



Erisvaldo Gadelha Saraiva Júnior

Faculdade de Tecnologia de João Pessoa (FATEC-JP)  
Especialização em Desenvolvimento para Dispositivos Móveis  
Disciplina: Tecnologias para Dispositivos Móveis

# Android (Parte 2)



E-mail: [erisvaldojunior@gmail.com](mailto:erisvaldojunior@gmail.com)

Site: <http://erisvaldojunior.com>

Twitter: [@erisvaldojunior](https://twitter.com/erisvaldojunior)



# Roteiro da Aula



Desenvolvendo para a plataforma Android



# PRIMEIROS PASSOS

Preparando o Ambiente de Desenvolvimento e Primeiros Passos em Android

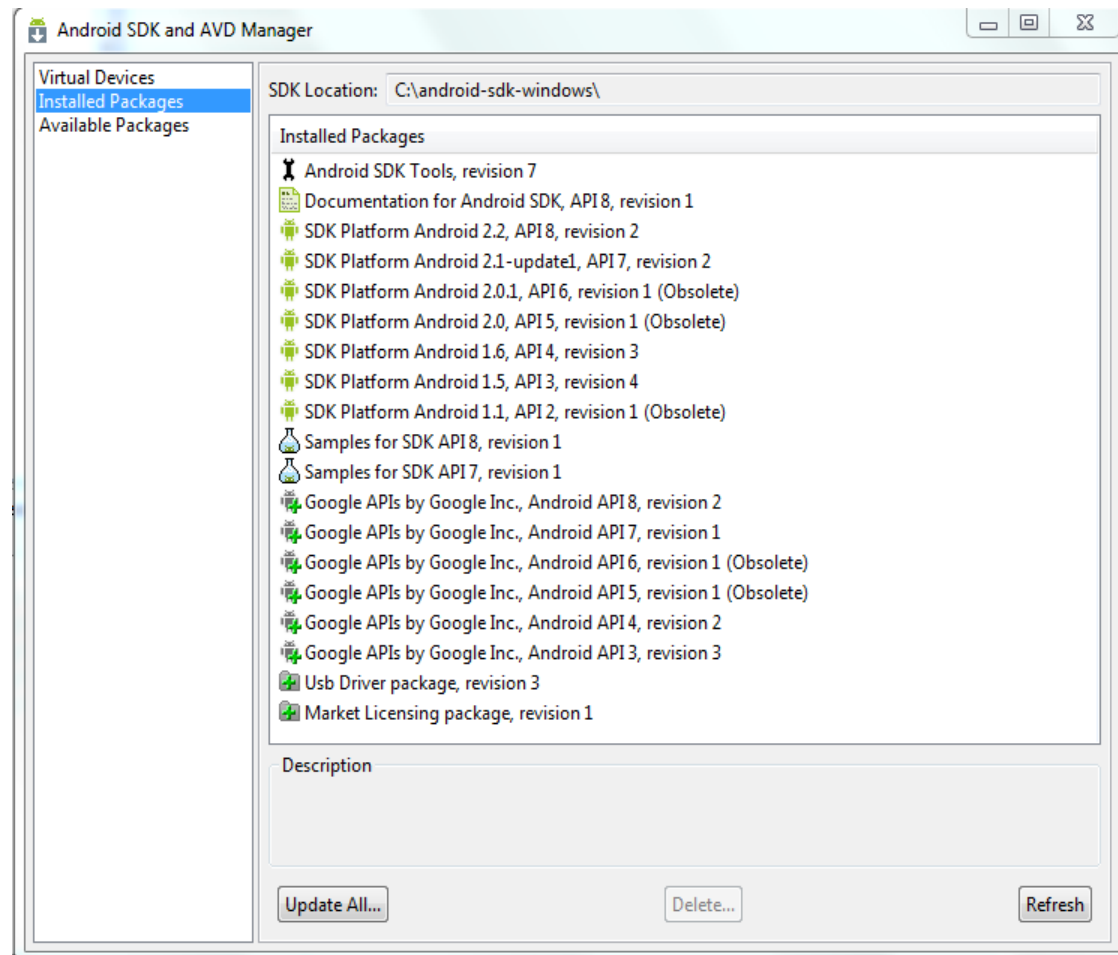
## O que é necessário?

1. Última versão do Java Development Kit (JDK) instalado
2. Efetuar o download de:
  - Android SDK (<http://developer.android.com/sdk>)
  - Eclipse IDE (<http://www.eclipse.org/downloads>)
3. Instalação do plugin Android Development Tools (ADT) através do Gerenciador de Plugins do Eclipse:
  - ADT Plugin para Eclipse  
(<https://dl-ssl.google.com/android/eclipse>)

## Adicionando plataformas Android no Eclipse

Menu **Window** ->  
**Android SDK and  
AVD Manager** ->  
**Installed Packages**  
-> **Update All.**

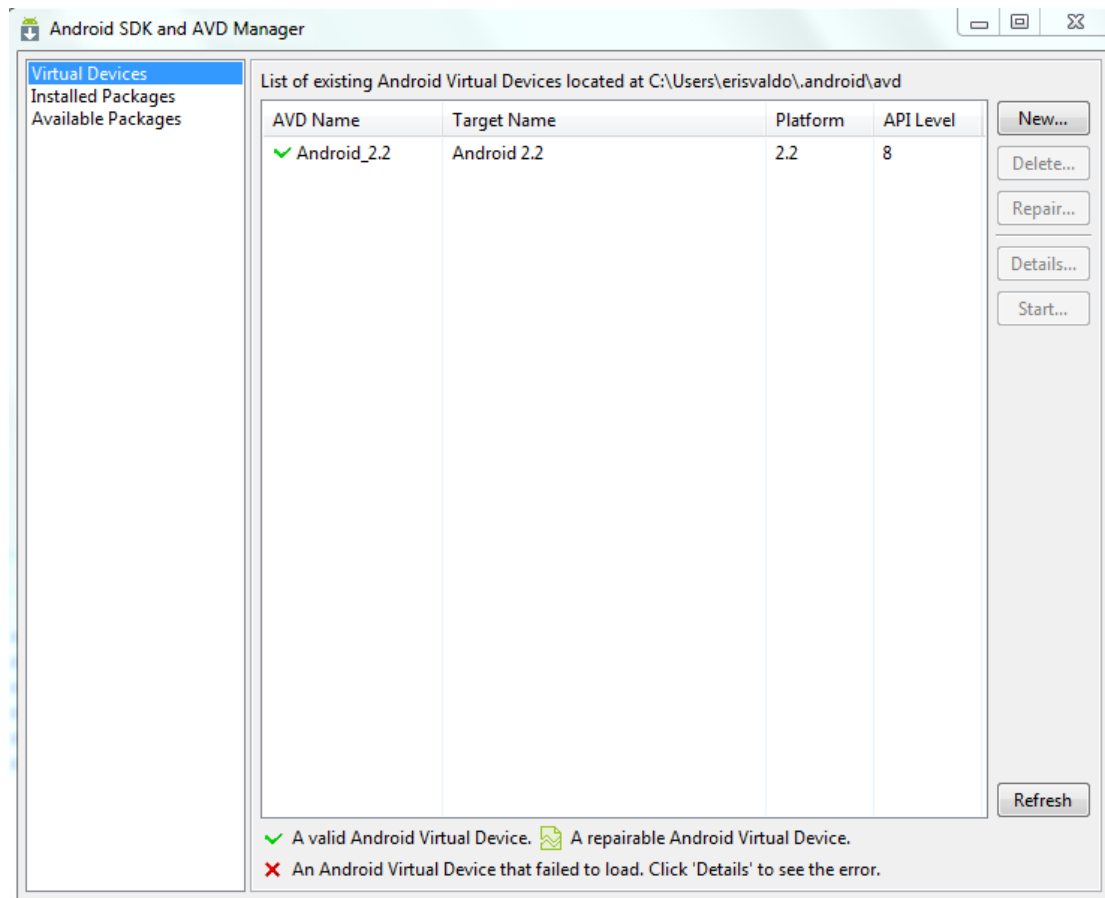
Selecionar  
plataformas e  
componentes  
desejados e  
efetuar o  
download.



## Adicionando AVDs (Android Virtual Devices)

Uma vez instalado o ADT Plugin e as plataformas desejadas, é hora de adicionar pelo menos um AVD para que você possa testar suas aplicações.

Acessa-se o menu Window do Eclipse e, em seguida, **Android SDK and AVD Manager**. Seleciona-se *Virtual Devices* e, finalmente, clica-se em *New*. Depois basta configurar o dispositivo virtual conforme se deseja e ele estará disponível para ser usado.



## Estrutura de um projeto Android

Quando se cria um projeto com o ADT plugin, obtém-se a seguinte estrutura de pastas:

- / - Raiz do projeto.
  - src/ - Classes Java
  - gen/ - Código Java gerado automaticamente.
  - res/ - Recursos da aplicação
    - drawable/ - Imagens
    - layout/ - Layouts de telas/formulários.
    - values/ - Arquivos de variáveis.
  - AndroidManifest.xml - Configuração do projeto.



## AndroidManifest.xml

Arquivo de configuração de uma aplicação Android. Identifica o nome e o ícone da aplicação, declara os componentes, realiza a conexão com bibliotecas extras que a aplicação necessita (além da biblioteca padrão do Android), define a versão mínima do Android na qual a aplicação pode ser executada e identifica quaisquer permissões que a aplicação espera obter.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
            android:icon="@drawable/small_pic.png"
            android:label="@string/freneticLabel"
            . . . >
        </activity>
        . . .
    </application>
</manifest>
```



## Layout – main.xml

A forma mais comum de esboçar uma tela é através de um arquivo de layout no formato XML.

Esse arquivo determina o tipo de layout utilizado e declara os elementos de interface que compõem a tela, correspondentes às classes *View* e suas subclasses.

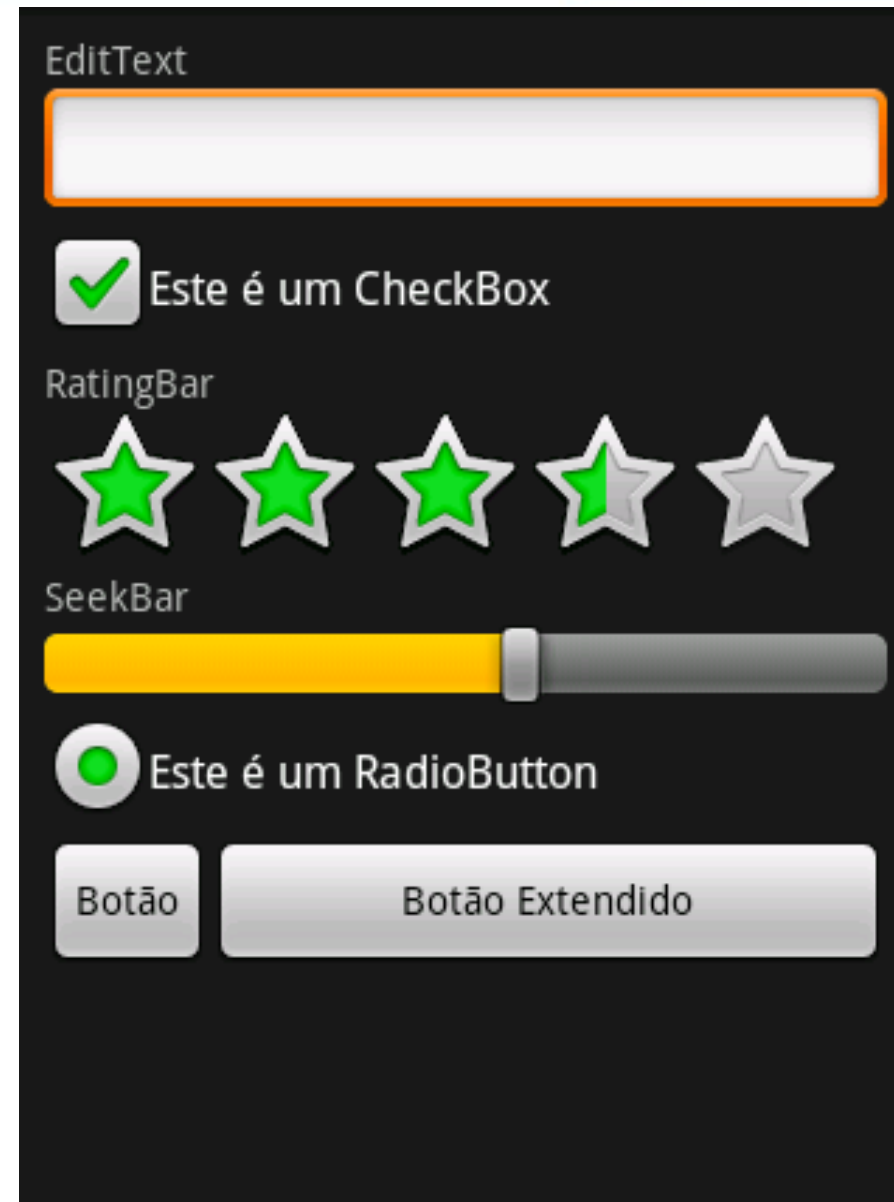
Além disso, a aplicação pode criar objetos *View* e *ViewGroup*, bem como manipular suas propriedades, em tempo de execução, através de código na Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
            android:icon="@drawable/small_pic.png"
            android:label="@string/freneticLabel"
            . . . >
        </activity>
        . . .
    </application>
</manifest>
```

# EXERCÍCIOS DE LAYOUT

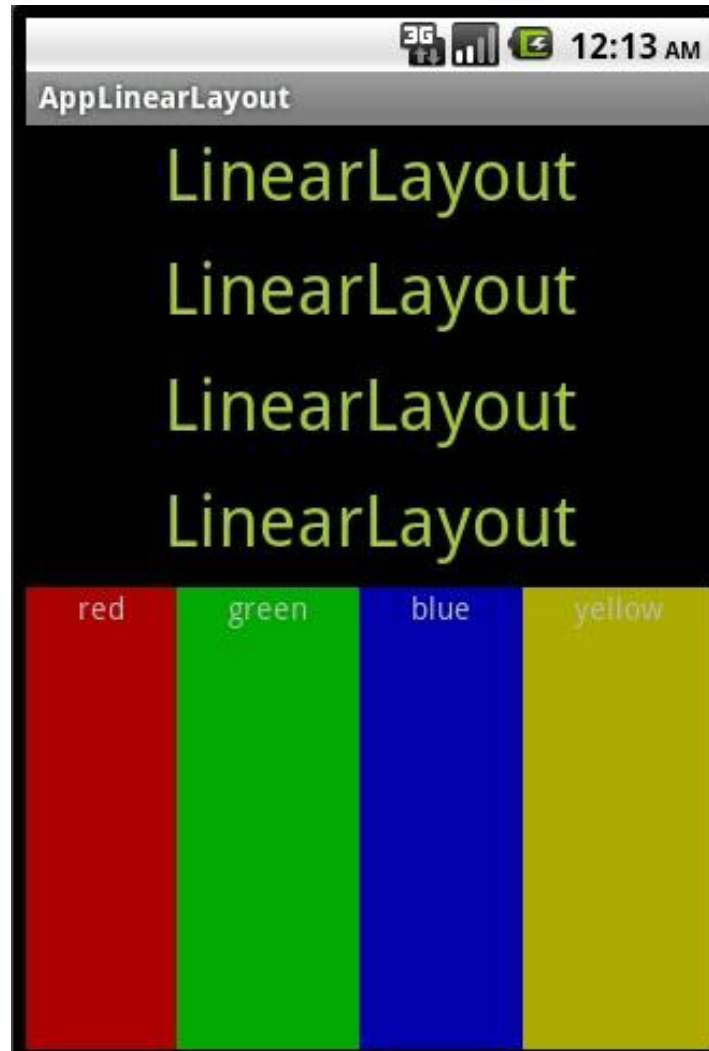
## Exercício 1.1

Alterar o layout **main.xml** para obter uma tela similar a esta imagem.



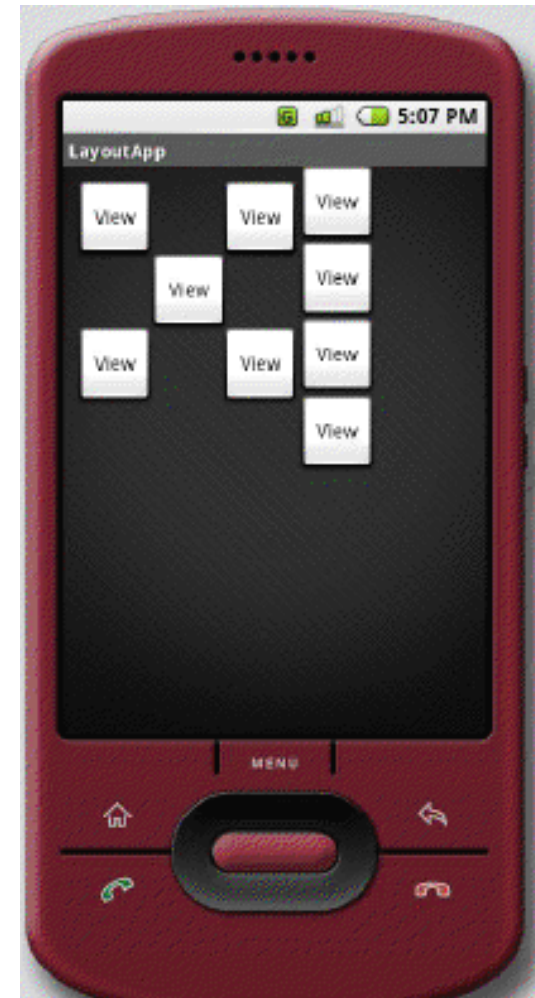
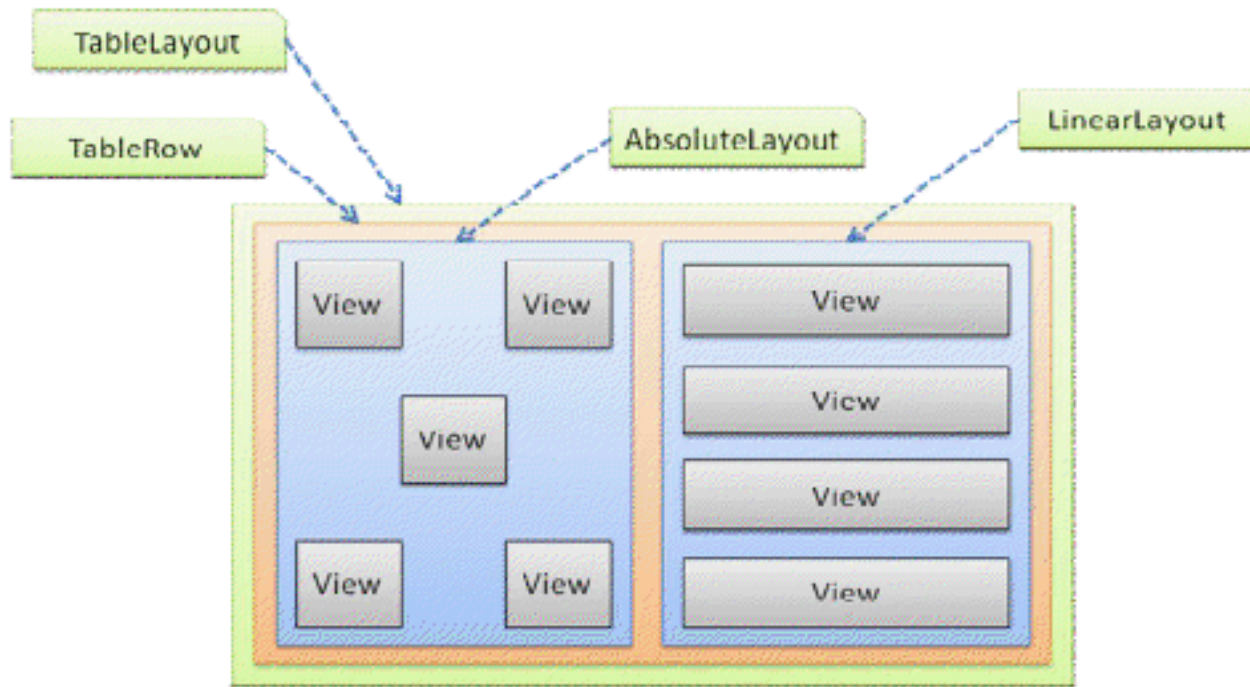
# EXERCÍCIOS DE LAYOUT

**Exercício 1.2:** criar um layout `linearlayout.xml` de forma a confeccionar a tela abaixo.



# EXERCÍCIOS DE LAYOUT

**Exercício 1.3:** criar o layout abaixo.



## Exercício 2.1

Criar a aplicação ao lado. Ao pressionar o botão, exibir um **AlertDialog** informando Álcool ou Gasolina.

Gasolina ou Alcool?  
Preço do Alcool

Preço da Gasolina

Calcular o melhor

## Exercício 2.2

Criar uma aplicação com duas Activities e uma Dialog. Uma Activity (FormActivity) é chamada através do clique no primeiro botão e a Dialog através do clique no segundo botão.

Minha primeira aplicação!

Mostrar Formulário

Mostrar Caixa de Diálogo

## Exercício 2.2 (FormActivity)

The image shows a screenshot of an Android application interface titled "Exercício 2.2 (FormActivity)". The interface is dark-themed and contains the following elements:

- A label "Nome:" followed by a text input field.
- A label "Telefone:" followed by a text input field.
- A label "E-Mail:" followed by a text input field.
- A label "Data de Nascimento:" followed by three date pickers. Each date picker consists of a top button with a "+" sign, a middle button, and a bottom button with a "-" sign.
- A "Salvar" (Save) button located at the bottom right of the form.

Formulário com campos que devem ser mapeados para a Activity.





# Persistência e Comunicação

Persistência e Comunicação com a Web em Android

## Armazenamento de Dados

O Android provê diversas opções para armazenar dados da aplicação. A escolha da melhor opção deve ser feita de acordo com a necessidade: dados privados ou públicos, quantidade de espaço necessário, etc.

- **Shared Preferences** – Armazena dados primitivos em um conjunto de pares do tipo chave – valor;
- **Internal Storage** – Armazena dados *privados* na memória do dispositivo;
- **External Storage** – Armazena dados *públicos* na mídia externa de armazenamento;
- **SQLite Databases** – Armazena dados estruturados em um banco de dados *privado*;
- **Network Connection** – Armazena dados na web com o seu próprio servidor de rede.

## Shared Preferences

A classe `SharedPreferences` provê um framework geral que permite salvar e recuperar pares de dados primitivos do tipo chave-valor. Os dados ficam armazenados mesmo quando a aplicação é finalizada.

Para obter uma instância de `SharedPreferences` para a sua aplicação, usa-se um dos métodos abaixo:

- **`getSharedPreferences()`** – use se você precisar de múltiplos arquivos de preferências identificados por um nome que é passado como parâmetro;
- **`getPreferences()`** – use se você precisar de um único arquivo de preferências para a sua Activity. Como se trata de um único arquivo, não é necessário fornecer um nome.

# Shared Preferences

```
public class Calc extends Activity {
    public static final String PREFS_NAME = "MyPrefsFile";

    @Override
    protected void onCreate(Bundle state){
        super.onCreate(state);
        . . .

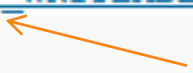
        // Restore preferences
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
        boolean silent = settings.getBoolean("silentMode", false);
        setSilent(silent);
    }

    @Override
    protected void onStop(){
        super.onStop();

        // We need an Editor object to make preference changes.
        // All objects are from android.context.Context
        SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean("silentMode", mSilentMode);

        // Commit the edits!
        editor.commit();
    }
}
```

MODE\_PRIVATE  
MODE\_WORLD\_READABLE  
MODE\_WORLD\_WRITEABLE



## Internal Storage

Para criar e escrever em um arquivo privado para a memória interna, chama-se o método **openFileOutput()** com o nome do arquivo e o tipo de operação, obtendo-se um `FileOutputStream`. Em seguida, basta escrever com o **write()** e finalizar com **close()**.

```
String FILENAME = "hello_file";  
String string = "hello world!";  
  
FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);  
fos.write(string.getBytes());  
fos.close();
```

MODE\_APPEND

Já para ler um arquivo da memória interna, chama-se o método **openFileInput()** passando o nome do arquivo a ser lido como parâmetro. Esse método retorna um `FileInputStream`. Assim, basta ler com **read()** e finalizar com **close()**.

## Internal Storage

Também é possível armazenar arquivos como *cache*, usando o método **getCacheDir()** para obter uma instância de File na qual sua aplicação possa salvar arquivos temporários de *cache*.

Quando o dispositivo está com pouca memória, o Android pode excluir os arquivos de *cache* para recuperar espaço. Contudo, é responsabilidade do desenvolvedor manter seus arquivos de *cache* e usar apenas um espaço razoável com os mesmos, como 1MB. Ao desinstalar o aplicativo, os arquivos de *cache* do mesmo são removidos.

Outros métodos interessantes:

- **getFilesDir()** – retorna o caminho absoluto dos arquivos salvos;
- **getDir()** – Cria ou abre um diretório para armazenamento interno;
- **deleteFile()** – Exclui um arquivo;
- **fileList()** – Lista de arquivos atualmente salvos por sua aplicação.

## External Storage

Todo dispositivo Android suporta uma mídia externa de armazenamento. Pode ser um cartão removível (como SD Card) ou um espaço de armazenamento interno (não removível). Em ambos os casos, os arquivos podem ser lidos e modificados por outras aplicações ou pelo usuário quando se conecta o dispositivo via USB para transferência de dados.

Antes de se trabalhar com o armazenamento externo, deve-se chamar o método **getExternalStorageState()** para verificar se a mídia está disponível:

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    // to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```



## Acessando arquivos em um External Storage

No caso da API Level 8 (Android 2.2) ou superior, usa-se o método **getExternalFilesDir()** para obter um **File** que representa o diretório da mídia externa em que se deve salvar seus arquivos. Esse método recebe como parâmetro o tipo de subdiretório desejado, tais como `DIRECTORY_MUSIC` ou `DIRECTORY_RINGTONES`. Caso se deseje armazenar na raiz, passa-se **null** como parâmetro.

Já no caso da API Level 7 (Android 2.1) ou inferior, usa-se o **getExternalStorageDirectory()**. Os dados serão armazenados em `/Android/data/nome_do_pacote/files/`. O nome do pacote é no estilo Java, por exemplo: “com.example.android.app”. Caso o usuário desinstale a aplicação, esse diretório e todo o seu conteúdo será excluído.

## Salvando arquivos que devem ser compartilhados

Caso se queira salvar arquivos que não são específicos da aplicação e devem permanecer mesmo após a aplicação ser desinstalada, basta salvá-los em diretórios de armazenamento públicos, tais como *Music/*, *Pictures/*, *Ringtones/* e outros.

Na API Level 8 ou superior, usa-se **getExternalStoragePublicDirectory()** passando como parâmetro o tipo de diretório público que se deseja, tais como: `DIRECTORY_MUSIC`, `DIRECTORY_PICTURES`, etc.

Já no caso da API Level 7 ou inferior, usa-se o **getExternalStorageDirectory()**. Em seguida, salva-se os arquivos compartilhados em algum desses diretórios: *Music/*, *Podcasts/*, *Ringtones/*, *Alarms/*, *Notifications/*, *Pictures/*, *Movies/* e *Download/*.

## Salvando arquivos externos de cache

Na API Level 8 ou superior, usa-se **getExternalCacheDir()**.

Já no caso da API Level 7 ou inferior, usa-se o **getExternalStorageDirectory()**. Em seguida, salva-se os dados de *cache* no seguinte diretório:

```
/Android/data/nome_do_pacote/cache/
```

O nome do pacote é no estilo Java. Por exemplo: “com.example.android.app”.

## Banco de Dados SQLite

Android provê suporte completo a bancos de dados **SQLite**. Os bancos criados serão acessíveis pelo nome para qualquer classe da aplicação, mas não poderão ser acessados externamente.

Para se criar um banco de dados, pode-se usar o método **openOrCreateDatabase()** do contexto. Contudo, a forma recomendada é criar uma subclasse de **SQLiteOpenHelper** e sobrescrever o método **onCreate()** para a criação das tabelas.

```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
        KEY_WORD + " TEXT, " +
        KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

# SQLiteOpenHelper

O construtor de SQLiteOpenHelper recebe quatro parâmetros, conforme especificado abaixo. A versão do banco de dados é muito útil para o caso de o banco já existir e precisar ser atualizado (através da adição de novas tabelas e/ou colunas, por exemplo).

## Parameters

<i>context</i>	to use to open or create the database
<i>name</i>	of the database file, or null for an in-memory database
<i>factory</i>	to use for creating cursor objects, or null for the default
<i>version</i>	number of the database (starting at 1); if the database is older, <a href="#"><code>onUpgrade(SQLiteDatabase, int, int)</code></a> will be used to upgrade the database

```
public abstract void onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)
```

## SQLiteOpenHelper

Pode-se, então, obter uma instância dessa subclasse através do construtor criado. Com a instância, tem-se acesso a dois métodos que retornam um SQLiteDatabase: **getWritableDatabase()**, para inserção, edição e remoção de dados, e **getReadableDatabase()**, para consulta de dados.

Pode-se executar consultas no SQLite usando o método **query()** de SQLiteDatabase, que permite a passagem de diversos parâmetros, tais como: tabela a ser consultada, projeção, seleção, colunas, agrupamento e outros. Para consultas complexas, pode-se usar uma instância de SQLiteQueryBuilder. Cada consulta retorna um objeto Cursor que aponta para todos os registros encontrados pela query. É através do objeto Cursor que se navega entre os resultados.

Por fim, usam-se os métodos **insert()** e **delete()** de SQLiteDatabase para inserção e remoção de registros, respectivamente.

## SQLiteDatabase - query()

As consultas com **query()** retornam um Cursor para navegação e recebem diversos parâmetros, cada qual com sua utilidade para a consulta, conforme explicitado abaixo.

```
public Cursor query (boolean distinct, String table, String\[\] columns, String selection, String\[\] selectionArgs, String groupBy, String having, String orderBy, String limit)
```

### Parameters

<i>distinct</i>	true if you want each row to be unique, false otherwise.
<i>table</i>	The table name to compile the query against.
<i>columns</i>	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
<i>selection</i>	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
<i>selectionArgs</i>	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
<i>groupBy</i>	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
<i>having</i>	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
<i>orderBy</i>	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
<i>limit</i>	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

### Returns

A Cursor object, which is positioned before the first entry



## Network Connection

Há, ainda, uma quinta forma de persistência em Android: o armazenamento remoto. Pode-se usar uma rede (quando disponível) para armazenar e recuperar dados de seus serviços baseados na Web.



Para realizar operações na rede, usam-se classes dos seguintes pacotes:

- `java.net;`
- `android.net.`

## Content Providers

Única maneira de compartilhar dados entre aplicações. Android vem com Content Providers nativos para áudio, vídeo, imagens, contatos, etc. Eles estão disponíveis no pacote *android.provider*.

Para tornar os dados de sua aplicação públicos, pode-se criar um Content Provider (herdando da classe `ContentProvider`) ou, ainda, adicionar os dados a um Content Provider já existente, desde que esse provedor controle o mesmo tipo de informação e você tenha permissão de escrita.

Todos os provedores de conteúdo implementam uma interface comum para adição, edição, remoção e consulta de dados. O cliente usa essa interface indiretamente, através de um `ContentResolver`. Um `ContentProvider` pode se comunicar com múltiplos objetos de `ContentResolver`, em diferentes aplicações e processos.

## Content Providers - URIs

Cada Content Provider expõe uma URI pública que identifica o conjunto de dados. Se um Content Provider controla múltiplos grupos de dados (múltiplas tabelas), há uma URI para cada um. A constante `CONTENT_URI` é padrão de todos os Content Providers nativos da plataforma, conforme mostrado abaixo.

```
android.provider.Contacts.Phones.CONTENT_URI  
android.provider.Contacts.Photos.CONTENT_URI
```

A URI é utilizada em todas as interações com o provedor de conteúdo. Todos os métodos de um Content Resolver recebem a URI do provedor de conteúdo como parâmetro. É a URI que identifica com qual provedor o ContentResolver deve se comunicar e que tabela desse provedor está sendo acessada.

`content://com.example.transportationprovider/trains/122`

The diagram illustrates the components of the URI `content://com.example.transportationprovider/trains/122` using curly braces and labels:

- A**: Points to the scheme `content`.
- B**: Points to the authority `com.example.transportationprovider`.
- C**: Points to the path `/trains`.
- D**: Points to the query parameter `/122`.

## Consultando um Content Provider

Através do método **query()** de **ContentResolver** ou **managedQuery()** de **Activity**. No caso da segunda opção, o **Activity** gerencia o ciclo de vida do **Cursor**. A solicitação é feita através do método **startManagingCursor()** de **Activity**.

O exemplo abaixo retorna o registro 23 dos contatos do usuário. Note que a URI obtida fica similar a:

`content://. . . ./23`

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI for the contact with _ID == 23.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

## Consultando um Content Provider (Query)

Os argumentos de uma **query()** ou **managedQuery()** são os seguintes:

Após o primeiro argumento (URI), existem outros quatro argumentos, que são explicitados a seguir:

- O número de colunas que deve ser retornado. **null** retorna todas as colunas. Os Content Providers definem constantes para as suas colunas. Por exemplo, *android.provider.Contatcs.Phones* define as seguintes constantes: `_ID`, `NUMBER`, `NUMBER_KEY`, `NAME`, etc;
- Um filtro detalhando que registros devem ser retornados, formatado como um WHERE do SQL. **null** retorna todos os registros;
- Argumentos do WHERE (argumentos de seleção);
- Ordem em que os registros devem ser retornados, formatado como um ORDER BY do SQL. **null** retorna uma ordem padrão.

## Consultando um Content Provider (Query)

No exemplo abaixo, a consulta retorna uma lista de contatos com os seus respectivos nomes e telefones (no caso o telefone principal).

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
    People._ID,
    People._COUNT,
    People.NAME,
    People.NUMBER
};

// Get the base URI for the People table in the Contacts content provider.
Uri contacts = People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
    projection, // Which columns to return
    null,       // Which rows to return (all rows)
    null,       // Selection arguments (none)
    // Put the results in ascending order by name
    People.NAME + " ASC");
```

## Consultando um Content Provider (Cursor)

O exemplo abaixo mostra como ler os nomes e telefones da lista de contatos obtida pela Query mostrada anteriormente. Note que o objeto cursor pode conter N registros e é necessário saber o tipo de dado dos campos que estão sendo obtidos.

```
import android.provider.Contacts.People;

private void getColumnData(Cursor cur){
    if (cur.moveToFirst()) {

        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;

        do {
            // Get the field values
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);

            // Do something with the values.
            ...

        } while (cur.moveToNext());

    }
}
```



## Modificando um Content Provider (Adição de Registros)

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");
// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

## Modificando um Content Provider (Adicionando novos valores a um registro)

```
Uri phoneUri = null;
Uri emailUri = null;

// Add a phone number for Abraham Lincoln. Begin with the URI for
// the new record just returned by insert(); it ends with the _ID
// of the new record, so we don't have to add the ID ourselves.
// Then append the designation for the phone table to this URI,
// and use the resulting URI to insert the phone number.
phoneUri = Uri.withAppendedPath(uri, People.Phones.CONTENT_DIRECTORY);

values.clear();
values.put(People.Phones.TYPE, People.Phones.TYPE_MOBILE);
values.put(People.Phones.NUMBER, "1233214567");
getContentResolver().insert(phoneUri, values);

// Now add an email address in the same way.
emailUri = Uri.withAppendedPath(uri, People.ContactMethods.CONTENT_DIRECTORY);

values.clear();
// ContactMethods.KIND is used to distinguish different kinds of
// contact methods, such as email, IM, etc.
values.put(People.ContactMethods.KIND, Contacts.KIND_EMAIL);
values.put(People.ContactMethods.DATA, "test@example.com");
values.put(People.ContactMethods.TYPE, People.ContactMethods.TYPE_HOME);
getContentResolver().insert(emailUri, values);
```

## Modificando um Content Provider (Atualizando e removendo registros)

É possível atualizar registros em lote (por exemplo, modificar uma informação de todos os contatos) através do método **update()** de ContentResolver, no qual se especifica as colunas e os valores que devem ser alterados.

Pode-se também excluir um registro, invocando o método **delete()** de ContentResolver, passando como parâmetro a URI do registro específico.

Além disso, o método **delete()** também possibilita a exclusão de múltiplos registros, passando como URI o tipo de registro (por exemplo *android.provider.Contacts.People.CONTENT\_URI*) e no terceiro argumento passando o filtro (“WHERE”) que fará com que um conjunto de registros seja excluído.

## Criando um Content Provider

A maioria dos Content Providers armazenam seus dados em SQLite, mas você também pode utilizar outro mecanismo de persistência de sua preferência.

O procedimento para criação de um provedor de conteúdo é: criar uma classe que herda de `ContentProvider` para prover acesso aos dados e declará-lo no `AndroidManifest.xml`.

Métodos que precisam ser implementados:

- **onCreate()** – Preparação dos dados oferecidos;
- **getType()** – Tipo de dado (MIME) que está sendo provido;
- **query()** – retorna um `Cursor` que permite navegar pelos registros;
- **insert()** – Inserção de novos dados;
- **update()** – Atualização dos dados;
- **delete()** – Exclusão de dados.

## Criando um Content Provider

Além dos métodos que devem ser implementados, há uma série de regras que devem ser seguidas para que o ContentProvider possa ser devidamente aproveitado por outras aplicações.

```
public static final Uri CONTENT_URI =  
    Uri.parse("content://com.example.codelab.transportationprovider");
```

- **CONTENT\_URI** – definir uma constante que especifica a URI do Content Provider, bem como uma constante para cada uma de suas tabelas;
- definir uma constante para cada coluna, incluindo uma `_ID` do tipo inteiro;
- Documentar cuidadosamente cada coluna. Os clientes precisam dessa informação para ler corretamente os dados.

# Android Networking Capabilities

Package	Description
java.net	Provides networking-related classes, including stream and datagram sockets, Internet Protocol, and generic HTTP handling. This is the multipurpose networking resource. Experienced Java developers can create applications right away with this familiar package.
java.io	Though not explicitly networking, it's very important. Classes in this package are used by sockets and connections provided in other Java packages. They're also used for interacting with local files (a frequent occurrence when interacting with the network).
java.nio	Contains classes that represent buffers of specific data types. Handy for network communications between two Java language-based end points.
org.apache.*	Represents a number of packages that provide fine control and functions for HTTP communications. You might recognize Apache as the popular open source Web server.
android.net	Contains additional network access sockets beyond the core java.net.* classes. This package includes the URI class, which is used frequently in Android application development beyond traditional networking.
android.net.http	Contains classes for manipulating SSL certificates.
android.net.wifi	Contains classes for managing all aspects of WiFi (802.11 wireless Ethernet) on the Android platform. Not all devices are equipped with WiFi capability, particularly as Android makes headway in the "flip-phone" strata of cell phones from manufacturers like Motorola and LG.
android.telephony.gsm	Contains classes required for managing and sending SMS (text) messages. Over time, an additional package will likely be introduced to provide similar functions on non-GSM networks, such as CDMA, or something like android.telephony.cdma.



# org.apache.http.client.methods.HttpGet

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

public class TestHttpGet {
    public void executeHttpGet() throws Exception {
        BufferedReader in = null;
        try {
            HttpClient client = new DefaultHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI("http://w3mentor.com/"));
            HttpResponse response = client.execute(request);
            in = new BufferedReader
                (new InputStreamReader(response.getEntity().getContent()));
            StringBuffer sb = new StringBuffer("");
            String line = "";
            String NL = System.getProperty("line.separator");
            while ((line = in.readLine()) != null) {
                sb.append(line + NL);
            }
            in.close();
            String page = sb.toString();
            System.out.println(page);
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Para adicionar parâmetros na sua requisição GET, basta inserí-los na URI. Exemplo:

```
HttpGet request = new
HttpGet("http://erisvald
ojunior.com/exemploget
.php?tipo=2");
```

```
client.execute(request);
```



# org.apache.http.client.methods.HttpPost

```
public void postData() {  
    // Create a new HttpClient and Post Header  
    HttpClient httpclient = new DefaultHttpClient();  
    HttpPost httppost = new HttpPost("http://www.yoursite.com/script.php");  
  
    try {  
        // Add your data  
        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);  
        nameValuePairs.add(new BasicNameValuePair("id", "12345"));  
        nameValuePairs.add(new BasicNameValuePair("stringdata", "AndDev is Cool!"));  
        httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));  
  
        // Execute HTTP Post Request  
        HttpResponse response = httpclient.execute(httppost);  
  
    } catch (ClientProtocolException e) {  
        // TODO Auto-generated catch block  
    } catch (IOException e) {  
        // TODO Auto-generated catch block  
    }  
}
```

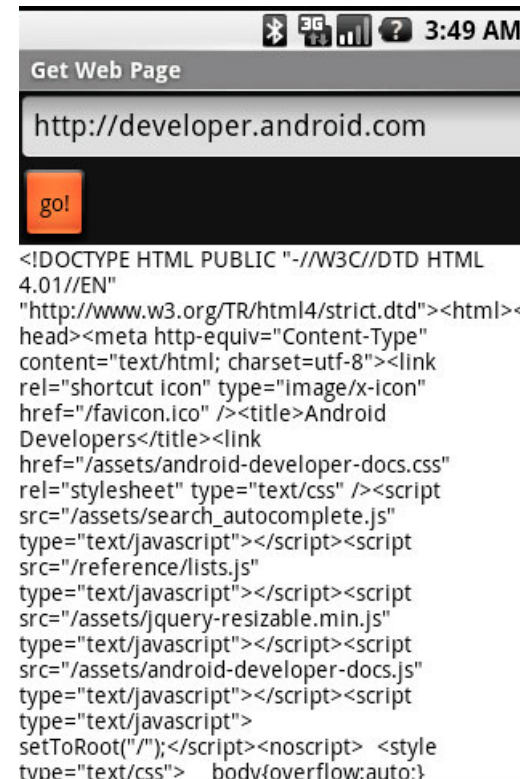
## URLConnection

```
java.net.URL url = new java.net.URL("http://developer.android.com");
```

```
java.net.URLConnection uc = url.openConnection();
```

```
BufferedReader br = new BufferedReader( new InputStreamReader (
uc.getInputStream() ) );
```

Faça uma aplicação que exiba o código-fonte de uma página qualquer cujo endereço é digitado em um EditText.



## Enviando um SMS

`android.telephony.SmsManager` (API Level 5 ou superior)

`Android.telephony.gsm.SmsManager` (API Level 4 ou inferior)

### Sending text messages

```
public void sendTextMessage (String destinationAddress, String scAddress,  
String text, PendingIntent sentIntent, PendingIntent deliveryIntent)
```

### Sending multi-part messages

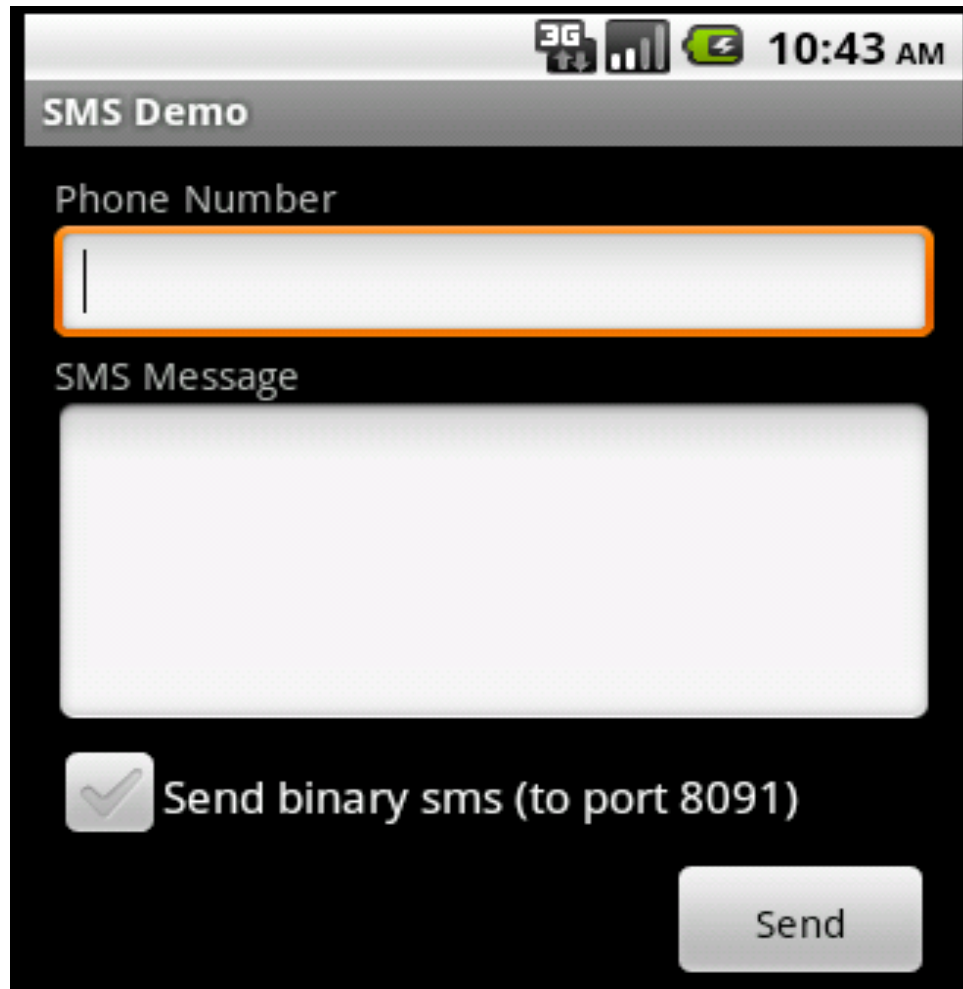
```
public void sendMultipartTextMessage (String destinationAddress,  
String scAddress, ArrayList parts, ArrayList sentIntents,  
ArrayList deliveryIntents)
```

### Sending binary SMS message

```
public void sendDataMessage (String destinationAddress, String scAddress,  
short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)
```

## Enviando um SMS

Faça uma aplicação que envie uma mensagem de texto (caso o texto tenha menos de 140 caracteres), envie uma mensagem em múltiplas partes (caso o texto tenha mais de 140 caracteres) e envie uma mensagem SMS binária (caso a opção Send binary SMS esteja ativada).



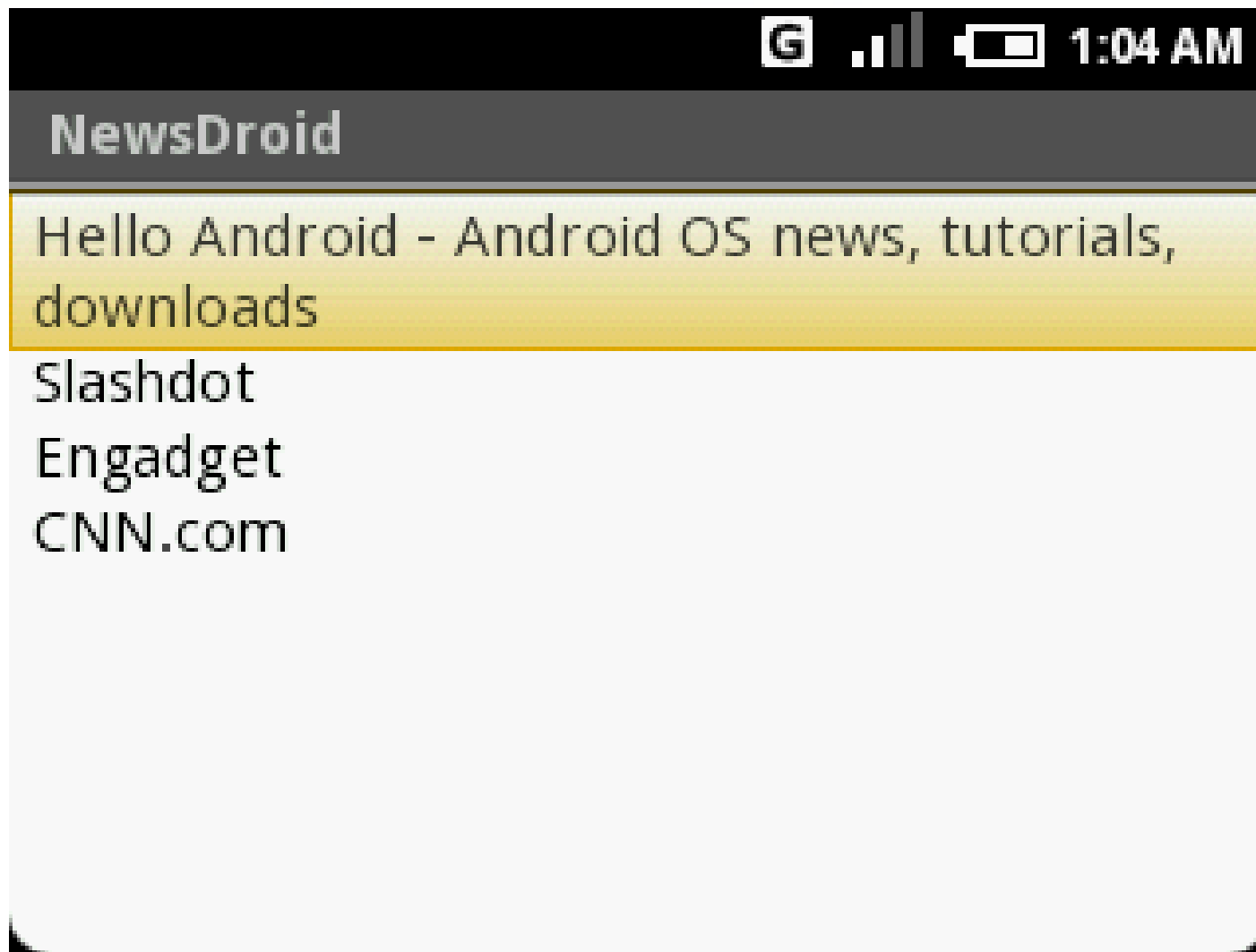
The screenshot shows an Android application titled "SMS Demo". At the top, the status bar displays "3G", signal strength, battery level, and the time "10:43 AM". The app interface has a black background. It features a "Phone Number" label above a white text input field with an orange border. Below this is an "SMS Message" label above a larger white text area. At the bottom left, there is a checked checkbox labeled "Send binary sms (to port 8091)". At the bottom right, there is a "Send" button.



# Primeira **Aplicação**

Desenvolvendo uma Aplicação Android com arquitetura MVC

# Leitor de Feeds (RSS)



## ListActivity

A lista de Feeds ou Artigos pode ser facilmente exibida por meio de uma especialização de Activity, o **ListActivity**.

Trata-se de um Activity que mostra uma lista de itens associados a uma fonte de dados, como um array ou um Cursor. Além disso, disponibiliza tratadores de eventos para quando o usuário seleciona um item.

O conteúdo do ListActivity é associado através do método **setListAdapter()** que recebe como parâmetro uma classe que implementa a interface ListAdapter. Pode ser um ArrayAdapter, um SimpleAdapter (Map) ou um SimpleCursorAdapter (Cursor).



## Estrutura do aplicativo (MVC)

O aplicativo **NewsDroid** consiste, basicamente, de sete classes:

Classes de modelo:

- **Artigo** – armazena os dados de um artigo, como o título e o endereço;
- **Feed** – armazena os dados de um feed, como o título e o endereço do arquivo XML.

Classes de visão:

- **ArtigosList** – ListActivity que mostra os artigos;
- **FeedsList** – ListActivity que mostra os feeds;
- **URLEditor** – Activity para que o usuário digite a URL do feed desejado.

Classes de controle:

- **NewsDB** – abstrai o uso de SQLite para persistência;
- **RSSHandler** – realiza o *parsing* de feeds RSS e captura apenas o necessário dos mesmos.

## Feed e Artigo

Entidades correspondentes às tabelas do SQLite. Um Feed possui um ID, título e uma URL. Um Artigo, por sua vez, possui um ID, o ID do feed a qual pertence, título e URL.

```
class Feed extends Object {  
    public long feedId;  
    public String title;  
    public URL url;  
}
```

```
class Article extends Object {  
    public long articleId;  
    public long feedId;  
    public String title;  
    public URL url;  
}
```

## NewsDB

Classe responsável pela criação do banco de dados, inserção e remoção de registros.

Para o banco de dados do **NewsDroid**, são necessárias duas tabelas: *feeds* e *artigos*. A primeira deve conter um id, o título do feed e a URL do arquivo XML do feed. A segunda, por sua vez, contém um id, o título do artigo e a URL do artigo.

Assim, o código SQL para criação dessas tabelas é o seguinte:

```
create table feeds  
(feed_id integer primary key autoincrement,  
title text not null,  
url text not null);
```

```
create table articles  
(article_id integer primary key autoincrement,  
feed_id int not null, title text not null, url text not null);
```

## NewsDB

Agora que se conhece a composição do banco de dados da aplicação, pode-se determinar quais as responsabilidades da classe **NewsDB** que irá abstrair o banco.

- **Criar banco de dados e tabelas** – A classe **NewsDB** deve criar o banco de dados e as tabelas necessárias para o armazenamento dos feeds e artigos;
- **Inserir dados** – Prover métodos para a inserção de um novo feed ou artigo no banco de dados;
- **Excluir dados** – Prover métodos para a remoção de um feed ou artigo do banco de dados;
- **Obter dados** – Prover métodos para a obtenção de um feed e seus respectivos artigos do banco de dados.

## RSSHandler

Classe responsável por baixar um RSS (arquivo XML) e realizar o *parsing* do mesmo, obtendo o seu conteúdo. Para isso, faz-se uso da API SAX (Simple API for XML), nativa do Android SDK.

```
1. <?xml version="1.0" encoding="utf-8" ?>
2. <rss version="2.0" xml:base="http://www.helloandroid.com"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
3.     <channel>
4.         <title>Hello Android - Android OS news, tutorials,
           downloads</title>
5.         <link>http://www.helloandroid.com</link>
6.         <description />
7.         <language>en</language>
8.         <item>
9.             <title>Biggest story of the year!</title>
10.            <link>http://www.helloandroid.com/node/59</link>
11.            <description>Here is a teaser for the
           story.</description>
12.            <comments>http://www.helloandroid.com/node/59#comments</comments>
13.            <pubDate>Sat, 17 Nov 2007 15:07:25 -0600</pubDate>
14.            <dc:creator>hobbs</dc:creator>
15.        </item>
16.    </channel>
17. </rss>
```

## RSSHandler (Feed e Artigo)

No arquivo XML, cada Feed está representado por uma tag `<channel>`, possuindo um título (`<title>`), uma URL (`<link>`), além de descrição e linguagem.

Para cada feed, há um conjunto de artigos. No arquivo XML, cada artigo está representado por uma tag `<item>`, possuindo um título (`<title>`), uma URL (`<link>`), uma descrição (equivalente ao conteúdo do artigo e representada pela tag `<description>`), URL de comentários (`<comments>`), data de publicação (`<pubDate>`) e o criador (`<dc:creator>`).

## RSS Handler (Parsing)

A API SAX é bastante simples de ser utilizada. Basicamente, ao analisar o trecho `<exemplo>teste</exemplo>`, o *Handler* trabalha com a chamada de três métodos:

- **startElement()** – Essa função é chamada quando o *parser* encontra um elemento. No exemplo acima, **startElement()** é invocado com a passagem de “exemplo” como parâmetro;
- **characters()** – Chamada logo após o **startElement()** e antes de se encontrar o final do elemento. No exemplo acima, **characters()** é invocado com a passagem de “teste” como parâmetro;
- **endElement()** – Chamada quando o *parser* encontra o final do elemento. No exemplo acima, **endElement()** é invocado após **characters()** com a passagem de “exemplo” como parâmetro



## RSS Handler – Criando um Feed

A partir de um SAXParserFactory, cria-se um XMLParser e, a partir deste, um XMLReader. Por fim, define-se o *ContentHandler* e então inicia-se o *parsing*.

Em código:

```
targetFlag = TARGET_FEED;
droidDB = new NewsDroidDB(ctx);
currentFeed.url = url;

SAXParserFactory spf = SAXParserFactory.newInstance();
SAXParser sp = spf.newSAXParser();
XMLReader xr = sp.getXMLReader();
xr.setContentHandler(this);
xr.parse(new InputSource(url.openStream()));
```

## RSS Handler – Técnica para leitura do XML

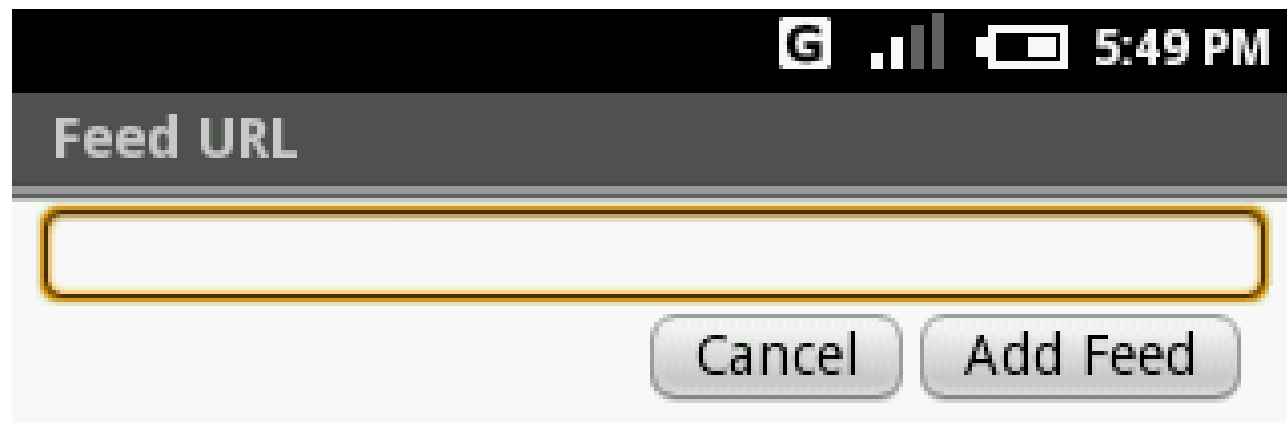
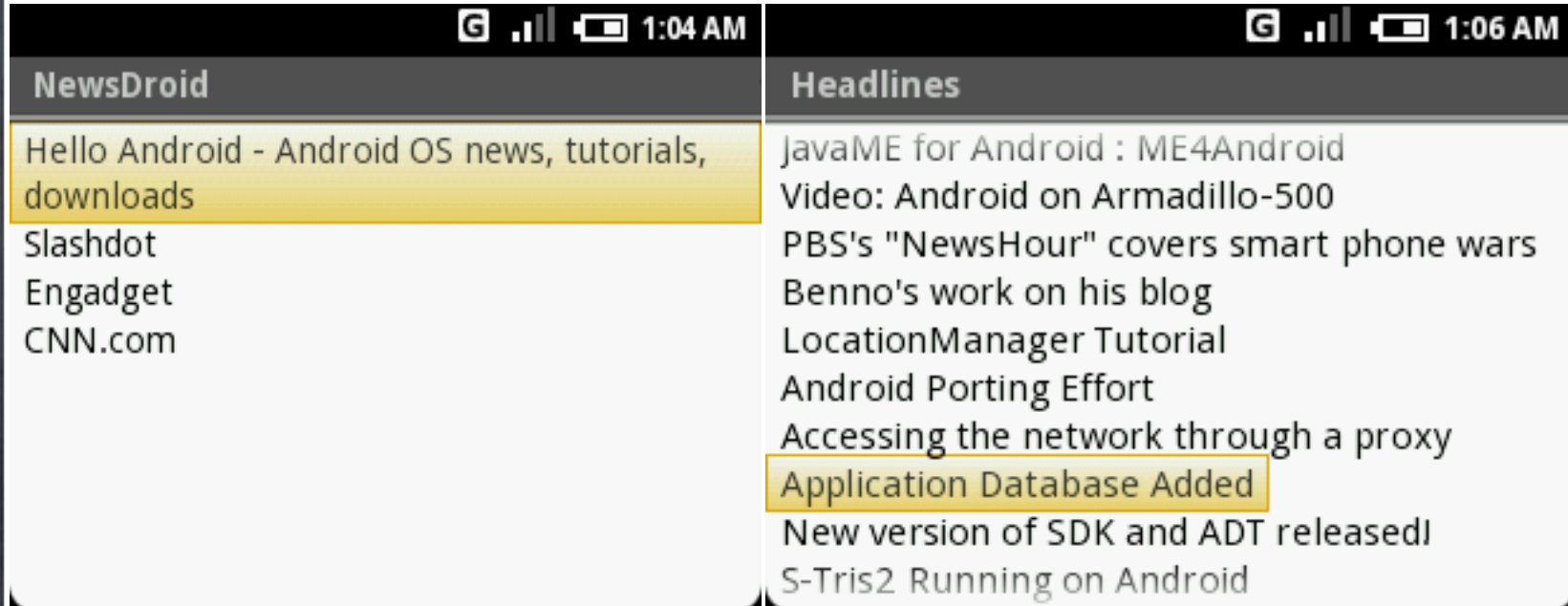
Criam-se campos booleanos para as tags que serão lidas (por exemplo: **inItem**, **inTitle** e **inLink**. Quando o *startElement()* é invocado, o campo correspondente se torna **true**. Já quando o *endElement()* é invocado, o campo correspondente se torna **false**.

Dessa forma, quando o **characters()** é chamado, sabe-se em que tag o *parser* se encontra e, então, pode-se atualizar os valores correspondentes ao artigo atual e o feed atual.

No **endElement()**, uma vez que o feed atual ou artigo atual já foi lido, pode-se armazenar os dados no banco de dados.

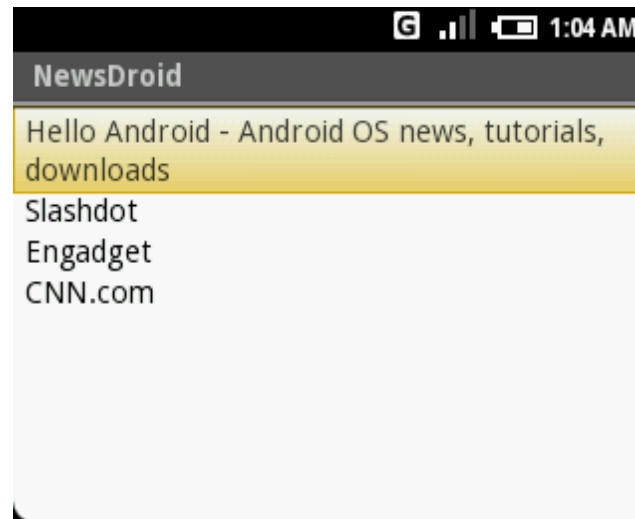
## Interface do Usuário

A interface da aplicação é composta por três telas: lista de Feeds, lista de Artigos e a tela para adição de um novo Feed.



## Interface do Usuário – Seleção de um Feed

Quando um feed é selecionado, cria-se um Intent que chama a Activity de listagem dos artigos desse feed. Para isso, é necessário embutir os dados do feed como parâmetros da Activity, conforme mostrado abaixo.



```
@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);

    Intent i = new Intent(this, ArticlesList.class);
    i.putExtra("feed_id", feeds.get(position).feedId);
    i.putExtra("title", feeds.get(position).title);
    i.putExtra("url", feeds.get(position).url.toString());
    startSubActivity(i, ACTIVITY_VIEW);
}
```

## Interface do Usuário – Seleção de um Feed

Uma vez selecionado o feed, o Activity de artigos é criado, recebendo os dados do feed pelo Intent. Isso é feito dentro do método **onCreate()** do ArtigosActivity.

```
@Override
protected void onCreate(Bundle icicle) {
    try {
        super.onCreate(icicle);
        droidDB = new NewsDroidDB(this);
        setContentView(R.layout.articles_list);

        feed = new Feed();

        if (icicle != null) {
            feed.feedId = icicle.getLong("feed_id");
            feed.title = icicle.getString("title");
            feed.url = new URL(icicle.getString("url"));
        } else {
            Bundle extras = getIntent().getExtras();
            feed.feedId = extras.getLong("feed_id");
            feed.title = extras.getString("title");
            feed.url = new URL(extras.getString("url"));

            droidDB.deleteArticles(feed.feedId);
            RSSHandler rh = new RSSHandler();
            rh.updateArticles(this, feed);
        }

        fillData();
    } catch (MalformedURLException e) {
        Log.e("NewsDroid", e.toString());
    }
}
```

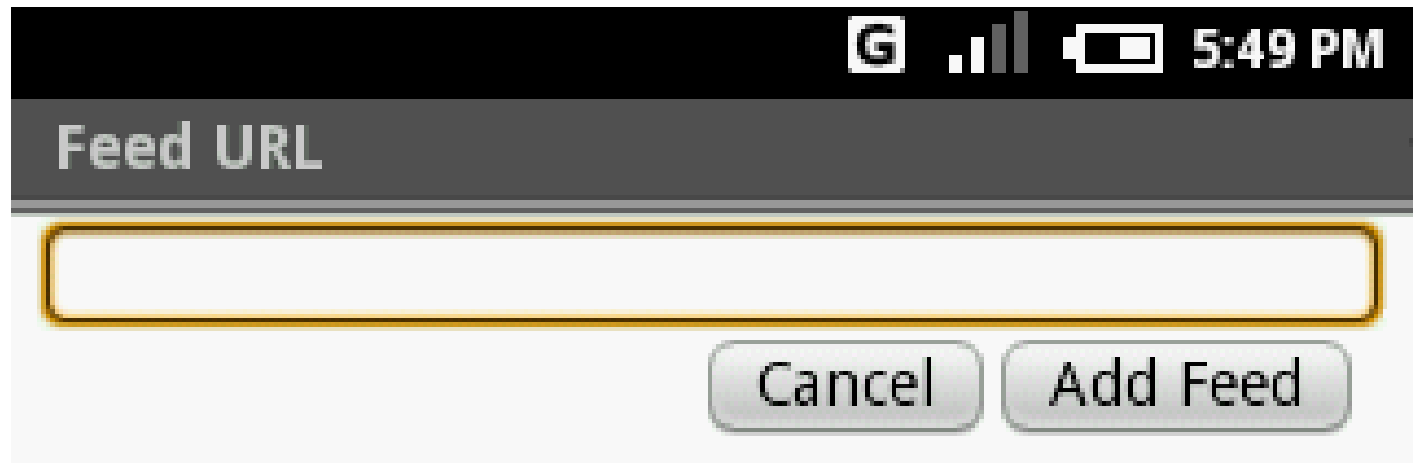
## Interface do Usuário – Seleção de um Feed

Para garantir que, quando o `ArtigosActivity` seja pausado ou destruído, ele não perca o seu estado, deve-se implementar o método **`onFreeze()`**, no qual se armazenam as informações desejadas.

```
@Override
protected void onFreeze(Bundle outState) {
    super.onFreeze(outState);
    outState.putLong("feed_id", feed.feedId);
    outState.putString("title", feed.title);
    outState.putString("url", feed.url.toString());
}
```

## Interface do Usuário – Adição de um Feed

Simple Activity na qual o usuário digita a URL do Feed e o mesmo é adicionado no banco de dados.

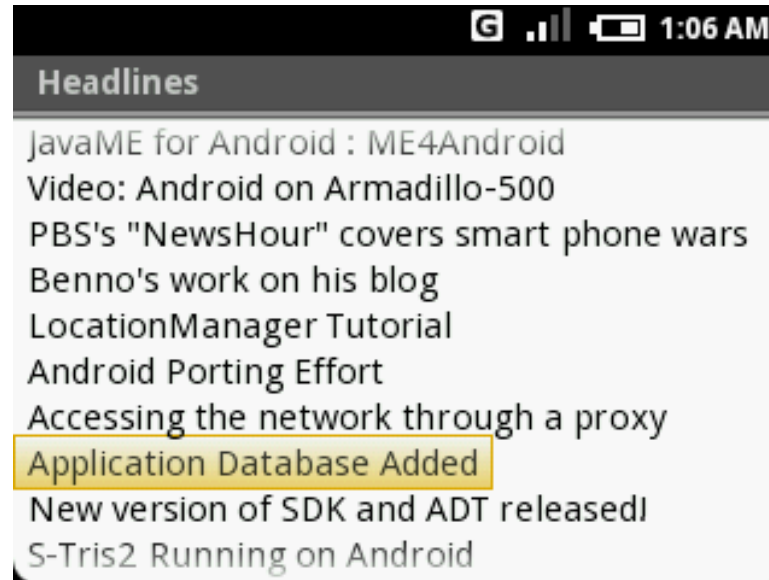


The screenshot displays the 'Feed URL' activity of the NewsDroid application. At the top, a black status bar shows the Google logo, signal strength, battery level, and the time 5:49 PM. Below this, a dark gray header contains the text 'Feed URL'. A large, empty text input field with a yellow border is positioned below the header. At the bottom right of the screen, there are two buttons: 'Cancel' and 'Add Feed'.



## Interface do Usuário – Seleção de um Artigo

Quando um artigo é selecionado, cria-se um Intent que é responsável por invocar o browser do Android, passando como parâmetro a URL do artigo escolhido.



```
@Override
protected void onItemClick(ListView l, View v, int position, long id) {
    super.onItemClick(l, v, position, id);
    Intent myIntent = new Intent(Intent.VIEW_ACTION,
ContentURI.create(articles.get(position).url.toString()));
    startActivity(myIntent);
}
```



**OBRIGADO!**

