Diferente das aplicações Java tradicionais, as aplicações Android possuem pouco controle sobre o seu ciclo de vida, isto é, sobre os estados que assumem desde o momento em que são inicializadas até o momento em que terminam a sua execução. Basicamente, elas monitoram possíveis mudanças no seu estado de execução e reagem a elas de forma apropriada, realizando algum tipo de processamento, como estudaremos adiante.

Neste tópico, conheceremos o ciclo de vida das aplicações Android e aprenderemos como se dá a atribuição de prioridades das aplicações Android.

Uma característica das aplicações Android é funcionar de modo a monitorar as mudanças que podem ocorrer durante o processo de execução. Para facilitar a reação a essas mudanças, os componentes (podem conter funções diferentes e específicas) que compõem as aplicações Android (Activities, Services, Content Providers e Broadcast Receivers) possuem métodos específicos chamados de métodos call-backs, que são invocados pelo Android quando determinadas mudanças no estado da aplicação ocorrem. Ao longo desse curso, nós conheceremos quais são esses métodos e como eles funcionam em cada um dos principais componentes que compõem uma aplicação Android.

As aplicações Android são executadas em processos dedicados, e cada um deles executa essas aplicações em uma instância particular de uma máquina chamada Dalvik, que é uma máquina virtual baseada em registradores.

**Você Sabia?**



A Dalvik foi criada por engenheiros do Google, entre eles, Dan Bornstein, sendo incorporada na plataforma Android e responsável por compilar bytecodes, ou seja, representação intermediária entre o código escrito pelo programador, o texto de código-fonte e de máquina de modo que o código possa ser executado em qualquer dispositivo Android.

No Android, o gerenciamento do processo e da memória é feito exclusivamente pelo Runtime do Android, que é um ambiente responsável por executar as aplicações em tempo de execução. Posteriormente este ambiente substituiu a máquina virtual Dalvik, sendo a principal diferença entre elas o tempo de compilação. Ao utilizar o Android Runtime, a compilação é realizada antes da execução, o que dá maior velocidade para a execução, enquanto que a máquina virtual Dalvik realiza este processo no momento da execução da aplicação.

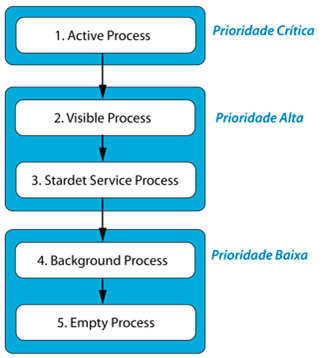
O Android adota uma política bem controlada de gerenciamento de recursos, visto que esta é realizada através do Android Runtime. Isso significa, por exemplo, que processos podem ser removidos da memória, sem qualquer aviso prévio, com o intuito de liberar recursos para outros processos associados a aplicações que possuem uma maior prioridade de execução em um determinado momento. Este processo é realizado ao remover-se da Pilha recursos que têm menor utilização ou prioridade, o que aprenderemos a seguir.

A ordem com a qual os processos são finalizados e removidos da memória, para que recursos sejam liberados, é determinada pela prioridade das aplicações que executam esses processos. Essa prioridade é atribuída com base:

* nas partes (componentes) da aplicação que estão executando, ou seja, nas Activities, services, content providers etc;
* na importância dessas partes para o usuário;
* na quantidade de memória disponível no dispositivo que está executando a aplicação.

A figura a seguir apresenta uma classificação priorizada para os processos, levando em consideração o seu estado de execução, além da sua importância para o usuário e para o sistema em um determinado momento. Na figura a seguir, os processos de maior prioridade são os do topo (Active Process com Prioridade Crítica) e os de menor prioridade são os localizados na parte inferior (Empty Process com Prioridade Baixa).

Atente-se para o conteúdo desta figura:

Figura 1: Prioridades dos processos Android.Fonte: Dell Computadores/LE@D.

Na figura, podemos observar os processos do tipo Active Process (também conhecidos como Foreground Process ou, em português, processo em primeiro plano), que são processos cujos componentes da aplicação estão interagindo com o usuário, por exemplo, uma Activity que o usuário está utilizando no momento. Os processos Active Process possuem Prioridade Crítica, ou seja, têm a maior prioridade possível entre as aplicações em execução, como está sendo representado no topo da figura. O Android tenta preservar os recursos desse tipo de processo evitando o quanto possível que ele seja finalizado e removido da memória do aparelho que está executando a aplicação.

Os processos do tipo Visible Process e Started Service Process possuem Prioridade Alta, ou seja, têm uma grande prioridade entre os processos em execução, abaixo somente dos processos de Prioridade Crítica. Os processos do tipo Visible Process são aqueles cujas Activities estão exibidas, mas inativas. Isso acontece quando uma Activity está visível (ou parcialmente visível), mas não está em primeiro plano ou não pode interagir com usuário. Como exemplo disso, temos a tela de uma aplicação que encontra-se por trás de uma tela lançada como diálogo, o que pode ser conferido na figura 2. Apesar de estar em segundo plano, a aplicação ainda está em execução e em exibição. Este tipo de processo só será removido da memória em condições extremas a fim de liberar recursos para processos do tipo Active Process.

Figura 2: Exemplo de uma tela lançada como diálogo.Fonte: Dell Computadores/LE@D.



De uma forma geral, dizemos que um processo está executando em primeiro plano quando está sujeito à interação direta com o usuário. Por outro lado, dizemos que um processo está executando em segundo plano quando este não está sujeito à interação direta com o usuário.

Os processos do tipo Started Service Process são processos nos quais componentes do tipo service foram inicializados. Nesse caso, como os componentes services não interagem diretamente com o usuário, eles recebem uma prioridade ligeiramente mais baixa que as do Visible Process. As aplicações que executam em Started Service Process são consideradas em primeiro plano, mas podem ser retiradas da memória pelo Runtime do Android a fim de liberar recursos para processos do tipo Visible Process e Active Process, que possuem prioridades superiores.

* Empty Process estão classificados como processos de Prioridade Baixa em relação a todos os outros tipos de processos, são os processos com menor prioridade na plataforma. Os processos do tipo Background Process não possuem Activities exibidas e nem services executando. Esse tipo de processo encontra-se em execução, porém em segundo plano, e é removido da memória sob demanda, utilizando o padrão last-seen-first-killed.

**Você Sabia?**



O padrão last-seen-first-killed (em português, última atividade - primeira a morrer) significa que o último processo que foi exibido para o usuário ou executado em primeiro plano será o primeiro a ser removido quando houver a necessidade de recursos para executar processos que estão em primeiro plano.

Por fim, os processos do tipo Empty Process representam processos que já finalizaram o seu ciclo de vida, porém são mantidos na memória pelo Runtime do Android para melhorar o desempenho do tempo de inicialização da aplicação quando esta for reinicializada. Esse tipo de processo possui a menor prioridade dentre todos os tipos de processo, podendo ser removido da memória sempre que existir a necessidade de recursos para a execução dos demais tipos de processos.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 9

Página 8 de 9

* Neste tópico, conhecemos como funciona o ciclo de uma aplicação Android e o modo como as prioridades são atribuídas aos tipos de processos onde elas são executadas.

Em geral, quase tudo que ocorre na vida estabelece um ciclo de processo, com prioridades predefinidas. Por exemplo, ao acordar pela manhã, qual sua prioridade? Tomar café? Tomar banho? Verificar o e-mail e redes sociais? Cada pessoa possui distintas prioridades e isso é algo subjetivo, que pode variar conforme o perfil da pessoa. Na plataforma Android, o funcionamento se dá de maneira semelhante, porém, por padrão, quem define as prioridades é o próprio sistema operacional Android, que terá o encargo de proporcionar a melhor escalabilidade para os aplicativos em execução. É extremamente relevante ter isso em mente, pois, ao construirmos aplicativos Android, devemos modelar o comportamento deles conforme os ciclos de vida disponibilizados pela plataforma em questão. Não se preocupe! Iremos detalhar passo a passo do fluxo de trabalho que devemos seguir conforme o contexto.

No próximo tópico, estudaremos em detalhes o componente Activity e o seu ciclo de vida. Por meio desse componente, iremos proporcionar bastante dinamicidade para os aplicativos e viabilizar o uso de diferentes telas, melhorando assim sua usabilidade. Entre os componentes, este é um dos mais relevantes para a construção de aplicativos.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 9

Considerando o nível que você se encontra agora, você já deve saber que as Activities representam as telas (interfaces gráficas que possibilitam alguma forma de interação com o usuário) das nossas aplicações, podendo existir mais de uma por aplicação. Além disso, programaticamente, uma Activity deve herdar (estender) da classe Android.app.Activity, ou de alguma subclasse desta, e que são responsáveis por tratar eventos de interface gráfica, como pressionar um botão ou selecionar um item de uma lista. Adicionalmente, verificou-se que toda Activity deve fornecer uma implementação obrigatória para o método onCreate (Bundle bundle), responsável por executar os comandos necessários para a inicialização da aplicação

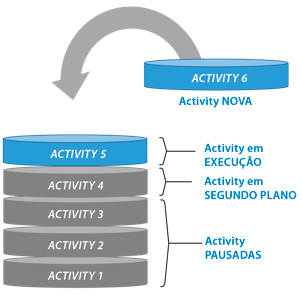
Neste tópico, conheceremos um pouco mais sobre o componente Activity, estudando em detalhes o seu ciclo de vida.

O ciclo de vida do componente Activity compreende os possíveis estados que uma Activity pode alcançar durante a sua execução. São eles:

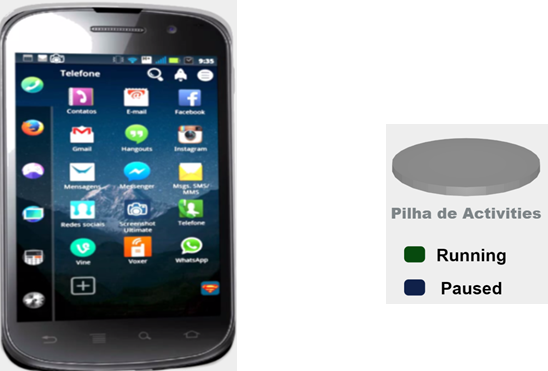
|  |  |
| --- | --- |
| RESUMED | Mais conhecido como Running (em português, Executando), nesse estado, a aplicação está executando em primeiro plano e tem o foco do usuário. |
| PAUSED | Em português, significa Pausado. Nesse estado, a Activity encontra-se temporariamente interrompida, desse modo, outra Activity encontra-se em primeiro plano, porém, a Activity em questão permanece exibida para o usuário. |
| STOPPED | Em português, significa Parado. Nesse estado, a Activity encontra-se totalmente encoberta por outra Activity e, nesse caso, a Activity em questão está executando em segundo plano, mas não está em utilização pelo usuário. |

Antes de conhecermos como funciona o fluxo de mudança de estados dentro do ciclo de vida de uma Activity e quais métodos de call-backs são invocados em cada mudança de estado, precisamos entender o conceito de Activity Stack (em português, Pilha de Activities), utilizado pelo Android para gerenciar as telas da aplicação.

No Android, cada Activity inicializada é colocada no topo da Pilha de Activities, como ilustrado na figura abaixo. Desse modo, toda vez que uma Activity é criada e inserida no topo da Pilha, a que estava em execução fica logo abaixo da nova Activity. Em função da sua posição na Pilha de Activities, uma delas pode assumir um possível estado, por exemplo, uma que esteja no topo da Pilha encontra-se em execução (estado Executando), já as demais podem estar executando em segundo plano ou podem estar nos estados Pausado e Parado. Quando a tecla BACK (tecla “voltar”) do dispositivo móvel é pressionada, a Activity que encontra-se no topo da Pilha é removida, e a Activity que encontra-se logo abaixo dessa, volta a executar em primeiro plano, passando a ser exibida para o usuário. Observe a figura, que melhor ilustra o raciocínio descrito.

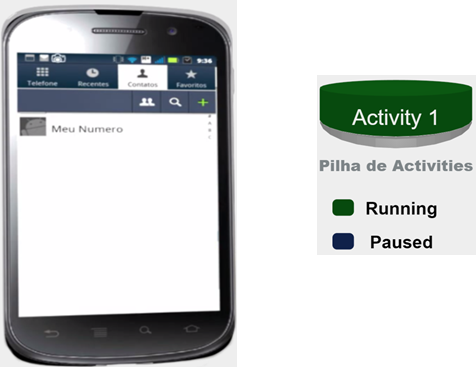
Figura 3: Pilha de Activities no Android.Fonte: Dell Computadores/LE@D.

Agora imagine uma aplicação que dê acesso aos contatos do telefone do usuário. A partir desta, vamos observar como o Android lida com a Pilha de Activities em relação às suas prioridades e ordens de execução. Observe a figura. Na aplicação, a Pilha de Activities está totalmente vazia, ou seja, não temos aplicação alguma em execução no momento. Perceba como isso é representado através do disco à direita do smartphone e note que a Activity é representada pelo disco.

Figura 4: Pilha de Activity vazia.Fonte: Dell Computadores/LE@D.

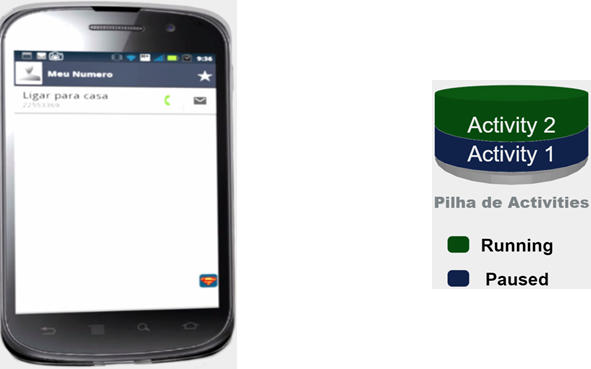
Vamos observar adiante o que acontece ao abrirmos a nossa primeira Activity e como se comportará a Pilha de Activities a partir disso.

Ao executarmos a aplicação de contatos, ocorre que a primeira Activity é aberta, esta é, então, adicionada à Pilha de Activities, sendo a primeira em nossa Pilha, representada por um disco na Pilha. Ao ser colocado na fila, a aplicação abre a Activity requisitada. Note na figura:

Figura 5: Activity 1 adicionada à Pilha de Activities.Fonte: Dell Computadores/LE@D.

Agora que temos a primeira Activity em execução, vamos abrir uma próxima Activity em nossa aplicação de contatos.

Podemos abrir uma nova funcionalidade desta aplicação, por exemplo, pressionar um dos contatos existentes na lista de contatos. Com esta ação, a nova Activity sobrepõe a anterior e é também adicionada à Pilha, representada por um novo disco que sobrepõe o disco anterior, já que este está em execução acima da Activity que estava sendo executada no momento.

Figura 6: Activity 2 adicionada à Pilha de Activities.Fonte: Dell Computadores/LE@D.

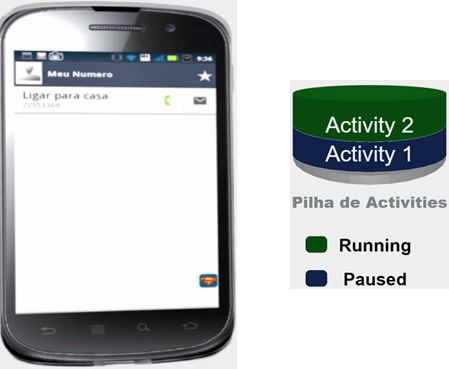
Que tal adicionarmos ainda mais uma Activity em nossa Pilha, para podermos observar o resultado? Em uma situação real, o usuário abrirá diversas Activities em uma aplicação enquanto navega.

O mesmo processo deve ocorrer caso seja aberta uma nova funcionalidade da aplicação, por exemplo, enviar uma mensagem para o contato atualmente aberto. Teremos uma nova Activity em execução e esta será adicionada à Pilha de Activities, que pode ser observada através do disco adicionado ao topo da Pilha.

Figura 7: Activity 3 adicionada à Pilha de Activities.Fonte: Dell Computadores/LE@D.

Repare que agora temos três discos na Pilha, ou seja, três Activities, entre estas, temos uma em execução, que está no topo, e outras duas em estado de espera, abaixo da Activities atual.

No caso de pressionarmos o botão de retorno, para a Activity anterior, então a Activity atualmente aberta será removida da Pilha, exibindo a que está logo abaixo, até que todas as Activities sejam removidas da Pilha, fechando a aplicação, ou seja, cada disco da Pilha de Activities será removido da Pilha, até que esta esteja novamente como no início: vazia.

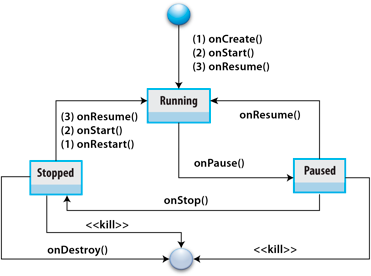
Figura 8: Activity 3 removida da Pilha de Activities.Fonte: Dell Computadores/LE@D.

Todo este processo é organizado através da plataforma Android pelo Android Runtime, como comentado anteriormente, para isto, existem métodos que auxiliam a criação de aplicações que lidam com esta Pilha de Activity.

Como observamos, fica a cargo do Runtime do Android remover as Activities da memória e liberar seus recursos. Desse modo, se não prepararmos as Activities da nossa aplicação para lidar com essas mudanças de estado, elas podem sofrer perdas de dados ou informações, implicando em aborrecimentos ou prejuízos para os usuários. Os métodos de call-back do componente Activity nos permite preparar as Activities para lidar com essas mudanças de estado. Esses métodos são: onCreate(), onStart(), onRestart(), onResume(), onPause(), onStop() e onDestroy().

A figura a seguir ilustra o ciclo de vida do componente Activity. O círculo escuro na parte superior da figura indica a inicialização da Activity e o círculo claro na parte inferior indica a finalização da Activity, momento em que ela é removida da memória. Os retângulos com extremidades arredondadas representam os estados que as Activities podem alcançar e as setas, entre os estados, indicam transições entre estados.

Nas transições entre estados, aparecem nomes dos métodos de call-backs, o que indica quais métodos da Activity são invocados durante a mudança de estado. Nos casos em que mais de um método é invocado, a numeração indica a ordem na qual esses métodos são invocados.

Figura 9: Ciclo de vida do componente Activity.Fonte: Dell Computadores/LE@D.

Para compreender melhor o ciclo de vida de uma Activity, podemos dividi-lo em três subníveis, que se repetirão durante toda a execução da aplicação. São eles: entire lifetime, visible lifetime e foreground lifetime. Vamos conhecer cada um desses subníveis a seguir:

* Entire lifetime: representa o ciclo de vida completo entre o início e a finalização da Activity. Ocorre apenas uma vez durante a execução da Activity. Ele compreende o que ocorre entre a invocação dos métodos onCreate() e onDestroy(). Você pode retornar à figura anterior para observar isso melhor. Note que esses métodos são invocados apenas uma única vez: no momento em que a Activity é criada (onCreate()) e no momento em que ela é finalizada (onDestroy()). Dessa forma, você pode programar sua Activity para fazer alguma configuração global no método onCreate() e fazer a liberação de algum recurso específico no método onDestroy(). Por exemplo, se a sua aplicação possuir uma thread que é executada em segundo plano, realizando um download da internet, você pode iniciar essa thread no método onCreate() e finalizá-la dentro do método onDestroy();
* Visible lifetime: nesse subnível, a Activity já encontra-se inicializada, porém pode estar no topo da Pilha de Activities, interagindo com o usuário ou temporariamente parada, executando em segundo plano. Esse ciclo compreende as transições que podem ocorrer entre a invocação dos métodos onStart() e onStop(), exemplificados na figura anterior. Durante esse ciclo, a Activity sempre está visível para o usuário, porém pode não estar em primeiro plano para que o usuário possa interagir com ela. De uma forma geral, esse ciclo compreende todo o período em que a Activity encontra-se no topo da Pilha de Activities ou executando em segundo plano à espera de que outra Activity conclua a sua execução. Nesse ciclo, por exemplo, podemos registrar a Activity em um broadcast receiver (componentes responsáveis por receber e tratar eventos provenientes do sistema ou de outras aplicações) no método onStart(), para monitorar mudanças que podem impactar na interface do usuário e fazer o cancelamento do registro no método onStop(), quando o usuário não conseguir mais visualizar o que está sendo exibido;
  + Foreground lifetime: nesse subnível, a Activity encontra-se no topo da Pilha de Activities, em pronta exibição e em interação com o usuário. Esse ciclo compreende sequências de invocações dos métodos onPause() e onResume(). Desse modo, o estado da Activity pode alternar entre os estados executando (running) e pausado (paused). Por exemplo, o método onPause() da Activity pode ser invocado quando outra aplicação for iniciada, quando o usuário atender a uma chamada telefônica ou mesmo quando o dispositivo móvel entrar em modo de repouso para economizar bateria. Em qualquer que seja o caso, o usuário pode ativar a Activity e o método onResume() é invocado para continuar a execução.

**Atenção**



A especificação da plataforma Android sugere que os métodos onPause() e onResume() contenham implementações “leves” para não causar uma sensação de demora para o usuário, uma vez que são executados várias vezes.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página12 de 16

Página 12 de 16

Para entender um pouco melhor a função de cada um dos métodos de call-backs do componente Activity, vamos descrever sobre cada um de forma mais detalhada.

|  |  |
| --- | --- |
| onCreate() | É um método obrigatório e é executado uma única vez durante o ciclo de vida da Activity. É nesse método que é criado uma View que representa a interface gráfica da Activity. Após a sua execução, o runtime do Android invoca o método onStart(). |
| onStart() | Esse método é invocado antes que a Activity possa ser exibida para o usuário. Ele pode ser invocado depois do método onCreate(), quando trata-se da inicialização da Activity, ou depois do método onRestart(), quando a Activity encontra-se no estado stopped (parado) e transita para o estado running (executando), como foi ilustrado na figura do ciclo de vida. Após sua execução, o método onResume() é imediatamente invocado. |
| onRestart() | Esse método é invocado quando uma Activity for temporariamente parada e depois necessitar ser reiniciada. Observe que o método é invocado durante a transição entre os estados stopped e running, como ilustrado na figura do ciclo de vida. Após o término de sua execução, o método onStart() é imediatamente invocado. |
| onResume() | Esse método é invocado quando a Activity encontra-se no topo da Pilha de Activities e está pronta para interagir com o usuário. Após executar esse método, a Activity assume os estados running e o único método que pode ser invocado após isso é o método onPause(), conforme abordado na figura do ciclo de vida. |

|  |  |
| --- | --- |
| onPause() | Esse método é invocado quando algum evento deseja pausar a aplicação, por exemplo, o celular entra em modo de descanso para economia de bateria e pausa a aplicação ou quando uma nova aplicação é iniciada. Isso significa que a Activity que estava executando no topo da Pilha de Activities será interrompida temporariamente. A invocação desse método é útil, pois permite que a Activity salve o estado da aplicação para que este possa ser recuperado, se necessário, quando a Activity voltar a executar. Nesse caso, o processo de recuperação do estado da aplicação ocorre dentro do método onResume(), durante a transição entre os estados paused e running, conforme ilustrado na figura do ciclo de vida. |
| onStop() | Esse método é invocado quando a Activity está sendo finalizada e não é mais exibida para o usuário. Essa transição consiste na mudança de estado da Activity de paused para stopped. Mesmo no estado stopped, a Activity pode ser reiniciada. Nesse caso, o método onRestart() é invocado, caso contrário, se a Activity passar muito tempo no estado stopped e houver necessidade de recursos por parte do Runtime do Android, ela pode ser finalizada, para isso, o método onDestroy() é invocado automaticamente e a Activity é removida da Pilha de Activities. |
| onDestroy() | Esse último método é invocado no final do ciclo de vida da Activity. Ele pode ser invocado automaticamente pelo Runtime do Android ou pela própria aplicação por meio do método finish() da classe Activity. Após sua execução a Activity é removida por completo da Pilha de Activities e o seu processo é encerrado pelo sistema operacional do Android. |

Agora que nós já temos uma boa ideia sobre o ciclo de uma Activity, por que não criamos uma? Realizaremos, como exemplo, a execução de um Hello Wolrd na plataforma Android, através do Android Studio. Não se engane, uma classe Activity é a base para a execução de nossa aplicação para dispositivos móveis. Pronto para se aventurar? Vamos lá!

### 2.1 Manipulação de Activity?

Além de conhecermos o fluxo de vida de uma Activity, também devemos ter em mente como construir e manipulá-la. O passo inicial é realizar a importação do pacote android.app.Activity. Por meio deste, teremos acesso à classe pai Activity, responsável por proporcionar acesso aos métodos do ciclo de vida estudados neste tópico.

Ao realizar a herança dessa classe, iremos observar que seremos obrigados a sobrescrever os métodos do ciclo de vida. O que será colocado em cada um poderá variar conforme o domínio de sua aplicação e também necessidade, mas tenha em mente que a estrutura é fixa e se dará da forma que é exibida no código [deste link](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551484684/aula/code/codigo1-link.txt).

Neste tópico, estudamos em detalhes o ciclo de vida do componente Activity. Conhecemos os estados que uma Activity pode alcançar e quais métodos de call-backs são invocados a cada mudança de estado que ocorre. Além disso, estudamos como funciona a Pilha de Activities. Por fim, durante a descrição dos métodos de call-backs, conhecemos alguns tipos de eventos que podem levar uma Activity a mudar de estado.

Tivemos a oportunidade de observar os ciclos e métodos que compõem a construção de uma acitivity. Devemos ter em mente essa estrutura, pois em grande parte dos casos ela é utilizada como padrão.

No próximo tópico, estudaremos como ocorre a utilização da ferramenta LogCat para imprimir informações sobre a execução de aplicações Android. Perceberemos que o acompanhamento e debug da nossa aplicação será uma grande chave para identificação de problemas e ajuda para solução.

## TÓPICO 3 – Ferramenta LogCat

Neste último tópico, conheceremos a ferramenta LogCat que utilizaremos para imprimir informações sobre a execução de aplicações Android e acompanharemos a forma mais adequada de imprimir mensagens no console da aplicação, o que se faz bastante importante, pois conheceremos o status atual da aplicação e, dessa forma, poderemos observar via terminal valores atuais dos objetos, além de imprimir mensagens de erros ou de sucesso.

Através desta ferramenta, podemos obter logs do sistema de forma prática e fácil, facilitando, assim, a correção de erros e remoção de dúvidas sobre o estado da aplicação.

Em um programa Java, quando desejamos imprimir alguma mensagem no terminal, utilizamos o comando System.out.println(). Na plataforma Android, essa tarefa é feita de uma forma diferente, pois nossas aplicações executam em um dispositivo móvel ou em um emulador. O que ocorre é que, quando usamos o comando System.out.println(), a Dalvik direciona a saída para um arquivo dentro da pasta dev/null.

Para imprimirmos mensagens de forma eficiente na plataforma Android, devemos utilizar a classe Android.util.Log. Assim, mensagens podem ser impressas com níveis de detalhes que são úteis para o desenvolvedor no acompanhamento do fluxo de execução da aplicação. A classe Android.util.Log permite que sejam criados logs de informação, debug (depuração), alertas e erros. Desse modo, é possível criar mensagens específicas e, posteriormente, recuperar apenas aquelas que desejamos.

Quando utilizamos a classeAandroid.util.Log, as mensagens geradas são enviadas para a ferramenta LogCat. O LogCat é uma ferramenta que vem junto com o Android SDK e tem como função gerenciar todos os logs gerados pelo sistema operacional do Android. Para que o LogCat seja exibido dentro do Android Studio, basta navegar até o Android Monitor no canto inferior da IDE, ou através do atalho Alt + 6. No campo de busca, utilizaremos a Activity ExemploLogCat, do código abaixo, para ilustrar o uso da classe android.util.Log, mostrando como é feito o registro de mensagens de log com diferentes níveis de severidades e como podemos ativar a exibição desses logs dentro do LogCat.

01 import android.app.Activity;

02 import android.os.Bundle;

03 import android.util.Log;

04 public class ExemploLogCat extends Activity {

05 private static final String FILTRO = "ExemploLogCat";

06 protected void onCreate (Bundle savedInstanceState) {

07 super.onCreate(savedInstanceState);

08 setContentView(R.layout.activity\_exemplo\_log\_cat);

09 // Verbose

10 Log.v(FILTRO, "Log de Verbose!");

11 // Debug

12 Log.d(FILTRO, "Log de Debug!");

13 // Info

14 Log.i(FILTRO, "Log de Info!");

15 // Warn

16 Log.w(FILTRO, "Log de Alerta!");

17 // Error

18 Log.e(FILTRO, "Log de ERRO!", new RuntimeException());

A Activity ExemploLogCat, no código anterior, utiliza métodos da classe Log (android.util.Log) para registrar mensagens da aplicação. Observe que nela foi definida uma constante do tipo String com nome FILTRO, que possui valor igual a “ExemploLogCat”. As chamadas aos métodos da classe Log estão dentro do método onCreate() de ExemploLogCat. Os métodos da classe Log são simples, seus nomes são exatamente a primeira letra do nível de severidade da mensagem que está sendo registrada. Por exemplo, para imprimir uma mensagem com nível de severidade de informação (info), utilizamos Log.i, categoria, mensagem que funciona como um identificador da mensagem, seu tipo e o campo da mensagem, representando o texto que se deseja registrar no log, ou seja, que será apresentado ao usuário. No caso da Activity ExemploLogCat, a nossa constante FILTRO representa a categoria, e “Log de Info!” representa a mensagem na chamada de método: Log.i(FILTRO, "Log de Info!"). A categoria do log é útil para uma posterior filtragem das mensagens na ferramenta LogCat.

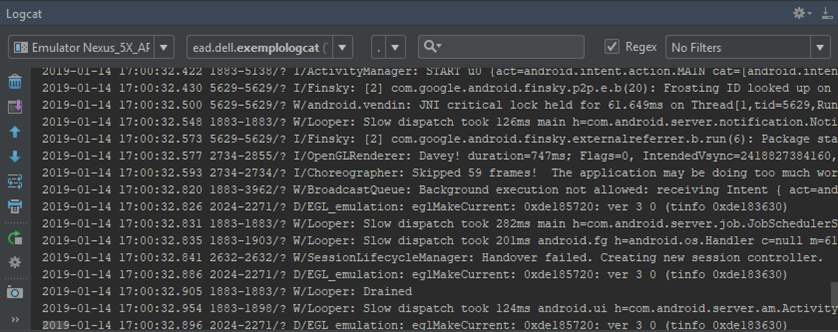
Como mencionado anteriormente, as mensagens de log possuem diferentes níveis de severidade. Os principais níveis de severidade são: debug (depuração), verbose (literal), info (informação), warn (alerta) e error (erro). Para facilitar a legibilidade, as mensagens de log em cada um desses níveis de severidade são exibidas em uma coloração diferente dentro da ferramenta LogCat. Além disso, cada nível de severidade possui uma sigla, representada pela primeira letra do nível de severidade. O quadro a seguir sumariza essa relação entre os níveis de severidade e suas respectivas siglas e cores.

Quadro 1: Níveis dos logs e suas respectivas coloração no LogCat.

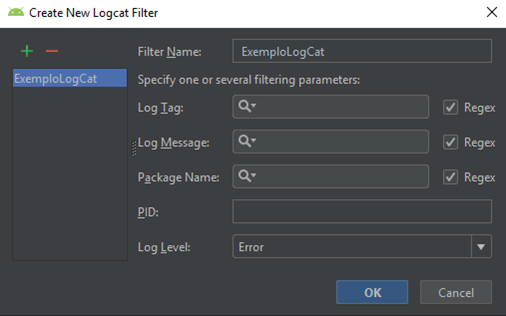
|  |  |  |
| --- | --- | --- |
| **Nível** | **Sigla** | **Cor** |
| Debug | D | Azul |
| Verbose | V | Preto |
| Info | I | Verde |
| Warn | W | Laranja |
| Error | E | Vermelho |

Fonte: Dell Computadores/LE@D

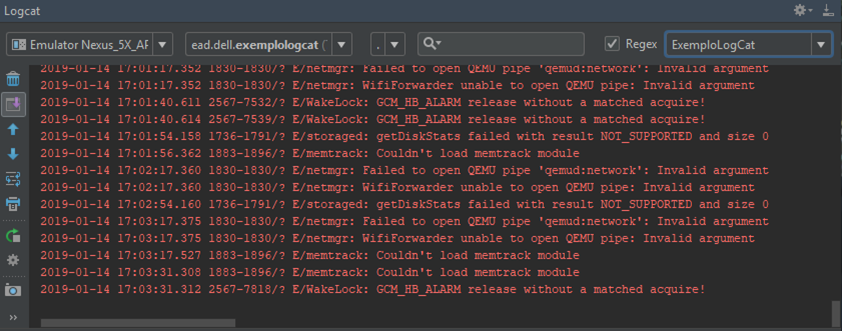
Após executarmos a nossa aplicação ExemploLogCat, o registro dos logs na ferramenta LogCat dentro do IDE Android Studio é exibido, como na figura a seguir. Observe que os logs exibidos não são apenas aqueles feitos pela nossa aplicação, mas todos os logs gerados pelo sistema operacional. Pode-se observar o campo level, apresentado anteriormente, de acordo com o tipo de log que está sendo apresentado; o campo time, que representa o tempo que o log foi exibido; e o campo PID, que representa o ID do processo que gerou o log. Também podemos notar o TID, que é o identificador de Thread caso a aplicação utilize esta tecnologia; a Aplication, que representa o campo que refere-se ao nome da aplicação; a Tag, que apresenta a tag que o log está sendo atribuído; e, por último, o campo text, que apresenta a mensagem do log para o usuário.

Figura 10: Janela do LogCat com todos os logs.Fonte: Dell Computadores/LE@D.

A mistura de logs do sistema operacional com os da nossa aplicação pode se tornar um problema, dificultando a identificação das mensagens que desejamos. Contudo, criamos um identificador (a constante FILTRO) que nos permite identificar exatamente quais são as mensagens geradas pela nossa aplicação. Dessa forma, podemos criar um filtro dentro do LogCat para selecionar apenas as mensagens que nos interessam. Isto é, aquelas que possuem a identificação “ExemploLogCat”. Para isso, basta pressionar o botão de adicionar filtro (apresentado no canto direito superior da janela de logs da figura anterior). Após pressionar o botão, a tela para criação do filtro irá aparecer. No nosso exemplo, demos o nome do filtro de “Filtro ExemploLogCat” e no campo “Log Tag” colocamos o nome “ExemploLogCat”. Também definiremos que este nosso filtro deverá mostrar apenas as mensagens de erro obtidas na execução do código através da opção Log Level. Observe a figura abaixo. Após definir o que o filtro deve exibir, basta pressionar o botão “OK”, então o filtro será criado.

Figura 11: Criação de filtro.Fonte: Dell Computadores/LE@D.

A figura a seguir  exibe os logs gerados apenas pela nossa aplicação dentro da ferramenta LogCat. Observe que essas mensagens de log foram filtradas com o filtro que acabamos de criar, o “Filtro ExemploLogCat”, e que são exibidas apenas mensagens de erros, como definimos anteriormente.

Figura 12: Janela do LogCat com os logs filtrados.Fonte: Dell Computadores/LE@D.

Como observado na figura anterior, temos a possibilidade de criar mensagens de retorno no sistema de forma que teremos um maior controle do que está ocorrendo. Estas mensagens, ou logs, podem ser filtradas através da IDE, como exibido anteriormente, assim, podemos buscar por termos ou qualquer atributos desta mensagem, por exemplo, nível da mensagem, hora de exibição e texto contido. Esta filtragem ocorre através de comparação do texto buscado e o texto contido no log.

Desta forma, através dos logs, podemos procurar e ajeitar problemas no código através das mensagens de retorno, observar o fluxo das Activities e observar o fluxo de dados da aplicação.

Neste tópico, nós aprendemos a imprimir mensagens de forma padrão dentro das aplicações Android. Realizamos esse processo através do LogCat, que pode apresentar as mensagens de forma organizada através de vários campos e filtros. Também aprendemos a utilizar a ferramenta LogCat para gerenciar e fazer exibir as mensagens de log geradas pela nossa aplicação e pelo sistema operacional Android.

Através desta ferramentas, podemos não só criar logs com diferentes mensagens, mas também filtros que dividem os diferentes tipos de mensagens que podemos obter na execução de uma aplicação Android.



Chegamos ao final de nossa aula. Nela, conhecemos como funciona o ciclo de vida de uma aplicação Android e como prioridades são atribuídas aos tipos de processos onde elas executam. Além disso, estudamos em detalhes o funcionamento do componente Activity e o seu ciclo de vida. Por fim, aprendemos a imprimir mensagens de log e a utilizar a ferramenta LogCat para gerenciar e ativar a exibição desses logs. Na próxima aula, estudaremos em detalhe o funcionamento das intents. Bons estudos e até lá!

* Android Development Site: Activities. Disponível em: <http://developer.android.com/guide/components/Activities.html>. Acesso em: 06 dez. 2018.
* LECHATA, R. R. Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 3. ed. São Paulo: Novatec Editora, 2013. 824 p.
* MEIER, R. Professional AndroidTM 4 Application Development. Indianapolis: Wiley Publishing, 2013. 864 p.
* MONTEIRO, J. B. Google Android: crie aplicações para celulares e tablets. 1. ed. São Paulo: Casa do Código, 2012. 250 p.

## TÓPICO 1 – O que são Intents?

**Objetivos**

* Compreender a função das Intents;
* Conhecer os atributos e métodos necessários para o uso de Intents.

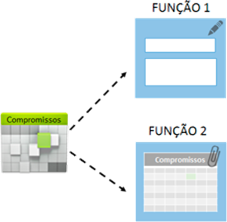
Você já deve saber que uma aplicação Android pode ter diversas Activities e que elas representam um ponto de interação com o usuário, ou seja, a principal característica de uma Activity é possuir uma interface gráfica.

Conforme os requisitos do sistema, nosso aplicativo poderá conter várias interfaces, mas como podemos configurar o aplicativo para que o usuário navegue entre as telas ou possa voltar para uma tela já encerrada?

Para que isso possa ocorrer, devemos esclarecer, conceitualmente e também na prática, o que são as Intents. Abordaremos, neste tópico, sua definição a fim de compreendermos sua função e de conhecer os atributos e métodos necessários para seu uso.

De fato, uma aplicação pode ter várias Activities. Cada uma dessas Activities normalmente equivale a um ponto diferente da aplicação, ou seja, cada Activity tem relação com uma ação específica que a aplicação pode executar.

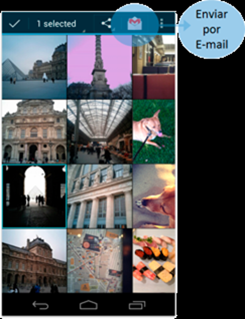
Vamos entender, a partir de um exemplo, que essa necessidade de várias funções de uma aplicação irá obrigar que nossa aplicação tenha várias telas para distribuir as funcionalidades de maneira adequada e com boa usabilidade para o usuário. Imagine que você precisa desenvolver uma aplicação para agendar compromissos. Esta aplicação deve ter pelo menos duas funções distintas: uma para criar e modificar compromissos, e outra para exibir a lista de compromissos existentes. Cada uma dessas funções é, de certa forma, independente, portanto pode ser codificada em Activities independentes.

Figura 1: Aplicação para agendar compromissoFonte: Dell Computadores/LE@D.

Você pode estar se perguntando: qual a vantagem de implementar os módulos da aplicação em Activities separadas?

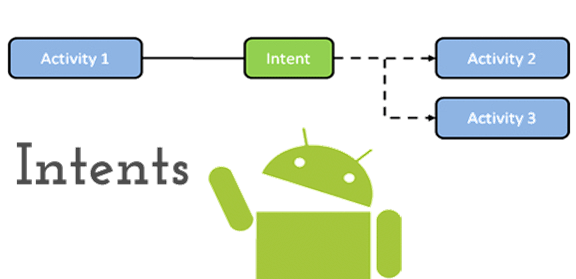
Implementando aplicações com Activities independentes, podemos permitir que o usuário entre nessa aplicação a partir da Activity mais adequada para ele no momento. Assim, para o nosso exemplo anterior, quando o usuário quiser apenas criar um novo compromisso, ele não será obrigado a acessar a lista de compromissos primeiro, para, só depois, poder criar um compromisso, por exemplo.

Se você tem ou já teve um dispositivo Android, deve conhecer esse tipo de comportamento citado no exemplo. Então você entender que, para enviar por e-mail uma foto que você acabou de tirar, por exemplo, você não precisa necessariamente abrir a aplicação de e-mail e procurar o botão “Novo E-mail”. Diretamente do aplicativo de Galeria é possível executar a aplicação de e-mail na tela de composição de mensagem, inclusive com a imagem já anexada. Perceba a realização disso na figura a seguir.

Figura 2: Compartilhamento de foto via e-mail.Fonte: Dell Computadores/LE@D.

Tudo isso é possível por conta das Intents. A Intent representa a intenção de executar um componente, seja ele Activity, Service ou Broadcast Receiver. Intents trabalham como uma espécie de fio que liga componentes independentes de uma mesma aplicação Android ou até mesmo de aplicações distintas, fazendo parecer que todos são um só.

A Intent funciona da seguinte forma: determinada aplicação deseja ativar um componente, seja ele Activity, Service ou Broadcast Receiver. Essa aplicação informa sua intenção de acesso ao sistema operacional Android, então este último trata a solicitação, encontra o componente e o executa. Também é possível passar dados para esse componente que vai ser executado, mas isso é algo que abordaremos depois, então você poderá concluir que essa é uma das principais vantagens das Intents. Agora observe a figura a seguir.

Figura 3: Exemplo de funcionamento de uma Intent.Fonte: Dell Computadores/LE@D.

Perceba que a Activity 1, por meio da Intent, tenta interagir com outras duas Activities, Activity 2 e Activity 3. A função das Intents é exatamente esta: proporcionar uma interação entre componentes, no caso deste exemplo, entre Activities.

Perceba, na figura 3, que uma Activity não ativa outra diretamente. Esta é uma tarefa do sistema operacional Android. Este detalhe é essencial para manter a flexibilidade do sistema, pois permite que o Android perceba todas as Activities disponíveis e trate a Intent da melhor forma possível, podendo até mesmo oferecer mais uma opção de Activity para o usuário

**Você Sabia?**



Você já tentou abrir um link ou determinado tipo de arquivo em um dispositivo Android e foi apresentado a uma lista de aplicações que poderiam fazer esta tarefa para você? Isso ocorre porque o Android, via tratamento de Intents, consegue apontar que há mais de uma aplicação para fazer o que você precisa e lhe dá a chance de escolher o que mais for de seu agrado. No próximo tópico, detalharemos melhor essa possibilidade.

No entanto, como o Android encontra o componente que queremos ativar dentre todos os demais? Ele analisa todos os dados que são anexados a uma Intent. Uma Intent pode ser considerada uma estrutura de dados que pode carregar várias informações sobre o componente que deseja executar. Essas informações são distribuídas em atributos que são descritos no quadro a seguir.

Quadro 1: Descrição dos atributos de uma Intent.

|  |  |
| --- | --- |
| **Atributo** | **Descrição** |
| Component Name (Nome do Componente | Corresponde ao nome da classe (incluindo o pacote) que representa o componente que se deseja ativar caso a classe seja conhecida. Quando este atributo está em branco, o Android analisa as demais informações para encontrar o componente buscado. |
| Action (Ação) | Este atributo permite ao Android compreender se a Intent em questão está direcionada para uma Activity, um Service ou um Broadcast Receiver. Existem algumas Actions padrão do Android como ACTION\_CALL (efetua uma chamada), ACTION\_PASTE (colar um conteúdo), entre outros, mas o desenvolvedor pode criar suas próprias actions*.* |
| Data (Dados) | Corresponde ao dado principal que é carregado pela Intent e que deve ser usado na ativação do componente, como uma mensagem ou identificador de contato. Este atributo também pode estar em branco. |
| Category (Categoria) | Este atributo ajuda o Android a identificar o componente buscado em uma das suas categorias predefinidas, como CATEGORY\_HOME (aplicações que estão na tela principal) ou CATEGORY\_ALTERNATIVE (aplicações que devem ser consideradas como opção para ações do usuário). |
| Type (Tipo) | Identifica o tipo de dado do conteúdo que está no atributo Data. |
| Extra | Este atributo pode conter uma lista de valores auxiliares que podem ser usados pelocomponente ativado pela Intent, complementando a informação do atributo Data. |

Fonte: Dell Computadores/LE@D

Como você pôde perceber na descrição do atributo Component Name, uma vez que o nome do componente alvo da Intent é conhecido, fica fácil para o sistema operacional encontrar tal componente. Uma Intent que possui o nome do componente alvo definido é chamada de Intent Explícita. Quando o atributo não está disponível, dizemos que se trata de uma Intent Implícita e, nesse caso, o Android irá precisar dos outros atributos para localizar o componente alvo no sistema. Abordaremos o funcionamento dessa busca no próximo tópico, no qual falamos sobre Intent Filters.

Em termos de programação Java, uma Intent é, na verdade, um objeto da classe Intent (android.content.Intent). Após instanciar uma Intent, devemos dar a ela todas as informações necessárias para encontrar o componente que desejamos. Depois, utilizamos um dos métodos disponíveis para disparar a Intent (fazendo-a ser executada), dependendo do seu componente alvo, como descrito no quadro adiante, que relaciona os métodos de execução das Intents conforme seu componente.

Quadro 2: Métodos utilizados para a ativação de componentes via Intents.

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| startActivity(Intent) | Ativa um componente do tipo Activity baseado na Intent fornecida. |
| startService(Intent) | Ativa um componente do tipo Service conforme a Intent enviada como parâmetro. |
| sendBroadcast(Intent) | Ativa um componente do tipo Broadcast Receiver segundo a Intent fornecida. |

Fonte: Dell Computadores/LE@D

Todos os métodos descritos no quadro anterior fazem parte da classe Context. Portanto, a partir de uma subclasse de Activity, ou seja, qualquer Activity em execução, é possível executar estes métodos sem a necessidade de um objeto do tipo Context.

## TÓPICO 2 – Intent Filters

**Objetivo**

* Conhecer os tipos de Intents;
* Compreender o que são Intent Filters;
* Entender como os componentes são identificados através de Intent Filters;
* Conhecer a metodologia de teste de combinação dos Intent Filters.

No tópico anterior, conhecemos as Intents e seu papel dentro da plataforma Android, que é fazer de forma flexível a ligação entre componentes. Também aprendemos que uma Intent possui diversos atributos que servem para que o sistema operacional Android encontre um componente que possa tratá-lo em uma das aplicações instaladas no dispositivo.

Um componente pode se registrar por meio de um Intent Filter para uma ação específica e dados objetivos. Um Intent Filter determina os tipos de Intents às quais uma Activity, Service ou Broadcast Receiver podem responder.

Imagine que, em sua Activity, você deseja tirar uma foto do usuário. O que é mais fácil? Desenvolver o código para tirar uma foto do zero ou simplesmente solicitar ao aplicativo de câmera do dispositivo que registre uma foto por nós e retorne para o aplicativo? A seguir, vamos entender como os Intent Filters torna aplicações como essas possíveis!

### 2.1 Tipos de Intents

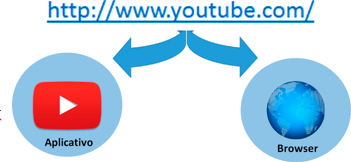
Podemos dizer que existem dois tipos de Intents: Explícitas, quando o nome do componente está disponível dentro do nosso aplicativo, e Implícitas, quando o nome do componente não é informado.

Toda Intent Explícita é entregue diretamente ao seu componente de destino, contanto que este esteja instalado no dispositivo Android.

Você pode registrar seus componentes do Android por meio de Intent Filters para determinados eventos. Se um componente não define o tipo de serviço que ele pode prover, logo ele só poderá ser chamado por Intents Explícitas. Por exemplo, seu aplicativo pode iniciar um componente do navegador web para uma determinada URL por meio de uma Intent, pois o navegador possui um Intent Filter que pode ser acionado quando o sistema operacional Android identifica uma URL que foi selecionada pelo usuário. Assim, quando temos um componente que pode ser acionado por outros componentes, dizemos que o primeiro possui uma Intent Implícita. Por outro lado, quando temos um componente que não pode ser acionado por Intents de outros aplicativos, denomina-se Intent Explícita, que ocorre geralmente dentro de uma mesma aplicação.

Vamos a um exemplo: você está utilizando uma aplicação que possui uma tela de login para validar sua identidade antes de acessar a aplicação em si. A tela de login corresponde a uma Activity, pois se trata de uma tela na qual ocorre interação com o usuário. Após o processo de login, esta Activity dispara uma Intent que solicita a ativação da Activity principal do aplicativo, e é por essa ativação que você deve ter acesso a todas as funcionalidades.

Então você pode se questionar sobre a utilidade ou mesmo sobre a necessidade de uma Intent Implícita que não tem um componente alvo predefinido. Porém, este é um tipo muito comum de Intent e você, provavelmente, já se deparou com alguma delas sem perceber, enquanto utilizava um dispositivo Android. Por exemplo, se você navegar na internet em um dispositivo Android e acessar um link que aponta para um vídeo no YouTube, provavelmente o sistema operacional vai disponibilizar a você uma janela com, pelo menos, duas opções para abrir o link, podendo ser um browser (navegador web) instalado no dispositivo e o aplicativo do YouTube (que geralmente já vem instalado nos dispositivos Android). Isto é possível devido ao uso de Intent Filters.

Figura 4: Opções comuns do Android para abrir um link.

### 2.2 Como Ocorre a Resolução de Intents?

Quando uma nova Intent é criada, o Android a intercepta e tem que encontrar um componente que possa tratá-la. Então ele carrega a lista de Intent Filters dos componentes instalados no dispositivo e começa a comparar as características deles com as características da Intent criada. Por fim, gera uma lista de componentes que podem tratar essa Intent, contendo os componentes que passaram pelo processo de filtragem.

Um detalhe importante é que o primeiro nível de separação da filtragem é baseado no tipo de componente que a Intent precisa. Se a Intent em questão visa ativar uma Activity para exibir uma nova tela ao usuário, por exemplo, a filtragem não analisa Services e Broadcast Receivers. Da mesma forma, se uma Intent é voltada para um Broadcast Receiver para notificar o usuário sobre o status da bateria, por exemplo, Activities e Services não são levados em consideração, e assim por diante.

### 2.3 O que são Intent Filters?

Intent Filters funcionam como filtros, contendo descrições estruturadas de Intents que podem ser tratadas pelos componentes aos quais pertencem. Os Intent Filters são definidos dentro da declaração de Activities, Services e Broadcast Receivers no arquivo AndroidManifest.xml, através da tag.

Quando uma Intent precisa ativar um componente, mas não conhece seu nome exatamente, ou quando intencionalmente existe o interesse em deixar que diversos componentes tenham a chance de tratar a Intent, criamos uma Intent Implícita. De todos os atributos desta, apenas Action (Ação), Data (Dados) e Category (Categoria) são levados em consideração no processo de seleção do componente que pode atendê-la. Chamamos esse processo de Resolução de Intent (do inglês, Intent Resolution). Que tal definirmos esses três atributos? Vamos lá!

|  |  |
| --- | --- |
| Category | String que contém informações adicionais sobre o tipo de componente que deve processar Intent. Qualquer número de descrições de categoria pode ser inserido em uma Intent, mas a maioria das Intents não requer categoria alguma. Esse atributo declara a categoria da Intent aceita no atributo name (da categoria <category android:name="android.intent.category.DEFAULT"/>). O valor deve ser o valor literal da string de uma ação, e não a constante da classe. Um componente só pode tratar uma Intent se possuir todas as categorias da Intent no seu filtro. Nenhuma categoria da Intent pode faltar no componente. |
| Action | Declara a ação da Intent aceita no atributo name. O valor deve ser o valor literal da string de uma ação, e não a constante da classe. Se ambos, componente e Intent, não possuem um atributo Action definido, o componente pode tratar a Intent. Se esta possui atributo Action, ao menos uma das possíveis Actions do componente deve corresponder ao da Intent. |
| Data | Este é o critério mais complexo, pois o atributo Data de um filtro é dividido em Scheme (em português, Esquema) e Type (em português, Tipo). O atributo Data do Intent Filter do componente deve acompanhar exatamente o que for definido no atributo Data da Intent, caso contrário, a filtragem falhará. Por exemplo, se a Intent define apenas o Scheme (sem Type), o Intent Filter do componente deve o mesmo Scheme e apenas ele (Type) deve estar em branco também. |

Não se preocupe! Vamos detalhar melhor esses conceitos por meio de exemplos, assim, esses atributos serão melhor compreendidos. Vamos lá?

Observe o seguinte exemplo de código XML para um Intent Filter. Nele, estamos definindo uma Activity chamada VisualizadorDeImagem com um único Intent Filter. Este irá aceitar Intents de categoria padrão, conforme é definido pelo atributo category do tipo DEFAULT na linha 04 do código XML. Além disso, permite a visualização de conteúdo do tipo imagem, de acordo com o atributo Action do tipo VIEW, definido na linha 03. E, por último, o tipo de formato aceitável definido neste Intent é declarado na linha 05, atributo Data que aceita qualquer mimeType relacionado à imagem, como jpeg, png, TIFF etc.

01 <activity android:name="VisualizadorDeImagem" ... >

02 <intent-filter>

03 <action android:name="android.intent.action.VIEW" />

04 <category android:name="android.intent.category.DEFAULT" />

05 <data android:mimeType="image/\*" />

06 </intent-filter>

07 </activity>

#### Saiba Mais!



Multipurpose Internet Mail Extensions (MIME) são extensões de tipos de dados usados principalmente na Internet. Aplicações para dispositivos móveis também utilizam as definições dos padrões MIME. Algumas das principais extensões são: text/plain (texto simples), text/html (página web), image/jpeg (imagem JPEG), audio/mp3 (música em formato MP3), video/mpg (vídeo em formato MPG). Saiba mais sobre MIME em: <http://www.freeformatter.com/mime-types-list.html>.

Por meio desse exemplo de filtro de Intent para realizar uma visualização de imagem, podemos perceber que os três atributos (Ação, Categoria e Dado) serão os responsáveis por guiar o objetivo geral dos Intents que venhamos a desenvolver. De antemão, já podemos imaginar que existem diversos casos envolvendo Intents, tais como: despertador, agendas, temporazidores, alarmes, câmera, edição de contato em uma agenda, composição de e-mail com anexo opcional, armazenamento de arquivo, obtenção de rotas em um mapa, reprodutores de áudio, vídeo, criação de anotações, configurações do dispositivo etc.

Quando você desejar criar outro tipo de filtro de Intent envolvendo algum desses contextos, deverá ter conhecimento dos atributos específicos para o caso em questão. Agora, vamos codificar mais um exemplo, envolvendo a definição de um filtro de Intent para um alarme de despertador.

Para o caso do filtro de Intent de um alarme de despertador, a estrutura será a mesma do visualizador de imagem. Todavia, o atributo Action deverá possuir o valor de SET\_ALARM, permitindo assim, que outras Intents tenham acesso a esse filtro.

01 <activity ...>

02 <intent-filter>

03 <action android:name="android.intent.action.SET\_ALARM" />

04 <category android:name="android.intent.category.DEFAULT" />

05 </intent-filter>

06 </activity>

#### Saiba Mais!



Ficou curioso sobre os valores que os atributos Ação, Categoria e View podem receber? Então dê uma conferida neste website <https://developer.android.com/guide/components/intents-common> que exemplifica diversos tipos de Intents e filtros de Intents.

## TÓPICO 3 – Criação de Intents

**Objetivo**

* Acompanhar o processo de criação de Intents Explícitas e Implícitas;
* Aprender a tornar possível navegação entre Activities por meio de Intents.

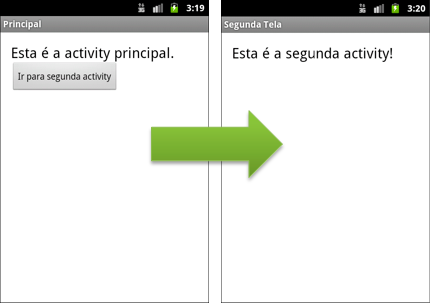
Agora que já compreendemos as funções e as principais características de Intents e Intent Filters, podemos implementar alguns exemplos que sirvam para demonstrar como utilizá-las na prática. Já abordamos o uso de Intents e principalmente seu objetivo, que é permitir que o usuário navegue entre Activities, todavia, ainda não definimos como realizar isso por meio de código Java.

Neste tópico, iremos implementar Intents Implícitas e Explícitas, percebendo, através de exemplos, que podemos permitir que o usuário da aplicação possa navegar entre as telas de maneira simples.

### 3.1 Criando Intents Explícitas

Uma Intent pode ser considerada Explícita quando possui o nome do componente ao qual se destina. Lembre-se de que o nome do componente corresponde ao nome da classe que representa o componente, juntamente com seu pacote.

Com base nisso, vamos criar um projeto Android, chamado TrocarActivity, no qual iremos utilizar Intents Explícitas para efetuar a navegação entre Activities, com o objetivo de permitir a navegação entre telas do aplicativo. Nosso aplicativo será bem simples, conterá apenas duas Activities, chamadas Principal e SegundaTela, com um botão que dispara a navegação da Activity Principal para a Activity SegundaTela.



O código da classe Principal possui um relacionamento de herança para a classe Activity, assim, toda classe que for uma Activity deve herdar da classe Activity do Android. Na classe Principal, podemos perceber que o método trocarActivity() instancia um objeto do tipo Intent (android.content.Intent), que aponta para a classe SegundaTela, exatamente na linha 09 do código a seguir.

01 public class Principal extends Activity {

02 @Override

03 protected void onCreate(Bundle savedInstanceState) {

04 super.onCreate(savedInstanceState);

05 setContentView(R.layout.activity\_principal);

06 }

07

08 public void trocarActivity(View v){

09 Intent intent = new Intent(this, SegundaTela.class);

10 startActivity(intent);

11 finish();

12 }

13 }

Na linha 10 do código, é executado o método startActivity(), que recebe o objeto Intent como parâmetro, disparando a ativação da segunda Activity (representada pela classe “SegundaTela.java”). Nesse momento, o Android analisa a Intent, localiza a classe SegundaTela e exibe o layout associado a esta. Por fim, o método finish(), na linha 11, encerra a Activity Principal.

O objeto Intent recebe dois parâmetros de instanciação: a origem, ou seja, a classe Principal, que tem seu contexto representado pelo ”this” e a classe destino, ou seja, SegundaTela.class.

Perceba que a criação de Intents Explícitas, ou seja, aquelas que já apontam para uma Activity destino, Service ou Broadcast Receiver específico, é simples; basta criar o objeto do tipo Intent na Activity atual e usar o método sendBroadcast() para iniciar o componente desejado.

**Atenção**



Caso o método finish() não seja usado após o disparo da Intent, a Activity atual ficará aguardando em segundo plano. Assim, se pressionarmos o botão “retornar” quando a Activity SegundaTela estiver ativada, a Activity Principal será trazida de volta ao primeiro plano.

### 3.2 Criando Intents Implícitas

Já aprendemos a criar Intents Explícitas e percebemos que o processo é bem simples, certo? Agora, imagine uma situação em que sua aplicação precisa que um determinado tipo de dado seja tratado, não importando qual aplicação fará isso pelo usuário. Nesses casos, utilizamos Intents Implícitas.

Quer um exemplo prático do cenário que acabamos de descrever? Suponha que você tem uma aplicação que oferece um botão para levar o usuário a um website. O objetivo da aplicação é abrir o link para o usuário, não importando por qual aplicativo ele irá fazer isso. De início, a aplicação não sabe qual navegador (web browser) está instalado no dispositivo do usuário. Além disso, este usuário pode ter mais de um navegador instalado, e cabe a ele decidir qual navegador prefere para abrir a página. Como resolver esta situação?

É bastante simples! Basta criarmos uma Intent que contenha o link que desejamos abrir, definir o tipo de ação que deve ser aplicada a esse link, a categoria de componente que pode tratar esse link e deixar o Android resolver a Intent. Acompanhe o trecho de código abaixo.

01 public class MainActivity extends Activity {

02

03 ...

04

05 public void abrirLink(View v){

06 Intent intentLink = new Intent();

07 // Definindo ação: VIEW (visualização)

08 intentLink.setAction(Intent.ACTION\_VIEW);

09 // Definindo categoria: DAFEAULT (Padrão)

10 intentLink.addCategory(Intent.CATEGORY\_DEFAULT);

11 // Definindo o endereço do link a ser aberto

12 intentLink.setData(Uri.parse("http://www.google.com"));

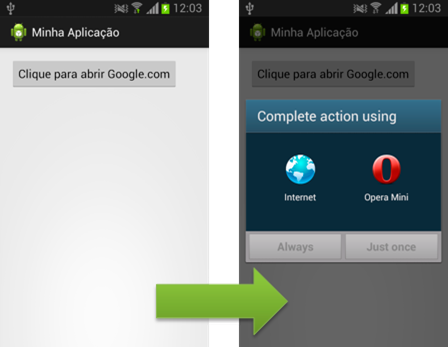
13

14 startActivity(intentLink);

15 }

16 }

Neste exemplo, iremos abrir o link “[http://www.google.com](http://www.google.com/)” para o usuário através de um botão. O método abrirLink() é executado quando o botão “abrir Google.com” é requisitado, que existe na interface gráfica da aplicação. Da linha 6 à 12, instanciamos uma Intent e definimos três de seus atributos: Action, Category e Data. No atributo Data, definimos o link a ser aberto, o website do Google. No atributo Action, definimos que a ação a ser executada sobre este link é de visualização (VIEW) e que qualquer Activity de categoria padrão (DEFAULT) pode fazer este trabalho. Na linha 14, disparamos a Intent através do método startActivity().



* Se houvesse apenas um navegador disponível, o Android iria abri-lo diretamente, sem perguntar ao usuário. E como o Android sabe que precisamos de um navegador de internet para essa Intent? Ele analisa o tipo de dado que foi fornecido no atributo Data e infere que tipo de aplicativo deve tratar a Intent. Devido à presença do protocolo “http” no link, o Android consegue identificar o String do atributo Data como um link.

De maneira semelhante, caso seja um arquivo de extensão PDF, o sistema operacional Android irá buscar por um aplicativo que possa ter o Intent Filter capaz de receber o documento como parâmetro.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 14

### 3.3 Tornando Componentes Visíveis para Intents Externas

Até agora, acompanhamos o processo de criação de Intents em aplicações que podem ser tratadas por componentes internos (dentro da nossa própria aplicação) ou externos (por aplicações de terceiros). Que tal, agora, oferecermos nossa aplicação como opção para tratamento de Intents externos para o dispositivo Android? Faremos isso através dos Intents Filters.

Vamos tomar o seguinte exemplo: criamos um aplicativo para visualização de arquivos de texto simples. E, para que o usuário utilize este aplicativo, basta oferecê-lo como opção para visualização de arquivos de texto através de Intent Filters. Observe o trecho de código a seguir do arquivo “AndroidManifest.xml”.

01 <activity >

02 ...

03 <intent-filter>

04 <action android:name="android.intent.action.VIEW" />

05 <category android:name="android.intent.category.DEFAULT" />

06 <data android:mimeType="text/plain" />

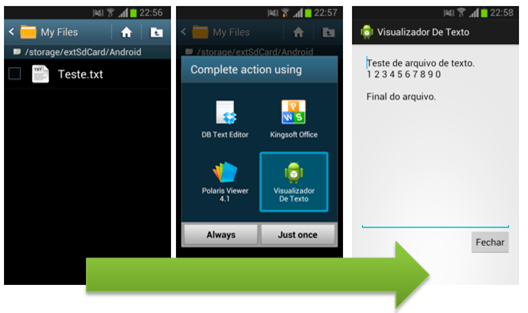
07 </intent-filter>

08 </activity>

Vamos prosseguir para entender melhor esse Intent Filter.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página10 de 14

* Definimos um Intent Filter com três atributos: Action, Category e Data. No atributo Action, definimos que nossa Activity é indicada para visualização de dados. A categoria desta Activity é padrão e ela é destinada a tratar dados do tipo texto simples, representado pelo MIME text/plain. Quando tentamos abrir um arquivo do tipo texto (extensão .txt) que está armazenado no dispositivo, por exemplo, é gerada uma Intent direcionada para uma Activity que exibe o conteúdo de um arquivo de texto. Nesse momento, o Android apresenta as aplicações que podem abrir arquivos de texto, incluindo nossa aplicação de exemplo, como pode ser observado na figura a seguir.

Figura 7: Aplicação Visualização de Texto como opção oferecida pelo Android.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página11 de 14

* Neste tópico, aprendemos a implementar Intents e Intent Filters. Podemos concluir que, para usar Intents Explícitas apenas precisamos do nome do componente que desejamos ativar. Já para Intents Implícitas, é preciso criar Intent Filters que descrevam o tipo de componente que pode tratar a Intent. Também aprendemos que os Intent Filters servem tanto para utilizarmos componentes das nossas e de outras aplicações quanto para que outras aplicações possam acessar nossos componentes.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página12 de 14

### fechamento



Chegamos ao fim de nossa aula. Nela, estudamos as Intents, que são utilizadas pela plataforma para ligar componentes como Activities, Services e Broadcast Receivers. Através da combinação de Intents e Intent Filters, é possível fazer a aplicação encontrar e ativar componentes dela mesma ou de outras aplicações instaladas no dispositivo. Além disso, é possível disponibilizar seus componentes para outras aplicações instaladas. Mas não é só isso! Ainda há muito para explorar sobre esse assunto, como ativação de Services via Intents, acesso a provedores de conteúdo, como banco de dados e catálogos de contatos do dispositivo via URI (atributo Data das Intents), criação de Intent Filters para Broadcast Receivers, entre outros.

Continue estudando! Caso tenha dúvidas, entre em contato com o seu tutor. Até mais!

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página13 de 14

#### Referências

* + Android, Plataforma. O Sistema Operacional móvel mais conhecido do mundo. Disponível em: [https://www.android.com](https://www.android.com/). Acesso em: 6 dez. 2018.
  + Developers, Plataforma. Android Developer. Disponível em: [https://developer.android.com](https://developer.android.com/). Acesso em: 6 dez. 2018.
  + Developers, Plataforma. Android Studio. Disponível em: <https://developer.android.com/studio/?hl=pt-br>. Acesso em: 10 dez. 2018.
  + Developers, Plataforma. Intent Filters. Disponível em: <https://developer.android.com/guide/components/intents-filters?hl=pt-br>. Acesso em: 10 dez. 2018.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página14 de 14