Dispositivos Móveis

Aula 04 - Activities I

Apresentação

Olá! Se você recorda da primeira aula, deve lembrar que discutimos os quatro componentes existentes no Android. Dentre esses componentes, destacamos o mais utilizado e mais importante: a Activity. Nessa aula, começaremos os estudos sobre esse componente. Como fizemos com o conteúdo da aula anterior, dividiremos o estudo das Activities em mais de uma aula, pois precisamos falar bastante sobre elas. Nesta primeira aula, estudaremos inicialmente como criar uma Activity e declará-la dentro de sua aplicação. Em seguida, estudaremos o diagrama de estados e o seu comportamento ao longo de todo o seu ciclo de vida. Essa parte é crucial para o desenvolvimento de boas aplicações Android e será bem detalhada em diversas subseções. Para finalizar, discutiremos as mudanças que ocorrem na Activity graças a mudanças na configuração do celular. Portanto, bastante atenção nesta aula!

Vídeo 01 - Apresentação

Como o desenvolvimento de interfaces gráficas para o Android é uma parte importante e com bastante conteúdo, preferimos dividir o estudo desses componentes em três aulas, intercalando-as com aulas sobre Activities, para garantir que será possível ir desenvolvendo aplicações básicas, à medida que o conteúdo for sendo desenvolvido.

Objetivos

Ao final desta aula, você será capaz de:

- Criar novas Activities nas aplicações Android.
- Tratar corretamente as mudanças ocorridas ao longo do ciclo de vida de uma Activity.
- Tratar eventos de mudança de configurações do aparelho.

O que é uma Activity e como implementá-la?

Como já discutimos anteriormente, a Activity é o componente do Android responsável por mostrar telas ao usuário e permitir que ele interaja com essas telas através de toques e cliques. Cada Activity representa uma tela e a programação por trás dela. Por exemplo, no caso de uma aplicação de e-mails, uma Activity seria responsável pela tela da listagem de e-mails, outra responsável pela exibição de um e-mail que venha a ser selecionado na tela anterior, outra responsável pela criação de novos e-mails a serem enviados e assim por diante. Essas Activities, apesar de trabalharem para um objetivo comum, são independentes. Uma Activity, mesmo que seja parte de sua aplicação, pode ser inicializada por uma outra Activity, até mesmo de uma outra aplicação. Como fazer isso é cena dos próximos capítulos, mas já tenha em mente que é possível. Um exemplo comum disso é o caso de uma aplicação precisar mandar um e-mail ao longo de sua execução. Essa aplicação, para não ter o trabalho de refazer algo que já está pronto, pode simplesmente fazer uma chamada ao cliente de e-mails padrão do celular. Prático!

Agora que já refrescamos a memória e relembramos os conceitos de Activity vistos na primeira aula, vamos finalmente aprender a implementá-la. A **Listagem 1** mostra a criação de uma Activity simples, apenas com os componentes básicos. Vejamos:

```
1  package br.ufrn.metropole;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class Aula04Activity extends Activity {
7      /** Called when the activity is first created. */
8      @Override
9      public void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.main);
12     }
13 }
```

Listagem 1 - Código da Activity Aula04Activity.

Dessa Activity básica, podemos tirar alguns pontos que são comuns a todas as Activities. O primeiro deles é que toda Activity deve estender da classe android.app.Activity, ou de uma classe que estenda dela (aqui vale relembrar os conceitos de herança, vistos em Programação Orientada a Objetos!). Isso é necessário para que os comportamentos padrões das Activities possam ser tratados, independentemente do que sua Activity se proponha a fazer. Também por esse motivo, cada um dos callbacks que forem implementados na Activity, como vemos no exemplo do onCreate() da **Listagem 1**, deve conter uma chamada ao seu pai, através da utilização do *super*. O onCreate(), por sua vez, é o único callback que deve ser obrigatoriamente implementado. Ele é o responsável pela criação da Activity, como estudaremos adiante, no ciclo de vida da Activity.

Outro ponto a ser destacado é a nomeação do pacote na qual sua aplicação estará contida. Esse ponto não está relacionado diretamente com Activities, mas é um ponto importante a se ter em mente desde o começo do desenvolvimento. Como vimos, o Android aconselha sempre utilizar um endereço web que seja seu como nome do pacote. Isso evita que, por acidente, duas aplicações trabalhem com o mesmo nome de pacote, já que dois endereços web não podem ser iguais. Caso você não possua um endereço web, apenas tenha em mente que sua aplicação deve estar em um pacote que não venha a colidir no futuro com uma outra aplicação instalada no aparelho.

Uma vez com a Activity criada e programada da maneira que se deseja, o próximo passo é declará-la no AndroidManifest.xml. Esse arquivo, padrão em todas as aplicações Android e criado automaticamente ao se criar um novo projeto Android, é o arquivo responsável por guardar as configurações que são de interesse de toda a aplicação. É nesse arquivo que se define quais serão os modos de tela que a aplicação vai suportar, quais as funcionalidades do telefone que serão utilizadas pela aplicação, qual a versão atual da aplicação, entre outras informações e configurações. Outra coisa que esse arquivo contém é a declaração das Activities que são implementadas pela aplicação, bem como as intenções (ou eventos, que veremos em mais detalhes em aulas futuras) que cada uma dessas Activities deve responder. Vejamos na **Listagem 2** a declaração da nossa Activity da **Listagem 1** no AndroidManifest.

```
<manifest ... >

   <uses-sdk ... />

   <application ... >
<activity android:name=".Aula04Activity" />
</application>

</manifest>
```

Listagem 2 - Declaração da Aula04Activity no AndroidManifest.xml. Veja que algumas partes não relevantes para a declaração da Activity foram omitidas.

Após ser declarada no AndroidManifest, a Activity está pronta para ser utilizada pela aplicação, ou até mesmo por outras aplicações, caso assim seja configurada. É importante lembrar-se de colocar cada nova Activity criada na sua aplicação dentro do Manifest, ou então erros que podem não fazer muito sentido, a princípio, podem ocorrer durante a execução de sua aplicação. Uma boa dica é colocar a Activity no Manifest assim que o arquivo for criado, antes mesmo de desenvolvê-lo! Por padrão, a IDE costuma fazer essa configuração assim que uma nova Activity é criada, mas não custa nada verificar!

Uma boa dica é criar a Activity pela parte gráfica do Android Studio. Para fazer isso, acesse o menu File -> New -> Activity e escolha o modelo desejado (por exemplo, Empty Activity, se você pretende criar uma Activity do zero). Ao utilizar esse método de criação para suas Activities, o Android Studio já se encarrega de declará-la adequadamente no AndroidManifest! Prático!

Agora que já vimos como criar e declarar uma Activity, precisamos conhecer os estados que ela pode ter ao longo de sua execução, bem como seu ciclo de vida completo, desde a criação, até a destruição. Esse conhecimento vai permitir criarmos aplicações que não deixam de funcionar, caso o usuário receba uma ligação, vire a tela ou até mesmo receba um aviso de bateria fraca.

Atividade 01

---

1. Crie um novo projeto e, dentro dele, crie uma Activity chamada HelloActivity.

2. Declare essa Activity no AndroidManifest.

Estados de uma Activity

Quando vimos, na primeira aula desse curso, as diferenças entre a programação para dispositivos móveis e a programação desktop, observamos que uma das principais diferenças é a possibilidade da ocorrência de eventos assíncronos, no caso da programação mobile. Esses eventos ocorrem sem que tenhamos controle sobre eles, e uma boa aplicação desenvolvida para dispositivos móveis deve sempre se preocupar em tratá-los corretamente.

Pensando nesses eventos e no ciclo normal da aplicação, o Android disponibiliza diversos métodos que devem ser implementados nas Activities para que respondam corretamente a cada estado em que a Activity possa ser colocada enquanto durar o seu ciclo de vida. Para compreender esses métodos, é necessário antes entender cada um dos possíveis estados da Activity. São eles:

- **Ativa**: Uma Activity nesse estado foi inicializada e está em primeiro plano, sendo a responsável por receber eventos e interagir com o usuário no momento.

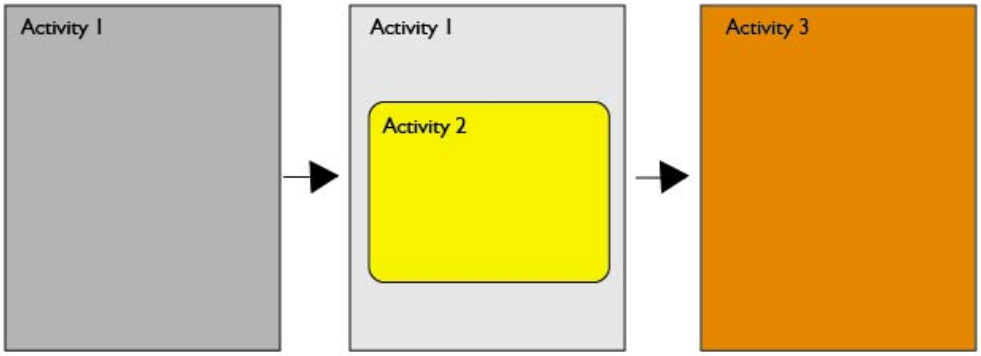
- **Pausada**: Quando uma janela toma a frente da Activity, mas não ocupa a tela inteira (uma notificação, por exemplo), a Activity que ficou para trás é colocada no estado de pausa. Nesse estado, ela ainda está visível, mas não está sendo executada.

- **Parada**: No caso quando uma nova janela de tela inteira toma a frente da aplicação, a Activity é então colocada no estado de parada. Nesse estado, ela não está sendo executada, nem mostrada na tela. É importante lembrar que nesse estado ela ainda é válida e pode voltar a ser executada e exibida na tela quando a que tomou a sua frente for finalizada.

- **Morta**: Esse estado indica que a Activity não foi inicializada (apesar de estar presente na aplicação) ou então que, por falta de memória, foi encerrada, ou ainda que foi encerrada graças ao usuário, que finalizou sua execução. Em todos esses casos, ela não mais existe na memória do dispositivo.

Veja que a mudança de estado pode ocorrer tanto de maneira controlada, como , por exemplo, uma Activity iniciando outra e sendo parada por esse motivo, quanto de uma maneira totalmente inesperada, como uma que estava sendo executada ser parada por uma ligação que está chegando e então morta pela falta de memória gerada ao atender essa chamada. Ambas as maneiras vão gerar na Activity o mesmo ciclo, passando pelos mesmos métodos e estados. Por isso, cuidado na hora de planejar os estados de sua Activity. Para facilitar o entendimento de cada um dos estados, vejamos a **Figura 1**.

Figura 01 - Estados de Activities



Na **Figura 1**, vemos, à esquerda, a Activity 1 em execução, ou seja, Ativa. As Activities 2 e 3 estão no estado Mortas, pois ainda não foram iniciadas. Ao centro da **Figura 1**, vemos um exemplo onde a Activity 2 passou a ser Ativa, enquanto a de número 1 passou a Pausada, pois está na tela, mas sem o foco. À direita da **Figura 1**, a Activity 3 foi inicializada, passando ao estado Ativa, enquanto as Activities 1 e 2, que ainda estão executando porém não estão visíveis, foram para o estado Paradas.

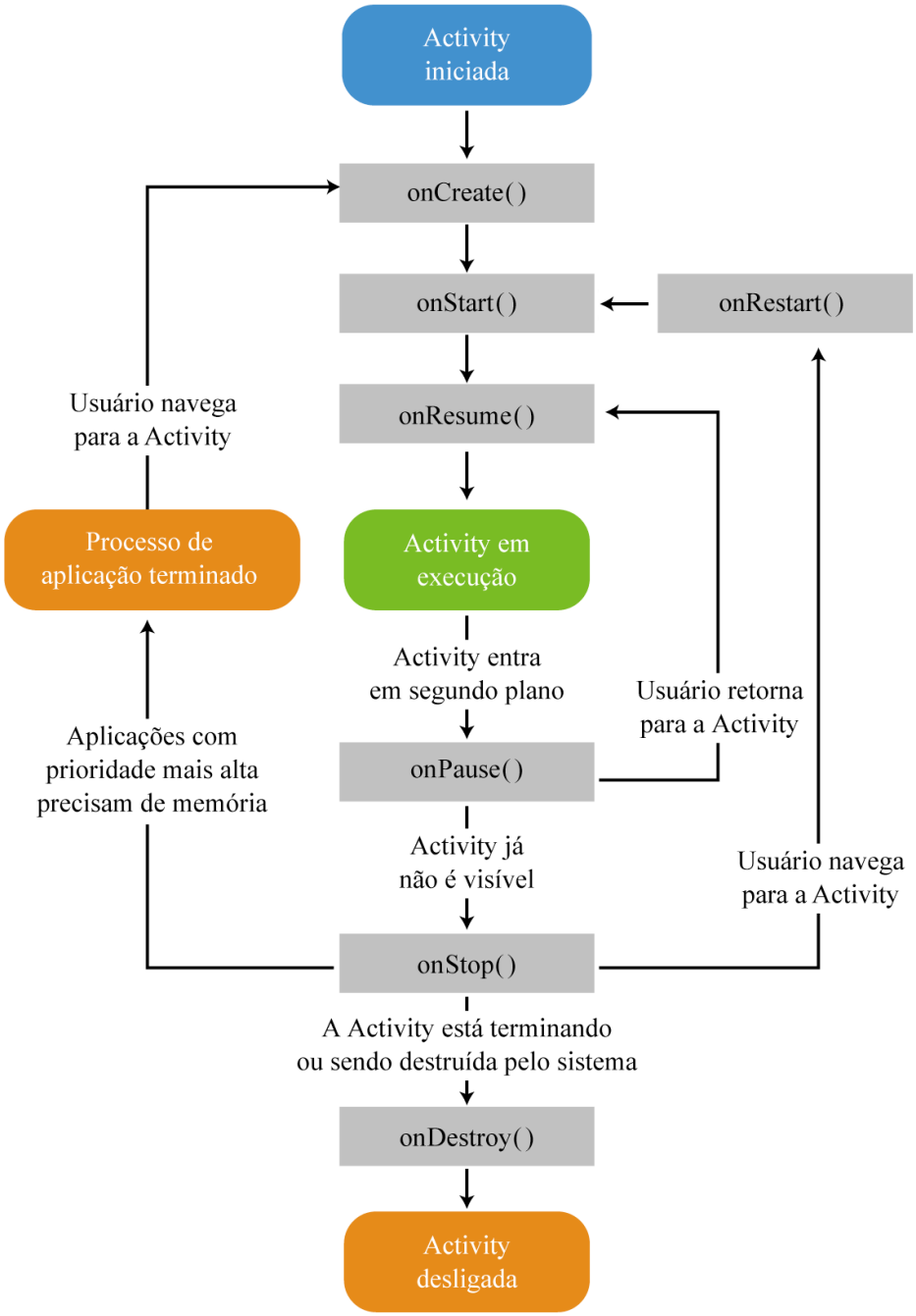
Ao iniciarmos uma nova Activity, ela será colocada no topo da pilha de exibição das Activities. É como se fossemos empilhando as telas da aplicação, sendo que cada nova Activity será colocada no topo, e as anteriores ainda irão existir, mas estarão paradas. Elas só irão voltar a estar ativas quando a(s) Activity(ies) que estão acima delas forem finalizadas. Uma vez finalizada, uma Activity será removida do topo da pilha e a Activity imediatamente abaixo voltará a ficar ativa.

Conhecendo os estados da Activity, podemos estudar o seu ciclo de vida e quais são os métodos que o Android disponibiliza para prepará-la para passar por cada fase do ciclo.

Ciclo de vida de uma Activity

Para começarmos nossos estudos sobre o ciclo de vida de uma Activity, vejamos a **Figura 2**. Essa imagem mostra todo o caminho que a aplicação segue, desde quando é lançada pela primeira vez, passando por todas as interrupções, até ser finalizada, seja pelo sistema ou pelo usuário.

Figura 02 - Ciclo de vida de uma Activity



Fonte: Adaptada de Android Developers

Vídeo 02 - Ciclo de Vida da Activity

Como vemos na **Figura 2**, o Android nos dispõe diversos métodos para tratar os diversos eventos que podem acontecer durante a execução da aplicação num dispositivo. A partir de agora, vamos discutir mais detalhadamente cada um desses métodos e qual a importância de implementá-los em sua aplicação.

Mas, antes disso, é importante lembrar alguns pontos que são comuns a todos esses métodos. Primeiramente, como já foi dito na primeira seção, esses métodos, ou callbacks, devem sempre conter uma chamada ao método do qual ele herda, utilizando a palavra-chave *super*. Essa chamada deve ser feita para que o Android possa tratar internamente o evento que está acontecendo, além de executar os passos que o usuário definiu para sua Activity.

Outro ponto importante é que esses métodos são todos invocados automaticamente pelo ambiente e não pelo usuário. Ao acontecer um evento que modifique o estado da Activity, o próprio Android se encarrega de detectar qual método deve ser chamado para notificar a Activity dessa mudança e então o faz, passando pelas instruções que o usuário tenha criado para esse callback da Activity. Com esses pontos comuns em mente, vamos estudar os métodos em particular.

Método onCreate(Bundle savedInstanceState)

Esse é o primeiro método a ser chamado quando uma Activity é inicializada e é o único dos callbacks que deve ser obrigatoriamente implementado em todas as Activities. Esse método deve ser utilizado para executar tarefas executadas apenas uma vez durante a execução de uma Activity. Um exemplo de ação que deve ser tomada nesse método é a inicialização da interface gráfica, bem como a definição das ações de clique e toque a serem tratadas pela Activity.

Existem três situações que nos levam à execução desse método. São elas:

- Quando a Activity é criada pela primeira vez;
- Caso a Activity seja finalizada prematuramente (graças a uma requisição de memória extra, feita por uma aplicação que tenha uma maior prioridade dentro do sistema, por exemplo) e precise ser executada novamente.

- Após alguma alteração nas configurações do dispositivo, como estudaremos adiante.

Mas, como você deve imaginar, pode haver grandes diferenças no que deve ser feito por um método responsável por iniciar a Activity (primeiro caso) para um método que deve recuperá-la após a Activity ter sido encerrada pelo sistema (segundo e terceiro casos). Para onde vão os dados que o usuário já tinha adicionado à tela? Como recuperar a tela no estado em que estava e não criá-la novamente? Tudo isso pode ser resolvido pelo parâmetro passado ao método onCreate.

O parâmetro savedInstanceState é o responsável por salvar qualquer dado que seja importante para a Activity antes que ela seja encerrada e então, também através desse parâmetro, os dados são devolvidos à Activity através do seu método onCreate. Ou seja, caso o savedInstanceState seja nulo, a aplicação pode assumir que não há nenhum dado especial a ser carregado e então simplesmente carregar os dados iniciais. Caso esse parâmetro venha com algum valor dentro dele, pode-se então utilizar esse valor para restaurar o estado anterior da aplicação.

Para entender melhor esse parâmetro, pense em uma Activity que exibe ao usuário um calendário, um mês de cada vez. O usuário pode navegar dez meses para frente e então, após passar por todos esses meses, receber uma ligação. Por algum motivo, essa ligação toma memória demais do celular e a aplicação é então encerrada pelo sistema. Caso não haja o tratamento do valor desse parâmetro, o sistema irá recriar a aplicação mostrando o mês inicial e não o mês para o qual o usuário havia navegado. Mas, caso o programador tenha tido o cuidado de implementar o método para salvar o estado da aplicação antes da mesma se encerrar, é possível utilizar esse valor salvo para retornar o calendário diretamente ao mês para o qual o usuário havia navegado.

O método **onSaveInstanceState(Bundle outState)** é o método responsável por salvar esses dados importantes antes do encerramento da aplicação. Uma vez que a aplicação necessite ser encerrada por um motivo qualquer, o Android tenta chamar esse método antes de destruir a Activity. O parâmetro desse método pode então ser modificado com os dados que devem ser salvos antes da aplicação encerrar. Uma vez reiniciada, esse mesmo Bundle será passado para o onCreate, permitindo ao

programador carregar qualquer dado salvo de volta na aplicação. A **Listagem 3** tem o código do exemplo do calendário, citado anteriormente. É importante ressaltar que também nesse método é necessário chamar o *super*.

Perceba que, para a Listagem 3 funcionar, precisamos implementar uma classe Calendário. Como esse não é o objetivo dessa aula, veja a listagem apenas como um exemplo de utilização do método onSaveInstanceState().

```
1  public class Aula04Activity extends Activity {
2      private int mesAtual = Calendar.getInstance().get(Calendar.MONTH);
3      private Calendario cal = new Calendario();
4
5      @Override
6      public void onCreate(Bundle savedInstanceState) {
7          super.onCreate(savedInstanceState);
8           setContentView(R.layout.main);
9
10     if (savedInstanceState != null) {
11         mesAtual = savedInstanceState.getInt("mesAtual");
12      }
13      calendario.mostrarMes(mesAtual);
14 }
15
16     @Override
17     protected void onSaveInstanceState(Bundle outState) {
18         super.onSaveInstanceState(outState);
19          outState.putInt("mesAtual", mesAtual);
20     }
21 }
```

Listagem 3 - Salvando e recuperando o estado da Activity

Perceba que no método onSaveInstanceState, o valor do mês atual é salvo dentro do Bundle para que, durante a recuperação da tela, a aplicação possa voltar ao mês que o usuário estava visualizando antes de encerrar-se a aplicação. Já no método onCreate, a aplicação testa se há algum valor salvo no Bundle. Caso haja algum valor lá, ela busca esse valor e altera o mês atual. Se o parâmetro passado for nulo, sabe-se então que é a primeira vez que a Activity está sendo criada, portanto o mês a ser mostrado no calendário é o atual. Salvar o estado da aplicação antes que ela seja encerrada é muito importante. Procure sempre detectar quais valores deverão ser salvos nas telas que você desenvolver!

Também é muito importante que o método onSaveInstanceState seja rápido em sua execução, uma vez que o sistema poderá utilizá-lo quando estiver precisando de recursos. Uma execução demorada desse método faria com que a liberação dos recursos fosse mais lenta e isso provavelmente resultaria em um travamento de alguma maneira para o usuário.

Método onDestroy()

Oposto ao onCreate, esse método é chamado quando a aplicação está sendo encerrada. Ele deve ser utilizado principalmente quando se é necessário liberar algum recurso que tenha sido alocado durante o ciclo de vida da aplicação. Esse método deve ser o responsável por encerrar conexões pendentes, fechar arquivos que tenham sido abertos, ou seja, liberar recursos que a Activity tenha passado a usar em qualquer fase do seu ciclo de vida.

É importante lembrar que nem sempre esse método é chamado. Apesar de ser raro, há casos de extrema necessidade de memória nos quais o Android pode encerrar a Activity sem passar por esse método. Por isso, evite ao máximo tentar salvar o estado da aplicação, ou fazer alguma coisa que seja crucial à Activity no método onDestroy.

O método pode ser chamado em duas situações distintas. A primeira delas é a situação rotineira, quando a Activity está sendo encerrada pelo usuário, através da tecla voltar, ou de algum caminho da Activity que cause o encerramento da mesma, como o toque em um botão ou mesmo o fim da execução do que a Activity se propunha a fazer. Já o outro caso, que é bem menos comum, é a chamada a esse método após a Activity ter sido finalizada pelo Android por falta de recursos, como falta de memória, por exemplo. Nesse caso, o método deve se portar do mesmo jeito, até porque não há como diferenciar os dois casos dentro do método.

Os Métodos onStart() e onRestart()

Esses métodos, apesar de bem parecidos, têm funcionalidades diferentes. Ambos são chamados quando uma Activity ganha a frente, ou seja, quando uma Activity passa a ser visível. A Activity pode se tornar visível em dois casos. No primeiro caso, ela acaba de ser criada, passou pelo método onCreate e agora está à frente, na tela. Nesse caso, como é a primeira vez que ela foi criada, o método onStart é chamado, para indicar que ela está agora visível ao usuário. Dentro desse método, deve-se fazer a inicialização de componentes visuais que não tenham sido inicializados no onCreate, caso ainda exista algum.

O segundo caso em que a Activity ganha a frente é quando ela estava visível, foi encoberta por uma outra Activity e agora está recuperando a frente. Nesse caso, o método onRestart é chamado antes do método onStart. Como não há indicações via Bundle, como acontece no método onCreate, esse é o meio que o Android tem de dizer ao programador se a Activity está se tornando visível pela primeira vez ou se está sendo reiniciada. É importante notar que mesmo que a Activity esteja sendo recriada, o método onStart será chamado. A diferença é apenas que o método onRestart será chamado antes.

Resumindo, se uma Activity não existia e está sendo criada a primeira vez, ou se ela foi destruída e precisa ser recriada, a sequência de métodos chamados é ilustrada na **Figura 3a**, sendo chamados os métodos onCreate e onStart, nesta sequência. Se a Activity estava no estado Parado, a sequência será a chamada do método onRestart, depois o método onStart, como mostrado na **Figura 3b**.

Figura 03a - Activity sendo criada pela primeira vez

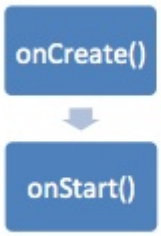
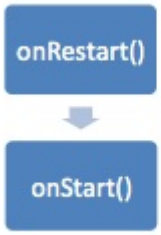


Figura 03b - Activity sendo reiniciada, voltando do estado Parado



Método onStop()

O método onStop é o método oposto ao onStart. Ele é chamado pelo Android quando a Activity sai completamente da visibilidade do usuário. Essa desaparição da Activity pode gerar a necessidade da liberação de alguns recursos, salvamento do estado da Activity, ou mesmo a parada de algumas atividades que estejam sendo executadas nela, para não descontextualizá-las ou para evitar a perda de informações por parte do usuário. Por exemplo, imagine que há uma música tocando em um jogo que está sendo executado na Activity mostrada. Caso ela saia completamente de vista, não faz sentido a música de fundo do jogo continuar tocando, pois isso pode até mesmo atrapalhar a execução da nova Activity. Porém, lembre-se que a Activity mesmo que não visível, poderá ainda estar sendo executada, a menos que seja destruída. Evite operações pesadas no onStop, pois a qualquer momento pode ser necessário voltar à Activity.

Normalmente, o método onStop é responsável por desfazer tudo que foi feito no onStart, o seu equivalente. Vale lembrar também que, assim como no onDestroy, em casos extremos, o Android pode finalizar a Activity sem passar por esse método. Por isso, evite utilizar esse método para salvar dados importantes, ou executar qualquer operação que seja crucial ao salvamento do estado da Activity. Isso poderá acarretar perda de dados por parte do usuário.

Método onResume()

O onResume é o último método chamado pelo Android antes da Activity se tornar novamente visível. Esse método é o responsável por fazer os últimos ajustes de tela antes da Activity passar a ser utilizada pelo usuário. Esse método é o único atingido em qualquer um dos fluxos antes da Activity estar completamente visível, sendo atingido tanto no fluxo normal de inicialização da aplicação, quanto na recuperação da tela, após a mesma ter sido omitida completamente ou parcialmente (Activity Parada ou Pausada, respectivamente). Se a Activity estava no estado Pausado, ao ser retornada, o método onResume será chamado imediatamente, mas se a Activity estava no estado Parado, antes serão chamados os métodos onRestart e onStart, como visto anteriormente, para somente depois ser chamado o onResume.

Esse método é o mais indicado para fazer atualizações na tela que sejam necessárias após alguma interrupção. Imagine que se está mostrando na tela um contador de tempo, atualizado de acordo com o tempo de visualização da Activity. Ao parar a Activity, o tempo para e, no método onResume, ele deve voltar a ser computado. Outro exemplo da utilidade do método onResume é a recuperação das animações em um jogo. Caso a Activity entre em estado de pausa, é pertinente pausar todas as animações que vinham sendo exibidas no jogo e então recuperá-las quando a Activity passe a ser novamente exibida por completo, sendo usado o método onResume para isso. Por fim, é importante recuperar recursos de uso exclusivo, como a câmera, nesse método. Esses recursos devem ser liberados assim que a Activity for pausada e podem então ser recuperados no método onResume, enquanto a Activity recupera o foco.

Método onPause()

O último callback que estudaremos é o onPause. Esse método é o oposto do onResume e é chamado imediatamente antes da Activity perder o foco. Diferentemente dos outros métodos de finalização da Activity (onStop e onDestroy), o onPause é sempre chamado, mesmo em casos extremos. Esse método deve

então, por esse motivo, ser um método de rápida execução, pois será executado por completo e irá bloquear uma nova Activity que tenha tomado a frente até terminar a sua execução. Uma vez terminada essa execução, o Android passa então a ter liberdade para finalizar a Activity a qualquer momento, caso um processo de maior prioridade requisite a memória que está sendo utilizada pela sua Activity. São nesses casos que a Activity é encerrada passando pelo onPause, mas não pelo onStop ou onResume.

Nesse método, deve-se desfazer tudo que foi feito no método onResume. Utilizando os mesmos exemplos citados na subseção anterior, podemos dizer que esse método é o responsável por parar o relógio que esteja sendo mostrado para contar o tempo da Activity, ou por parar as animações que estavam sendo exibidas no jogo, ou ainda liberar recursos de uso exclusivo (como a câmera, por exemplo) que estivessem sendo utilizados pela Activity para que a outra que tomou a frente possa utilizá-los. Conhecendo esse método, concluímos então o nosso estudo dos callbacks de uma Activity no Android.

O vídeo a seguir mostra, com um exemplo prático, o ciclo de vida de uma Activity. Apesar do vídeo utilizar o Eclipse, é possível acompanhá-lo, utilizando o mesmo código, no Android Studio, caso seja de seu interesse. O logcat, no Android Studio, fica localizado na parte inferior esquerda, na aba Android Monitor.

Vídeo 03 - Ciclo de Vida da Activity - pt.2

Atividade 02

1. Suponha que uma Activity foi inicializada pelo usuário e, assim que passou a ser executada, recebeu o comando para encerrar. Descreva, na ordem, por quais callbacks essa Activity passou.

2. Imagine que essa Activity deva utilizar a câmera (um recurso de uso exclusivo), carregar uma interface gráfica, realinhar essa interface, uma vez que tenha sido criada, e salvar o número de cliques que o usuário tenha

dado na tela, que está guardado em uma variável. Diga em qual callback cada uma dessas ações deve ser realizada e quais devem ser desfeitas, também citando o ponto em que devem se desfazer.

Mudanças nas Configurações

No Android, algumas configurações do dispositivo podem ser alteradas durante a execução de sua Activity. Vamos estudar nessa sessão quais os impactos que essas mudanças acarretam na Activity e como ela se comporta após essas mudanças.

Durante a execução da aplicação, o aparelho pode mudar a orientação da tela, graças a um posicionamento diferente do dispositivo. Outro exemplo é a mudança na disponibilidade de um teclado físico, causada pelo usuário que o conecta ao dispositivo. Esses são alguns exemplos de mudanças que podem ocorrer durante a execução da sua aplicação e que causam alterações tão grandes nos recursos que, para se adaptar a essas mudanças, o Android reinicia automaticamente a Activity que estava sendo executada.

Para fazer essa reinicialização, o Android faz uma chamada ao método onDestroy e, em seguida, uma nova chamada ao método onCreate. Para evitar a perda de dados e informações, deve-se utilizar o método onSaveInstanceState, estudado na sessão anterior. Uma vez destruída e com os dados salvos, a Activity é recriada orientada à nova tela, ou com as informações de disponibilidade de teclado físico alterada, por exemplo. Tenha bastante cuidado no tratamento dessas mudanças de configuração para evitar a perda de dados.

Vídeo 04

Leitura Complementar

ACTIVITIES. Disponível em: <http://developer.android.com/guide/topics/fundamentals/activities.html>. Acesso em: 03 dez. 2015.

Resumo

Nesta aula, começamos os nossos estudos sobre Activities. Iniciamos relembrando o que são e como devem ser implementadas e declaradas no AndroidManifest. Entendemos também um pouco do que está contido nesse arquivo. Na sequência, estudamos os estados nos quais uma Activity pode estar, detalhando cada um deles. Com essas informações, foi possível estudar o ciclo de vida da Activity e entender quando cada um dos callbacks são chamados, assim como o que deve ser feito em cada um deles. Para finalizar a aula, vimos alguns eventos que geram a reinicialização de uma Activity para que ela se readapte a mudanças que tenham ocorrido no dispositivo.

Autoavaliação

1. Qual a principal função de uma Activity?
2. Descreva os quatro estados de uma Activity.
3. Separe os callbacks de uma Activity em grupos de acordo com a função que realizam. O grupo deve conter o método que inicializa e o equivalente, utilizado na parada. Em seguida, dê um exemplo de recurso que deve ser inicializado/destruído no método.
4. Crie um novo projeto. Dentro desse projeto, crie uma Activity e declare-a no AndroidManifest. Nessa Activity, crie cada um dos callbacks. Dica: utilize o autocompletar.

Referências

ANDROID Developers. 2012. Disponível em: <http://developer.android.com>. Acesso em: 6 mai. 2015.

DIMARZIO, J. **Android:** a programmer's guide. São Paulo:McGraw-Hill, 2008. Disponível em: <http://books.google.com.br/books?id=hoFI5pxjGesC>. Acesso em: 8 maio 2015.

HASEMAN, C. **Android essentials**. Berkeley, CA. USA: Apress, 2008.

LECHETA, Ricardo R. **Google android:** aprenda a criar aplicaçoes. 2. ed. São Paulo: Novatec, 2010.

LECHETA, Ricardo R. **Google android para tablets**. São Paulo: Novatec, 2012.

MEIER, R. **Professional Android 2 application development**. New York: John Wiley & Sons, 2010. Disponível em: <http://books.google.com.br/books?id=ZthJlG4o-2wC>. Acesso em: 5 maio 2012.