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.

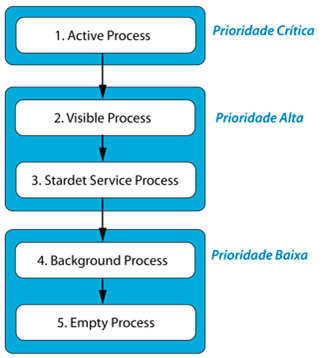
O Android adota uma política bem controlada de gerenciamento de recursos, visto que esta é realizada através do Android Runtime. Isso significa, por exemplo, que processos podem ser removidos da memória, sem qualquer aviso prévio, com o intuito de liberar recursos para outros processos associados a aplicações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

* Os processos do tipo Background Process e 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

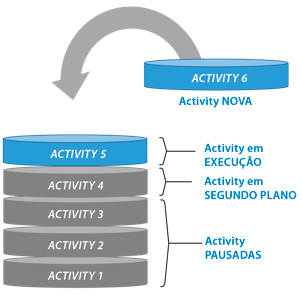
Neste tópico, conheceremos um pouco mais sobre o componente Activity, estudando em detalhes o seu ciclo de vida.

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

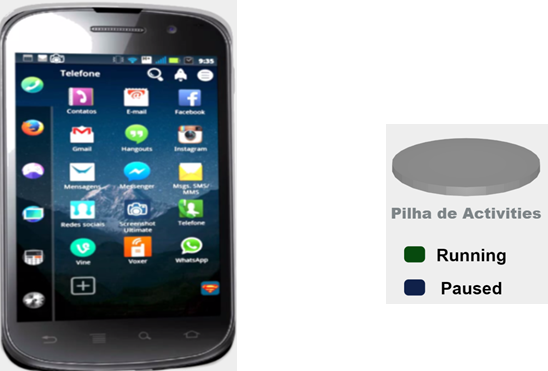
|  |  |
| --- | --- |
| RESUMED | Mais conhecido como Running (em português, Executando), nesse estado, a aplicação está executando em primeiro plano e tem o foco do usuário. |
| PAUSED | Em português, significa Pausado. Nesse estado, a Activity encontra-se temporariamente interrompida, desse modo, outra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

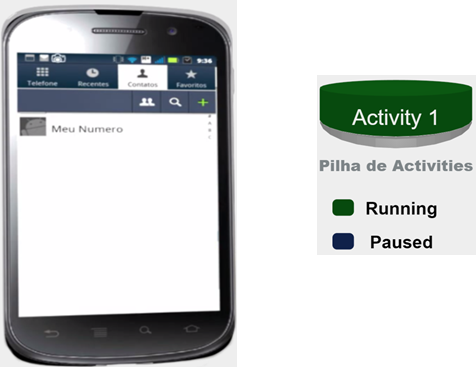
Figura 3: Pilha de Activities no Android.Fonte: Dell Computadores/LE@D.

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.

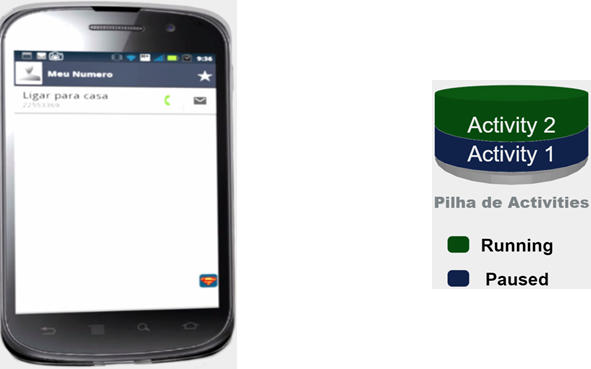
Vamos observar adiante o que acontece ao abrirmos a nossa primeira Activity e como se comportará a Pilha de Activities a partir disso.

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.

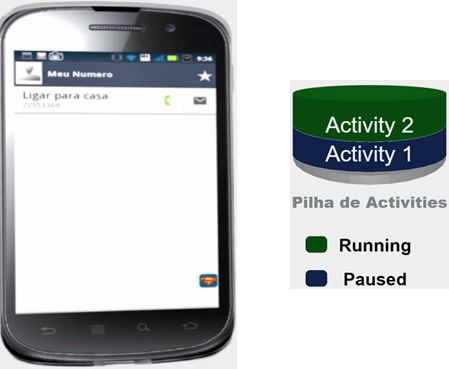
Que tal adicionarmos ainda mais uma Activity em nossa Pilha, para podermos observar o resultado? Em uma situação real, o usuário abrirá diversas Activities em uma aplicação enquanto navega.

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.

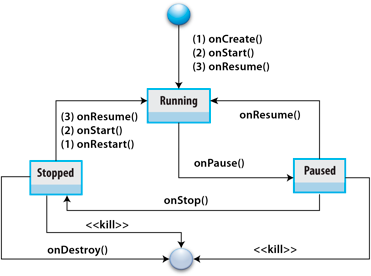
Figura 8: Activity 3 removida da Pilha de Activities.Fonte: Dell Computadores/LE@D.

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.