciclo de vida de uma aplicação Android.
- Conhecer algumas dicas para tirar proveito dela para melhorar a aplicação.
- Usar conscientemente a *Activity Stack*.
- Conhecer e usar os métodos de gerenciamento do ciclo de vida da *Activity*.
- Codificar automaticamente os métodos do ciclo de vida da *Activity*.
- Usar recursos de depuração com a ferramenta *LogCat*.
- Testar funcionalidades básicas do emulador com o *Emulator Control*.

INTRODUÇÃO

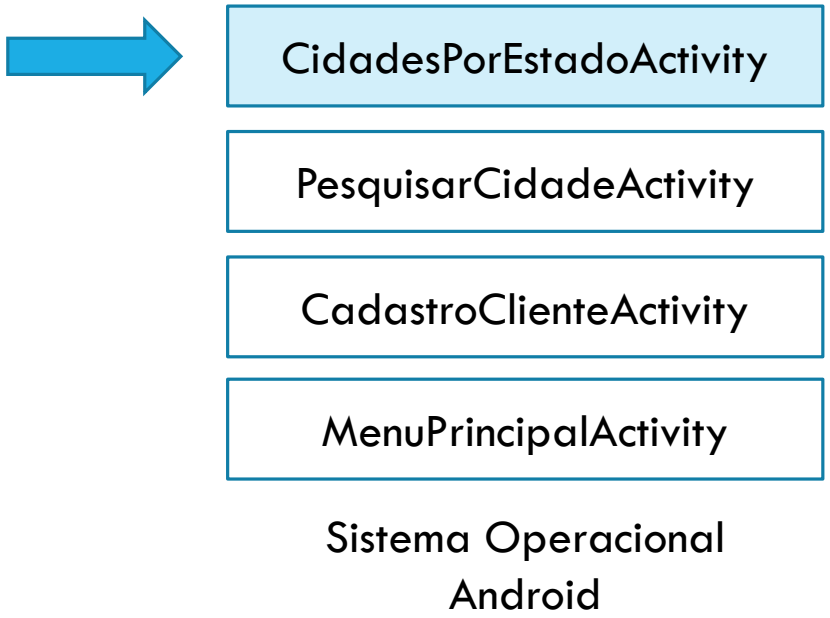
- Uma das maiores diferenças entre o desenvolvimento para computadores e para dispositivos móveis é o tratamento do ciclo de vida desta última.
- Em uma aplicação móvel, os recursos são bem mais limitados. O compartilhamento de recursos pode comprometer a performance do dispositivo.
- Na própria plataforma Android não é possível alternar entre várias telas como em uma tela de computador normal, usando como exemplo as teclas [ALT] + [TAB].
- Um pequeno exemplo: quando estamos jogando e uma ligação é recebida, o jogo fica parado até que a ligação seja encerrada. Aparentemente, as duas aplicações estavam sendo executadas, quando na verdade, apenas uma delas estava e a outra estava parada.

FUNCIONAMENTO DA ACTIVITY STACK

- Toda aplicação Android pode ser formada por uma ou mais Activities. Essas são, na verdade, pequenas "atividades" executadas uma de cada vez.
- A maneira mais fácil de entender as *Activities* é associar cada tela a uma *Activity*.
- Além das *Activities* codificadas pelos programadores e que fazem parte de uma aplicação específica, existem outras *Activities* e estas fazem parte do pacote do sistema operacional Android.
- Para controlar todas essas *Activities*, já que apenas uma pode ser executada de cada vez, existe a Activity Stack ou pilha de *Activities*.

FUNCIONAMENTO DA ACTIVITY STACK

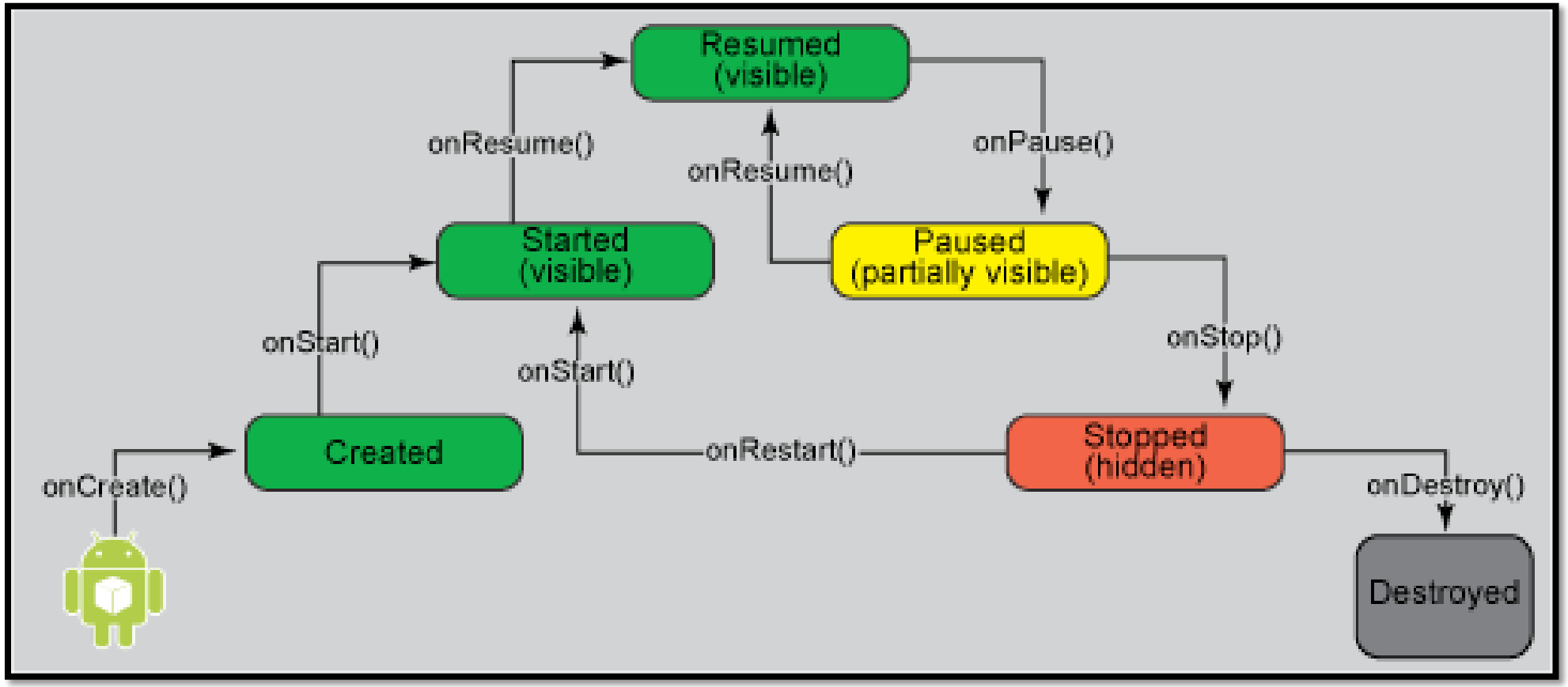
- A figura abaixo apresenta o funcionamento da Activity Stack.



CICLO DE VIDA DE UMA APLICAÇÃO ANDROID

- De forma resumida, uma aplicação móvel pode estar em **quatro estados distintos**: em execução, pausada, parada ou destruída.
- No Android, o ciclo de vida de uma aplicação é algo mais complexo e quando uma aplicação está em execução, ela costuma tomar conta da tela do *device*.
- Quando a aplicação está pausada, o aplicativo tem apenas parte de sua tela visível.
- No estado de parada, ela não está mais visível parta o usuário.
- O estado de destruída é chamado para retirar o aplicativo da memória.
- Para tratar esses quatro estados, sete métodos podem ser codificados: onCreate, onStart, onResume, onPause, onStop, onRestart e onDestroy.

CICLO DE VIDA DE UMA APLICAÇÃO ANDROID



CICLO DE VIDA DE UMA APLICAÇÃO ANDROID

- Dos sete métodos citados, o único obrigatório para a codificação é onCreate(). Esse método é executado quando a *Activity* é chamada e sua principal função costuma ser apresentar a tela associada à *Activity*.
- Depois da execução do método onCreate(), outros dois métodos são executados: onStart() e onResume().
- Para o estado de pausa, os métodos onPause() e onResume() são executados.
- Para o estado de parado, os métodos onPause() e onStop() são executados e na sequencia uma nova aplicação ou *Activity* assume o topo da pilha de *Activities*.
- O método onDestroy() é chamado quando a *Activity* é encerrada e esse método é antecedido pelos métodos onPause() e onStop().

INICIANDO E DESTRUINDO UMA APLICAÇÃO ANDROID

- O método onCreate() é o único método do ciclo de vida que necessita obrigatoriamente de codificação e por este motivo, é o método mais conhecido dos programadores Android.
- No código abaixo temos um exemplo clássico de utilização do método onCreate() em um aplicativo Android.

```
1  package pm25s.aula09.ciclodevida;
2
3  import android.app.Activity;
9
10 public class MainActivity extends Activity {
11     private Button btLancamento;
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         btLancamento = (Button) findViewById(R.id.btLancamento);
19         btLancamento.setOnClickListener(new View.OnClickListener() {
20             @Override
21             public void onClick(View v) {
22                 Toast.makeText(getApplicationContext(), String.valueOf(Build.VERSION.SDK_INT), Toast.LENGTH_LONG).show();
23             }
24         });
25     }
26 }
```

INICIANDO E DESTRUINDO UMA APLICAÇÃO ANDROID

- O método onDestroy() é chamado quando uma *Activity* é finalizada. Um exemplo de utilização deste método pode ser visto logo abaixo.

```
private Button btSair;
```

```
btSair = (Button) findViewById(R.id.btSair);
```

```
btSair.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        finish();
    }
});
```

```
@Override
public void onDestroy() {
    super.onDestroy();
    Toast.makeText(getApplicationContext(), "Aplicação finalizada", Toast.LENGTH_LONG).show();
}
```

PAUSANDO E RETORNANDO APLICAÇÃO ANDROID

- Durante a execução de uma *Activity*, em algumas situações, esta pode estar "parcialmente visível" para o usuário, como, por exemplo, quando a *Activity* chama um *Dialog* ou quando uma tela automática é mostrada, como a do despertador. Nesta situação, o método onPause() é chamado automaticamente.
- Quando a *Activity* chama este método o aplicativo ficará por alguns momentos sem executar. O usuário poderá retornar a ela e continuar utilizando a mesma.
- O método onPause() é o primeiro a ser chamado após uma aplicação sair do estado de execução.
- Ele é comumente usado para parar a execução de vídeos ou animações, para evitar o consumo desnecessário do processador ou liberar recursos que serão utilizados pela outra tela que foi executada, por exemplo.

PAUSANDO E RETORNANDO APLICAÇÃO ANDROID

- Uma dica importante é, sempre que possível, não utilizar comandos que prejudiquem a performance da *Activity* no método onPause() como, por exemplo, persistir os dados em um banco de dados ou no cartão de memória. Tais recursos costumam ocupar certo tempo de processamento e podem dar a sensação de que o dispositivo travou na troca entre diferentes telas. O código abaixo apresenta um exemplo de implementação do método onPause().

```
@Override
public void onPause() {
    super.onPause();
    Toast.makeText(getApplicationContext(), "Aplicação pausada", Toast.LENGTH_LONG).show();
}
```

- Experimente configurar um alarme e deixe sua aplicação em foco. Quando o alarme for acionado, este método será chamado.

PAUSANDO E RETORNANDO APLICAÇÃO ANDROID

- Quando o usuário liberar novamente a tela do aplicativo, como, por exemplo, fechando a caixa de diálogo aberta, o mesmo executará o método onResume(). Este método é executado em duas situações:
  - Quando a Activity é executada, sendo chamado o onResume() logo após o método onCreate();
  - Quando o aplicativo retorna de um estado de pausa.

- Por este motivo, é interessante não utilizar códigos que devem ser executados apenas quando retorna de uma pausa neste método ou, ainda, que devem ser executados apenas na primeira vez quando o programa é chamado. Exemplo de implementação deste método:

```
@Override
public void onResume() {
    super.onResume();
    Toast.makeText(getApplicationContext(), "Aplicação resumida", Toast.LENGTH_LONG).show();
}
```

PARANDO E REINICIANDO APLICAÇÃO ANDROID

- Uma *Activity* possui dois estados em que ela não está rodando: quando está em estado de pausa ou em estado de parada.
- Uma *Activity* fica em estado de parada quando a mesma não está mais presente na tela do dispositivo, como, por exemplo, quando este chama uma nova *Activity*, quando retornar para a tela principal da aplicação ou, ainda, quando chega uma ligação. Nestas situações, a *Activity* chama o método onStop(), que é executado logo após o método de onPause().
- Para um melhor gerenciamento da memória do dispositivo, o sistema operacional Android possui a permissão de finalizar as aplicações Android que estão em estado de "parada", isso quando, por exemplo, o dispositivo ficou com pouca memória para a execução de programas.

PARANDO E REINICIANDO APLICAÇÃO ANDROID

- A situação apresentada anteriormente, embora difícil de acontecer, é possível.
- Não existe garantia de que uma *Activity* parada retorne normalmente com os mesmos dados na tela após a primeira execução.
- Por este motivo é aconselhável a persistência dos dados digitados na tela, isso, claro, se houver interesse do programador.
- Um exemplo de implementação do método onStop() é mostrado abaixo:

```
public void onStop() {
    super.onStop();
    Toast.makeText(getApplicationContext(), "Aplicação parada", Toast.LENGTH_LONG).show();
}
```

PARANDO E REINICIANDO APLICAÇÃO ANDROID

- Ao contrário do método onPause(), onde se deve evitar a utilização de comandos que prejudiquem a performance da aplicação, no método onStop(), esses comandos podem ser utilizados sem problemas.
- Após retornar de um estado de parada, uma *Activity* executa dois métodos associados ao ciclo de vida dela:
  - Método onRestart(): esse método só é executado quando uma aplicação retorna do estado de parada, assim, os códigos específicos do retorno podem ser tratados aqui;
  - Método onStart(): esse método pode ser chamado no retorno de um estado de parada, assim como na primeira execução de um aplicativo (após o método onCreate()).

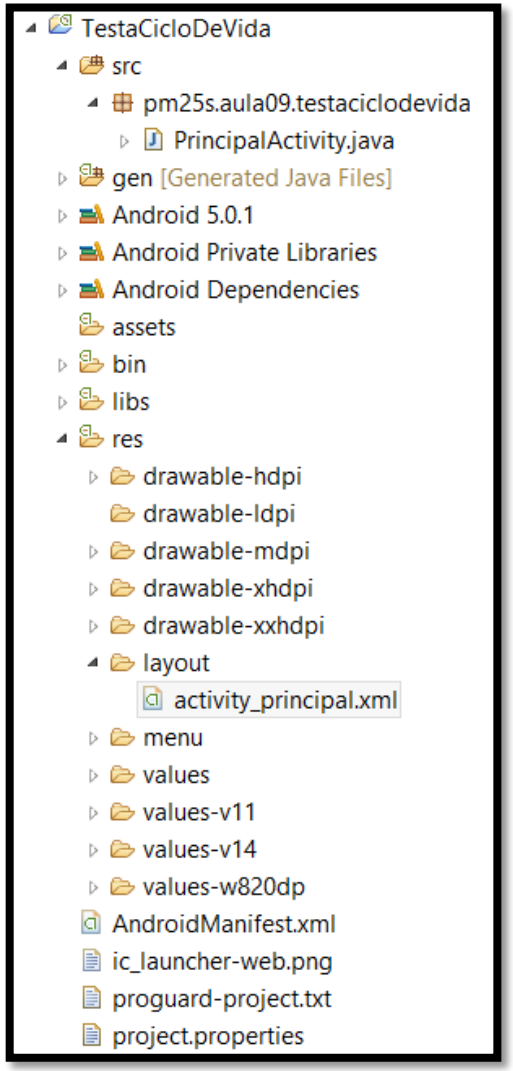
PARANDO E REINICIANDO APLICAÇÃO ANDROID

- O código abaixo apresenta um exemplo de uso do método onRestart() (chamado apenas após retornar do estado de parado) e do método onStart() (chamado sempre quando a aplicação é iniciada e também quando retorna do estado de parada).

```
public void onStart() {
    super.onStart();
    Toast.makeText(getApplicationContext(), "Aplicação iniciada", Toast.LENGTH_LONG).show();
}

public void onRestart() {
    super.onRestart();
    Toast.makeText(getApplicationContext(), "Aplicação reiniciada", Toast.LENGTH_LONG).show();
    // Código que deve ser executado exclusivamente quando a aplicação vem do estado de parado.
}
```

ESTUDO DE CASO

- Para testar todos os métodos do ciclo de vida de uma aplicação Android, criaremos uma aplicação chamada TestaCicloDeVida.
- Nossa aplicação terá uma única tela, podendo ter um botão central com o texto "Apresenta Dialog", já que nosso objetivo não é trabalhar com os componentes visuais da plataforma e sim, entender o ciclo de vida da aplicação.
- Esta interface gráfica terá o nome de *activity_principal.xml*, já a classe da *Activity* terá o nome de *PrincipalActivity.java*, conforme imagem ao lado.



ESTUDO DE CASO

- O código referente à interface gráfica da aplicação pode ser o apresentado na imagem abaixo, gerado automaticamente pelo Eclipse com o plug-in do ADT na criação de um novo projeto Android.

```
1  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
2      xmlns:tools="http://schemas.android.com/tools"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:paddingBottom="@dimen/activity_vertical_margin"
6      android:paddingLeft="@dimen/activity_horizontal_margin"
7      android:paddingRight="@dimen/activity_horizontal_margin"
8      android:paddingTop="@dimen/activity_vertical_margin"
9      tools:context="pm25s.aula09.testaciclodevida.PrincipalActivity" >
10
11     <TextView
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:layout_centerHorizontal="true"
15         android:layout_centerVertical="true"
16         android:text="@string/hello_world" />
17
18 </RelativeLayout>
```

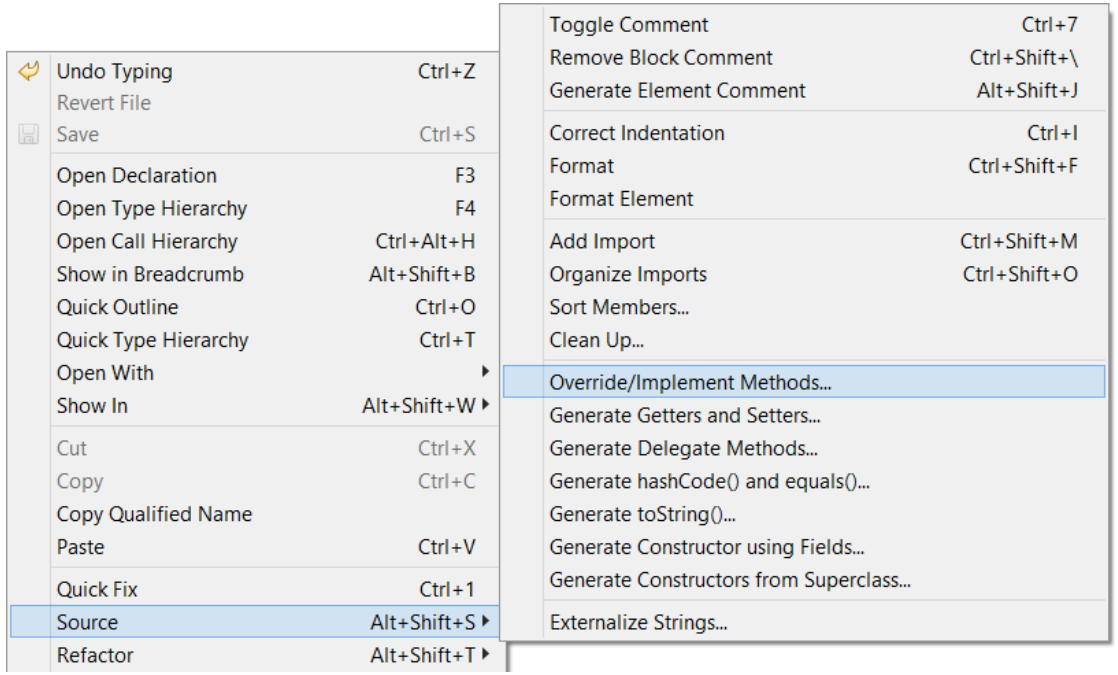
ESTUDO DE CASO

- O código referente à *Activity* principal é apresentado abaixo:

```java
package pm25s.aula09.testaciclodevida;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

public class PrincipalActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_principal);
        System.out.println("Método onCreate() executado.");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        System.out.println("Método onDestroy() executado.");
    }

    @Override
    protected void onPause() {
        super.onPause();
        System.out.println("Método onPause() executado.");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        System.out.println("Método onRestart() executado.");
    }

    @Override
    protected void onResume() {
        super.onResume();
        System.out.println("Método onResume() executado.");
    }

    @Override
    protected void onStart() {
        super.onStart();
        System.out.println("Método onStart() executado.");
    }

    @Override
    protected void onStop() {
        super.onStop();
        System.out.println("Método onStop() executado.");
    }
}
```

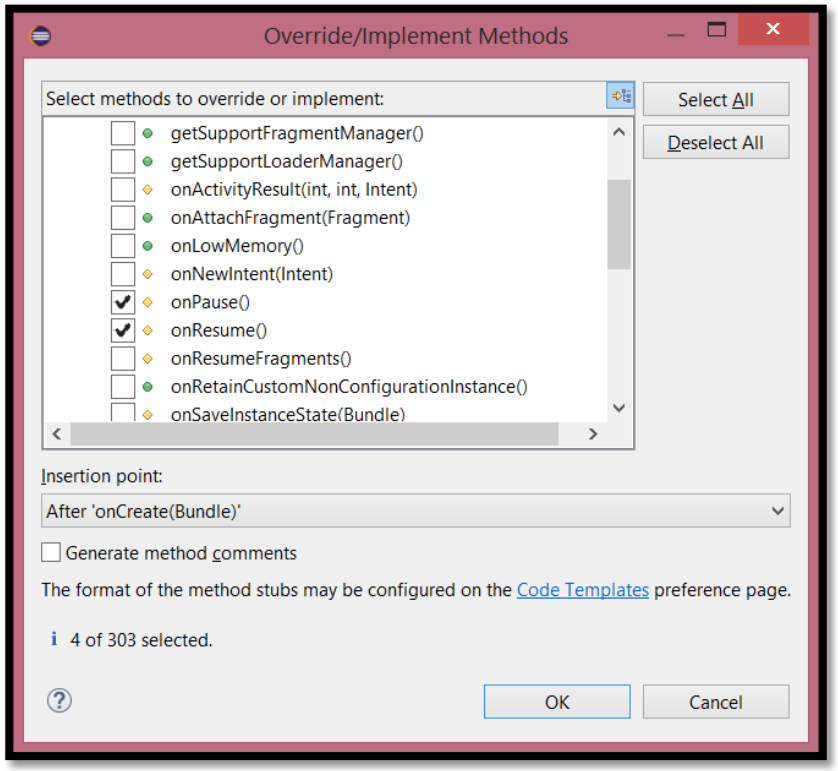
ESTUDO DE CASO

- Para facilitar a codificação dos métodos apresentados, podemos clicar com o botão direito no código-fonte, escolhendo a opção **Source > Override/Implement Methods...**, conforme apresentado abaixo:



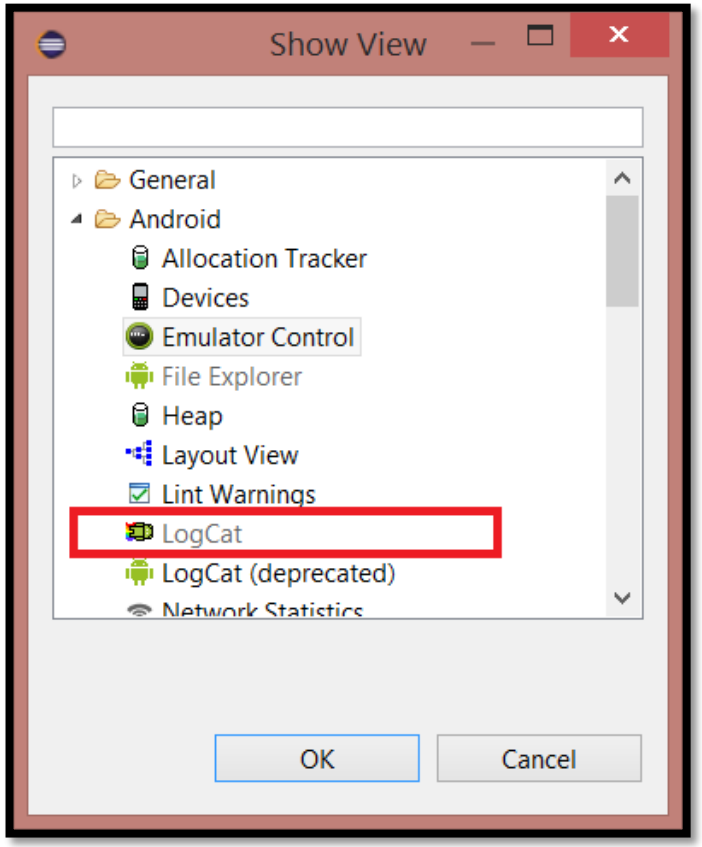
ESTUDO DE CASO

- Na janela apresentada é possível escolher os seis métodos faltantes e que não são obrigatórios. Após a seleção, basta clicar em OK.



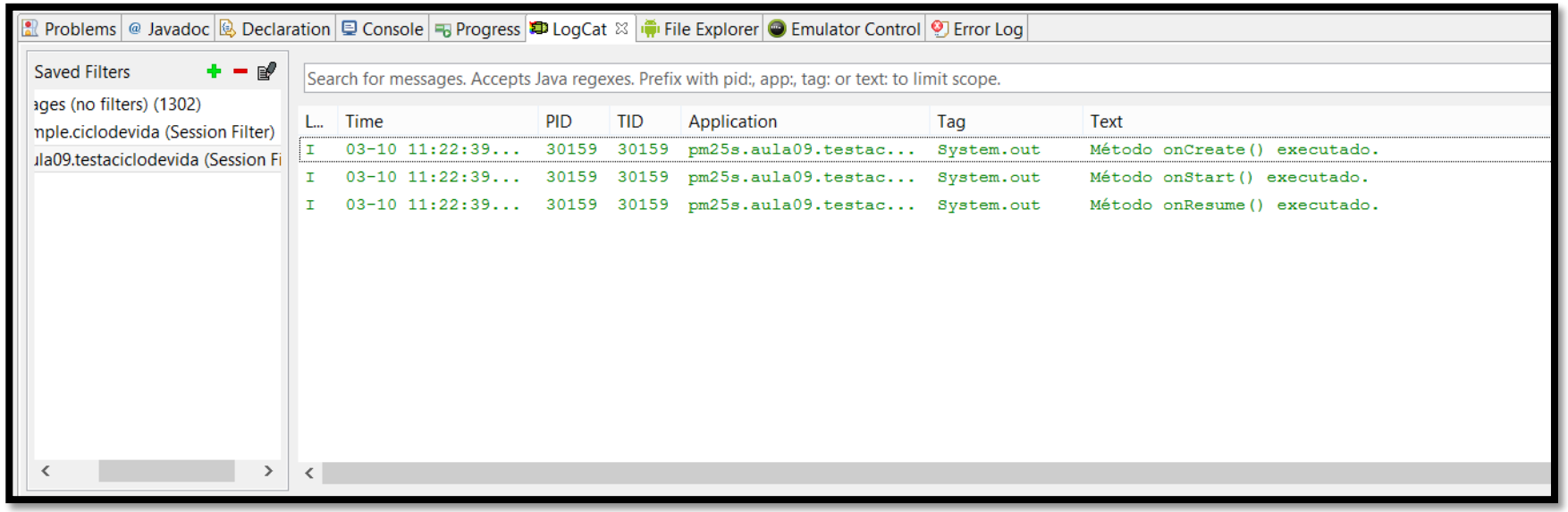
ESTUDO DE CASO

- Executando o aplicativo, as mensagens geradas com o comando System.out.println são visíveis na janela do **LogCat** do IDE Eclipse, por exemplo.
- Para apresentar a tela, basta acessar o menu **Window > Show View > Other** e na categoria Android, escolher a opção **LogCat**, conforme imagem abaixo.



ESTUDO DE CASO

- A janela **LogCat** que aparece após a execução do aplicativo desenvolvido é apresentada abaixo. Verifique que todas as mensagens foram apresentadas no mesmo local, diferenciando apenas pela *tag* das mensagens geradas pelo sistema operacional Android para a execução do aplicativo.



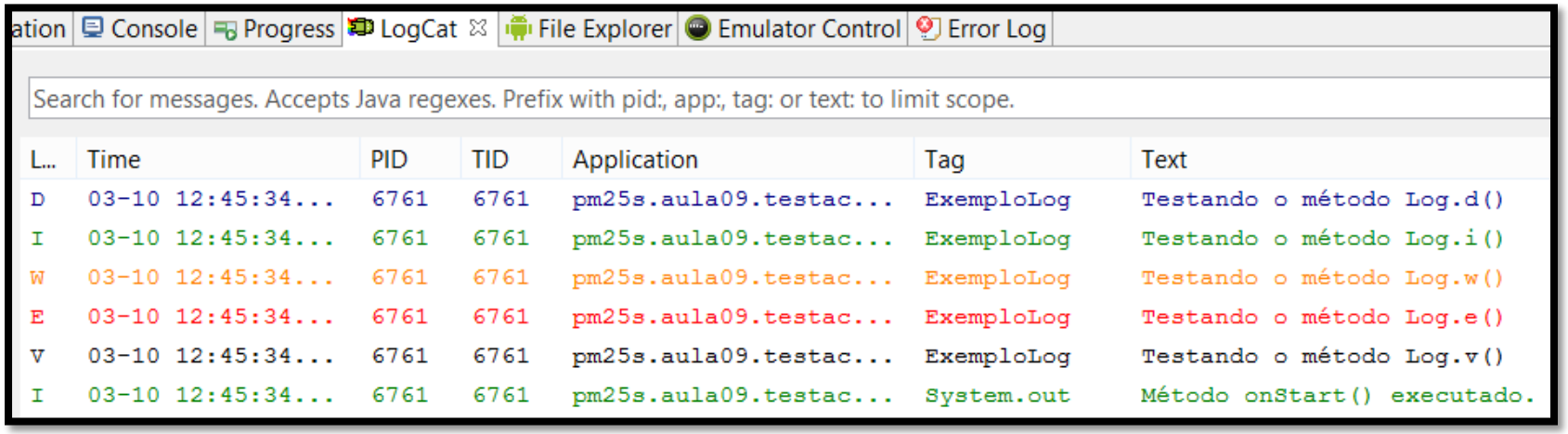
ESTUDO DE CASO

- Você pode utilizar também a classe *Log* do pacote *java.util* para exibir mensagens na tela do **LogCat**. O conteúdo impresso com a classe *Log* é apresentado na tela de **LogCat** e está dividido em:
  - Log.d(): informações referentes a *Debug*.
  - Log.i(): apresenta dados de informação.
  - Log.w(): informações referentes a *warning*.
  - Log.e(): informações referentes a erros.
  - Log.v(): informações gerais.
- Para utilizar tais comandos, basta fazer o importar do pacote *java.util* e digitar o comando como segue, passando como primeiro parâmetro a tag que identificará a mensagem e o segundo parâmetro a mensagem que se deseja apresentar.

ESTUDO DE CASO

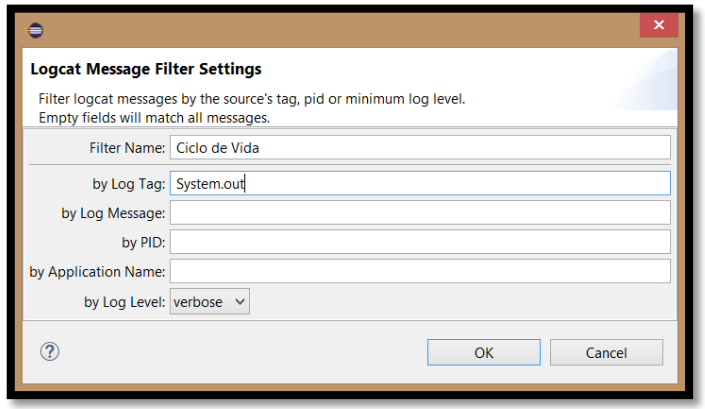
- Na imagem abaixo é mostrado um exemplo de uso da classe *Log* e o seu respectivo resultado na tela do **LogCat**.

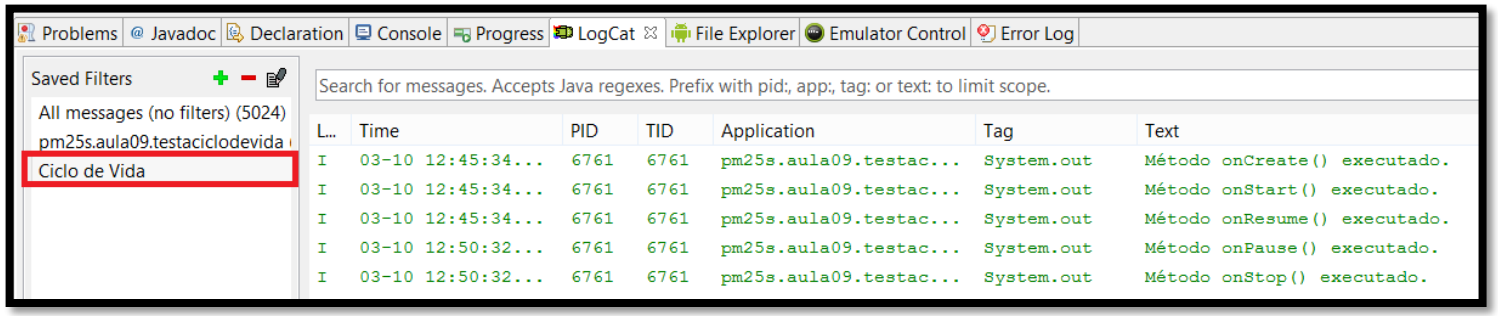
```
Log.d("ExemploLog", "Testando o método Log.d()");
Log.i("ExemploLog", "Testando o método Log.i()");
Log.w("ExemploLog", "Testando o método Log.w()");
Log.e("ExemploLog", "Testando o método Log.e()");
Log.v("ExemploLog", "Testando o método Log.v()");
```



ESTUDO DE CASO

- Para filtrar as mensagens, apresentando apenas aquelas com uma determinada *tag*, devemos clicar no sinal de mais (*Add a new logcat filter*). Uma janela então é apresentada.
- Entre os campos utilizados para o filtro, usamos apenas o nome do filtro, o qual identificará no lado esquerdo da tela do **LogCat**, e também o campo Log Tag, o qual possui o nome da tag a ser procurada.

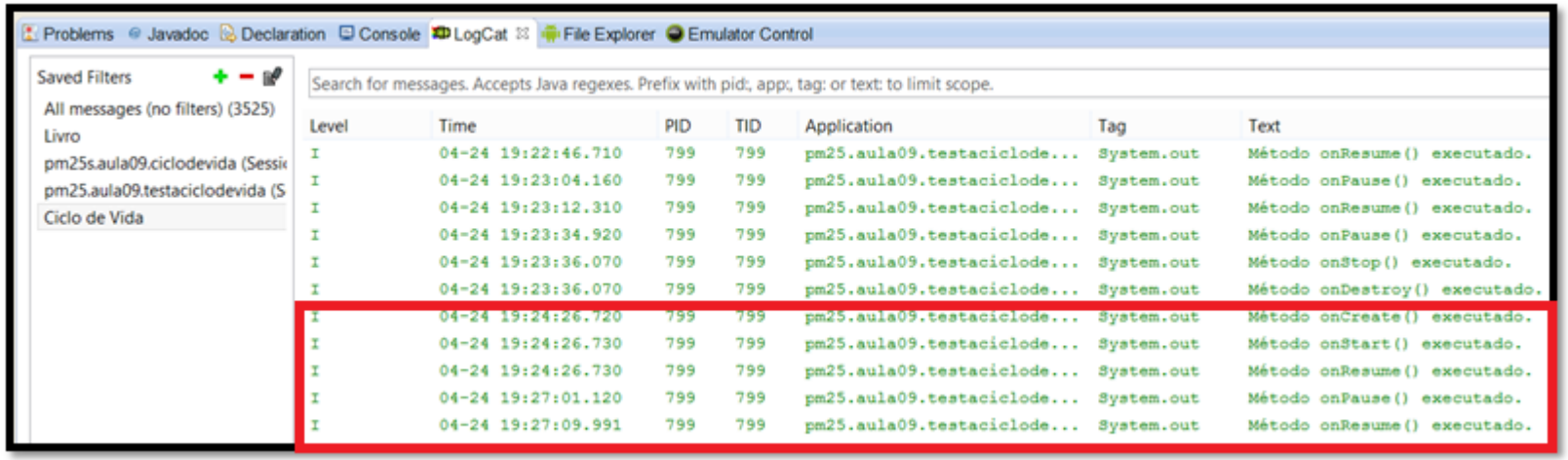




ESTUDO DE CASO

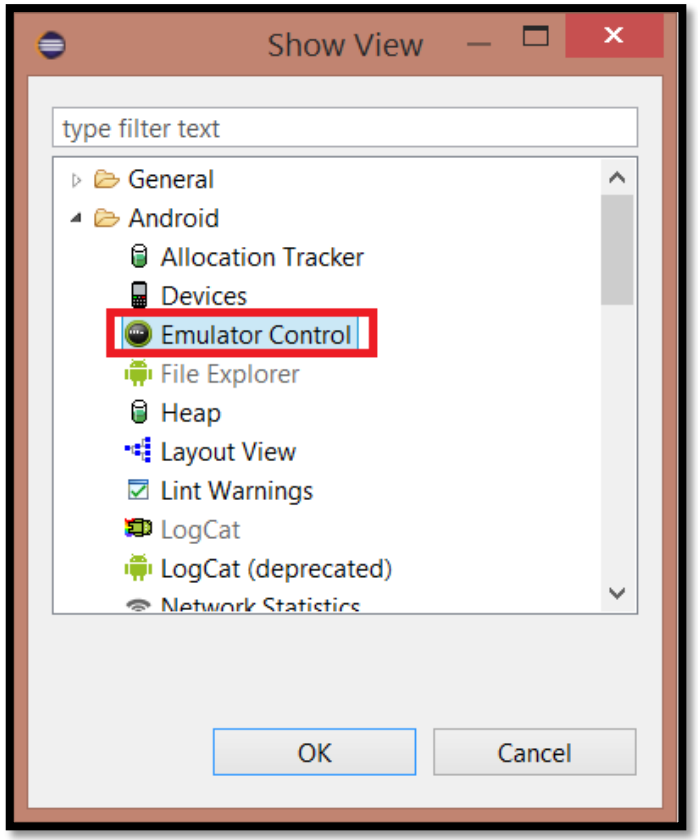
- O primeiro teste do ciclo de vida é testar os métodos **onPause()** e **onResume()**.
- Para isto, acesse o menu do aparelho celular, entre na opção de alarme/despertador e agende-o para tocar um ou dois minutos a mais do que o horário atual.
- Após isso, confirme o alarme/despertador e abra novamente sua aplicação, aguardando a chegada do alarme/despertar.
- Quando isto acontecer, uma tela com os dados do despertador (um *Dialog*) tomará parcialmente a tela do dispositivo e este ficará em estado de pausa, como mostra a imagem presente no slide seguinte.

ESTUDO DE CASO



ESTUDO DE CASO

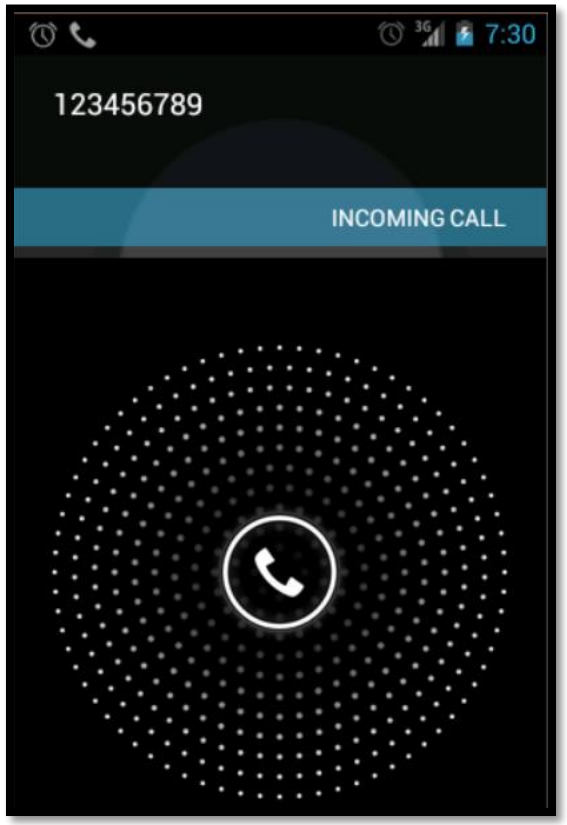
- Para testar os métodos onStop(), onRestart() e onStart(), apresentaremos uma tela que permite simular as características externas do emulador, tais como, realizar ligações, enviar mensagens SMS, entre outras funcionalidades. Para isso acesse o menu **Window > Show View > Other** e na categoria **Android**, selecione **Emulator Control**, conforme imagem ao lado.

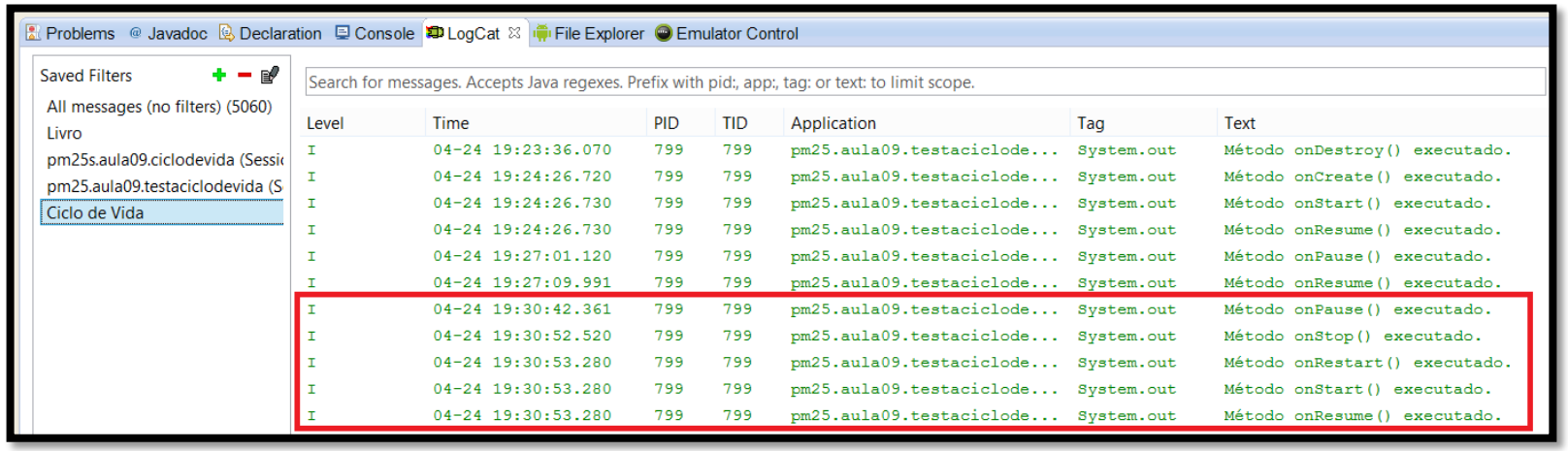


ESTUDO DE CASO

- Desta forma, podemos simular o que acontece com a *Activity* em execução quando o dispositivo recebe uma ligação, por exemplo.
- Para esse teste, basta estar com a *Activity* em execução e preencher os campos **Incoming Number** com o número do telefone que está fazendo a ligação, seguido do clique no botão **Call**.
- O emulador receberá a ligação, conforme apresentado no slide seguinte, e o aplicativo ficará em estado de parado.

ESTUDO DE CASO





ESTUDO DE CASO

- Outra maneira interessante de testar o ciclo de vida de uma aplicação Android é clicando no botão Home, apresentado em destaque na imagem abaixo.

- Com a aplicação em execução, ao clicar na tecla Home, a *Activity* executa os métodos onPause() e onStop(), ficando, assim, em estado de parada. Já a tela do dispositivo apresenta a tela principal do dispositivo, o "desktop".
- Ao retornar ao aplicativo, o mesmo não executa o método onCreate() novamente, pois vem do estado de parado, executando apenas os métodos onRestart(), onStart() e onResume().

ESTUDO DE CASO