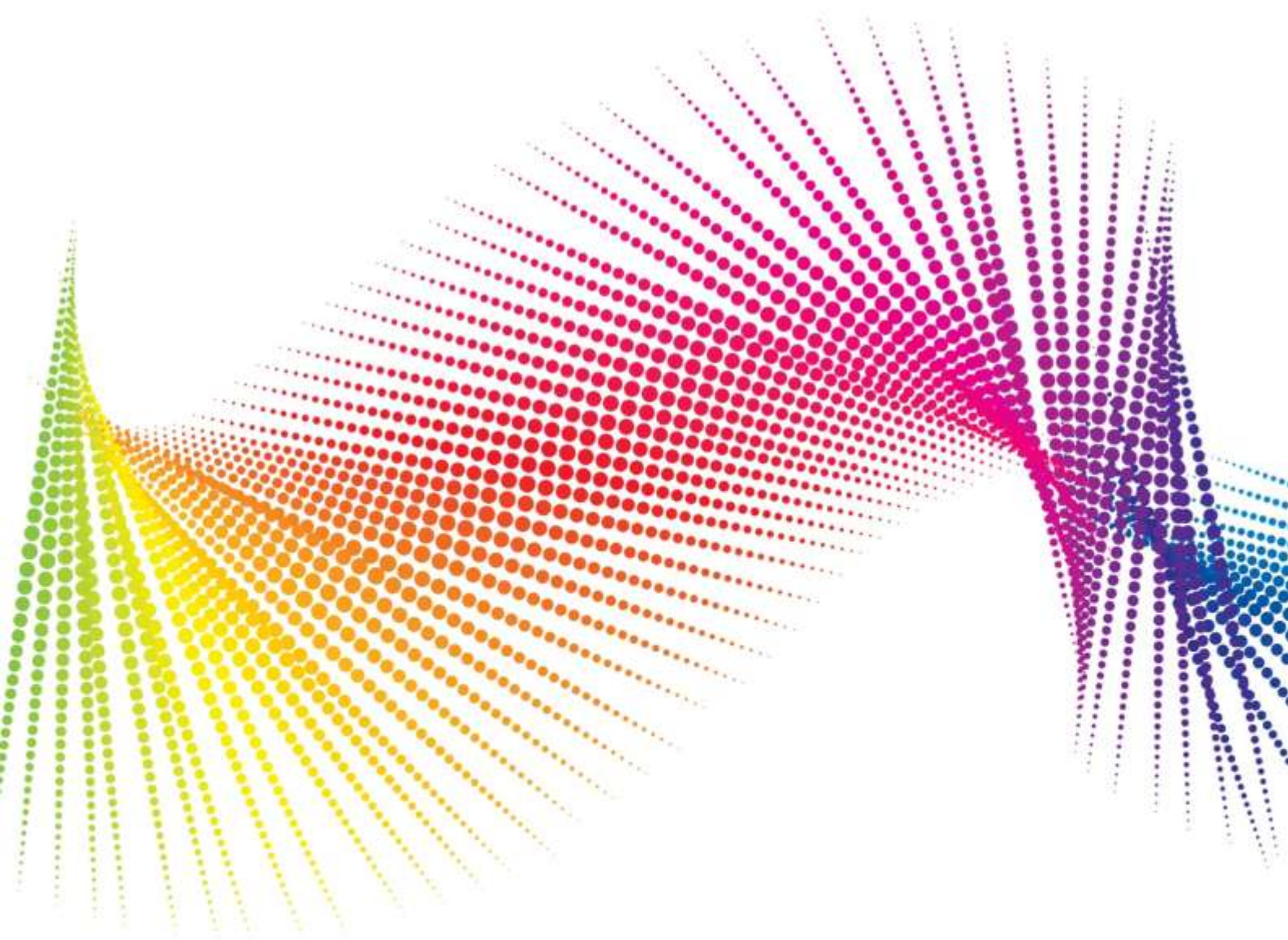


Computação Móvel

Aula 07



Este material é parte integrante da disciplina oferecida pela UNINOVE.

O acesso às atividades, conteúdos multimídia e interativo, encontros virtuais, fóruns de discussão e a comunicação com o professor devem ser feitos diretamente no ambiente virtual de aprendizagem UNINOVE.

Uso consciente do papel.

Cause boa impressão, imprima menos.

Aula 07: Componentes do Android: atividades, serviços, repositórios de conteúdo, mensagens e arquivo de manifesto

Objetivo: Apresentar os componentes principais do desenvolvimento de um aplicativo na plataforma Android, demonstrando suas funcionalidades, bem como os efeitos produzidos pelas interações elaboradas durante a criação do projeto e de um conjunto de processos associados às diretrizes de funcionamento que compõem as configurações de manuseio em uma aplicação Android.

Activity

É a classe principal utilizada no gerenciamento da interface com o usuário. Os aplicativos feitos para o sistema operacional Android começam, normalmente, mostrando uma activity, quando esta aplicação é executada, ou seja, a activity principal é visualizada na tela, se este for o caso.

Uma das coisas mais importante que se deve conhecer sobre uma activity é o ciclo de vida, em que está contido o tempo em que ela fica ativa na execução da aplicação.

Deste modo, sempre que uma aplicação é executada, uma activity pode ser utilizada como padrão e, se este for o caso, esta será executada, o que não impede que outras activitys sejam executadas também (o que será abordado posteriormente).

Mas, para que se possa imaginar, caso se deseje iniciar uma nova activity sobre uma existe, basta utilizar as funções `startActivity()` e `startActivityForResult()`.

Para melhor entendimento, verifique o Exemplo 1.

```
static final int PICK_CONTACT_REQUEST = 0;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    startActivityForResult(
        new Intent(Intent.ACTION_CONTACT_REQUEST,
            new Uri("content://contacts")),
        CONTACT_REQUEST);
}

protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if (requestCode == CONTACT_REQUEST) {
        if (resultCode == RESULT_OK) {
            // fazer alguma coisa...
        }
    }
}
```

Serviços

Sempre que desejarmos executar uma operação que necessite de duas propriedades específicas, ser executada em segundo plano e um período de duração prolongado, devemos utilizar um serviço, pois este componente foi elaborado para isso e, portanto, não possui interface com o usuário. Para inicializarmos sua execução, isso pode ocorrer em conjunto com determinado componente da aplicação e, se este for o caso, o serviço ficará ligado a este componente para que mesmo quando o usuário inicie outra aplicação, o serviço continue sua execução até que a tarefa seja concluída.

Além disso, um componente ligado a um serviço para interagir com ele pode, inclusive, realizar processos de comunicação entre processos. Por exemplo: um serviço pode querer manusear transações de rede, tocar música, fazer uma troca de arquivo ou interagir com um provedor de conteúdo, tudo em modo background.

Um serviço pode se inicializado de duas formas:

Started

Um serviço é considerado dessa forma quando um componente da aplicação dá início a sua execução, chamando o método `startService()`. Deste modo, o serviço será executado em modo background por tempo indeterminado, mesmo que o componente que deu início a sua execução seja encerrado. Normalmente, um serviço realiza uma única tarefa e não retorna nenhum resultado para quem o chamou. Pode-se fazer download ou upload de arquivos em uma rede. Quando a operação termina, o serviço é encerrado, sem necessidade de interação com o usuário.

Bound

Um serviço é considerado dessa forma quando um componente de aplicação está ligado a ele por meio da chamada do método `bindService()`. Ao contrário de um serviço started, um bound possui uma interface cliente-servidor que permite ao componente realizar interação com o serviço em execução, enviando requisições, obtendo resultados ou fazendo processos de comunicação entre processos, se necessário. Um serviço bound executa somente enquanto o componente que o iniciou estiver ligado. Múltiplos componentes podem ser ligados a um mesmo serviço num mesmo instante, mas o serviço será interrompido somente quando todos componentes forem desligados.

Como um serviço é executado na thread principal do processo que aloja a aplicação, ou seja, não cria sua própria thread e não é executado em um processo exclusivo, se for constatado que ele está usando muita CPU ou bloqueando operações, deve-se criar uma nova thread dentro dele para que seja executado.

Arquivo de manifesto

O arquivo de manifestos é o local da aplicação Android em que serão informados todos os recursos presentes na aplicação, bem como os limites a serem aplicados durante sua execução.

Por exemplo, para uma atividade, devem-se declarar todos os serviços que ela possui no arquivo de manifesto da aplicação, sendo que, para isso, adiciona-se um elemento `<service>` subordinado a um elemento `<application>`, como segue:

```
< manifest ... >
...
<application ... >
  <service android:name=".ExampleService" />
...
</application>
< /manifest>
```

Veja a seguir os elementos válidos no arquivo de manifesto:

`<action>`, `<activity>`, `<activity-alias>`, `<application>`, `<category>`, `<data>`, `<grant-uri-permission>`, `<instrumentation>`, `<intent-filter>`, `<manifest>`, `<meta-data>`, `<permission>`, `<permission-group>`, `<permission-tree>`, `<provider>`, `<receiver>`, `<service>`, `<supports-screens>`, `<uses-configuration>`, `<uses-feature>`, `<uses-library>`, `<uses-permission>`, `<uses-sdk>`.

Recebendo mensagens

Para que se possa entender melhor o recebimento de mensagens no Android, precisamos primeiro entender o funcionamento dos processos de `BroadcastReceiver`.

Um `BroadcastReceiver` é a extensão da classe `content.BroadcastReceiver`, responsável pelo gerenciamento dos eventos gerados por um `Intent`, sem que seja necessária a utilização de uma interface gráfica. Os eventos gerados correspondem a mensagens enviadas pelo sistema Android que podem ser avisos, como o que diz que a bateria está baixa ou o comunicado de que uma mensagem sms foi recebida.

Para melhor entendimento do recebimento de mensagens sms, veremos sua utilização no envio uma mensagem sms, como segue:

A primeira coisa que devemos fazer é autorizar o recebimento de mensagens sms, o que deve ser informado no arquivo AndroidManifest.xml e, posteriormente, configurar o recebimento de um BroadcastReceiver, para melhor entendimento, veja o Exemplo 2.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="br.com.EnviaSMS"
android:versionCode="1"
android:versionName="1.0">
<application android:icon="@drawable/icon"
                android:label="@string/app_name">
<activity android:name=".EnviaSMS"
android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<receiver android:name=".ReceberSMS" android:enabled="true">
<intent-filter>
<action
                android:name="android.provider.Telephony.SMS_RECEIVED" >
</action>
</intent-filter>
</receiver>
</application>
<uses-sdk android:minSdkVersion="7" />
<uses-permission
                android:name="android.permission.READ_SMS">
</uses-permission>
<uses-permission
                android:name="android.permission.SEND_SMS">
</uses-permission>
```

```
<uses-permission  
    android:name="android.permission.WRITE_SMS">  
</uses-permission>  
</manifest>
```

Observe a mensagem gerada pelo receiver informando qual classe receberá o evento e qual tipo de filtro de intent que, neste caso, é uma mensagem sms. Agora, podemos inserir uma classe e herdar da classe BroadcastReceiver. Para melhor entendimento, veja o Exemplo 3.

```
import android.app.PendingIntent;  
import android.content.BroadcastReceiver;  
import android.content.Context;  
import android.content.Intent;  
import android.os.Bundle;  
import android.telephony.SmsManager;  
import android.telephony.SmsMessage;  
import android.widget.Toast;  
public class ReceberSMS extends BroadcastReceiver{  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        // TODO Auto-generated method stub  
        Bundle bundle = intent.getExtras();  
        Object messages[] = (Object[]) bundle.get("pdus");  
        SmsMessage smsMessage[] = new  
            SmsMessage[messages.length];  
        for(int n=0 ; n < messages.length; n++)  
        {  
            smsMessage[n] =  
                SmsMessage.createFromPdu((byte[])messages[n]);  
            String celular =  
                smsMessage[n].getDisplayOriginatingAddress();  
            String Mensagem =  
                smsMessage[n].getDisplayMessageBody();
```



```
        Toast.makeText(context,"Celular: " +
        smsMessage[n].getDisplayOriginatingAddress() +
        "Mensagem: " +
        smsMessage[n].getDisplayMessageBody() ,
        Toast.LENGTH_LONG).show();
    }
}
}
```

Provedores de conteúdo

Os provedores de conteúdo têm o objetivo de gerenciar o acesso aos dados. Eles encapsulam os dados e proporcionam mecanismos para a definição da segurança destes. Os provedores de conteúdo são a interface-padrão que conecta dados com código em execução.

Para acessar dados em um provedor de conteúdo, usa-se o objeto `ContentResolver`, que se comunica com o objeto do provedor, uma instância de uma classe que implementa `ContentProvider`. O objeto do provedor recebe solicitações de dados de clientes, realiza a ação solicitada e retorna os resultados.

O Android, em si, inclui provedores de conteúdo que gerenciam dados, como áudio, vídeo, imagens e informações de contato pessoal.

- **Provedor básico de conteúdo:** um provedor de conteúdo gerencia o acesso a um repositório central de dados. Um provedor é parte de um aplicativo Android, o que muitas vezes fornece a sua própria interface de usuário para trabalhar com os dados. No entanto, os provedores de conteúdo são, principalmente, destinados a serem utilizados por outras aplicações, que acessam o provedor usando os objetos cliente e provedor.
- **Acessando um provedor:** um aplicativo acessa os dados de um provedor de conteúdo com um objeto `ContentResolver`. Este objeto tem métodos que fornecem a base CRUD (criar, recuperar, atualizar e excluir) e funções de armazenamento persistente.

- **Provedor de permissões de conteúdo:** a aplicação de um provedor pode especificar permissões que outros aplicativos devem ter para acessar os dados do provedor. Estas permissões garantem que uma aplicação saiba quais dados ela pode acessar.
Para obter as permissões necessárias para acessar um provedor, um aplicativo deve configurar o elemento <uses-permission> em seu arquivo de manifesto.
- **Inserção de dados:** para inserir dados em um provedor, chama-se o método ContentResolver.insert(), que insere uma nova linha para o provedor e retorna um URI conteúdo para essa linha. Para melhor entendimento, veja o Exemplo 4.

```
// Define um objeto Uri novo que recebe o resultado da inserção
Uri mNewUri ;

...

// Define um objeto para conter os novos valores para inserir
contentValues mNewValues = new ContentValues ();

/*
 * Define os valores de cada coluna e insere o texto. Os argumentos para a "put" são
 "nome da coluna" e "valor"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");

mNewUri = getContentResolver . () inserir (
    UserDictionary . Word. CONTENT_URI , // o conteúdo do dicionário URI
    mNewValues);                // do usuário com os valores para inserir
```

- **Atualizando dados:** para atualizar uma linha, são necessários um objeto ContentValues com os valores atualizados e os critérios de seleção. O método é o ContentResolver.update(). Só é preciso adicionar valores ao objeto ContentValues para colunas e elas serão atualizadas. Para melhor visualização, veja o Exemplo 5.

```
// Define um objeto para conter os valores atualizados
ContentValues mUpdateValues = novo ContentValues ();

// Define critérios de seleção para as linhas que você deseja atualizar
String mSelectionClause = UserDictionary.words.LOCALE + "como?" ;
String[] mSelectionArgs = { "% pt_BR" };

// Define uma variável para conter o número de linhas atualizadas
int mRowsUpdated = 0 ;

...

/*
 * Define o valor atualizado e atualiza as palavras selecionadas.
 */
mUpdateValues . putNull ( UserDictionary . words . LOCALE );

mRowsUpdated = getContentResolver(). update(
    UserDictionary . words . CONTENT_URI , // dicionário do usuário
    mUpdateValues // as colunas para atualizar
    mSelectionClause // coluna para selecionar em
    mSelectionArgs // o valor para comparar
);
```

- **Exclusão de dados:** para excluir dados, devem-se especificar critérios de seleção para as linhas que se deseja excluir e do método de cliente retorna o número de linhas excluídas. Para melhor entendimento, veja o Exemplo 6.

```
// Define critérios de seleção para as linhas que você deseja excluir
String mSelectionClause = UserDictionary . words. app_id + "como?" ;
String [] mSelectionArgs = { "user" };

// Define uma variável para conter o número de linhas excluídas
int mRowsDeleted = 0 ;

...

// Exclui as palavras que correspondem aos critérios de selecção
mRowsDeleted = getContentResolver (). delete(
    UserDictionary . words. CONTENT_URI ,
        // o conteúdo do dicionário do usuário
    mSelectionClause          // coluna para selecionar
    mSelectionArgs );         // o valor para comparar
```



Resumo

Nesta aula, apresentamos as atividades que constituem a principal estrutura da aplicação Android: serviços que permitem a execução de processos em background, repositórios de conteúdo onde se encontram os dados que uma aplicação tem acesso, mensagens para os usuários e as configurações possíveis para o arquivo de manifesto.

Próxima aula


Na próxima aula, detalharemos os recursos associados à utilização de imagens e strings no processo de construção de aplicações Android, bem como os principais recursos desses elementos.

Para termos certeza de que entendemos a aula, vamos realizar uma atividade sobre os componentes do Android?



EXERCÍCIOS

Agora, veja os exercícios disponíveis acessando o AVA, ou via QR Code*. Não deixe de visualizar esses exercícios, pois eles fazem parte da sequência desta aula e, portanto, são essenciais para a aprendizagem.





* O QR Code é um código de barras que armazena links às páginas da web. Utilize o leitor de QR Code de sua preferência para acessar esses links de um celular, tablet ou outro dispositivo com o plugin Flash instalado.

REFERÊNCIAS

LECHETA, Ricard R. *Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK*. 2. ed. São Paulo: Editora Novatec, 2010.

MEIKE, Blake et al. *Desenvolvimento de aplicações Android*. São Paulo: Editora Novatec, 2009.