Android - Cards de Referência

Activity

Bem vindo ao nosso terceiro card sobre Android! :-) Desta vez, vamos conversar sobre as Activities. Elas são essenciais na criação de aplicativos Android e não tem como você ficar alheio a esse assunto!

O QUE É UMA ACTIVITY?

De forma bem simplificada, podemos dizer que uma Activity corresponde a uma tela da sua aplicação, através da qual o usuário pode interagir. Normalmente, uma aplicação é formada por múltiplas Activities.

Você pode construir toda a tela do seu sistema, usando os mais diversos tipos de campo, como EditText, Spinners e etc, usando somente as Activities. Contudo, como veremos, esta não é a melhor forma de se desenvolver para Android.

Motivo? Simples: o ideal é que você faça uma separação do código que define o layout da tela daquele código que define a lógica de apresentação dos dados. Neste contexto, podemos considerar uma Activity como sendo um controlador. Ele não constrói a tela, mas fica responsável em tratar a interação do usuário com a aplicação, como o clique em um botão, por exemplo.

Você é desenvolvedor Web com JSF? Então, deve estar percebendo que uma Activity lembra bastante os ManagedBeans, não é? Se você se sente confortável com esta analogia, tudo bem, ela faz bastante sentido. :-)

ANATOMIA DE UMA ACTIVITY

O trecho de código abaixo é a Activity padrão criada pelo wizard do ADT. Vamos analisar alguns pontos dela.

```
public class MainActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.activity_main, menu);
        return true;
    }

}
```

Primeiro, toda classe que se propõe a ser uma Activity deve herdar de **android.app.Activity**. Feito isto, você pode implementar alguns métodos do ciclo de vida. Não sabe o que é isso? Vamos discutir sobre isto no próximo tópico.

Saiba apenas que o método **onCreate()** é sempre chamado quando a Activity é criada pela primeira vez. Use ele para definir o layout da Activity, obter elementos e por aí vai. Enfim, faça todo trabalho de inicialização da Activity aqui.

Só não esqueça de chamar o método do objeto pai usando **super.onCreate(**savedInstanceState**)**. Caso você não faça isso, será lançada uma exceção. A segunda linha deste método é usada para definir qual o arquivo de layout será associado a esta Activity.

Conforme vimos nos cards anteriores, os layouts são recursos do seu projeto e ficam na pasta **/res/layouts**. Também vimos que todo recurso pode ser referenciado no código Java através das constantes no arquivo **R.java**. Portanto, a segunda linha está indicando que o layout desta Activity será o recurso que possui o nome **R.layout.activity_main**.

Por enquanto, vamos esquecer o segundo método desta classe. Abordaremos ele em outros cards. Ele simplesmente define como será o menu. A classe **android.app.Activity** possui algumas classes filhas que são específicas para determinadas situações, como **android.app.ListActivity**. Abordaremos elas depois.

CICLO DE VIDA

Vamos entender o ciclo de vida de uma Activity. O processo de criar e ativar uma Activity é custoso. Portanto, não faz sentido o Android ter todo este trabalho para imediatamente destruir esta Activity assim que você deixar ela.

Entender o ciclo de vida é importante, pois lhe permite fazer as coisas certas na hora certa. Evitar desperdício de processamento, melhorar a "responsividade" e "usabilidade" de seu sistema e por aí vai.

Primeiro, entenda que as Activities que você cria são colocadas em uma pilha de activities. Se uma nova Activity é criada, a anterior fica logo abaixo e só pode voltar a aparecer se a nova for removida da pilha.

A imagem abaixo define todo o ciclo de vida de uma Activity. Vamos analisar ela. Os retângulos são métodos que são chamados conforme há a migração entre os estados. O ciclo é baseado essencialmente em quatro estados: executando, pausado, parado e finalizado.

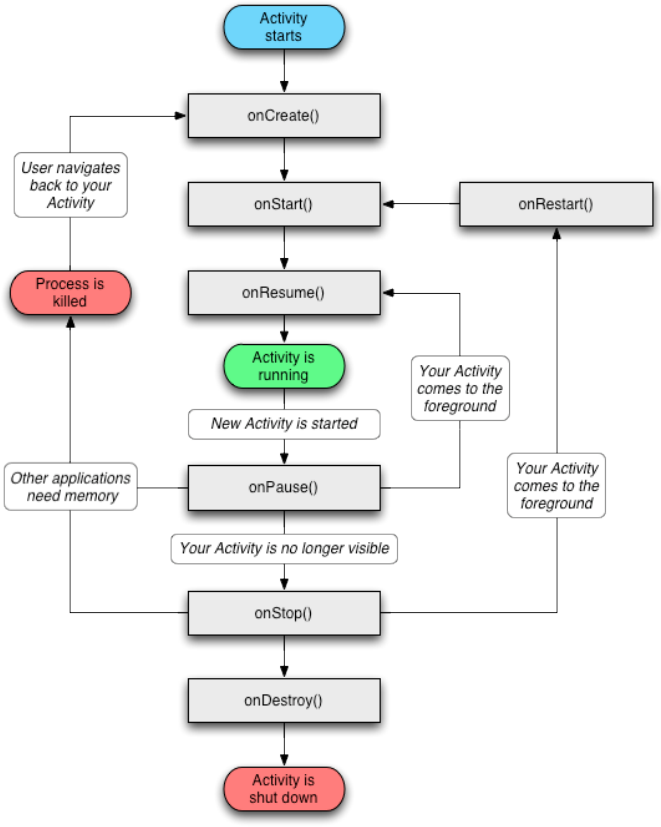
Executando: quando a Activity está visível para o usuário interagir com ela.

Pausado: quando uma outra Activity está por cima da atual. Ela continua visível para o usuário, mas não possui o foco e o usuário não pode interagir com ela diretamente. A Activity mantém todos os seus recursos e seu estado. Esta Activity pode ser destruída pelo Android em casos de extrema necessidade de memória.

Parado: quando a Activity não é mais visível. Ela foi totalmente encoberta por outra Activity. Isto significa que o Andorid pode matar, a qualquer momento, esta Activity, caso necessite de memória.

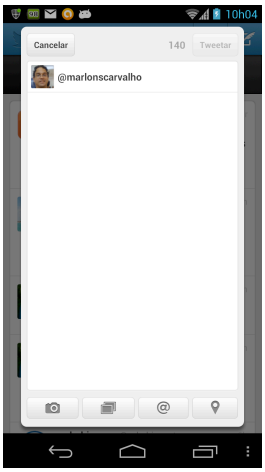
Finalizado: a Activity foi removida da memória. Sempre que houver necessidade de exibir ela de novo, terá que passar por todo o ciclo mais uma vez.

Vamos analisar a imagem do ciclo de vida, que está logo abaixo.



Toda Activity é inicialmente criada e, para isso, o método **onCreate()** é chamado. Observe que este método é chamado apenas uma vez durante todo o ciclo de vida e deve ser usado para você fazer o setup inicial da Activity: carregar dados em listas, exibir dados do banco de dados e etc. Logo após vem o método **onStart()**, seguido de **onResume()**. Observe, na imagem, que eles podem ser chamados mais de uma vez.

Após a chamada a este último método, temos a Activity sendo executada e exibindo dados para o usuário. Agora, o usuário clica em um botão e você exibe uma nova Activity mas que não cobre totalmente a anterior. Veja como exemplo a imagem a seguir, do aplicativo do Twitter.



O método **onPause()** é chamado. O usuário interage com esta nova Activity e fecha ela. Então, o método **onRestart()** é chamado e voltamos para os métodos **onStart()** e **onResume()**.

E se, ao invés de abrir uma Activity parcial, abríssemos uma que encobre totalmente a anterior? Agora, o método **onPause()** continuaria sendo chamado, mas também seria chamado o **onStop()**. Neste caso, a Activity pode ser retirada da memória a qualquer momento, basta o Android precisar de memória livre. Caso isto não aconteça, volta para o **onRestart()**, senão, temos que recomeçar todo o ciclo pelo **onCreate()**.

Por fim, temos o método **onDestroy()**, chamado antes de a Activity ser retirada da memória. Isto pode acontecer por dois motivos. O primeiro, você chama o método **finish()** da Activity.

Qual a importância de entender este ciclo de vida? Suponha que você tenha três campos na tela e não quer depender que o usuário clique em um botão SALVAR para guardar estes dados. Isto é bastante comum em telas de Configurações do aplicativo. Como você faria isso?

A cada vez que o usuário digita algo, grava? Não parece custoso demais? E se você fizer isto no **onPause()**? Melhor, não acha?

Toda vez que sua Activity for encoberta, você aproveita pra gravar os dados digitados. Existem muitos outros casos e quando estiver programando, vai começar a compreender melhor e usar também cada vez melhor o ciclo de vida.

Para fechar este tópico, vamos entender o que é esse parâmetro passado para o método **onCreate(Bundle)**. Suponha que você queira salvar algum estado da sua Activity antes de ela ser removida da memória. Você quer salvar esse estado para que da próxima vez que esta Activity for colocada visível, possa recuperar esse estado. Como fazer isso?

Primeiro, você precisa implementar o método **onSaveInstanceState(Bundle)** que recebe como parâmetro um objeto do tipo **Bundle**. Este método será chamado antes de sua Activity ser removida da memória. É garantido!

O **Bundle** é um objeto que se parece muito com um **HashMap**. Nele, você guarda as informações que quiser. Depois, na próxima vez que esta Activity for colocada visível, o método **onCreate(Bundle)** receberá uma instância de **Bundle** contendo estas informações.

Atenção. **onSaveInstanceState()** não é um método do ciclo de vida. Ele pode parecer com o método **onPause()** ou **onStop()**, mas é diferente.

Outro método que você pode sobrescrever para tratar a recuperação dos dados da Activity é o **onRestoreInstanceState(Bundle)**. Ele será chamado antes da chamada a **onStop()**. Contudo, tenha cuidado com uma coisa: este método possui uma implementação padrão que já trata da recuperação do estado da View quando uma Activity encobre parcialmente a outra.

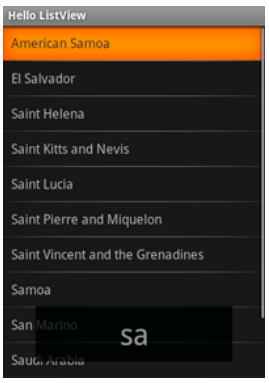
Observe que quando isto acontece, você não precisa tratar de recuperar o que o usuário digitou nos campos, certo? Isto acontece porque este método já tem a implementação que faz isto. Ou seja, se você sobrescreve e esquece de fazer uma chamada **super.onRestoreInstanceState(Bundle)**, terá que tratar você mesmo a recuperação do estado de TODAS as Views desta Activity.

OUTROS TIPOS DE ACTIVITIES

Existem outras classes de Activity? Sim, existem. Uma para cada propósito específico. Elas trazem facilitadores para cada situação. Vamos analisar algumas delas.

android.app.ListActivity
Já possui um ListView associado a ela e que pode ser usado para exibir listas de dados para o usuário. Ela não define a posição onde ficará a lista, mas apenas que existe uma lista associada.

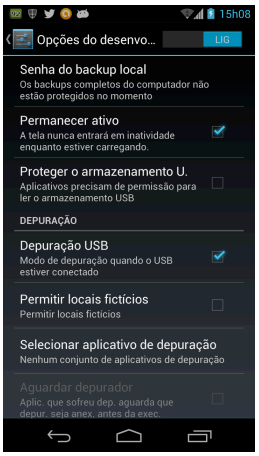
Você pode usar um layout padrão, já fornecido no SDK ou fazer o seu próprio. O layout padrão encontra-se em **android.R.layout.activity_list_item**. Caso você forneça seu próprio layout, a ListView que você criar deve ter o atributo **android:id** como **@android:id/list**.



android.support.v4.app.FragmentActivity
Usada somente em Activities que possuem fragmentos dentro dela. Não sabe ainda o que são fragmentos? Teremos um Card específico para isto.

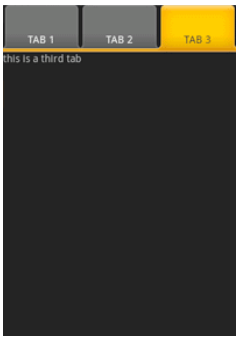
Fique atento, pois esta classe não existe, por padrão, nas versões anteriores à 3.0 do Android. Caso você queira usar ela nestas versões mais antigas, deve adicionar a biblioteca de suporte.

android.preference.PreferenceActivity
Em versões anteriores à 3.0, era uma classe que deveria ser estendida por Activities responsáveis em exibir as telas de preferências do aplicativo. O Android provê facilitadores para criar telas deste tipo e que abordaremos no card sobre Preferências do Usuário.

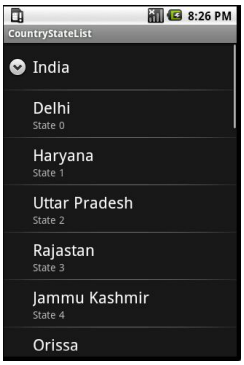


Contudo, a partir das novas versões do Android, sugere-se o uso de Fragments no lugar, usando a classe **PreferenceFragment**.

android.preference.TabActivity
Tipo de Activity que separa a tela em abas, conhecidas como Tabs. Contudo, esta classe já está depreciada e sugere-se que passe a usar as novas features, como a nova **ActionBar**.



android.preference.ExpandableListActivity
Aqui as coisas funcionam de forma muito semelhante à ListActivity, só que você vai ter um ExpandableListView no lugar da ListView. A ExpandableListView permite você exibir dados de forma mais agrupada, com uma hierarquia.



Acabou?
Existem outras classes que estendem de Activity, contudo, não vamos abordar todas elas aqui. São classes mais específicas para situações que na grande maioria dos casos, você não necessitará.

MUDANÇA DE CONFIGURAÇÕES

O que acontece caso o usuário force alguma mudança de configuração no smartphone dele? Por exemplo, caso haja uma mudança na orientação da tela. Ou mudança de Locale. Isto significa que toda a Activity será recarregada. Tudo recomeça no ciclo de vida, chamando novamente o método **onCreate(Bundle)** e daí em diante.

Fique atento a isto, para evitar possíveis erros. Por exemplo, você pode lançar uma thread para obter um conteúdo na internet. O usuário muda a orientação da tela e a thread perde a referência para a Activity e os objetos desta Activity. Provavelmente, você verá um erro.

LANÇANDO ACTIVITIES

E como eu inicio uma nova Activity? Bom, vamos adiantar brevemente um assunto, que é o caso das **Intents**. Para lançar uma nova Activity você deve usar o trecho de código a seguir.

```
public class MainActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Intent intent = new Intent(this, MainActivity2.class);
        startActivity(intent);

    }

}
```

Observe o trecho de código onde temos a criação da Intent, informando, no segundo parâmetro, qual a nova Activity que queremos exibir. Depois, basta passar esta Intent como parâmetro para o método **startActivity()**.

Sobre o Autor

Marlon Silva Carvalho

É um programador de longa data. Já peregrinou por diversas linguagens e hoje se considera um programador agnóstico. Atualmente, está fascinado pelo mundo mobile e suas consequências para a humanidade. Está mais focado no desenvolvimento para Android, embora também goste de criar aplicativos para iOS.

Trabalha em projetos sobre mobilidade no SERPRO, tendo atuado no projeto Pessoa Física, para a Receita Federal do Brasil. Você pode encontrar ele no Twitter, no perfil @marlonscarvalho, em seu blog, no endereço http://marlon.silvacarvalho.net e através do e-mail marlon.carvalho@gmail.com.

Sobre os Cards

Caso você já tenha visto os excelentes RefCards da DZone, deve ter percebido a semelhança. E é proposital. A ideia surgiu após acompanhar estes RefCards por um bom tempo. Contudo, eles são mais gerais. O objetivo destes Cards é tratar assuntos mais específicos. Trazendo informações relevantes sobre eles.

Caso tenha gostado, continue nos acompanhando! Tentaremos trazer sempre novos cards sobre os mais variados assuntos. Inicialmente, focaremos no Android.

O ícone principal deste card é de autoria de Wallec e foi obtido em seu profile no DeviantArt.
http://wwalczyszyn.deviantart.com/