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.

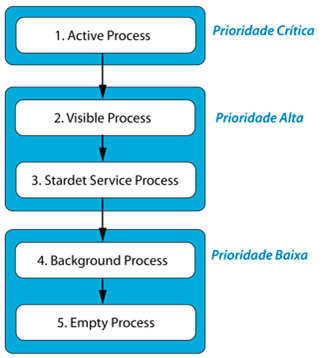
O Android adota uma política bem controlada de gerenciamento de recursos, visto que esta é realizada através do Android Runtime. Isso significa, por exemplo, que processos podem ser removidos da memória, sem qualquer aviso prévio, com o intuito de liberar recursos para outros processos associados a aplicações 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

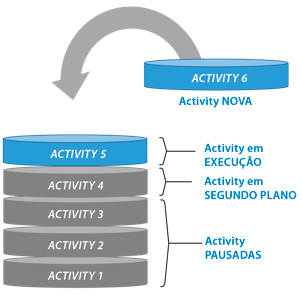
Neste tópico, conheceremos um pouco mais sobre o componente Activity, estudando em detalhes o seu ciclo de vida.

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

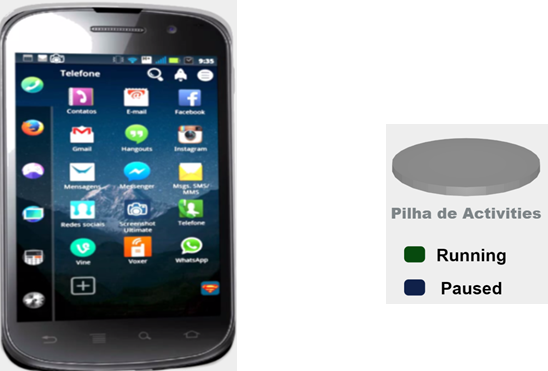
|  |  |
| --- | --- |
| RESUMED | Mais conhecido como Running (em português, Executando), nesse estado, a aplicação está executando em primeiro plano e tem o foco do usuário. |
| PAUSED | Em português, significa Pausado. Nesse estado, a Activity encontra-se temporariamente interrompida, desse modo, outra 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.

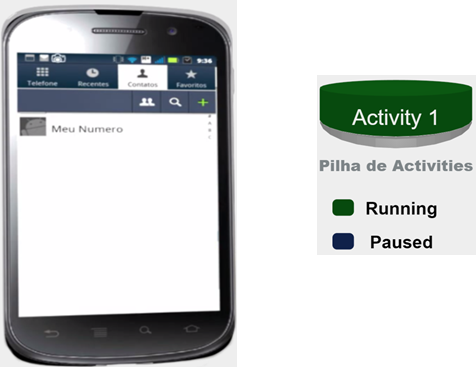
Figura 3: Pilha de Activities no Android.Fonte: Dell Computadores/LE@D.

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.

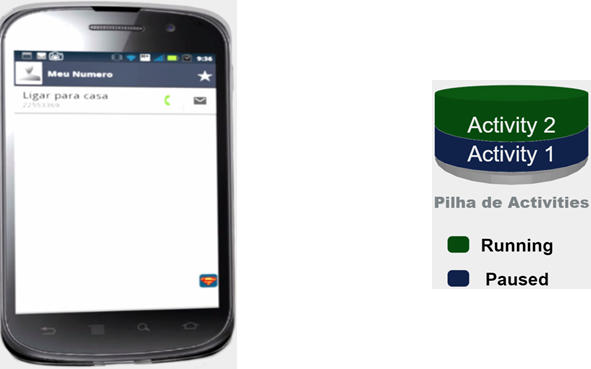
Vamos observar adiante o que acontece ao abrirmos a nossa primeira Activity e como se comportará a Pilha de Activities a partir disso.

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.

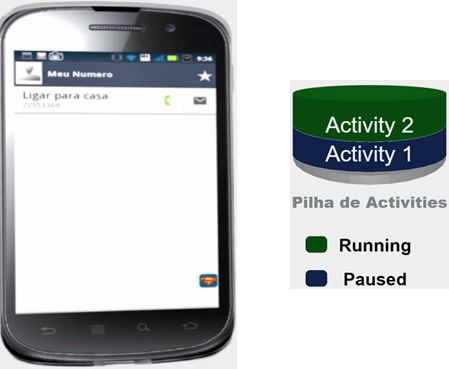
Que tal adicionarmos ainda mais uma Activity em nossa Pilha, para podermos observar o resultado? Em uma situação real, o usuário abrirá diversas Activities em uma aplicação enquanto navega.

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.

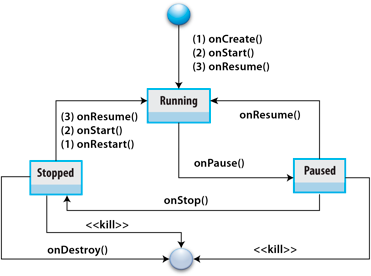
Figura 8: Activity 3 removida da Pilha de Activities.Fonte: Dell Computadores/LE@D.

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.

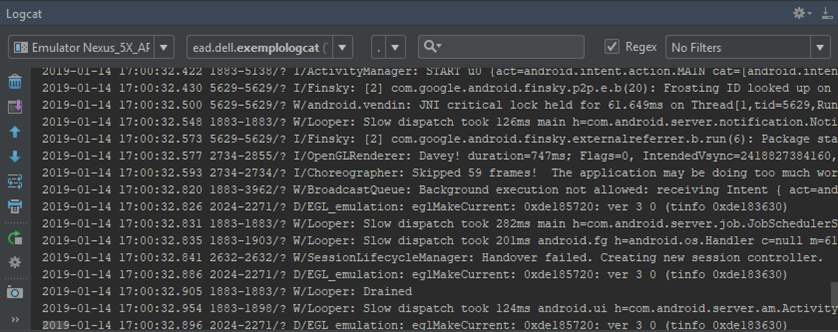
Como mencionado anteriormente, as mensagens de log possuem diferentes níveis de severidade. Os principais níveis de severidade são: debug (depuração), verbose (literal), info (informação), warn (alerta) e error (erro). Para facilitar a legibilidade, as mensagens de log em cada um desses níveis de severidade 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.

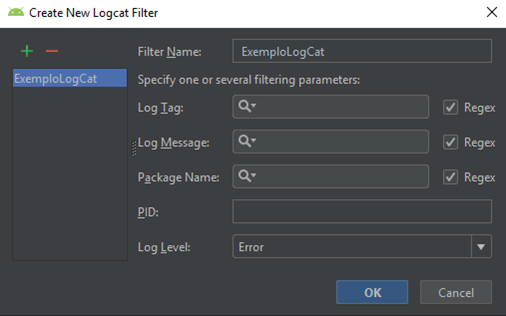
|  |  |  |
| --- | --- | --- |
| **Nível** | **Sigla** | **Cor** |
| Debug | D | Azul |
| Verbose | V | Preto |
| Info | I | Verde |
| Warn | W | Laranja |
| Error | E | Vermelho |

Fonte: Dell Computadores/LE@D

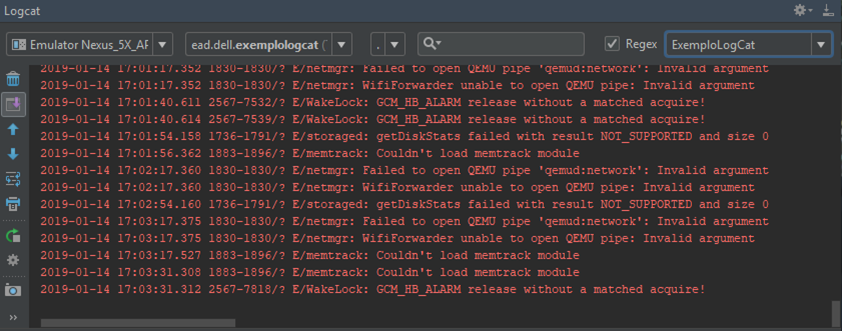
Após executarmos a nossa aplicação ExemploLogCat, o registro dos logs na ferramenta LogCat dentro do IDE Android Studio é exibido, como na figura a seguir. Observe que os logs exibidos não são apenas aqueles feitos pela nossa aplicação, mas todos os logs gerados pelo sistema operacional. Pode-se observar 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.

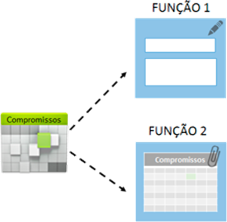
Você já deve saber que uma aplicação Android pode ter diversas Activities e que elas representam um ponto de interação com o usuário, ou seja, a principal característica de uma Activity é possuir uma interface gráfica.

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.

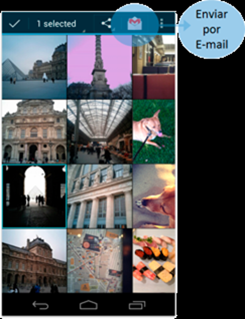
Vamos entender, a partir de um exemplo, que essa necessidade de várias funções de uma aplicação irá obrigar que nossa aplicação tenha várias telas para distribuir as funcionalidades de maneira adequada e com boa usabilidade para o usuário. Imagine que você precisa desenvolver uma aplicação para agendar 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?

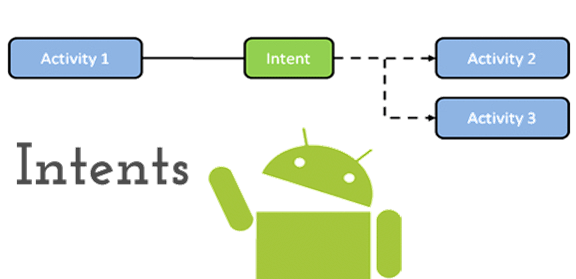
Implementando aplicações com Activities independentes, podemos permitir que o usuário entre nessa aplicação a partir da Activity mais adequada para ele no momento. Assim, para o nosso exemplo anterior, quando o usuário quiser apenas criar um novo compromisso, ele não será obrigado a acessar a lista de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

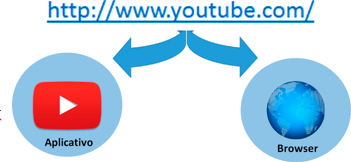
Podemos dizer que existem dois tipos de Intents: Explícitas, quando o nome do componente está disponível dentro do nosso aplicativo, e Implícitas, quando o nome do componente não é informado.

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.

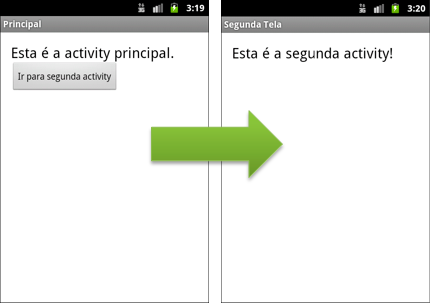
Agora que já compreendemos as funções e as principais características de Intents e Intent Filters, podemos implementar alguns exemplos que sirvam para demonstrar como utilizá-las na prática. Já abordamos o uso de Intents e principalmente seu objetivo, que é permitir que o usuário navegue entre Activities, 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"));

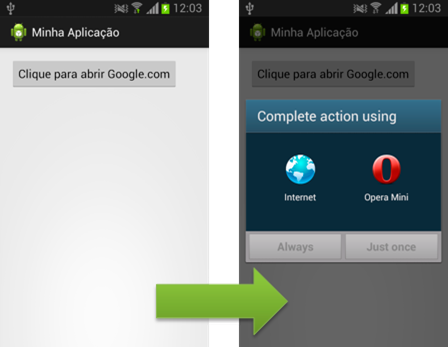
13

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" />

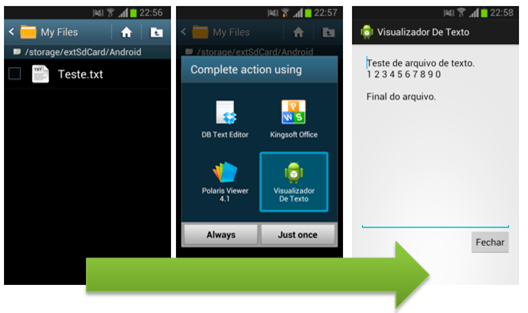
07 </intent-filter>

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

## TÓPICO 1 – Fundamentos da Interface Gráfica

**Objetivos**

* Conhecer as classes que formam a base dos elementos gráficos do Android;
* Entender como funciona a relação entre componentes gráficos visuais (widgets) e não visuais (gerenciadores de layout) no Android.

Para entendermos melhor o funcionamento dos gerenciadores de layout, que serão abordados posteriormente, estudaremos, neste tópico, a base dos elementos gráficos utilizados para a construção da interface gráfica, canal de interação entre humanos e máquinas, das aplicações Android. Destacaremos os principais componentes utilizados para desenvolver as interfaces dos aplicativos Android. Durante este tópico, vamos entender como esses elementos são responsáveis pela maior parte da interação entre o usuário e o aplicativo, pois os componentes possuem grande responsabilidade em definir o design da interface, para que seja agradável e com alta usabilidade, não somente por conta da estética, mas também pelo fato da organização gráfica dos objetivos da aplicação de maneira que permita ao usuário compreender facilmente o que deve ou pode fazer somente pela forma como os componentes do aplicativo estão organizados.

Que tal iniciarmos os estudos definindo os componentes gráficos de uma interface gráfica em Android? Você está preparado? Então vamos começar!

Com o intuito de padronizar a construção de interfaces gráficas utilizadas para interagir com o usuário, o Android propõe uma terminologia que permite identificar categorias de componentes gráficos e as suas funções. Essa terminologia estabelece dois grupos de componentes gráficos fundamentados em classes específicas da API Android, sendo: visões (Views) e grupos de visões (View Groups).

Quadro 1: Componentes gráficos baseados em classes específicas da API Android.

|  |  |  |
| --- | --- | --- |
|  | Visões (Views) | Grupos de Visões (View Groups) |
| CONHECIDOS COMO | Widgets. | Gerenciadores de layout. |
| DESCRIÇÃO TÉCNICA | São subclasses diretas da classe View (android.view.View) e encontram-se disponíveis para uso na biblioteca de classes do Android. Pelo fato de serem subclasses de View, a terminologia classifica os widgets como visões (views). | Os gerenciadores de layout são subclasses da classe ViewGroup (android.view.ViewGroup), que, por sua vez, é subclasse da classe View. Dessa forma, a classe ViewGroup é a base de todos os gerenciadores de layout presentes na plataforma Android. Pelo fato de serem subclasses de View, os widgets são referenciados como visões (views). |
| OBJETIVO | Representar os componentes gráficos visuais dos aplicativos. | Representar os elementos gráficos não visuais, isto é, que são invisíveis para o usuário. |
| FUNÇÃO | Utilizados para a construção da interface gráfica das aplicações Android. | Responsáveis por agrupar, organizar e controlar os componentes gráficos visuais, os widgets, da interface gráfica da aplicação. |
| EXEMPLOS | Botões (ImageButton, ToggleButton RadioButton), selecionador de opções (CheckBoxes), caixas de texto, menus de itens, imagens, dentre outros. | LinearLayout e RelativeLayout. |

Fonte: Dell Computadores/LE@D

Entre alguns exemplos de Views, também conhecidos como widgets, estão o ImageButton, ToggleButton , RadioButton e CheckBox. Com esses widgets, poderemos compor nossa interface gráfica de um aplicativo Android de maneira mais eficiente, ou seja, poderemos criar formulários com botões e selecionadores de opções (CheckBoxes), para que o usuário possa informar dados, por exemplo, gênero, preferências, etc. e, ao final, pressionar o botão Enviar para submeter o formulário de dados ou o botão Cancelar para limpar os campos do formulário.

Sabendo disso, o que você acha de verificar a função de alguns widgets constantemente utilizados para construção de aplicações? Note:

* ImageButton: com esse widget, podemos definir uma imagem que tenha a função de botão que, ao ser invocado, poderá realizar uma ação estabelecida dentro de algum método Java que venhamos a desenvolver. Para que possamos usar o ImageButton, devemos inserir a tag <ImageButton /> dentro de um layout XML;
* ToggleButton: permite que o usuário troque de configuração entre dois estados. De forma abstrata, funciona como um disjuntor de energia, que pode possuir dois status, por exemplo, ligado e desligado. Para que possamos fazer uso desse widget, devemos aplicar a tag <ToggleButton />;
* RadioButton: proporciona ao usuário a seleção de uma única opção dentro de um conjunto de opções predefinidas. Para isso, é necessário usar a tag XML <RadioButton />;
* CheckBox: disponibiliza uma lista de opções predefinidas pelo desenvolvedor, para que o usuário selecione aquilo que lhe convém, por exemplo, caso nossa aplicação necessite de um formulário que recupere todas as preferências do usuário como uma forma de lista, podemos criar vários checkboxes por meio da tag xml <CheckBox />, disponibilizando ao usuário sua lista de opções.

Todos esse widgets herdam de uma mesma classe em comum, que tal detalharmos melhor a hierarquia entre eles? Vamos lá.

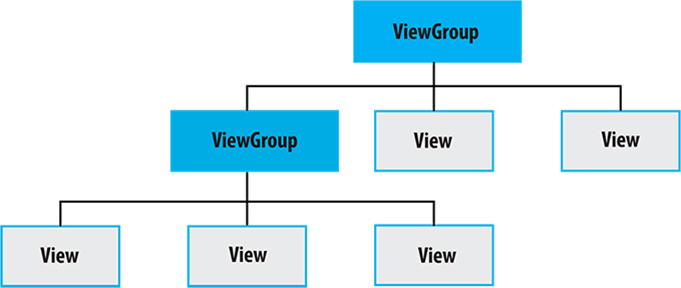
Como comentado no quadro 1, os gerenciadores de layout são componentes gráficos invisíveis que herdam da classe abstrata ViewGroup. Na hierarquia da figura 1, os elementos do tipo ViewGroup representam os gerenciadores de layout e funcionam como containers que podem armazenar vários objetos do tipo View e ViewGroup. Essa hierarquia pode assumir uma estrutura simples ou complexa de acordo com a necessidade do desenvolvedor em construir a interface gráfica desejada para sua aplicação. Contudo, por questões de desempenho, recomenda-se que essa hierarquia seja a mais simples possível e não gere códigos emaranhados.

De forma geral, a interface gráfica de uma aplicação Android pode ser definida usando uma hierarquia de objetos do tipo View e ViewGroup, melhor detalhada a seguir.

#### Você Sabia?



Toda subclasse de View deve implementar o método onDraw (Canvas canvas), responsável por desenhar o componente visual na tela do dispositivo.

Figura 1: Hierarquia de visões e grupo de visões.Fonte: <http://twixar.me/69J3> (adaptada).

No Android, a classe Activity que você já deve conhecer funciona como uma janela ou formulário, no qual os elementos gráficos visuais são exibidos para o usuário na tela do dispositivo móvel. Dessa forma, para que a interface gráfica de uma aplicação Android seja exibida, é preciso que ela esteja associada a uma Activity. Isso significa que algum componente visual (widget), subclasse de View, ou algum gerenciador de layout, subclasse de ViewGroup, esteja associado à Activity em execução. Via de regra, essa associação é feita por meio de uma chamada ao método setContentView() dentro do método onCreate() da classe Activity.



Se você já estudou um pouco de Java Swing, saiba que no desenvolvimento de interfaces gráficas para aplicações Android, a classe Activity possui função análoga a da classe Form da biblioteca Java Swing, utilizada para construir interfaces gráficas de aplicações desktop.

É certo que sem a Activity não poderemos executar a lógica do aplicativo. Somado a isso, também é importante ter uma interface gráfica que interaja com o usuário de maneira fácil e de boa qualidade, por isso estudamos Views e Viewgroups.

Neste tópico, estudamos a terminologia utilizada para classificar os elementos gráficos empregados na construção de interfaces gráficas das aplicações Android. Além disso, conferimos quais classes da API Android são utilizadas para construir essas aplicações. Devemos ter em mente esses dois elementos, pois servirão de base para construir as interfaces gráficas dos nossos aplicativos. Assim, iremos abordar como fazer uso desses componentes fundamentais para proporcionar a interação entre o usuário e o sistema.

No próximo tópico, estudaremos em detalhes os principais gerenciadores de layout da plataforma Android. Afinal, esses widgets devem ser agrupados em algum container, certo? Lembre-se: organização deve caminhar junto à programação.

## TÓPICO 2 – Gerenciadores de Layout

**Objetivo**

* Conhecer os principais tipos de gerenciadores de layout do Android;
* Entender como os principais tipos de gerenciadores de layout do Android são configurados e utilizados nas aplicações.

Neste tópico, iremos abordar um pouco sobre os gerenciadores de layout. Para obtermos uma certa organização na aplicação, devemos posicionar corretamente aquilo que iremos agregar à tela do aplicativo, ou seja, caso desejemos implementar um sistema que possua um formulário responsável em coletar as informações pessoais de um usuário, como nome, telefone, idade, endereço etc., devemos organizar os componentes do formulário, como caixas de texto, caixas de seleção, botões e outros de forma adequada. Essa organização, em geral, pode ser horizontal, vertical, aleatória ou pode variar conforme o contexto, contudo é importante sempre manter a tela com uma boa usabilidade para as pessoas, facilitando o manuseio da aplicação. Essa organização, posicionamento ou arranjo dos elementos no layout é realizada pelos gerenciadores de layout que conheceremos neste tópico de forma conceitual e também prática. Vamos lá?

Os gerenciadores de layout (ou simplesmente layouts) são classes específicas do Android, responsáveis por gerenciar a forma como os componentes visuais da aplicação serão dispostos na tela do dispositivo móvel. Como aprendido no tópico anterior, os gerenciadores de layout são subclasses da classe ViewGroup. No Android, os componentes visuais, as visões (Views), podem ser organizados de diferentes maneiras na tela do dispositivo. Por exemplo, os componentes visuais podem ser organizados de forma horizontal, com um componente após o outro, ou de forma vertical, com um componente abaixo do outro, ou ainda seguindo uma estrutura tabular onde os componentes visuais são dispostos em linhas e colunas. Essa organização vai depender do tipo de gerenciador de layout utilizado para agrupá-los, por exemplo, FrameLayout, LinearLayout, TableLayout e RelativeLayout que serão estudados adiante.

É importante mencionar que os gerenciadores de layout podem ser declarados de duas formas. A primeira, e mais recomendada, é a declaração via arquivo XML, que permite desacoplar a camada de apresentação do código restante da aplicação. Nessa abordagem, tanto os widgets quanto os gerenciadores de layout são declarados em um arquivo XML por meio de uma sintaxe própria. Esse arquivo fica armazenado no diretório res/layout, dentro do projeto da aplicação.

Isso nos fornece uma modularização melhor, para que não venhamos a misturar códigos da lógica do sistema com a interface gráfica.

A segunda forma de declarar os gerenciadores de layout e os widgets é a forma programática. Nessa abordagem, fica a cargo do programador escrever o código dentro da Activity responsável por instanciar e configurar os gerenciadores de layout e os widgets. Nas próximas seções deste tópico, estudaremos alguns dos principais gerenciadores de layouts do Android, sendo: FrameLayout, LinearLayout, TableLayout e RelativeLayout.

#### Saiba Mais!



O site do Android voltado para os desenvolvedores disponibiliza uma documentação detalhada sobre a sintaxe utilizada para declarar gerenciadores de layout em arquivos XML. Para mais informações, visite o site neste endereço: <http://developer.android.com/guide/topics/resources/layout-resource.html>.

### 2.1 FrameLayout

FrameLayout (android.widget.FrameLayout) é a classe mais simples dentre os demais gerenciadores de layout disponíveis na plataforma Android. Ela é utilizada quando se deseja preencher a tela inteira com um único componente gráfico visual, como demonstrado nas figuras 3 e 4. O componente gráfico visual adicionado à FrameLayout, via de regra, ocupará o canto superior esquerdo da tela e, dependendo do seu tamanho, isto é, da sua altura e largura, pode vir a ocupar toda a extensão da tela.

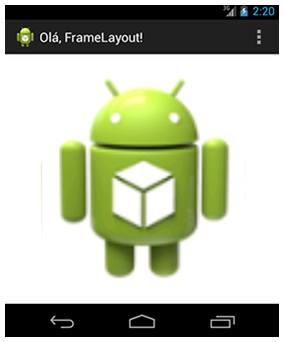
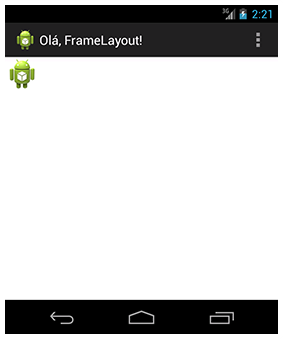
Figura 2: Tela da aplicação usando FrameLayout com largura e altura em fill\_parent (depreciado e renomeado para MATCH\_PARENT a partir da API Level 8 e superior).Fonte: Dell Computadores/LE@D.

Figura 3: Tela da aplicação usando FrameLayout com largura e altura em wrap\_content.  
Fonte: Dell Computadores/LE@D.



Os atributos match\_parent, fill\_parent, wrap\_content servem para dimensionar os elementos na tela, vamos conceituar cada um:

match\_parent: a View deve ocupar o espaço igual ao da sua Parent View (View Pai). Em qualquer dimensão (altura e/ou largura).

fill\_parent: mesma função do match\_parent, porém era utilizada no android 2.1; quando foi feito o update para o 2.3, foi depreciada em uso do match\_parent.

wrap\_content: a View deve ocupar apenas o espaço que necessitar(altura e/ou largura) para exibir suas informações no layout.

O exemplo a seguir mostra a descrição no arquivo XML de um gerenciador de layout do tipo FrameLayout. Nesse exemplo, adicionamos ao nosso gerenciador de layout um componente gráfico visual, o ImageView. Trata-se de um componente que possibilita a visualização de uma imagem na tela da aplicação.

Em nosso caso, a imagem é referenciada por android:src="@drawable/ic\_launcher", onde ic\_launcher corresponde ao nome da imagem, e @drawable corresponde ao diretório em que a imagem se encontra, cujo padrão é res/drawable.

01 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"

02 xmlns:tools="http://schemas.android.com/tools"

03 android:layout\_width="wrap\_content"

04 android:layout\_height="wrap\_content"

05 tools:context=".FrameLayoutActivity" >

06

07 <ImageView

08 android:id="@+id/imageView1"

09 android:layout\_width="wrap\_content"

10 android:layout\_height="wrap\_content"

11 android:src="@drawable/ic\_launcher" />

12

13 </FrameLayout>

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* Os parâmetros android:layout\_height e android:layout\_width são utilizados para informar a altura e largura do componente gráfico, respectivamente, quer seja gerenciador de layout ou widget. Podemos atribuir valores padrão para esses parâmetros, tais como wrap\_content, que estabelece que o componente gráfico vai ocupar apenas o tamanho necessário da tela, e match\_parent, que estabelece que o componente gráfico deve ocupar todo o tamanho definido por seu pai, isto é, o gerenciador de layout.

O exemplo anterior apresenta a tela da aplicação seguindo o layout descrito na página anterior. Já a figura 2 ilustra a tela da aplicação com uma pequena alteração no layout descrito neste código. No caso da tela da figura 2 que ocupa toda a extensão da tela, o valor atribuído aos parâmetros android:layout\_height e android:layout\_width, tanto do gerenciador de layout quanto do componente gráfico ImageView, foi alterado para match\_parent. Por esse motivo, a imagem preenche toda a tela do dispositivo.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 20

Página 7 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

### 2.2 LinearLayout

O LinearLayout (android.widget.LinearLayout) é o gerenciador de layout mais utilizado dentre os disponíveis na plataforma Android. Ele é utilizado quando se deseja organizar os componentes gráficos visuais em uma sequência horizontal (estabelecido por padrão) ou vertical.

No LinearLayout, é possível ajustar outros parâmetros além do android:layout\_height e do android:layout\_width. Particularmente, é possível controlar se os componentes serão exibidos na horizontal ou na vertical. Essa orientação é definida ajustando o parâmetro android:orientation. O código a seguir representa o uso do LinearLayout com a orientação na horizontal.

01 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

02 xmlns:tools="http://schemas.android.com/tools"

03 android:layout\_width="fill\_parent"

04 android:layout\_height="fill\_parent"

05 android:orientation="horizontal"

06 tools:context=".LinearLayoutActivity" >

07 <ImageView

08 android:id="@+id/imageView1"

09 android:layout\_width="wrap\_content"

10 android:layout\_height="wrap\_content"

11 android:src="@drawable/ic\_launcher" />

12 <ImageView

13 android:id="@+id/imageView2"

14 android:layout\_width="wrap\_content"

15 android:layout\_height="wrap\_content"

16 android:src="@drawable/ic\_launcher" />

17 </LinearLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 20

Página 8 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* Repare nas figuras 4 e 5. A primeira ilustra a tela da aplicação seguindo o layout descrito no código anterior. Já a segunda ilustra a tela da aplicação com uma pequena alteração no layout. Essa alteração consiste na modificação do valor do parâmetro android:orientation de horizontal para vertical.

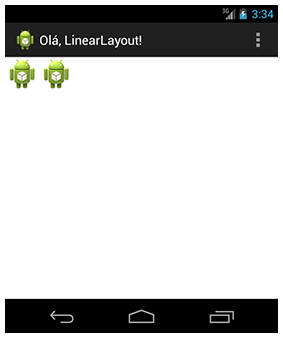
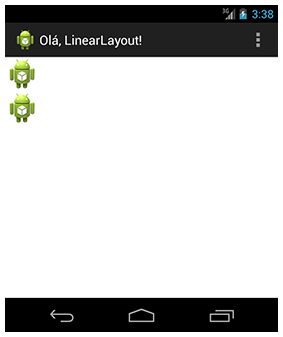
Figura 4: Tela da aplicação usando LinearLayout com orientação em horizontal.Fonte: Dell Computadores/LE@D.

Figura 5: Tela da aplicação usando LinearLayout com orientação em vertical.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 20

Página 9 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* Além do parâmetro android:orientation, o LinearLayout possui outro parâmetro que permite controlar o alinhamento dos seus componentes gráficos, que é o parâmetro android:layout\_gravity, este posiciona o componente (View) em relação ao layout que o contém, ou seja, com ele poderemos posicionar uma caixa de texto (TextView) ao lado direito, esquerdo, centralizado etc. na tela do aplicativo. Com isso, podemos organizar melhor os componentes da tela. Alguns dos valores disponíveis para esse atributo layout\_gravity podem ser observados a seguir.

bottom: posiciona o componente na parte inferior do layout;

* + center: posiciona o componente na parte central (verticalmente e horizontalmente) do layout;
  + right: posiciona o componente na parte direita do layout;
  + left: posiciona o componente na parte esquerda do layout;
  + top: posiciona o componente na parte superior do layout;
  + center\_horizontal: posiciona o componente na parte centro horizontal do layout;
  + center\_vertical: posiciona o componente na parte centro vertical do layout;

É interessante citar que este atributo layout\_gravity não tem qualquer efeito em um RelativeLayout, consequentemente apenas deve ser utilizado em LinearLayout e FrameLayout. Vamos observar um exemplo através do código a seguir.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página10 de 20

Página 10 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* Observe como utilizar esse atributo dentro do nosso arquivo .xml, especificamente nas linhas 13, 20 e 27:
* 01 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
* 02 xmlns:tools="http://schemas.android.com/tools"
* 03 android:layout\_width="fill\_parent"
* 04 android:layout\_height="fill\_parent"
* 05 android:orientation="vertical"
* 06 tools:context=".LinearLayoutActivity" >
* 07
* 08 <ImageView
* 09 android:id="@+id/imageView1"
* 10 android:layout\_width="wrap\_content"
* 11 android:layout\_height="wrap\_content"
* 12 android:src="@drawable/ic\_launcher"
* 13 android:layout\_gravity="left" />
* 14
* 15 <ImageView
* 16 android:id="@+id/imageView2"
* 17 android:layout\_width="wrap\_content"
* 18 android:layout\_height="wrap\_content"
* 19 android:src="@drawable/ic\_launcher"
* 20 android:layout\_gravity="center" />
* 21
* 22 <ImageView
* 23 android:id="@+id/imageView3"
* 24 android:layout\_width="wrap\_content"
* 25 android:layout\_height="wrap\_content"
* 26 android:src="@drawable/ic\_launcher"
* 27 android:layout\_gravity="right" />
* 28
* 29 </LinearLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página11 de 20

Página 11 de 20

Todas as páginas

>>>

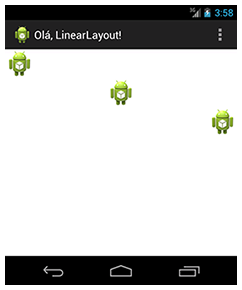
Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* A tela gerada a partir do código anterior pode ser observada graficamente na figura 6. Atente-se para a posição das imagens. Perceba que elas ficaram alinhadas conforme o que foi definido no atributo layout\_gravity.

Figura 6: Tela da aplicação usando LinearLayout com o parâmetro android:layout\_gravity ajustado.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página12 de 20

Página 12 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

### 2.3 TableLayout

O TableLayout (android.widget.TableLayout) é um dos gerenciadores de layout mais utilizados para a construção de telas que possuem a forma de um formulário. O TableLayout é uma subclasse de LinearLayout e organiza os componentes gráficos no formato de uma tabela. No TableLayout, cada linha da tabela é representada por TableRow (android.widget.TableRow), que também é uma subclasse de LinearLayout. Por esse motivo, componentes gráficos podem ser adicionados dentro de TableRow. Cada componente gráfico adicionado em TableRow representa uma coluna da tabela.

O gerenciador de layout TableLayout possui dois parâmetros adicionais, são eles: android:shrinkColumns e android:stretchColumns.

O parâmetro android:shrinkColumns faz as colunas especificadas sempre serem exibidas. Já o parâmetro android:stretchColumns faz as colunas especificadas preencherem todo o espaço disponível. O código a seguir ilustra o uso do TableLayout com três linhas e quatro colunas cada.

01 <TableLayout

02 xmlns:android="http://schemas.android.com/apk/res/android"

03 xmlns:tools="http://schemas.android.com/tools"

04 android:layout\_width="fill\_parent"

05 android:layout\_height="fill\_parent"

06 tools:context=".TableLayoutActivity" >

07 <TableRow>

08 <TextView android:text="Coluna 1" />

09 <TextView android:text="Coluna 2" />

10 <TextView android:text="Coluna 3" />

11 <TextView android:text="Coluna 4" />

12 </TableRow >

13 <TableRow >

14 <TextView android:text="Coluna 1" />

15 <TextView android:text="Coluna 2" />

16 <TextView android:text="Coluna 3" />

17 <TextView android:text="Coluna 4" />

18 </TableRow >

19 <TableRow >

20 <TextView android:text="Coluna 1" />

21 <TextView android:text="Coluna 2" />

22 <TextView android:text="Coluna 3" />

23 <TextView android:text="Coluna 4" />

24 </TableRow>

25 </TableLayout >

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página13 de 20

Página 13 de 20

Todas as páginas

>>>

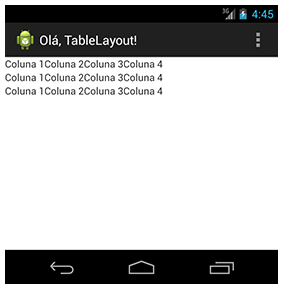
Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* A figura 7 ilustra a tela da aplicação seguindo o layout TableLayout descrito no código anterior.

Figura 7: Tela da aplicação usando TableLayout.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página14 de 20

Página 14 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

### 2.4 RelativeLayout

O gerenciador de layout RelativeLayout (android.widget.RelativeLayout) permite posicionar componentes gráficos ao lado, abaixo ou acima de outro componente gráfico previamente adicionado ao layout. Para que esse posicionamento relativo seja feito, é preciso que cada componente gráfico possua um identificador. Além disso, existe a necessidade de que o componente gráfico utilizado como referência para os outros apareça antes na descrição do layout dentro do arquivo XML. A seguir, você pode observar uma lista de parâmetros disponíveis para informar a posição do componente gráfico (por exemplo, uma View) que está sendo inserido no layout em relação ao componente gráfico já existente (por exemplo, outra View do mesmo layout).

* + layout\_below: posiciona a borda superior da View embaixo da View especificada no parâmetro, por exemplo, android:layout\_below="@+id/outra\_view", significa que a View responsável em possuir este parâmetro faz referência a outra View, dentro do mesmo layout, que tem por id = outra\_view. Consequentemente a view que leva o atributo layout\_below="@+id/outra\_view" ficará embaixo da view de id = outra\_view;
  + layout\_above: posiciona a borda inferior da View acima da outra View especificada;
  + layout\_toRightOf: posiciona a borda esquerda da View ao lado direito da outra View especificada;
  + layout\_toLeftOf: posiciona a borda direita da View ao lado esquerdo da outra View especificada;
  + layout\_alignParentTop: faz a borda superior da View se equiparar à borda superior do componente pai;
  + layout\_alignParentBottom: faz a borda inferior da View se equiparar à borda inferior do componente pai.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página15 de 20

Página 15 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* O código abaixo representa o uso do gerenciador de layout RelativeLayout. Nesse exemplo, foram adicionados dois componentes gráficos ao layout, uma imagem (ImageView) e um texto (TextView). Os parâmetros do ImageView foram configurados da seguinte forma: android:layout\_alignParentTop="true" e android:layout\_alignParentTop="true". O componente ImageView serviu de referência para o componente TextView através da configuração do parâmetro android:layout\_below="@+id/imageView1".
* 01 <RelativeLayout xmlns:android=
* 02 "http://schemas.android.com/apk/res/android"
* 03 xmlns:tools= "http://schemas.android.com/tools"
* 04 android:layout\_width= "fill\_parent"
* 05 android:layout\_height= "fill\_parent"
* 06 android:paddingBottom= "@dimen/activity\_vertical\_margin"
* 07 android:paddingLeft= "@dimen/activity\_horizontal\_margin"
* 08 android:paddingRight= "@dimen/activity\_horizontal\_margin"
* 09 android:paddingTop= "@dimen/activity\_vertical\_margin"
* 10 tools:context= ".RelativeLayoutActivity" >
* 11 <ImageView
* 12 android:id= "@+id/imageView1"
* 13 android:layout\_width= "wrap\_content"
* 14 android:layout\_height= "wrap\_content"
* 15 android:layout\_alignParentLeft= "true"
* 16 android:layout\_alignParentTop= "true"
* 17 android:src= "@drawable/ic\_launcher" / >
* 18 <TextView
* 19 android:id= "@+id/textView1"
* 20 android:layout\_width= "wrap\_content"
* 21 android:layout\_height= "wrap\_content"
* 22 android:layout\_below="@+id/imageView1"
* 23 android:layout\_alignParentLeft="true"
* 24 android:text="Olá, Android!" / >
* 25 </RelativeLayout >

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página16 de 20

Página 16 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* A figura 8 ilustra a tela da aplicação seguindo o layout RelativeLayout descrito no código anterior.

Figura 8: Tela da aplicação usando RelativeLayout.Fonte: Dell Computadores/LE@D.

Com isso, terminamos nosso conteúdo de gerenciadores de layout. Agora estamos aptos para praticar por meio da ferramenta Android Studio.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página17 de 20

Página 17 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

* Neste tópico, aprendemos os conceitos relacionados aos gerenciadores de layout do Android. Entendemos também como utilizá-los e configurá-los. É relevante possuir este conhecimento, pois posteriormente, ao modelarmos as interfaces gráficas através dos layouts XML da aplicação, deveremos posicionar adequadamente alguns componentes para garantir a correta usabilidade da aplicação. Observe a plataforma que você está utilizando, existe uma certa sequência lógica de como as páginas estão organizadas, por exemplo, a tela de edição do perfil, que organiza os campos do formulário de forma não aleatória e dentro de um container que coloca cada campo um ao lado do outro, acima ou abaixo um do outro. Grave estes conceitos para que possamos aplicá-los em outros contextos.

A partir destes gerenciadores de layouts, podemos organizar interfaces gráficas para aplicações que iremos desenvolver futuramente, visto que estes são a base para organização dos itens na tela (layout) do usuário.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página18 de 20

Página 18 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

### fechamento



Chegamos ao final de nossa aula. Nela, aprendemos como funcionam os gerenciadores de layout de interfaces gráficas do Android e estudamos em detalhes alguns deles (FrameLayout, LinearLayout, TableLayout e RelativeLayout). Além disso, conhecemos as classes do Android que representam a base dos elementos gráficos utilizados para construir interfaces gráficas de aplicações Android. Também detalhamos um pouco sobre as views, como a ImageView e TextView, que, sem dúvidas, utilizaremos constantemente em nossos aplicativos Android.

Bons estudos e até lá!

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página19 de 20

Página 19 de 20

Todas as páginas

>>>

Fim da página

Menu de tópicos

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba1)
* [TÓPICO 1Fundamentos da Interface Gráfica](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba2)
* [**TÓPICO 2Gerenciadores de Layout**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba3)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551486970/aula/index-fisico.html#aba4)

Inicio do Conteudo

#### Referências

* + Android Development Site. Intents and Intent Filters. Disponível em: <http://developer.android.com/guide/components/intents-filters.html>. Acesso em: 04 dez. 2018.
  + Intent Class Reference. Disponível em: <http://developer.android.com/reference/android/content/Intent.html>. Acesso em: 04 dez. 2018.
  + JACKSON, Wallace. Android Apps for Absolute Beginners. 2. Apress, 2012. 404 p.
  + LEE, Wei-Meng. Beginning AndroidTM 4 Application Development. Wrox, 2012. 560 p.
  + MEIER, R. Professional AndroidTM 4 Application Development. Indianapolis: Wiley Publishing, 2013. 864 p.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página20 de 20

Página 20 de 20

Todas as páginas

>>>

Fim da página

Olá!  
Estando nesta aula, você já deve saber como funcionam os gerenciadores de layout de interfaces gráficas do Android, como FrameLayout, LinearLayout, TableLayout e RelativeLayout, certo? Além disso, você já deve conhecer as classes do Android que representam a base dos elementos gráficos utilizados para construir interfaces gráficas de aplicações Android.

Pois bem!  
Partindo desses conhecimentos, nesta aula, estudaremos ainda sobre a construção de interfaces gráficas, e iniciaremos o estudo sobre um subconjunto dos componentes gráficos visuais do Android, que são mais utilizados na construção de interfaces gráficas. No mais, será apresentado um conjunto de boas práticas e princípios de design de interfaces gráficas para o Android, o que irá nos ajudar a construir melhores interfaces para nossas aplicações. Esperamos que você aproveite as discussões e as reflexões que serão conduzidas. Vamos à aula?

## TÓPICO 1 – Rótulos, Caixas de Texto, Botões de Ação e Botões de Seleção

**Objetivos**

* Conhecer Widgets de rótulos, caixas de texto, botões de ação e de seleção;
* Aprender como tratar as interações entre os usuários e esses Widgets.

Como você já deve saber, os componentes gráficos visuais (Widgets) são utilizados para construir a interface gráfica das aplicações Android. Esses componentes são subclasses da classe View (android.view.View) da API do Android.

Neste tópico, estudaremos detalhadamente as seguintes categorias de componentes gráficos:

* Rótulos e caixas de texto: TextView e EditText;
* Botões de ação: Button e ImageButton;
* Botões de seleção: CheckBox, ToggleButton e RadioButton.

O principal objetivo é descrever cada Widget e também demonstrar na prática como utilizá-lo para criar as interfaces dos aplicativos mobiles. Pronto para começar essa aventura? Vamos lá!

### 1.1 Rótulos e Caixas de Texto

Um dos componentes gráficos visuais mais simples da plataforma Android são os rótulos (label, em inglês). Eles exibem porções de textos fixos na tela da aplicação com o intuito de informar, por exemplo, o significado de algum ícone da tela ou de deixar claro que tipo de informação deve ser fornecida pelo usuário em um campo de texto.

Os rótulos podem ser utilizados textualmente como informação para o usuário em uma aplicação e são representados pela classe TextView (android.widget.TextView), que é uma subclasse da classe View.

#### Você Sabia?



No Android, a classe TextView possui função igual à da classe JLabel da biblioteca de classes Java Swing, que é utilizada para construir interfaces gráficas de aplicações desktop.

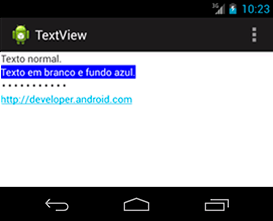
* Você já deve ter aprendido que tanto os layouts quanto os componentes gráficos visuais (Widgets) podem ser declarados via arquivos XML ou de forma programática e que a forma mais recomendada é a declaração via arquivo XML, pois permite desacoplar a camada de apresentação do código restante da aplicação. Desse modo, o trecho de código exibido a seguir apresenta a declaração de quatro rótulos (TextView). Note.
* 01 <LinearLayout
* 02 xmlns:android="http://schemas.android.com/apk/res/android"
* 03 xmlns:tools="http://schemas.android.com/tools"
* 04 android:layout\_width="match\_parent"
* 05 android:layout\_height="match\_parent"
* 06 android:orientation="vertical" >
* 07 <TextView
* 08 android:layout\_width="wrap\_content"
* 09 android:layout\_height="wrap\_content"
* 10 android:text="Texto normal." />
* 11 <TextView
* 12 android:layout\_width="wrap\_content"
* 13 android:layout\_height="wrap\_content"
* 14 android:background="#0000ff"
* 15 android:textColor="#ffffff"
* 16 android:text="Texto em branco e fundo azul."/>
* 17 <TextView
* 18 android:layout\_width="wrap\_content"
* 19 android:layout\_height="wrap\_content"
* 20 android:password="true"
* 21 android:text="minha senha" />
* 22 <TextView
* 23 android:layout\_width="wrap\_content"
* 24 android:layout\_height="wrap\_content"
* 25 android:autoLink="web"
* 26 android:text="http://developer.android.com" />
* 27 </LinearLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 18

Página 3 de 18

* Na primeira declaração do código descrito anteriormente, temos a descrição de um texto simples. Na segunda declaração, as cores do pano de fundo (android:background) e a cor do texto (android:textColor), propriamente ditos, são alterados. É atribuído o código da cor azul ("#0000ff") para o pano de fundo e o código da cor branca ("#ffffff") para o texto. Na terceira declaração, é feita uma pequena alteração no texto para que ele seja exibido no formato de uma senha, isto é, o texto será camuflado, e somente aparecerão os caracteres de senha. Por fim, a última declaração mostra como formatar o texto como um link para um endereço Web.

Podemos observar o resultado dessa escrita de código na figura a seguir.

Figura 1: Exemplo de tela de TextView.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 18

* Além dos rótulos, existe outro tipo de componente gráfico visual responsável por manipular textos na interface gráfica da aplicação. Trata-se da caixa de texto, na qual o usuário pode inserir informação textual. A classe responsável por isso é a EditText (android.widget.EditText), que é uma subclasse de TextView.

As caixas de texto podem ser utilizadas para entrada de textos normais, números ou para campos de senha.

O código a seguir apresenta uma declaração de duas caixas de texto. A primeira é utilizada para receber o nome de usuário (id="@+id/login"), e a outra é usada para receber uma senha (id="@+id/senha"). Observe que, na caixa de texto da senha, o atributo android:password está habilitado (true). Isso significa que uma máscara será utilizada para camuflar a senha digitada.

01 <TableLayout

02 xmlns:android="http://schemas.android.com/apk/res/android"

03 xmlns:tools="http://schemas.android.com/tools"

04 android:layout\_width="match\_parent"

05 android:layout\_height="match\_parent"

06 android:stretchColumns="1" >

07 <TableRow>

08 <TextView android:text="Usuario" />

09 <EditText android:id="@+id/login" />

10 </TableRow>

11 <TableRow>

12 <TextView android:text="Senha" />

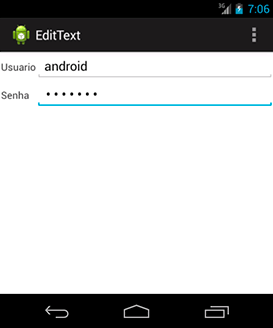
13 <EditText android:id="@+id/senha" android:password="true" />

14 </TableRow>

15 </TableLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 18

* Após conferir o código anterior, observe a figura a seguir. Trata-se da tela que será exibida ao usuário como resultado da declaração descrita no código.

Figura 2: Exemplo de tela de EditText.Fonte: Dell Computadores/LE@D.

A seguir, detalharemos melhor o Widget responsável em captar ações do usuário que utilizará nossa aplicação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 18

### 1.2 Botões de Ação

No Android, os botões de ação são utilizados para disparar ações ou comandos a partir da interface gráfica. Os botões de ação, quando pressionados pelo usuário, geram eventos que fazem determinadas partes do código da aplicação serem executadas. Basicamente, existem dois tipos de botões de ação no Android, button (android.widget.Button) e ImageButton (android.widget.ImageButton). A principal diferença é que o ImageButton permite que seja utilizada uma imagem para desenhar o botão de ação através da classe ImageButton.

Observe o código a seguir, ele é um exemplo de descrição de um formulário de login com um botão de ação do tipo Button.

01

<TableLayout

02 xmlns:android="http://schemas.android.com/apk/res/android"

03 xmlns:tools="http://schemas.android.com/tools"

04 android:layout\_width="match\_parent"

05 android:layout\_height="match\_parent"

06 android:stretchColumns="1" >

07

<TableRow>

08 <TextView android:text="Usuario" />

09 <EditText android:id="@+id/login" />

</TableRow>

11 <TableRow>

12 <TextView android:text="Senha" />

13 <EditText

14 android:id="@+id/senha"

15 android:password="true" />

16</TableRow>

17 <TableRow android:gravity="right" >

18 <Button

19 android:id="@+id/botaoLogin"

20 android:text="Login" />

21 </TableRow>

22 </TableLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 18

* Agora vamos a mais um exemplo de código. Este apresenta a descrição do mesmo formulário de login, porém com um botão de ação do tipo ImageButton. Perceba que, no ImageButton, o ícone padrão do Android ("@drawable/ic\_launcher") é utilizado como imagem para o botão de ação.
* 01 <TableLayout
* 02 xmlns:android="http://schemas.android.com/apk/res/android"
* 03 xmlns:tools="http://schemas.android.com/tools"
* 04 android:layout\_width="match\_parent"
* 05 android:layout\_height="match\_parent"
* 06 android:stretchColumns="1" >
* 07 <TableRow>
* 08 <TextView android:text="Usuario" />
* 09 <EditText android:id="@+id/login" />
* 10 </TableRow>
* 11 <TableRow>
* 12 <TextView android:text="Senha" />
* 13 <EditText
* 14 android:id="@+id/senha"
* 15 android:password="true" />
* 16 </TableRow>
* 17
* <TableRow android:gravity="right" >
* 18 <ImageButton
* 19 android:id="@+id/botaoLogin"
* 20 android:src="@drawable/ic\_launcher" />
* 21 </TableRow>
* 22 </TableLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 18

* Já que você conferiu os códigos, vamos, agora, verificar o resultado deles?

As figuras a seguir representam a implementação dos códigos de Button e ImageButton, respectivamente.

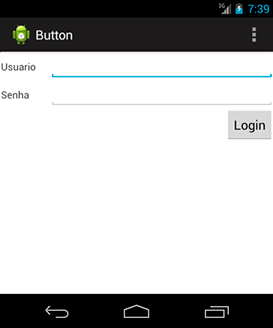
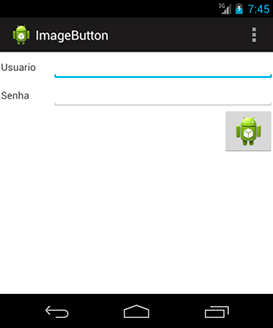
Figura 3: Exemplo de tela de Button.Fonte: Dell Computadores/LE@D.

Figura 4: Exemplo de tela de ImageButton.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 18

* Ao serem selecionados, os botões recebem eventos do tipo on click que devem ser tratados para que ações específicas sejam executadas, assim, uma resposta a esta ação pode ser enviada ao usuário. A vinculação das respostas aos eventos de cliques em botões podem ser feitas via arquivo de configuração XML ou de forma programática (por meio do código Java).

Quando utilizamos arquivos XML, o atributo android:onClick deve ser alterado, informando qual método da Activity em que o botão está inserido será invocado quando o botão for pressionado. Como exemplo disso, observe que, no trecho de código a seguir, o atributo android:onClick do botão de ação, cujo identificador é @+id/botaoLogin, indica que o método fazAlgo da Activity em questão será invocado quando o botão for pressionado.

01 <TableLayout

02 xmlns:android="http://schemas.android.com/apk/res/android"

03 xmlns:tools="http://schemas.android.com/tools"

04 android:layout\_width="match\_parent"

05 android:layout\_height="match\_parent"

06 android:stretchColumns="1" >

07 (...)

08 <TableRow android:gravity="right" >

09 <Button

10 android:id="@+id/botaoLogin"

11 android:text="Login"

12 android:onClick="fazAlgo"/>

13 </TableRow>

14 </TableLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página10 de 18

* Os métodos que respondem aos eventos de clique de botão devem ser obrigatoriamente implementados pela Activity, conforme as seguintes recomendações:
  + O método deve ser público;
  + O método deve retornar void;
  + O método deve receber View como seu único parâmetro formal.

Para o nosso exemplo, temos o método fazAlgo(), que deve seguir a especificação descrita neste código:

01 public void fazAlgo(View view) {

02 //Código executado em resposta ao acionamento do botão.

03 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página11 de 18

* Por outro lado, na forma programática, cabe ao desenvolvedor fazer a vinculação entre o evento de clique do botão e o código que fará o tratamento deste evento. Para isso, algumas convenções de programação devem ser seguidas.

Em primeiro lugar, o botão deve ser instanciado e, em seguida, um ouvinte, ou seja, um método capaz de verificar quando o botão será ativado e o que ele deve realizar, deve ser vinculado à essa instância do botão. O código a seguir exemplifica como isso é feito.

01 (...)

02 Button button = (Button) findViewById(R.id.botaoLogin);

03 button.setOnClickListener(new View.OnClickListener() {

04 public void onClick(View view) {

05 //Código executado em resposta ao clique do botão.

06 }

07 });

08 (...)

Primeiramente, uma instância do botão é recuperada por meio do método findViewById, que recebe como argumento o identificador do botão R.id.botaoLogin via classe R. Logo em seguida, é atribuído como ouvinte do clique do botão uma instância da implementação da interface OnClickListener. No corpo do método onClick, ficará o código responsável pelo tratamento do evento do botão.

Adiante, iremos abordar alguns Widgets de seleção que irão facilitar para nosso usuário do aplicativo a seleção de determinadas opções disponíveis, seja com exclusividade ou não.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página12 de 18

### 1.3 Botões de Seleção

Os principais botões de seleção do Android são: CheckBox (android.widget.CheckBox), ToggleButton (android.widget.ToggleButton) e RadioButton (android.widget.RadioButton).

O que você acha de acompanhar a declaração desses três tipos de botões de seleção?

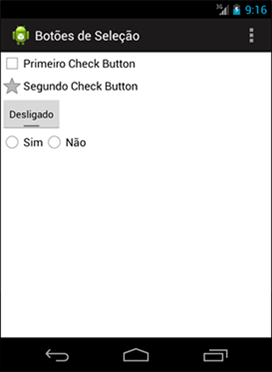
Para isso, faça o download do código neste link: [[download do código]](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101362551491284/aula/code/Cod1.txt).

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página13 de 18

* Após conferir o código, perceba que os dois primeiros componentes são do tipo CheckBox. A diferença do primeiro componente CheckBox (android:id="@+id/check1") para o segundo (android:id="@+id/check2") é que este utiliza um estilo para modificar o formato do campo de seleção. Neste caso, o novo formato do campo de seleção é uma estrela.

Já o terceiro componente declarado é ToggleButton. Na declaração do ToggleButton, são atribuídos dois textos ao botão, um para cada estado que ele pode assumir. O primeiro texto ("Desligado") está associado ao estado inicial do botão (android:textOff). Já o segundo texto ("Ligado") está associado ao estado do botão após pressionado (android:textOn).

Repare na figura a seguir. Ela apresenta o resultado dos botões de seleção descritos no código citado na página anterior.

Figura 5: Exemplo de tela de botões de seleção.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página14 de 18

* Ainda sobre o código anterior, as ações para tratar o evento de clique do botão são especificadas dentro do arquivo XML.

Agora atente-se para o código a seguir. Ele apresenta uma possível implementação para o método fazAlgoOnChecked, referenciado no atributo android:onClick dos botões do tipo CheckBox, definidos anteriormente. Na primeira parte da implementação do método fazAlgoOnChecked, é verificado se o checkbox foi marcado ou desmarcado pelo usuário. Em seguida, é feita a identificação de qual dos dois botões foi selecionado com base no seu identificador. Uma vez identificado qual dos checkboxes foi selecionado, o código do caso (case) específico é executado. Neste exemplo, se o botão estiver marcado, ele “fará uma coisa”, caso contrário, ele “fará outra coisa”.

01 (...)

02 public void fazAlgoOnChecked(View view) {

03 //O checkbox foi marcado?

04 boolean checked = ((CheckBox) view).isChecked();

05 //verifica qual checkbox foi selecionado

06 switch(view.id) {

07 case R.id.check1:

08 if(checked) {

09 //faz uma coisa

10 } else {

11 //faz outra coisa

12 }

13 break;

14 case: R.id.check2:

15 if(checked) {

16 //faz uma coisa

17 } else {

18 //faz outra coisa

19 }

20 break;

21 }

22 }

23 (...)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página15 de 18

* O Código a seguir apresenta uma possível implementação para o método fazAlgoOnToggleClicked, apresentado anteriormente e referenciado no atributo android:onClick do botão do tipo ToggleButton definido no código que implementa os botões. Na primeira parte da implementação do método fazAlgoOnToggleClicked, é verificado se o botão está no estado habilitado ou não. Se o botão estiver no estado habilitado, ele “fará uma coisa”, caso contrário, “ele fará outra coisa”.
* 01 (...)
* 02 public void fazAlgoOnToggleClicked(View view) {
* 03 //O toggle está habilitado?
* 04 boolean on = ((ToggleButton) view).isChecked();
* 05 if (on) {
* 06 //faz uma coisa
* 07 } else {
* 08 //faz outra coisa
* 09 }
* 10 }
* 11 (...)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página16 de 18

* Então vamos a mais um exemplo de código. O exemplo a seguir apresenta uma possível implementação para o método fazAlgoOnRadioButtonClicked, referenciado no atributo android:onClick dos botões do tipo RadioButton, definidos no código que implementa os botões. Na primeira parte da implementação do método fazAlgoOnRadioButtonClicked, é feita uma verificação para conferir se o botão foi selecionado pelo usuário. Em seguida, é feita a identificação de qual dos dois botões foi selecionado com base no seu identificador. Uma vez que é identificado qual dos botões foi selecionado, o código do caso (case) específico é executado. Neste exemplo, a ação só é executada se o botão estiver selecionado.
* 01 (...)
* 02 pubic void fazAlgoOnRadioButtonClicked(View view){
* 03 // o botão está selecionado?
* 04 boolean checked = ((RadioButton) view).isChecked();
* 05 // verifica qual radiobutton está selecionado
* 06 switch(view.getId()){
* 07 case R.id.radioSim:
* 08 if(checked)
* 09 // faz uma coisa
* 10 break;
* 11 case R.id.radioNao:
* 12 if(checked)
* 13 // faz outra coisa
* 14 break;
* 15 }
* 16 }
* 17 (...)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página17 de 18

* Neste tópico, estudamos o primeiro grupo de componentes gráficos visuais: os rótulos, as caixas de texto e os botões de ação e seleção. Assim, aprendemos como eles funcionam e como podemos utilizá-los no desenvolvimento da interface gráfica das nossas aplicações.

No próximo tópico, estudaremos o segundo grupo de componentes gráficos visuais: a lista suspensa, as mensagens de alerta e as telas de diálogo, que são recursos interessantes para viabilizar a interação do usuário com a aplicação, principalmente quando opções devem ser escolhidas ou exceções devem ser geradas.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página18 de 18

## TÓPICO 2 – Lista Suspensa, Mensagens de Alerta e Telas de Diálogo

**Objetivo**

* Aprender a tratar as interações entre os usuários e os Widgets;
* Conhecer os Widgets de lista suspensa, mensagens de alerta e telas de diálogo.

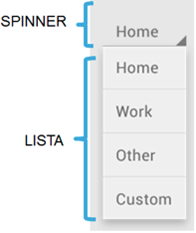
Continuando o nosso estudo sobre os principais Widgets disponíveis na plataforma Android, neste tópico, estudaremos as seguintes categorias de componentes gráficos:

* Lista suspensa: Spinner;
* Mensagens de alerta: Toast;
* Telas de diálogo: AlertDialog.

Por meio deles, perceberemos que a interação usuário-aplicação ficará mais tranquila e a usabilidade do aplicativo irá melhorar, proporcionando uma navegação eficiente para qualquer usuário.

### 2.1 Lista Suspensa

As listas suspensas providenciam uma maneira rápida para selecionar um valor dentro de um conjunto de valores pré-estabelecidos. No Android, o componente responsável para construir listas suspensas é o spinner (android.widget.Spinner). No estado padrão, um spinner exibe o item selecionado, no estado não padrão, exibe uma lista suspensa na qual o usuário pode selecionar um item. Assim, o spinner, ao ser selecionado, sai do estado padrão e exibe uma lista suspensa com todos os itens disponíveis, a partir da qual o usuário pode selecionar um novo item. Todo esse contexto pode ser melhor ilustrado por meio da figura abaixo.

Figura 6: Spinner.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 14

Página 2 de 14

* O código a seguir exibe a declaração de um spinner, no qual temos a lista de planetas do sistema solar. Observe que, no atributo android:entries, existe uma referência para um recurso @array/array\_planetas, que está fora do arquivo de configuração de layout. O array de planetas (array\_planetas) deve ser declarado como recurso no arquivo strings.xml, no diretório res/values.
* 01 <LinearLayout
* 02 xmlns:android="http://schemas.android.com/apk/res/android"
* 03 xmlns:tools="http://schemas.android.com/tools"
* 04 android:layout\_width="fill\_parent"
* 05 android:layout\_height="fill\_parent"
* 06 android:orientation="vertical">
* 07 <TextView
* 08 android:layout\_width="wrap\_content"
* 09 android:layout\_height="wrap\_content"
* 10 android:text="Selecione um dos platetas" />
* 11 <Spinner
* 12 android:id="@+id/spinner"
* 13 android:layout\_width="fill\_parent"
* 14 android:layout\_height="wrap\_content"
* 15 android:drawSelectorOnTop="true"
* 16 android:entries="@array/array\_planetas" />
* 17 </LinearLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 14

* Agora, a seguir, verifique a descrição de como o array de planetas foi declarado.
* 01 <?xml version="1.0" encoding="utf-8"?>
* 02 <resources>
* 03 <string name="app\_name">Spinners</string>
* 04 <string-array name="array\_planetas">
* 05 <item>Mercúrio</item>
* 06 <item>Vênus</item>
* 07 <item>Terra</item>
* 08 <item>Marte</item>
* 09 <item>Júpiter</item>
* 10 <item>Saturno</item>
* 11 <item>Urano</item>
* 12 <item>Netuno</item>
* 13 </string-array>
* 14 </resources>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 14

* A seguir, confira o resultado da tela do spinner declarado nos códigos anteriores. Nesse caso, a lista suspensa está aberta esperando que o usuário selecione um dos itens (planetas) listados.

Figura 7: Exemplo de tela de botões de spinner.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 14

* Quando um dos valores da lista suspensa é selecionado, um evento de item selected é disparado. Para tratar esse evento, é preciso que a Activity associada ao spinner em questão implemente a interface OnItemSelectedListener (android.widget.AdapterView.OnItemSelectedListener). Essa interface obriga que os métodos onItemSelected e onNothingSelected sejam implementados. O método onItemSelected é invocado quando um item da lista é selecionado. Por outro lado, quando nenhum item da lista é selecionado, o método onNothingSelected é invocado. O código a seguir descreve um exemplo de classe que implementa a interface OnItemSelectedListener e os seus métodos.

* 01 public class SpinnerActivity extends Activity implements
* 02 OnItemSelectedListener {
* 03 (...)
* 04 public void onItemSelected(AdapterView parent, View view,
* 05 int pos, long id) {
* 06 //Um item foi selecionado. Você pode recuperar o item
* 07 //usando parent.getItemAtPosition(pos)
* 08 }
* 09 public void onNothingSelected(AdapterView parent) {
* 10 //Nenhum item foi selecionado
* 11 }
* 12 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 14

### 2.2 Mensagens de Alerta e Telas de Diálogo

As mensagens de alerta e as telas de diálogo são importantes ferramentas para dar algum tipo de retorno aos usuários sobre o andamento da execução de algumas partes da aplicação.

Quanto a isso, aprenderemos a usar as classes Toast (android.widget.Toast) e AlertDialog (android.widget.AlertDialog) para dar e colher algum tipo de retorno do usuário.

O Toast provê um simples retorno para o usuário sobre alguma operação em uma pequena popup (janela de informação). Ele só preenche a quantidade de espaço requerida para exibir a mensagem, e a Activity em execução permanece visível e disponível para interação. Quando uma mensagem de alerta ou informação do tipo Toast é mostrada ao usuário, ela aparece como uma mensagem, flutuando sobre a tela da aplicação sem interromper a utilização da aplicação. A ideia é que ela seja o mais discreta possível, enquanto permanece exibindo a informação que o desenvolvedor quer que o usuário tenha acesso naquele momento.

As mensagens do tipo Toast desaparecem automaticamente após um certo período de tempo (timeout), que pode ser configurado em milissegundos. As janelas de AlertDialog têm funcionalidades diferentes, o que iremos conferir mais adiante.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 14

* Abordaremos, agora, por meio do exemplo de tela de login de uma aplicação, como podemos fazer uso de mensagens do tipo Toast.

O código a seguir apresenta uma configuração da tela de login do nosso exemplo. O atributo android:onClick do botão de ação indica que, quando este for pressionado, o método showToast da Activity será invocado. Nesse momento, uma mensagem de informação utilizando Toast será exibida na tela da aplicação.

01 <TableLayout

02 xmlns:android="http://schemas.android.com/apk/res/android"

03 xmlns:tools="http://schemas.android.com/tools"

04 android:layout\_width="match\_parent"

05 android:layout\_height="match\_parent"

06 android:stretchColumns="1" >

07 <TableRow>

08 <TextView android:text="Usuário" />

09 <EditText android:id="@+id/login" />

10 </TableRow>

11 <TableRow>

12 <TextView android:text="Senha" />

13 <EditText

14 android:id="@+id/senha"

15 android:password="true" />

16 </TableRow>

17 <TableRow android:gravity="right" >

18 <Button

19 android:id="@+id/botaoLogin"

20 android:text="Login"

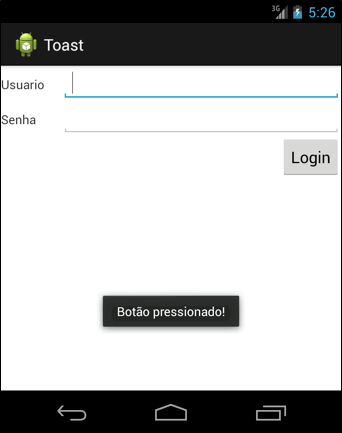
21 android:onClick="showToast" />

22 </TableRow>

23 </TableLayout>

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 14

* O código a seguir mostra a implementação do método showToast, e a figura exibe o resultado da tela da aplicação após o botão de login ter sido pressionado.
* 01 (...)
* 02 public void showToast(View view) {
* 03 Context context = view.getContext();
* 04 //mensagem a ser exibida
* 05 CharSequence text = "Botão pressionado!";
* 06 //tempo de duração da exibição da mensagem
* 07 int duration = Toast.LENGTH\_SHORT;
* 08 Toast toast = Toast.makeText(context, text, duration);
* 09 //exibição da mensagem
* 10 toast.show();
* 11 }
* 12 (...)

Figura 8: Exemplo de tela de mensagem Toast.Fonte: Dell Computadores/LE@D.

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

* Os diálogos de alerta (AlertDialog) são pequenas janelas que permitem que o usuário tome algum tipo de decisão ou forneça alguma informação adicional. Os diálogos de alerta não preenchem toda a tela do dispositivo móvel e são, normalmente, usados quando existe a necessidade de que o usuário tome alguma ação antes da aplicação prosseguir com sua execução. Considere a configuração da tela de login descrita nos códigos anteriores. Suponha que, ao invés do método showToast, outro método showAlertDialog (apresentado no código abaixo) esteja configurado no atributo android:onClick do botão de ação.
* 01 (...)
* 02 public void showAlertDialog(View view) {
* 03 AlertDialog.Builder alerta = new AlertDialog.Builder(this);
* 04 alerta.setIcon(R.drawable.ic\_launcher);
* 05 alerta.setTitle("Confirmação de Login.");
* 06 alerta.setMessage("Deseja fazer o Login?");
* 07 //Se escolher sim
* 08 alerta.setPositiveButton("Sim", new
* 09 DialogInterface.OnClickListener() {
* 10 public void onClick(DialogInterface dialog, int which) {
* 11 Toast toast = Toast.makeText(ExemploAlertDialog.this, "Sim!",
* 12 Toast.LENGTH\_SHORT);
* 13 toast.show();
* 14 }
* 15 });
* 16 //Se escolher não
* 17 alerta.setNegativeButton("Não", new
* 18 DialogInterface.OnClickListener() {
* 19 public void onClick(DialogInterface dialog, int which) {
* 20 Toast toast = Toast.makeText(ExemploAlertDialog.this, "Não!",
* 21 Toast.LENGTH\_SHORT);
* 22 toast.show();
* 23 }
* 24 });
* 25 alerta.show();
* 26 }
* 27 (...)

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

* O código do método showAlertDialog, descrito no código anterior, mostra uma instanciação de um AlertDialog. A variável alerta é utilizada para configurar o ícone, o título e a mensagem de texto da janela de alerta. Em seguida, são adicionados dois botões. O primeiro se trata de um botão de confirmação, que é configurado na variável alerta através da chamada alerta.setPositiveButton. O código adicionado nesse botão é responsável por tratar o evento de onclick. Nesse caso, se o botão Sim da janela de diálogo for pressionado, uma mensagem do tipo Toast é criada com o texto “Sim!”. Já o segundo se trata de um botão de negação, configurado com uma chamada alerta.setNegativeButton. Quando o botão Não da janela de diálogo é pressionado, uma mensagem do tipo Toast é criada com o texto “Não!”. Vamos conferir como isso funciona.

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

* A seguir, temos duas figuras. A primeira apresenta a janela de diálogo criada após pressionar o botão Login. Já a outra exibe a mensagem “Sim!”, evidenciando que o botão Sim da janela de diálogo foi pressionado.

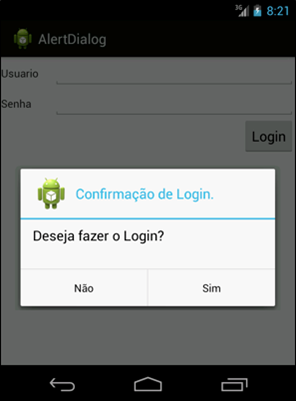
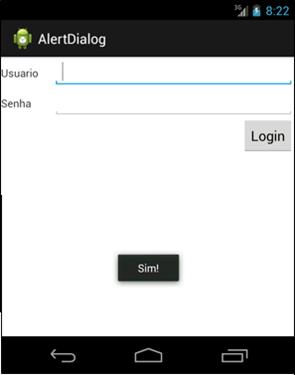
Figura 9: Exemplo de tela de mensagem AlertDialog.Fonte: Dell Computadores/LE@D.

Figura 10: Tela de mensagem após confirmação de login.Fonte: Dell Computadores/LE@D.

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

#### Saiba Mais!



Existem outros Widgets disponíveis na plataforma Android. Você pode encontrar outros Widgets e aprender mais sobre os que foram apresentados neste tópico visitando o site oficial dos desenvolvedores Android: <http://developer.android.com/guide/topics/ui/index.html>.

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

* Até aqui, estudamos os principais componentes gráficos visuais (Widgets) do Android. Neste tópico, aprendemos a utilizar listas suspensas, mensagens de alerta e janelas de diálogo. Além disso, estudamos o tratamento de eventos oriundos da interação do usuário com os Widgets da interface gráfica das aplicações.

No próximo tópico, discutiremos princípios de design de interfaces gráficas para o desenvolvimento de aplicações Android. Por meio de exemplos e contextualizações, iremos entender que existem princípios a serem obedecidos para que a interface tenha boa usabilidade e seja responsiva, ou seja, ela deve se adaptar corretamente a diferentes tamanhos de tela.

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

## TÓPICO 3 – Design de Interfaces Gráficas em Android

**Objetivo**

* Conhecer as principais metas de design de aplicações Android;
* Conhecer os princípios de design para a construção de interfaces gráficas para aplicações Android.

O projeto do Android está focado em torno de três objetivos gerais (metas) que se aplicam tanto aos aplicativos básicos quanto ao sistema operacional de um modo geral. Alguns princípios devem ser seguidos para que seja garantida a usabilidade adequada nas aplicações Android, costumes adotados pelos próprios criadores do sistema operacional que devem ser perpetuados também nos aplicativos utilitários desenvolvidos por nós, desenvolvedores.

* Dessa forma, a equipe de interface gráfica do Android sugere que você leve em consideração as seguintes metas para projetar aplicações corretamente.
  + Enchant me (Encante-me, em português): beleza é mais do que uma boa aparência. Aplicações Android devem ser elegantes e esteticamente agradáveis em todos os aspectos. Por exemplo, as transições entre telas, conteúdos e informações devem ser rápidas e claras, o layout e a tipografia devem ser nítidos e significativos. Ícones de aplicações devem ser desenhados com bastante precisão, como se fossem verdadeiras obras de arte. Além de uma boa ferramenta, sua aplicação deve combinar beleza, simplicidade e propósito para criar uma experiência de uso agradável, que seja fácil de usar e eficiente.
  + Simplify my life (Simplifique minha vida, em português): aplicações Android devem ser fáceis de usar e simples de serem entendidas, o que “facilitará (ou simplificará) a vida do usuário”, no sentido de que a aplicação não deve ser um atraso, ou não seja perda de tempo para o usuário. Quando as pessoas usam a aplicação pela primeira vez, devem, naturalmente, captar dela as características mais importantes. Aplicações Android devem buscar remover tarefas repetitivas e cansativas do conhecimento do usuário, como o gerenciamento de arquivos e sincronização. Tarefas simples nunca devem exigir procedimentos complexos, e tarefas complexas devem ser adaptadas para que o usuário consiga fazê-las de forma simples e criativa. Pessoas de todas as idades e culturas devem possuir controle das suas ações na aplicação, e nunca devem ser sobrecarregadas com muitas escolhas ou distrações irrelevantes.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 10

Página 2 de 10

* + Make me amazing (Faça-me incrível, em português): fazer um aplicativo que seja fácil de usar não é o suficiente. Aplicações Android devem despertar em seus usuários a curiosidade de buscar e experimentar novas formas de interação. O Android permite que as pessoas combinem aplicações em novos fluxos de trabalho através de multitarefas, notificações e compartilhamento de recursos, dados, informações etc. Ao mesmo tempo, a sua aplicação deve ser personalizada, dando às pessoas acesso à tecnologia com nitidez e elegância.

Com isso, a partir desses objetivos (metas), a equipe de experiência do usuário do Android (Android User Experience Team) estabeleceu um conjunto de princípios de design que foram concebidos para manter os principais interesses dos usuários em mente durante o design de uma aplicação. Serão apresentados três quadros a seguir: o primeiro apresenta os princípios de design relacionados com a meta Enchant me, o segundo apresenta os princípios de design relacionados com a meta Simplify my life e, por fim, o terceiro quadro apresenta os princípios de design relacionados com a meta Make me amazing. Vamos lá!

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 10

* Quadro 1: Princípios de design relacionados com a meta Enchant me.

|  |  |
| --- | --- |
| **Enchant me** (Encante-me) | |
| **Princípio de Design** | **Descrição** |
| me in surprising ways (Agrade-me de maneiras surpreendentes). | Uma superfície bonita, uma animação cuidadosamente inserida, ou um efeito sonoro bem cronometrado são características que possibilitam uma melhor experiência de uso. Efeitos sutis contribuem para uma sensação de menos esforço e uma sensação de ter uma ferramenta eficiente em uso. |
| Real objects are more fun than buttons and menus (Objetos reais são mais divertidos que botões e menus). | Permita que as pessoas toquem e manipulem objetos em seu aplicativo. Isso reduz o esforço cognitivo necessário para realizar uma tarefa ao mesmo tempo em que a torna mais empolgante e satisfatória. |
| Let me make it mine (Deixe-me personalizá-lo). | Pessoas adoram adicionar toques pessoais em suas aplicações, porque isso as ajuda a se sentirem à vontade e no controle. Forneça padrões bonitos em sua aplicação, considerando a diversão e as personalizações opcionais que não prejudiquem as tarefas primárias. |
| Get to know me (Conheça-me). | Aprenda sobre as preferências dos usuários ao longo do tempo. Ao invés de pedir-lhes para fazer as mesmas escolhas repetidas vezes, coloque escolhas anteriores de fácil acesso. |

* Fonte: Android Developers (tradução adaptada)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 10

* Quadro 2: Princípios de design relacionados com a meta Simplify My Life.

|  |  |
| --- | --- |
| **Simplify My Life** (Simplifique Minha Vida) | |
| **Princípio de Design** | **Descrição** |
| Keep it brief (Seja breve). | Use frases curtas com palavras simples. As pessoas (usuários) tendem a ignorar sentenças longas. |
| Pictures are faster than words (Imagens são mais rápidas do que palavras). | Considere o uso de imagens para explicar ideias. Elas tendem a chamar mais a atenção das pessoas e podem ser muito mais eficientes do que palavras. |
| Decide for me but let me have the final say (Decida por mim, mas deixe-me dar a palavra final). | A sua aplicação deve ser capaz de selecionar opções e apresentá-las ao usuário, trabalhando de forma proativa ao invés de ficar sempre solicitando ao usuário que tome alguma decisão ou realize alguma ação. A necessidade de muitas escolhas pode deixar o usuário aborrecido, desse modo, a aplicação deve tomar as decisões. Em caso, ainda, de ocorrer algo errado ou de o usuário discordar de uma das suas ações, a aplicação deve dar a possibilidade de desfazer aquela ação ou decisão. |

* Fonte: Android Developers (tradução adaptada)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 10

* Quadro 2: Princípios de design relacionados com a meta Simplify My Life.

|  |  |
| --- | --- |
| Only show what I need when I need it (Somente mostre-me o que eu preciso quando eu necessitar). | Os usuários tendem a ficar sobrecarregados quando veem muitas coisas de uma só vez. Segmente as tarefas e as informações em partes de tamanho suficiente para uma boa assimilação. Oculte opções que não são essenciais no momento e faça a aplicação ensinar como eles devem fazer determinada tarefa. |
| I should always know where I am (Eu devo sempre saber onde estou). | Dê às pessoas confiança de que elas sabem o caminho de volta. Crie locais em seu aplicativo, observe a distinção e use transições para mostrar relações entre as telas. Forneça um feedback sobre as tarefas em andamento. |
| Never lose my stuff (Nunca perca minhas coisas). | Salve o que os usuários levaram tempo para fazer e deixe que eles possam acessar de qualquer lugar. Lembre-se de configurações e toques pessoais nos telefones, tablets e computadores. Busque simplificar a atividade de atualização das coisas criadas pelos usuários. |
| If it looks the same, it should act the same (Se ele tem a mesma aparência, deve fazer a mesma atividade) | Ajude as pessoas a discernirem as diferenças funcionais, tornando-as distintas visualmente, em vez de visualmente sutis. Evite modos de uso nos quais lugares/telas são visualmente similares e funcionam de forma diferente em resposta à mesma entrada/interação. |
| Only interrupt me if it's important (Só me interrompa se for importante). | Como um bom assistente pessoal, proteja os usuários de minúcias sem importância. Os usuários querem manter o foco em suas tarefas, e uma interrupção pode ser incômodo para ele. |

* Fonte: Android Developers (tradução adaptada)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 10

* Quadro 3: Princípios de design relacionados com a meta Make me amazing.

|  |  |
| --- | --- |
| **Make me amazing** (Faça-me incrível) | |
| **Princípio de Design** | **Descrição** |
| Give me tricks that work everywhere (Dê-me “truques” que funcionam em todos os lugares). | As pessoas se sentem muito bem quando elas descobrem as coisas por si. Faça o seu aplicativo mais fácil de aprender, aproveitando padrões visuais e de memória de outros aplicativos do Android. |
| It's not my fault (Não é minha culpa). | Seja gentil na maneira como a aplicação leva os usuários a fazerem correções. Eles querem se sentir espertos quando usam a sua aplicação. Se alguma coisa der errado, dê claras instruções de reparação, mas poupe os detalhes técnicos. |
| Sprinkle encouragement (Encoraja-me). | Segmente tarefas complexas em pequenos passos que podem ser facilmente realizados. Dê feedbacks sobre ações, mesmo que seja apenas uma dica sutil. |
| Do the heavy lifting for me (Faça o trabalho pesado para mim). | Faça os novatos se sentirem como especialistas, permitindo-lhes fazer coisas que nunca pensaram que poderiam. Por exemplo, atalhos que combinam múltiplos efeitos fotográficos podem fazer fotos amadoras se tornarem incríveis em apenas alguns passos. |
| Make important things fast (Faça o que é importante com rapidez). | Nem todas as ações são iguais. Decida o que é mais importante na sua aplicação, torne-o fácil de encontrar e fácil de usar. Por exemplo, o botão de pausa em um leitor de música possui uma função importante e pode ser necessário a qualquer momento durante a execução de uma música, logo, ele precisa ser fácil de ser achado e simples de usar. |

* Fonte: Android Developers (tradução adaptada)

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 10

#### Saiba Mais!



Existem mais informações sobre o desenvolvimento de interfaces gráficas em aplicações Android no site dos desenvolvedores Android. Sugerimos que você visite o endereço <http://developer.android.com/design/style/index.html> para conhecer os estilos de interfaces gráficas do Android, e o endereço <http://developer.android.com/design/patterns/index.html> para conhecer os padrões de design e interação de interfaces gráficas das aplicações Android. Esses websites irão ajudar você a conhecer melhor, como usuário, o funcionamento das aplicações e do sistema Android em termos de design e interação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 10

### fechamento



Chegamos ao final de nossa aula. Aqui, estudamos os principais componentes gráficos visuais (Widgets) do Android. Aprendemos como podemos utilizar rótulos, caixas de texto, botões de ação, botões de seleção, lista suspensa, mensagens de alerta, janelas de diálogo e como tratar eventos oriundos da interação do usuário com esses Widgets. Além disso, estudamos alguns princípios de design de interfaces gráficas para o desenvolvimento de aplicações Android.

Mas não é só isso. Continue estudando e pesquisando sobre o assunto. Caso tenha dúvidas quanto ao conteúdo desta aula, converse com o seu tutor.

Bons estudos! Até mais!

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 10

#### Referências

* + Android Development Site: User Interface. Disponível em: <http://developer.android.com/guide/topics/ui/index.html>. Acesso em: 16 jan. 2019.
  + 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.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página10 de 10

## INTRODUÇÃO

Olá, aluno (a),  
Você já deve ter estudado os conceitos para utilização de interfaces gráficas, assim como já deve saber como implementá-las adequadamente, certo? Afinal, é importante que venhamos a produzir interfaces com boa usabilidade e de forma que transpareça objetividade para o usuário.

Sem dúvidas, o conceito de usabilidade não se restringe somente a boas interfaces gráficas, mas também na navegação, como evitar que o usuário sempre precise digitar sua senha para realizar login. Uma possível solução para isso é criar uma opção “Lembrar-me”, assim, o fluxo de acesso até as telas principais será reduzido. O que você acha de criarmos essa opção? Para isso, precisaremos de um banco de dados.

Sendo assim, nesta aula, aprenderemos a criar um banco de dados local para a nossa aplicação, usando SQLite e as classes do pacote android.database.sqlite. Vamos começar?

**Objetivos**

* + Conhecer o banco de dados SQLite;
  + Compreender como ocorre a criação de um banco de dados SQLite em aplicações Android;
  + Aprender a inserir e manipular dados em um banco de dados SQLite.

1 de 1

Página 1 de 1

## TÓPICO 1 – Aspectos Gerais sobre SQLite

**Objetivos**

* + Conhecer as características e as limitações do SQLite.

Durante esta aula, aprenderemos a criar e manipular bancos de dados locais em aplicações da plataforma Android. Diferentemente de aplicações Java para Desktop ou Web, na plataforma Android, não podemos escolher qual dos diversos Sistemas de Gerenciamento de Banco de Dados (SGBDs) do mercado de tecnologia iremos utilizar. Apenas o banco de dados SQLite está disponível. Mas, por que o SQLite? Vamos conhecer um pouco sobre esse banco de dados antes de começar a utilizá-lo?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página1 de 8

Página 1 de 8

* De acordo com o site oficial, <https://www.sqlite.org/index.html>, o SQLite é uma biblioteca que implementa um banco de dados relacional que não necessita de servidor ou configuração, e que é compatível com a linguagem SQL. Por ser um software livre, de código aberto, indicado para ser utilizado em aplicações que precisam manipular um pequeno volume de dados, o SQLite foi escolhido para fazer parte da plataforma Android. Confira na figura a seguir a logo utilizada para representar o banco de dados.

Figura 1: Logo do banco de dados SQLite.Fonte: <https://www.sqlite.org/index.html>.

#### Você Sabia?



SQLite é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 8

* Falamos que o SQLite é um software livre, mas você ainda lembra o que isso quer dizer?

#### Saiba Mais!



Um software livre não é necessariamente gratuito, ou seja, que você pode utilizar sem pagar. Na verdade, um software é considerado livre quando contempla as 4 liberdades definidas pela Free Software Foundation (FSF), que são: usar o software para qualquer propósito, poder estudá-lo e modificá-lo para atender a necessidades próprias, poder redistribuí-lo e aperfeiçoá-lo para necessidades da comunidade e distribuir sua versão modificada.

Saiba mais no portal Software Livre Brasil, em <http://softwarelivre.org/portal/o-que-e>.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 8

* Entre algumas das principais características do SQLite, podemos citar:
  + Implementa o modelo relacional de banco de dados, com suporte a transações (transactions) e linguagem SQL;
  + Ocupa pouco espaço em memória – a biblioteca completa tem cerca de 350Kb de tamanho;
  + Todo o banco de dados fica em um único arquivo;
  + Não necessita de configuração ou de um servidor para ser executado – a biblioteca é o próprio sistema do banco de dados (SGBD).

O SQLite possui 3 tipos de dados básicos: TEXT (equivalente a String em Java), INTEGER (equivalente a Integer em Java) e REAL (equivalente a Float/Double em Java). Para armazenar dados com formatação especial, como data e hora, utilizaremos uma das funções do SQLite para formatar o valor antes de inseri-lo no banco, transformando-o para TEXT, INTEGER ou REAL.

E sobre a segurança de armazenamento desses dados (TEXT, INTEGER e REAL) em nosso SGBD SQLite? As outras aplicações podem acessar dados da nossa aplicação? Vamos descobrir.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 8

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba1)
* [**TÓPICO 1Aspectos Gerais sobre SQLite**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba2)
* [TÓPICO 2Configurando o Acesso ao SQLite](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba3)
* [TÓPICO 3Manipulando Dados em um Banco de Dados](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba4)
* [TÓPICO 4Mapeamento Objeto Relacional no Android](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba5)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba6)

Inicio do Conteudo

* Todo dispositivo móvel que possui o sistema operacional Android já vem com o SQLite. Cada aplicação pode criar um ou mais arquivos no seu espaço de armazenamento interno para funcionar como banco de dados. Os dados que serão armazenados nesses arquivos estarão seguros, pois o Android não permite que uma aplicação acesse os arquivos do espaço de armazenamento interno de outra aplicação.

Agora que sabemos disso, de que forma podemos começar a construir esse banco de dados? É necessário termos o conhecimento de quais recursos precisaremos para começar a configurar o banco de dados. Qual pacote e classes utilizaremos?

Tudo o que é preciso para manipular um banco de dados SQLite em uma aplicação Android está disponível no pacote android.database.sqlite. Segundo a sugestão do guia de desenvolvimento do Android, o que pode ser conferido neste site: <https://developer.android.com/training/basics/data-storage/databases.html>, para criar e manipular o banco de dados, precisaremos de três integrantes: uma classe de contrato, uma classe Helper e uma classe SQLiteDatabase.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 8

Página 5 de 8

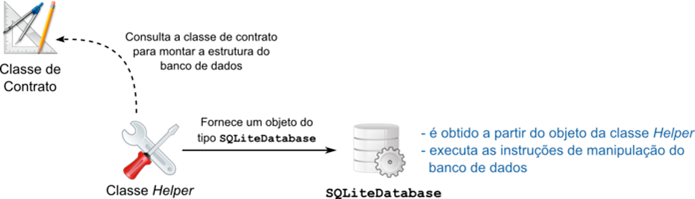
* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba1)
* [**TÓPICO 1Aspectos Gerais sobre SQLite**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba2)
* [TÓPICO 2Configurando o Acesso ao SQLite](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba3)
* [TÓPICO 3Manipulando Dados em um Banco de Dados](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba4)
* [TÓPICO 4Mapeamento Objeto Relacional no Android](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba5)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba6)

Inicio do Conteudo

* A classe de contrato conterá os nomes do banco de dados e das tabelas que ele irá conter, além dos nomes das colunas das tabelas. É comum armazenar também, na classe de contrato, alguns Strings contento consultas (queries) de manutenção, como geração do banco, criação e exclusão de tabelas e inserção de dados padrão.

A classe Helper deve ser uma subclasse de SQLiteOpenHelper e tem como função sobrescrever os métodos responsáveis pela construção do banco de dados no primeiro acesso e reconstrução do banco quando houver atualização de versão. Além disso, é através de uma instância da classe Helper que a nossa aplicação terá acesso ao banco de dados, para, então, executar as instruções de inserção, remoção, atualização e busca de dados.

A relação entre esses integrantes é ilustrada na figura a seguir:

Figura 2: Esquema de interação entre a classe de contrato, Helper e da aplicação.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 8

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba1)
* [**TÓPICO 1Aspectos Gerais sobre SQLite**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba2)
* [TÓPICO 2Configurando o Acesso ao SQLite](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba3)
* [TÓPICO 3Manipulando Dados em um Banco de Dados](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba4)
* [TÓPICO 4Mapeamento Objeto Relacional no Android](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba5)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba6)

Inicio do Conteudo

* Já na classe da nossa aplicação, teremos um objeto do tipo SQLiteDatabase, obtido a partir de uma instância da classe Helper. Com esse objeto, poderemos executar os métodos que permitem a manipulação do banco de dados. Nos próximos tópicos, abordaremos como a classe de contrato, a classe Helper e a classe da nossa aplicação interagem para que possamos criar e manipular um banco de dados SQLite.

#### Saiba Mais!



<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 8

* [INTRODUÇÃO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba1)
* [**TÓPICO 1Aspectos Gerais sobre SQLite**](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba2)
* [TÓPICO 2Configurando o Acesso ao SQLite](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba3)
* [TÓPICO 3Manipulando Dados em um Banco de Dados](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba4)
* [TÓPICO 4Mapeamento Objeto Relacional no Android](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba5)
* [GLOSSÁRIO](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/index-fisico.html#aba6)

Inicio do Conteudo

* Neste tópico, nós conhecemos o banco de dados SQLite, suas principais características e por que ele foi incorporado ao Android. Estudamos que o SQLite é um banco de dados relacional leve, simples e com suporte a transações e linguagem SQL. Conhecemos também a infraestrutura básica que deve ser provida para acesso a um SGBD SQLite em uma aplicação Android e como ocorre a interação entre suas classes principais.

No tópico seguinte, aprenderemos a configurar o acesso a um banco de dados SQLite a partir da nossa aplicação exemplo. Preste bastante atenção em como vamos realizar isso, pois o procedimento é padrão e possibilitará que as demais aplicações Android que necessitam de acesso a esse banco de dados se conectem facilmente. Vamos lá?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 8

## TÓPICO 2 – Configurando o Acesso ao SQLite

**Objetivo**

* + Compreender a função da classe de contrato e da classe Helper;
  + Aprender a criar um banco de dados SQLite.

No tópico anterior, conhecemos as características do banco de dados SQLite. Agora, conheceremos as classes que formam a infraestrutura de acesso a um SGBD SQLite. Aprenderemos em que momento da execução da aplicação o banco de dados é criado e em que casos ele pode ser modificado.

Para entender melhor os conceitos estudados até aqui e o que detalharemos neste tópico, nada melhor do que praticar o processo de configuração do acesso ao banco de dados e definir um exemplo de operações diretas no esquema desse banco. Então vamos praticar?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página1 de 14

Página 1 de 14

### 2.1 Contextualização da Aplicação Exemplo

Para compreender melhor a configuração de acesso ao banco dados, é necessário que você já tenha um certo conhecimento sobre alguns assuntos, principalmente sobre a linguagem SQL. O que você deve ter estudado sobre bancos de dados como MySQL, Postgres, Oracle, entre outros, irá facilitar bastante a compreensão do conteúdo desta aula. Caso você queira revisar o assunto, acesse o link disponível no Saiba mais e relembre alguns conceitos básicos a respeito.

#### Atenção



Para aproveitar esta aula ao máximo, você precisa ter uma boa noção sobre linguagem SQL. Se você ainda tem dúvidas sobre esse assunto, procure um material de estudo e revise os conceitos de tabela, relacionamento e consultas SQL. Este é um bom tutorial: <https://www.w3schools.com/sql/sql_intro.asp>.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 14

* Para estudarmos o processo de criação de um banco de dados e, posteriormente, inserirmos e manipularmos esses dados, usaremos um exemplo de uma aplicação que analisaremos a partir de agora até o final da aula. Essa aplicação tem como objetivo guardar lembretes do usuário num banco de dados SQLite. Cada lembrete tem um título e um texto que corresponde à sua descrição. A aplicação será capaz de criar, modificar e excluir lembretes.

#### Atenção



Vamos manter o foco na criação do banco de dados e na implementação da manipulação de dados. Você pode conferir os detalhes da aplicação fazendo o download do projeto neste link: <https://d11s0xq1vqg039.cloudfront.net/videos_conteudo/devmoveljava/aula07/arquivos/Lembretes.zip>.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 14

* Ao pulsar no ícone do aplicativo, é exibida uma tela de listagem dos lembretes e um botão Novo Lembrete, que possibilita criarmos um novo lembrete com título e descrição. Sendo assim, devemos criar um novo lembrete e, após isso, podemos editá-lo através do botão Editar.

Lembretes são muito úteis para nos notificar que há um compromisso importante, como um jantar marcado, o pagamento de uma conta, ou quando é necessário estudar para uma prova.

Ao selecionarmos a opção Novo Lembrete do nosso aplicativo já desenvolvido, este possibilita uma tela com dois campos de texto: título e descrição. Nos campos, é possível definir o assunto principal do lembrete no campo Título e seus respectivos detalhes em Descrição. Feito isso, podemos invocar o botão Salvar para que o novo lembrete seja armazenado pela aplicação e listado na tela principal.

Caso venhamos a perceber que alguma palavra está errada no título ou descrição do lembrete, podemos acessá-lo na listagem dos lembretes e selecionar a opção Editar. Após isso, a mesma tela de criação do lembrete é exibida com as informações atuais nos campos de título e descrição, ambos editáveis para que possamos fazer as devidas alterações. E caso o compromisso já tenha ocorrido? Podemos selecioná-lo na tela de listagem e escolher a opção Excluir, com isso, o lembrete será removido da aplicação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 14

### 2.2 Criação da Estrutura do Banco de Dados

O primeiro passo para a criação de um banco de dados SQLite é a definição de uma classe de contrato. Uma classe de contrato consiste basicamente em um conjunto de String que identifique os elementos do banco de dados. A sugestão da documentação do Android é que essa classe possua:

* + Um String para o nome do banco de dados;
  + A versão atual do esquema do banco de dados (valor inteiro);
  + Uma classe interna que implemente BaseColumns (interface que provê nome para as colunas mais comuns em tabelas: ID e COUNT) para cada tabela do banco, contendo, por sua vez:
    - Um String para o nome da tabela;
    - Um String para o nome de cada coluna da tabela, exceto ID;
    - Um conjunto de Strings de consultas predefinidas de criação da tabela.

Que tal colocarmos praticarmos com base nessas dicas? Vamos começar implementando a classe Contrato da aplicação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 14

* Na nossa aplicação, o pacote aula09.lembretes.db possui a classe de contrato chamada ContratoBanco e contém o nome e versão do banco de dados, além da classe TabelaLembrete, que, por sua vez, contém os nomes das colunas da tabela Lembrete e uma consulta do tipo CREATE. Ainda no mesmo pacote, temos a classe Helper, chamada LembretesHelper. Essa classe é uma extensão de SQLiteOpenHelper e, a partir da estrutura definida na classe de contrato, permite a criação do banco de dados no dispositivo móvel. Nela, definimos um método construtor que utiliza o nome do banco de dados e a versão definidos em ContratoBanco, assim como uma referência ao contexto da aplicação.

Vamos observar o código da classe ContratoBanco a seguir. Perceba que na linha 05 é definido o nome do banco e, na linha 06, a versão dele (que pode variar conforme fazemos modificações no esquema do banco de dados). Além disso, entre as linhas 08 e 19, é definido o nome da tabela, suas colunas e uma consulta para criação da tabela.

01 package aula.lembretes.db;

02 import android.provider.BaseColumns;

03 public class ContratoBanco {

04 // Informações Gerais do banco

05 public static final String NOME\_BANCO = "lembretes.db";

06 public static final int VERSAO\_BANCO = 1;

07 // Informações da tabela 'lembretes’

08 public static abstract class TabelaLembrete implements BaseColumns{

09 // Colunas da tabela 'lembretes’

10 public static final String NOME\_TABELA = "lembretes";

11 public static final String COLUNA\_TITULO = "titulo";

12 public static final String COLUNA\_DESCRICAO = "descricao";

13 // Query de criação da tabela

14 public static final String SQL\_CRIAR\_TABELA\_LEMBRETES =

15 "CREATE TABLE " + NOME\_TABELA + "(" +

16 TabelaLembrete.\_ID + " INTEGER PRIMARY KEY, " +

17 TabelaLembrete.COLUNA\_TITULO + " TEXT, " +

18 TabelaLembrete.COLUNA\_DESCRICAO + " TEXT " + ")";

19 }

20 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 14

* A classe LembreteHelper tem outro objetivo: sobrescrever o método onCreate() e, opcionalmente, os métodos onConfigure(), onOpen(), onUpgrade() e onDowngrade(). O método onCreate() é executado quando o banco de dados não existe e, por isso, deve conter instruções que efetivem a sua criação. Por isso, devemos sobrescrever este método em virtude do seu uso ser essencial para criação do banco. Observe o código a seguir, atentando-se principalmente aos comentários entre as linhas 05 e 08 e entre as linhas 13 e 16.
* 01 package aula.lembretes.db;
* 02 import android.provider.BaseColumns;
* 03 public class LembretesHelper extends SQLiteOpenHelper {
* 04 public LembretesHelper(Context context) {
* 05 /\*
* 06 \* O terceiro parâmetro refere-se a um modelo de Cursor, que não será
* 07 \* estudado daqui. Portanto, fornecemos 'null' no lugar desse parâmetro.
* 08 \*/
* 09 super(context, ContratoBanco.NOME\_BANCO, null, ContratoBanco.VERSAO\_BANCO);
* 10 }
* 11 @Override
* 12 public void onCreate(SQLiteDatabase banco) {
* 13 /\*\*
* 14 \* Dentro do método onCreate devemos executar o código de construção
* 15 \* da tabela, definido na classe de contrato.
* 16 \*/
* 17 banco.execSQL(TabelaLembrete.SQL\_CRIAR\_TABELA\_LEMBRETES);
* 18 }
* 19 ...
* 20 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 14

#### Saiba Mais!



O método onUpdate() é importante para aplicações que possuem um banco de dados evolutivo, ou seja, que sofre modificações durante o tempo. Quando o número de versão do banco muda (indicando atualização), o método onUpgrade() é automaticamente executado para efetuar as devidas mudanças nas tabelas do banco, como alterar nomes e propriedades de colunas de tabelas. Uma prática comum é destruir tabelas e recriá-las com a nova estrutura, mas essa estratégia implica em perda de dados e deve ser usada com cautela.

Em nosso caso, apenas a tabela Lembretes será criada dentro do método onCreate(). Um objeto do tipo SQLiteDatabase é usado para executar o método execSQL(), que recebe como parâmetro uma consulta SQL pronta. Esse método pode receber também um conjunto de parâmetros para injetar na consulta, mas, em nosso caso, que consiste na criação de uma tabela, todas as informações necessárias já estão dentro da consulta.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 14

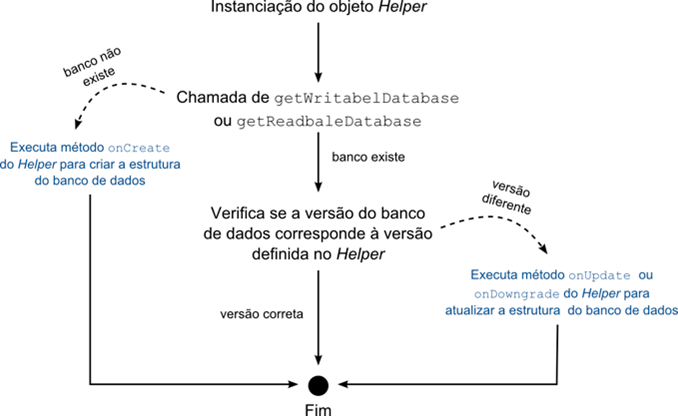
* A última peça para a gestão do banco de dados é uma classe que utilize a classe Helper. No nosso projeto, ela se chama LembretesDatabase. Esta classe terá apenas métodos estáticos que possam ser executados pelas Activities da aplicação. Cada método será responsável por uma ação na tabela de Lembretes, como recuperar, inserir ou excluir registros (operações típicas de um [CRUD](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=CRUD)).

A classe LembretesDatabase possui um objeto do tipo LembretesHelper. Quando esse objeto é instanciado, ocorre o processo de verificação da existência do banco de dados da aplicação. Dependendo da existência do banco de dados e da sua versão atual, ele pode ser criado ou atualizado. Podemos imaginar duas situações: aplicativo nunca foi instalado no dispositivo do usuário e, por isso, de imediato chamará o método onCreate() para ser criado; o aplicativo já está instalado, mas precisa, por exemplo, de uma atualização específica em uma coluna da tabela.

Em razão dessas situações, devemos programar nossa lógica de criação/atualização do banco de forma segura.

A seguir, perceba que a figura 3 representa o fluxo que acabamos de comentar.

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

* Figura 3: Verificação da existência do banco de dados no dispositivo móvel.Fonte: Dell Computadores/LE@D.

O objeto Helper será responsável por verificar se o esquema do banco de dados existe ou não, também se este se encontra em uma versão distinta. Caso isso ocorra, o método onUpdate() ou onDowngrade() do Helper será chamado para que a estrutura do banco de dados seja atualizada. Por outro lado, caso o esquema do banco de dados não exista, o método onCreate() será chamado para a criação do banco.

Com o banco de dados criado, devemos definir uma classe para gerenciar as operações de [CRUD](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=CRUD). Vamos lá?

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

* Com o acesso ao banco de dados configurado, definimos a classe LembretesDatabase responsável por possuir os métodos de inserção na linha 23, método de remoção na linha 29, método de atualização na linha 26 e o método de listagem na linha 32. Agora faça o download do código neste link: [[download do código].](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/code/Codigo_3_%20p_21.txt)

O trecho de código corresponde à classe LembretesDatabase. Neste momento, apenas iremos nos preocupar em declarar a assinatura dos métodos, posteriormente, cuidaremos da implementação detalhada de cada um deles.

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

* O método inicializar() é responsável por instanciar o objeto lembretesHelper, através do qual os métodos inserirLembrete(), atualizarLembrete(), excluirLembrete() e todosLembretes() podem ter acesso ao banco de dados. Todo o acesso é feito através do objeto banco, do tipo SQLiteDatabase, que é instanciado através dos métodos getWritableDatabase() e getReadableDatabase(), da classe Helper.

O método encerrarSessao() finaliza o acesso ao banco de dados, executando o método close() a partir do objeto banco. Os métodos inserirLembrete(), atualizarLembrete(), excluirLembrete() e todosLembretes(), responsáveis pela manipulação de dados do banco, serão estudados em detalhes no próximo tópico.

Podemos dizer, então, que a classe LembretesDatabase controla o acesso ao banco de dados. Tudo o que precisamos fazer é executar os métodos apropriados para cada ação (inserir, atualizar, excluir lembretes) a partir das Activities da aplicação. Abordaremos sobre como fazer isso no próximo tópico.

O banco de dados da aplicação só é criado ou atualizado quando um dos métodos getWritableDatabase() ou getReadableDatabase() é executado. Ambos devolvem uma instância de SQLiteDatabase, através da qual é possível submeter comandos ao banco.

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

#### Saiba Mais!



A abordagem que estudamos até aqui é uma forma organizada de armazenar as informações estruturais do banco de dados e tabelas da aplicação. Você também pode armazenar consultas prontas em Strings únicas, mas essa abordagem torna a manutenção do código difícil e demorada. Se um nome de alguma coluna precisar mudar, por exemplo, você pode precisar verificar diversas classes e métodos em busca de trechos que precisam ser atualizados.

O fluxo básico da criação do esquema de banco de dados foi definido neste tópico, dessa forma, já temos uma ideia de como esse SGBD trabalha. Isso se torna relevante para definirmos exatamente como o código de criação e atualização do esquema do banco de dados se comportará no momento em que for requisitado.

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

* Neste tópico, conhecemos as classes que formam a infraestrutura de acesso ao banco de dados SQLite para aplicações Android. Estudamos que a classe SQLOneHelper fornece acesso ao banco de dados através dos métodos getWritableDatabase() e getReadableDatabase().

No próximo tópico, iremos colocar em prática exemplos de como usar os métodos insert(), update(), delete() e query() da classe SQLiteDatabase para manipular dados do banco de dados em uma aplicação exemplo. Por meio desses métodos, poderemos fazer uso dos conceitos de [CRUD](https://leadfortaleza.com.br/ead2pcd/app/glossario?termo=CRUD), permitindo que o usuário manipule suas informações no banco de dados.

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

## TÓPICO 3 – Manipulando Dados em um Banco de Dados

**Objetivo**

* + Conhecer os métodos de acesso ao banco da classe SQLiteDatabase;
  + Conhecer as boas práticas de uso do SQLite a partir de Activities.

Nos tópicos anteriores desta aula, definimos alguns conceitos básicos do SQLite e também seu fluxo de trabalho para criação/atualização do banco de dados, utilizado na plataforma Android. Além disso, abordamos quais classes são utilizadas para oferecer uma infraestrutura de acesso ao banco de uma aplicação. Nos tópicos anteriores desta aula, definimos alguns conceitos básicos do SQLite e também seu fluxo de trabalho para criação/atualização do banco de dados, utilizado na plataforma Android. Além disso, abordamos quais classes são utilizadas para oferecer uma infraestrutura de acesso ao banco de uma aplicação.

Neste tópico, exploraremos cada um dos métodos de acesso ao banco de dados da classe SQLiteDatabase, utilizando-os em uma abordagem prática em nossa aplicação exemplo. Vamos detalhar os métodos utilizados e as possíveis situações que devemos programar para não prejudicar o fluxo de execução da aplicação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página1 de 28

Página 1 de 28

### 3.1 Boa Prática: Uso de AsyncTask

O acesso a banco de dados SQLite em um dispositivo Android consiste na manipulação de arquivos no disco do dispositivo móvel. Este tipo de operação pode ser bastante demorada. Por isso, para evitar que a interface gráfica fique bloqueada enquanto ocorre leitura ou gravação de dados no banco, utilizaremos instâncias especializadas de AsyncTasks para cada tipo de atividade: inserção, busca, atualização e exclusão de registros. O uso de AsyncTasks está relacionado com Threads (ambos trabalham juntos para proporcionar um processamento de instruções de forma mais eficiente e assíncrona).

Na classe ListaDeLembretes, que corresponde à Activity principal da aplicação, temos três classes internas que herdam de AsyncTask: InserirTask, ExcluirTask e CarregarLembretesTask. Essas classes têm a função de disparar o acesso ao banco de dados em uma Thread auxiliar, para evitar o travamento da aplicação. Já a classe ExibirLembretes, que corresponde à Activity de visualização de Lembrete, possui uma classe interna com o mesmo objetivo, chamada EditarTask.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 28

* Cada uma dessas classes com nome terminado em Task são subclasses de AsyncTask e tem como objetivo a execução de uma ação específica no banco de dados de forma assíncrona, ou seja, fora da UI thread, com o objetivo de não bloquear a interface gráfica da aplicação. A classe InserirTask, por exemplo, trabalha apenas com a inserção de novos registros no banco, enquanto a classe EditarTask, trabalha apenas com a atualização de registros que já estão no banco.

Em termos gerais, cada uma das classes terminadas em Task terão três métodos sobrescritos, são eles:

* 1. onPreExecute(): Irá exibir um diálogo de espera (classe Dialog) para que o usuário saiba que existe um processo longo em execução;
  2. doInBackground(): Executa a operação desejada no banco de dados;
  3. onPostExecute(): Fecha o diálogo de espera, efetua as alterações na interface gráfica da Activity atual e notifica o usuário sobre o fim da operação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 28

* A estratégia que utiliza AsyncTask é sugerida pela documentação oficial do Android, mas pode ser implementada de maneira similar, usando Handlers ou Threads manualmente gerenciadas pelo desenvolvedor. O importante é executar os métodos getReadableDatabase() e getWritableDatabase() fora da UI (User Interface) Thread.

Dissemos que podemos obter acesso ao banco de dados através de dois métodos da classe SQLiteDatabase: getWritableDatabase() e getReadableDatabase(). Esses dois métodos devolvem uma instância de SQLiteDatabase, através da qual podemos executar os métodos que são específicos para inserção, alteração, busca e exclusão de registros no banco de dados.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 28

### 3.2 Buscando Registros

Embora tanto getWritableDatabase() quanto getReadableDatabase() forneçam permissão para leitura e escrita, em casos muito específicos (como disco cheio ou problema de permissões de acesso ao disco), o método getReadableDatabase() pode devolver uma instância de SQLiteDatabase com acesso somente-leitura aos dados. Por questões de organização, usaremos getReadableDatabase() antes de executar consulta a dados no banco, ou seja, busca de dados. Para os demais casos, usaremos getWritableDatabase().

A busca de dados é feita através do método query() da classe SQLiteDatabase. Este método possui quatro versões diferentes. No projeto a ser desenvolvido nesta aula, usaremos a versão que recebe menos parâmetros. Confira-os a seguir.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 28

* + Nome da tabela alvo da busca (String);
  + Lista de colunas que devem ser trazidas no resultado (String[]);
  + Cláusula WHERE da consulta, sem os valores de referência (String);
  + Valores que serão injetados na cláusula WHERE (String[]);
  + Filtro GROUP BY de agrupamento de resultados (String);
  + Filtro HAVING de seleção de grupos no resultado (String);
  + Critério ORDER BY de ordenação dos resultados (String).

Lembre-se de que nossa consulta SQL pode variar conforme o caso e, não necessariamente, somos obrigados a utilizar todos esses parâmetros. Todavia, dependendo da complexidade da consulta, devemos aplicar recursos de filtragem, agrupamentos, entre outros.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 28

* No exemplo de código que apresentaremos mais à frente, não usaremos os parâmetros referentes à filtragem e agrupamento, ou seja, WHERE, GROUP BY e HAVING. Isso porque nosso método irá apenas carregar todos os registros de lembretes do banco de dados, ordenados de forma decrescente por ID, o que garante que os registros mais recentes virão antes dos mais antigos.

Todas as versões de query devolvem como resultado um objeto do tipo Cursor (do pacote android.database.Cursor), que corresponde à lista de resultados encontrados pela consulta. Um objeto do tipo Cursor possui métodos que ajudam a percorrer os resultados, semelhante aos métodos da classe ResultSet, usada na tecnologia Java Database Connectivity (JDBC).

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 28

* O código a seguir corresponde à classe CarregarLembretesTask, a implementação de AsyncTask, que recupera os registros de lembretes do banco de dados. Note que, na linha 20 (lembretes = LembretesDatabase.todosLembretes();), executamos o método estático todosLembretes() da classe LembretesDatabase, que devolve um objeto do tipo List, contendo objetos do tipo Lembrete, correspondente aos registros de Lembretes do banco. Faça o download do código neste link: [[download do código]](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/code/Codigo_4_p_32.txt).

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página8 de 28

* Agora, vamos observar o seguinte código, que contém o método todosLembretes() da classe LembretesDatabase. Esse método foi citado no tópico anterior, mas seu conteúdo foi omitido (assim como inserirLembrete(), atualizarLembrete() e excluirLembrete()). Agora vamos escrevê-lo de forma que possamos recuperar as informações do banco, tratá-las e retornar uma lista de objetos do tipo Lembrete. Faça o download do código neste link: [[download do código]](https://leadfortaleza.com.br/ead2pcd/conteudo/tmp/myopenolat_1_101500544362976/aula/code/Codigo_5_p_33.txt).

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página9 de 28

* O primeiro passo do método todosLembretes() é obter uma referência ao banco de dados. Como o acesso será apenas de leitura, pois se trata da busca de dados, usaremos o método getReadableDatabase(). Em seguida, montamos um vetor de String contendo os nomes das colunas que queremos no resultado da consulta. Na linha 15 (String criterioOrdenacao = ContratoBanco.TabelaLembrete.\_ID + ' DESC';), definimos nosso critério de ordenação como “\_ID DESC”, ou seja, resultados ordenados por ID em ordem decrescente.

Na linha 28 (Cursor c = banco.query(ContratoBanco.TabelaLembrete.NOME\_TABELA), executamos o método query(), que efetua uma busca no banco de dados. Fornecemos como argumentos o nome da tabela alvo, a lista de colunas que devem estar no resultado e o critério de ordenação. A sequência de quatro “null” da linha 28 (colunas, null , null, null, null, criterioOrdenacao);) corresponde aos parâmetros da cláusula WHERE e seus valores, da cláusula GROUP BY e da cláusula HAVING.

Na linha 33 (if (c != null && c.getCount() > 0) {), usamos o método getCount() para verificar a quantidade de registros encontrados. Se a quantidade for maior que zero, seguimos para a extração dos resultados do objeto c (Cursor), que ocorre entre as linhas 33 e 53.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página10 de 28

* A interface Cursor oferece métodos para navegação na lista de resultados e captura de valores das colunas, dentre os principais, podemos destacar:

Quadro 1: Métodos importantes da classe Cursor.

|  |  |
| --- | --- |
| **Método** | **Função** |
| moveToFirst() | Move o ponteiro do cursor para o primeiro resultado da lista de resultados. |
| moveToLast() | Move o ponteiro do cursor para o último resultado da lista de resultados. |
| moveToNext() | Move o ponteiro do cursor para o próximo resultado da lista de resultados. |
| getColumnIndex(String) | Fornece o índice de uma coluna dentro do resultado. |
| getInt(int), getString(int), getLong(int), getFloat(int), getBoolean(int), ... | Fornece o valor da coluna indicada no registro apontado atualmente pelo cursor. |

Fonte: Dell Computadores/LE@D

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página11 de 28

* Os três primeiros métodos do quadro 1 são usados para navegar dentre os resultados obtidos na consulta. Em geral, executamos o método moveToFirst() para acessar o primeiro registro do resultado e, depois, executamos o método moveToNext() para acessar os registros seguintes, até alcançar o final da lista.

Os dois últimos métodos do quadro 1 são usados para acessar os valores de um registro que esteja sendo apontado pelo cursor. Não existe uma maneira de acessar o valor de uma coluna diretamente a partir do seu nome. Então a solução é buscar o índice de uma coluna através do seu nome, usando o método getColumnIndex(), e, depois, usar esse índice para acessar o valor da coluna, com um dos métodos getters, por exemplo: getInt, getFloat, getString etc.

Que tal detalharmos melhor como fazer uso desses métodos?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página12 de 28

* Para compreendermos melhor o uso dos métodos descritos anteriormente, é importante saber que um objeto do tipo android. database.Cursor, que corresponde ao resultado de uma consulta ao banco de dados, pode ser obtido pela execução dos métodos query() e rawQuery() da classe SQLiteDatabase, por exemplo, Cursor c = db.query(...);

O objeto cursor, de forma abstrata, pode ser comparado a uma tabela, que contém os registros apresentados na consulta, e uma seta, que referencia o registro que está sendo acessado. Por exemplo, podemos possuir uma tabela Aluno com as seguintes colunas ID, NOME e MÉDIA. Observe a tabela 1 com algumas informações.

Tabela 1: Exemplo de tabela Aluno com as colunas ID, NOME e MÉDIA.

|  |  |  |
| --- | --- | --- |
| **ID** | **Nome** | **Média** |
| 1 | André | 9.3 |
| 2 | Bianca | 8.7 |
| 3 | David | 7.8 |

Fonte: Dell Computadores/LE@D

Ao ser criado o objeto cursor, este não está apontando para nenhum elemento dessa tabela do banco de dados, para que o cursor possa referenciar o primeiro elemento, ou seja, o de ID = 1, devemos fazer uso do método c.moveToFirst(). Agora que o primeiro registro está selecionado, acessamos os campos usando os métodos getInt(), getString(), getFloat() etc., conforme pode ser observado no seguinte trecho de código.

01 Aluno a = new Aluno(c.getInt(0), c.getString(1), c.getFloat(2));

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página13 de 28

* Os valores inteiros 0, 1 e 2, passados, respectivamente, aos métodos getInt(), getString() e getFloat(), fazem referência às posições das colunas na tabela, ou seja, na posição 0, temos a coluna ID (do tipo Int); na posição 1, temos a coluna NOME (do tipo String); na posição 2, temos a coluna MÉDIA (do tipo Float).

Para acessarmos os demais registros, um a um, devemos fazer uso do método moveToNext(), que irá fazer o cursor referenciar a próxima linha da tabela, ou seja, c.moveToNext().

Mas até quando devemos usar esse método? Até acabarem todos os registros da tabela! Em nosso exemplo, temos apenas três linhas na tabela, todavia, podem haver cenários em que existem milhares de linhas, certo? E então, será que se torna viável usar o método moveToNext() de forma explícita milhares de vezes? Não mesmo, por isso, devemos auxiliá-lo por meio de um laço de repetição, conforme pode ser observado no seguinte trecho de código.

01 Cursor c = db.query(...);

02 c.moveToNext();

03 while(!c.isAfterLast()) {

04 Aluno temp = new Aluno(c.getInt(), c.getString(), c.getFloat());

05 c.moveToNext();

06 }

O método isAfterLast(), presente na linha 03, indica se a seta já está depois do último registro do cursor, ou seja, se o final da lista de registro foi alcançada. Por isso, enquanto houver registros na tabela, o trecho de código da linha 04 até a linha 05 será executado, e os objetos serão instanciados conforme os dados retornados do banco de dados.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página14 de 28

* Toda vez que a Activity principal (ListaDeLembretes) é exibida, o carregamento de registros do banco de dados mencionado anteriormente é executado. Fazemos isso executando a tarefa assíncrona CarregarLembrestesTask() no método onResume() da Activity. Assim, sempre que a Activity entrar em primeiro plano, o carregamento será feito para garantir que a lista exibida ao usuário estará sempre atualizada em relação ao banco de dados. O seguinte código serve para realizar isso. Observe.
* 01 @Override
* 02 protected void onResume() {
* 03 super.onResume();
* 04 // Ao entrar em primeiro plano, a lista de lembretes
* 05 // deve ser recarregada direto do banco de dados
* 06 new CarregarLembretesTask().execute();
* 07 }

Isso é relevante, pois o banco de dados pode ser atualizado e garantir que a lista na interface gráfica sempre estará atualizada quando a Activity entrar em primeiro plano.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página15 de 28

### 3.3 Inserindo Novos Registros

Para inserir novos registros no banco de dados da nossa aplicação, precisamos navegar para a tela de edição de lembretes (Activity EditarLembrete) e, por meio do botão Novo Lembrete da tela principal, podemos adicionar um novo lembrete com título e descrição. Uma intent será disparada para executar a Activity EditarLembrete, como ilustrado na figura 4.

Figura 4: Adicionando um novo lembrete.Fonte: Dell Computadores/LE@D.

Isso é relevante, pois o banco de dados pode ser atualizado e garantir que a lista na interface gráfica sempre estará atualizada quando a Activity entrar em primeiro plano.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página16 de 28

* Após o preenchimento do formulário de Novo Lembrete, se o usuário pressionar o botão Salvar, será criada uma nova Intent, contendo os dados do formulário, que irá fazer a aplicação retornar à Activity ListaDeLembretes. O método onActivityResult() de ListaDeLembretes será, então, executado para tratar a Intent de retorno, como pode ser observado no trecho de código a seguir. Observe que, na instanciação do objeto Lembrete, da linha 07 à 11, os parâmetros enviados ao construtor da classe são provenientes da Intent.
* 01 @Override
* 02 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
* 03 super.onActivityResult(requestCode, resultCode, data);
* 04
* 05 if(resultCode == RESULT\_OK){
* 06 // Montagem do lembrete de referência, para adição ou exclusão
* 07 Lembrete lembrete = new Lembrete(
* 08 data.getExtras().containsKey("id")?
* 09 data.getExtras().getInt("id"):null,
* 10 data.getExtras().getString("titulo"),
* 11 data.getExtras().getString("descricao"));
* 12
* 13 if (requestCode == NOVO\_LEMBRETE) {
* 14 // Operação em segundo plano de adição de novo lembrete
* 15 new InserirTask().execute(lembrete);
* 16 } else if (requestCode == EXIBIR\_LEMBRETE) {
* 17 if (data.getExtras().getInt("acao") == EXCLUIR\_LEMBRETE) {
* 18 // Operação em segundo plano de exclusão do lembrete
* 19 new ExcluirTask().execute(lembrete);
* 20 }
* 21 }
* 22 }
* 23 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página17 de 28

* Na linha 15 (new InserirTask().execute(lembrete);) do método onActivityResult(), do código anterior, criamos uma tarefa assíncrona de inserção de novo registro. Da mesma forma que a tarefa assíncrona de busca de registros, a classe InserirTask irá disparar a interação com o banco de dados no seu método doInBackground(), sua estrutura pode ser observada no código a seguir.
* 01 @Override
* 02 protected Void doInBackground(Lembrete... params) {
* 03 // Cópia do lembrete para referência local
* 04 novoLembrete = params[0];
* 05 // Dispara a adição do registro do lembrete
* 06 long idNovoLembrete = LembretesDatabase
* 07 .inserirLembrete(novoLembrete);
* 08 novoLembrete.setId((int) idNovoLembrete);
* 09 return null;
* 10 }

O método doInBackground(), de InserirTask (classe interna de ListaDeLembretes), tem a função de realizar, de maneira paralela à execução da aplicação, a inserção de valores no banco de dados local. Na linha 6, executamos o método inserirLembrete() de LembretesDatabase. Esse método recebe um objeto do tipo Lembrete como parâmetro, extrai seus dados e repassa-os para o método insert (de SQLiteDatabase).

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página18 de 28

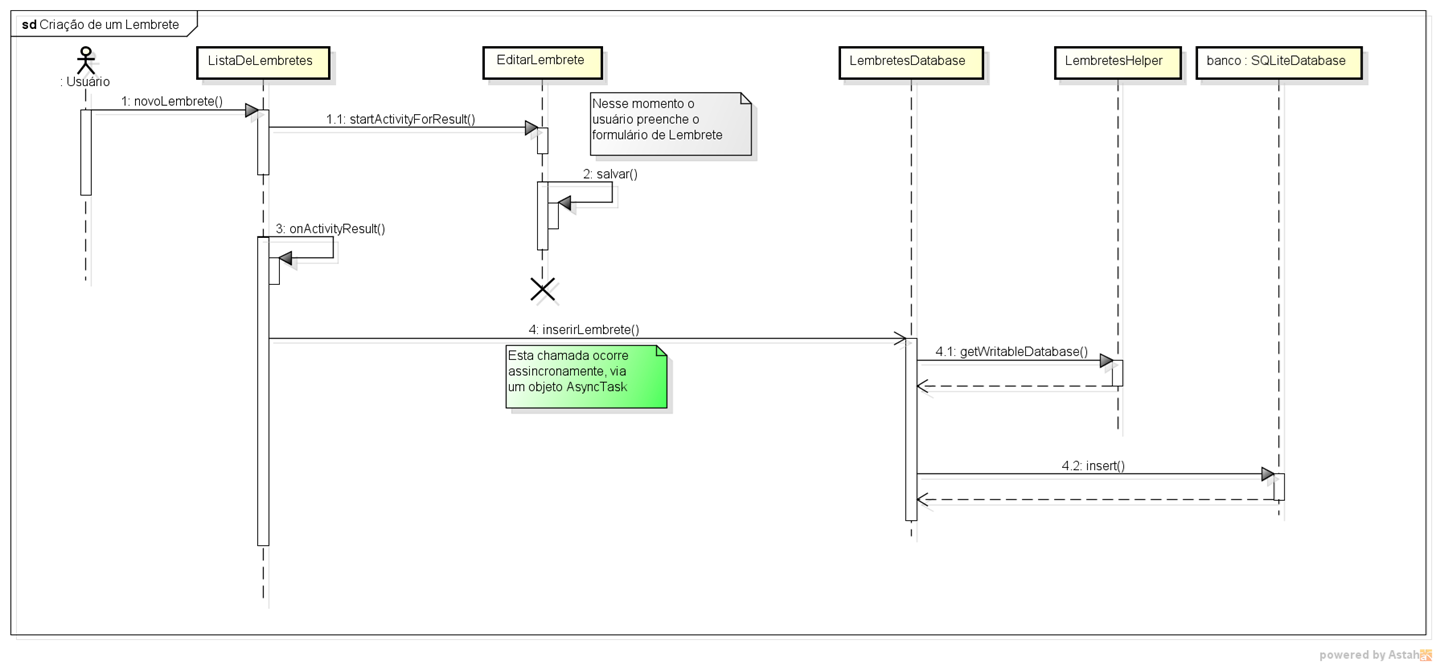
* O método insert() é bem mais simples que o query() e precisa apenas de três parâmetros, como pode ser observado no trecho de código a seguir, especificamente entre as linhas 11 e 13. Perceba que, desta vez, usaremos o método getWritableDatabase(), pois se trata de uma operação de gravação de dados.
* 01 public static long inserirLembrete(Lembrete lembrete) {
* 02 // Acesso à gravação necessário
* 03 banco = lembretesHelper.getWritableDatabase();
* 04
* 05 ContentValues dadosLembrete = new ContentValues();
* 06 dadosLembrete.put(ContratoBanco.TabelaLembrete.COLUNA\_TITULO,
* 07 lembrete.getTitulo());
* 08 dadosLembrete.put(ContratoBanco.TabelaLembrete.COLUNA\_DESCRICAO,
* 09 lembrete.getDescricao());
* 10
* 11 long idNovoLembrete = banco.insert(
* 12 ContratoBanco.TabelaLembrete.NOME\_TABELA,
* 13 ContratoBanco.TabelaLembrete.COLUNA\_DESCRICAO, dadosLembrete);
* 14
* 15 // Devolver o id que o banco gerou para o novo registro
* 16 return idNovoLembrete;
* 17 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página19 de 28

* O primeiro parâmetro de insert consiste no nome da tabela alvo, o terceiro parâmetro deve ser um mapa do tipo ContentValues, que contenha vários elementos compostos de chave e valor, no qual a chave deve ser o nome da coluna na tabela alvo, e o valor deve ser o conteúdo a ser gravado nessa coluna para o novo registro. Já o segundo parâmetro de insert corresponde ao nome de uma coluna em que o valor “null” pode ser injetado, caso o registro que se deseja criar seja completamente vazio. Esse caso é bem particular, mesmo assim o segundo parâmetro deve ser sempre fornecido.

Como esse trecho da aplicação possui troca de Activities, podemos definir o fluxo de navegação da seguinte forma: o usuário solicita a criação de um lembrete através da sequência de métodos novoLembrete(), em seguida, startActivityForResult() e, por último, o método salvar(). E a inserção de dados no lembrete com a sequência de métodos inserirLembrete(), depois o método getWritableDatabase() e, por último, o método insert(). Esse fluxo também pode ser ilustrado conforme o diagrama de sequência apresentado na figura 5, a seguir.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página20 de 28

* Figura 5: Diagrama de sequência da função Novo Lembrete.Fonte: Dell Computadores/LE@D.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página21 de 28

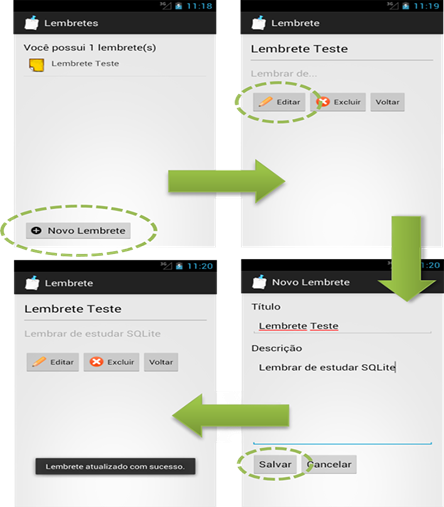
### 3.4 Edição de Registros (Atualização)

O processo de edição de um registro, que equivale ao comando UPDATE da linguagem SQL, tem um fluxo parecido com a criação de um novo registro. Para editar um Lembrete, a partir da Activity principal, devemos primeiro pressionar o item de um lembrete da lista. A tela de Visualizar Lembrete será, então, exibida.

Nessa tela, equivalente à Activity ExibirLembrete, podemos checar os dados de um Lembrete, o que implica em três opções: editar o lembrete, excluir o lembrete e voltar. Ao selecionar Editar, somos levados à tela de edição de lembretes, como no fluxo de criação que acabamos de detalhar. Todavia, nesse caso, o formulário estará preenchido, aguardando que o usuário efetue as modificações no lembrete.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página22 de 28

* Após editar o lembrete, o usuário deve fazer uso do botão Salvar. Nesse momento, o fluxo irá direcioná-lo de volta à Activity ExibirLembrete, onde será instanciada uma tarefa assíncrona para atualizar os dados do lembrete no banco de dados. Confira esse fluxo representado na figura a seguir.

Figura 6: Editando o lembrete.Fonte: Dell Computadores/LE@D.

Para que o lembrete seja corretamente editado, o método atualizarLembrete() deve ser implementado, para isso, devemos usar o método getWritableDatabase(). Vamos acompanhar como podemos fazê-lo.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página23 de 28

* Dentro do método doInBackground() da classe EditarTask (classe interna de ExibirLembrete), é executado o método atualizarLembrete(), da classe LembretesDatabase. A atualização no banco de dados é feita pelo método update(), que tem sintaxe correspondente a uma mistura de insert com query. Os parâmetros recebidos por esse método são o nome da tabela alvo, a lista de ContentValues contendo os valores a serem atualizados para cada coluna da tabela, a condição WHERE e os valores a serem injetados na condição WHERE.
* 01 public static boolean atualizarLembrete(Lembrete lembrete) {
* 02
* 03 // Acesso à gravação necessário
* 04 banco = lembretesHelper.getWritableDatabase();
* 05
* 06 // Novo valor por coluna
* 07 ContentValues dadosLembrete = new ContentValues();
* 08 dadosLembrete.put(ContratoBanco.TabelaLembrete.COLUNA\_TITULO,
* 09 lembrete.getTitulo());
* 10 dadosLembrete.put(ContratoBanco.TabelaLembrete.COLUNA\_DESCRICAO,
* 11 lembrete.getDescricao());
* 12
* 13 // Condição de atualização: ID do registro ser igual
* 14 // ao ID do lembrete que está sendo modificado.
* 15 String condicao = ContratoBanco.TabelaLembrete.\_ID + " = ?";
* 16 String[] valoresCondicao = { String.valueOf(lembrete.getId()) };
* 17
* 18 int registrosAtualizados = banco.update(
* 19 ContratoBanco.TabelaLembrete.NOME\_TABELA, dadosLembrete,
* 20 condicao, valoresCondicao);
* 21
* 22 // Se um registro foi atualizado, devolver true.
* 23 return registrosAtualizados == 1;
* 24 }

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página24 de 28

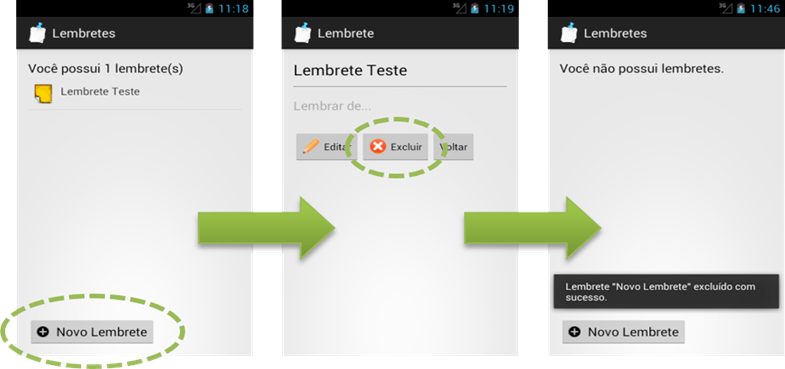
* A novidade nesse método está na linha 15 (String condicao = ContratoBanco.TabelaLembrete.\_ID + ' = ?';), onde definimos o critério de seleção WHERE. Perceba que não há necessidade de manter a palavra WHERE, basta listar os campos que farão parte da condição (você pode construir expressões complexas com conectores lógicos “OR” e “AND”) e usar o caractere-curinga ? no lugar em que devem ser injetados valores de parâmetro.

Na execução do método update, os caracteres-curinga ? serão, automaticamente, substituídos pelos valores fornecidos como parâmetros para a condição WHERE (objeto valoresCondicao, criado na linha 16 String[] valoresCondicao = { String.valueOf(lembrete.getId()) };). Esse mesmo mecanismo também vale para os métodos query() e delete(), como abordaremos em breve.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página25 de 28

### 3.5 Exclusão de Registros

A exclusão de registros é o último processo coberto pela nossa aplicação. Seu fluxo é similar ao de edição. Consiste em selecionar um lembrete na lista de lembretes e, na tela de visualização, pressionar o botão Excluir, conforme ilustrado na figura 7. Você será levado de volta à Activity ListaDeLembretes, Activity na qual a tarefa assíncrona de exclusão de lembrete pode ser criada, no método onActivityResult().

Figura 7: Excluindo lembretes.Fonte: Dell Computadores/LE@D.

Para que a exclusão de um registro possa ser efetivada com sucesso, o método excluirLembrete() deve ser invocado. Vamos detalhar melhor a estrutura desse método?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página26 de 28

* No método doInBackground() da classe ExcluirTask (classe interna de ListaDeLembretes), é executado o método excluirLembrete() de LembretesDatabase. A exclusão é feita pelo método delete(), linha 08 do código a seguir, que possui a sintaxe mais simples de todos os métodos de SQLiteDatabase que estudamos até agora. Só é necessário fornecer o nome da tabela alvo, a cláusula WHERE e os valores que serão injetados na cláusula WHERE. Observe o código a seguir do método excluirLembrete(), responsável por remover o lembrete de ID passado como parâmetro.
* 01 public static void excluirLembrete(int id) {
* 02 // Acesso à gravação necessário
* 03 banco = lembretesHelper.getWritableDatabase();
* 04
* 05 String condicao = ContratoBanco.TabelaLembrete.\_ID + " = ?";
* 06 String[] valoresCondicao = { String.valueOf(id) };
* 07
* 08 banco.delete(ContratoBanco.TabelaLembrete.NOME\_TABELA, condicao,
* 09 valoresCondicao);
* 10 }

#### Saiba Mais!



Para manter as boas práticas, estudamos o uso dos métodos insert(), delete(), update() e query() neste tópico. Porém, é possível executar comandos SQL puros através do método execSQL(), como abordamos para o caso da criação da tabela Lembrete (classe LembretesHelper, método onCreate()). Se você ainda não fez o download do projeto Lembretes, faça neste link: <https://d11s0xq1vqg039.cloudfront.net/videos_conteudo/devmoveljava/aula07/arquivos/Lembretes.zip> e confira mais detalhes de implementação.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página27 de 28

* Neste tópico, conhecemos os métodos query(), insert(), update() e delete() da classe SQLiteDatabase. Também estudamos a respeito de como utilizar AsyncTasks para executar o acesso a banco de dados em segundo plano, com o objetivo de evitar o bloqueio da aplicação.

Lembre-se de que o acesso a banco de dados é sempre um processo lento e que seguir a recomendação de acessá-lo em uma tarefa de segundo plano é essencial para a qualidade da aplicação. Por esse motivo, devemos sempre realizar tarefas em segundo plano.

No próximo tópico, iremos comentar sobre o conceito de Mapeamento Objeto Relacional na plataforma Android, dessa forma, poderemos reduzir o trabalho para criação de tabelas no esquema do banco de dados. Vamos descobrir como?

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página28 de 28

## TÓPICO 4 – Mapeamento Objeto Relacional no Android

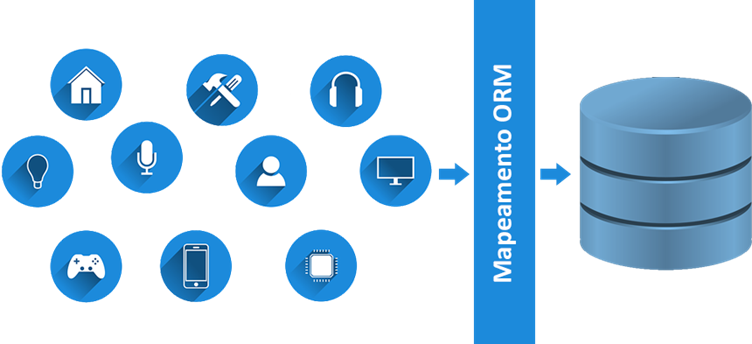
**Objetivo**

* Compreender o processo de Mapeamento Objeto Relacional na plataforma Android;
* Conhecer algumas das ferramentas de mapeamento para a plataforma Android.

Object Relational Mapper (ORM ou, em português, Mapeador Relacional de Objetos), é uma técnica utilizada para mapear objetos relacionais, ou seja, objetos que permitem fazer uma relação de objetos com os dados que os representam.

Existem algumas ferramentas que possibilitam o uso desta técnica para codificação Android, facilitando, assim, a escrita de códigos, além de simplificar, já que não é necessária a escrita de códigos extensos em SQL. Neste tópico, iremos compreender o conceito de ORM e entender de que forma essa técnica pode facilitar a persistência de dados na plataforma Android.

* A ideia principal do conceito de ORM é unir os dois mundos, relacional e orientado a objetos, uma vez que, no mundo relacional, os princípios matemáticos prevalecem para prover segurança no armazenamento e gerência de dados. Neste mundo, a linguagem SQL é amplamente utilizada para informar o que deve ou não ser feito para o banco de dados. O objetivo do ORM é modelar as entidades do mundo real, definidas em um código de programação, diretamente no banco de dados. Esse processo é representado pela figura a seguir. Perceba como ela engloba diversas entidades, como casa, ferramenta, computador, pessoa, celular etc., e modela essas entidades no banco de dados, tudo por meio do ORM.

Figura 8: Mapeamento do mundo de objetos para o mundo relacional.Fonte: Dell Computadores/LE@D.

A orientação a objetos facilita muito a organização do código, o reuso de informação, e deixa o código-fonte mais fácil de ser desenvolvido e mantido. Visando facilitar o uso de banco de dados relacional em aplicações, foram criados os frameworks de Mapeamento Objeto Relacional, que abstraem a parte relacional de acesso ao banco de dados. Para o programador, o acesso ao banco é feito de forma orientada a objetos, utilizando classes, objetos, e manipulando seus atributos e métodos.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página2 de 7

Página 2 de 7

* É importante esclarecer que a utilização de um framework ORM não substitui totalmente as consultas SQL, pois, em alguns casos em que a consulta seja complexa, é necessário escrever consultas SQL, que também são suportadas pelos frameworks.

Em geral, os frameworks ORM utilizam anotações para configurar as classes como entidades do banco. Para o programador, as tabelas do banco são representadas através de classes, e os registros de cada tabela são representados como instâncias das classes.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página3 de 7

* Utilizando uma ferramenta ORM, podemos ganhar produtividade, pois deixamos de escrever comando SQL para que o próprio ORM o faça. Repare que podemos partir do banco de dados para nossas classes, assim como podemos partir das classes diretamente para o banco de dados. Toda essa transição é realizada através do ORM.

#### Saiba Mais!



Existem algumas ferramentas ORM disponíveis para Android, algumas delas são:

* + ORMLite – disponível em: <http://ormlite.com/>.
  + Sugar ORM – disponível em: <http://satyan.github.io/sugar/index.html>.
  + greenDAO – disponível em: <http://greenrobot.org/greendao/>.
  + Active Android – disponível em: <http://www.activeandroid.com/>.
  + Realm – disponível em: <https://realm.io/blog/realm-for-android/>.

Apesar de haver diferenças em seus funcionamentos e métodos, todos estes propõem soluções ORM para a plataforma Android, facilitando o desenvolvimento de suas aplicações.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página4 de 7

* Saiba que o Mapeamento Objeto Relacional é uma técnica de programação para conversão de dados entre banco de dados relacionais e linguagens de programação orientada a objetos e que serve para facilitar a criação de banco de dados ou classes Java, reduzindo o trabalho do programador.

A princípio, não se preocupe em utilizar esses frameworks em seus projetos, mas é interessante que você esteja atento, pois eles podem contribuir na redução do seu trabalho e dar celeridade ao projeto.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página5 de 7

### fechamento



Chegamos ao final de nossa aula sobre banco de dados SQLite. Nela, conhecemos as características do SQLite e aprendemos como um banco de dados é criado para aplicações Android. Também conferimos boas práticas sobre a gestão de acesso ao banco de dados numa Activity, que deve utilizar sempre uma tarefa de segundo plano, como AsyncTask, para evitar que a aplicação pare.

Além de aprendermos mais sobre SQLite, também conhecemos algumas das principais ferramentas ORM para desenvolvimento Android, facilitando a implementação dos nossos projetos, através de uma maior facilidade de codificação, ganhamos tempo e simplicidade.

Coloque em prática os exemplos de códigos abordados nesta aula. Pesquise mais e tire dúvidas com o seu tutor. Bons estudos! Até mais!

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página6 de 7

#### Referências

* + Android Developer. Disponível em: <https://developer.android.com/?hl=pt-br>. Acesso em: 05 jan.2019.
  + SQLite. What is SQLite?. Disponível em: <https://www.sqlite.org/>. Acesso em: 05 jan. 2019.

<<<ir para a página anteriorir para a próxima páginair para a primeira páginair para a última página7 de 7

Adicionando o framework no build path do projeto:

implementation 'com.j256.ormlite:ormlite-android:5.1'

Criando a classe Livros:

package com.example.lead.ormlite;

import com.j256.ormlite.field.DatabaseField;

import com.j256.ormlite.table.DatabaseTable;

@DatabaseTable(tableName = "livro")

public class Livros {

@DatabaseField(columnName = "id", generatedId = true)

private int id;

@DatabaseField(columnName = "nome")

private String nome;

@DatabaseField

private String genero;

public Livros() {}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getNome() {

return nome;

}

public void setNome(String nome) {

this.nome = nome;

}

public String getGenero() {

return genero;

}

public void setGenero(String email) {

this.genero = genero;

}

}

Implementando a classe DataBaseManager:

package com.example.lead.ormlite;

import android.content.Context;

import android.database.sqlite.SQLiteDatabase;

import android.util.Log;

import com.j256.ormlite.android.apptools.OrmLiteSqliteOpenHelper;

import com.j256.ormlite.support.ConnectionSource;

import com.j256.ormlite.table.TableUtils;

public class DataBaseManager extends OrmLiteSqliteOpenHelper {

private static final String DATABASE\_NAME = "sistema.db";

private static final int VERSION = 1;

public DataBaseManager(Context context) {

super(context, DATABASE\_NAME, null, VERSION);

}

@Override

public void onCreate(SQLiteDatabase database, ConnectionSource connectionSource) {

try {

TableUtils.createTable(connectionSource, Livros.class);

}catch (Exception exception){

Log.e("DATABASE","Não foi possível criar a base de dados.",exception);

}

}

@Override

public void onUpgrade(SQLiteDatabase database, ConnectionSource connectionSource, int oldVersion, int newVersion) {

try {

TableUtils.dropTable(connectionSource, Livros.class, true);

onCreate(database, connectionSource);

}catch (Exception exception){

Log.e("DATABASE","Não foi possível atualizar a base de dados.",exception);

}

}

}

Criando a classe GenericDao:

package com.example.lead.ormlite;

import android.content.Context;

import com.j256.ormlite.dao.Dao;

import com.j256.ormlite.dao.DaoManager;

import java.util.List;

public class GenericDao<E> extends DataBaseManager{

protected Dao<E,Integer> dao;

private Class<E> type;

public GenericDao(Context context, Class<E> type){

super(context);

this.type = type;

setDao();

}

protected void setDao() {

try {

dao = DaoManager.createDao(getConnectionSource(),type);

}catch (Exception e){

e.printStackTrace();

}

}

public List<E> getAll(){

try{

List<E> result = dao.queryForAll();

return result;

}catch(Exception e){

e.printStackTrace();

return null;

}

}

public E getById(int id) {

try{

E obj = dao.queryForId(id);

return obj;

}catch(Exception e){

e.printStackTrace();

return null;

}

}

public void insert(E obj) {

try{

dao.create(obj);

}catch(Exception e){

e.printStackTrace();

}

}

public void delete(E obj) {

try{

dao.delete(obj);

}catch(Exception e){

e.printStackTrace();

}

}

public void update(E obj) {

try{

dao.update(obj);

}catch(Exception e){

e.printStackTrace();

}

}

}

Fazendo uso de herança através do extends:

package com.example.lead.ormlite;

import android.content.Context;

public class LivrosDAO extends GenericDao<Livros> {

public LivrosDAO(Context context){

super(context, Livros.class);

}

}

Criando Activities:

package com.example.lead.ormlite;

import android.content.Intent;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void inserirObjeto(View v){

LivrosDAO dao = new LivrosDAO(getApplicationContext());

Livros livro = new Livros();

livro.setGenero("Romance");

livro.setNome("Nome do Livro");

livro.setGenero("Acao");

livro.setNome("Nome do Livro 2");

dao.insert(livro);

//Activity extende Context

Intent intent = new Intent(this, SegundaActivity.class);

startActivity(intent);

}

}

Exibindo o nome dos livros da tabela Livros:

package com.example.lead.ormlite;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.widget.TextView;

import java.util.List;

public class SegundaActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_segunda);

LivrosDAO dao = new LivrosDAO(getApplicationContext());

List<Livros> ListaLivros = dao.getAll();

String msgTitulos = " ";

for (Livros l : ListaLivros) {

msgTitulos = msgTitulos + l.getNome() + "\n";

}

TextView textView = (TextView) findViewById(R.id.mensagem);

textView.setTextSize(40);

textView.setText(msgTitulos);

}

}