Tópicos Avançados de Programação

Aula 06

Prof. Marcelo Rodrigues do Nascimento

Conversa inicial

Caro aluno, estamos começando a sexta aula de **Tópicos Avançados de Programação**, seja bem-vindo!

Continuando pelos componentes de aplicação, trabalharemos nesta rota com os Broadcast Receivers, que nos permitirão uma interação mais rica com o sistema operacional Android. Aprenderemos também a criar notificações a nossos usuários, com a definição de ações-padrão.

Trabalharemos também o conceito de conectividade, criando aplicativos que interajam com a Internet, seja baixando imagens ou leitores de notícias. Para que possamos utilizar desse tipo de ação, devemos aprender primeiramente a respeito do conceito de linha de processamento de aplicativos Android, como criar linhas de processamento paralelas e como transferir informações sobre tarefas completadas de uma linha de processamento para outra.

Assista à videoaula, em que o professor Marcelo apresenta o conteúdo desta rota. Confira!

Contextualizando

Quando desenvolvemos aplicativos, é comum que tarefas utilizem alto processamento ou até mesmo funções que requerem algum tipo de bloqueio, por exemplo, conectividade com a Internet. Essas tarefas não devem ser atribuídas à linha principal de processamento de seu aplicativo, uma vez que esses processos podem, por exemplo, dar ao usuário a impressão de que seu aplicativo travou. É necessário que trabalhemos com novas linhas de processamento, em conjunto com a linha principal. Também durante a execução dos aplicativos eventualmente necessitamos chamar a atenção do usuário para um determinado evento.

A plataforma Android nos fornece ferramentas para isso, permitindo inclusive ao usuário determinar a ação a ser executada a partir daquela notificação. Dessa maneira, podemos monitorar eventos que ocorrem no

dispositivo e preparar nossos aplicativos para que respondam de acordo. Finalizaremos aproveitando todos os conceitos entendidos até agora, desenvolveremos também um leitor de notícias *on-line* e interativo.

Pesquise

Tema 1: Processamento Concorrente

Quando um aplicativo Android é iniciado, uma nova linha de processamento chamada UIThread é iniciada em seu dispositivo. A UIThread é a principal linha de execução de seu aplicativo. É nela que a maior parte de seu código é executado. Todos os seus componentes de aplicação (Activiy, Services, Content Providers e BroadcastReceivers) são criados nesta thread e várias chamadas de sistema a esses componentes são executadas nela.

Por exemplo, digamos que seu aplicativo seja composto de uma única classe Activity. Todos os métodos que controlam o ciclo de vida e a maior parte do gerenciamento de eventos é executado na UIThread, como os métodos de onCreate, onPause, onClick. Além disso, é nessa thread que todas as atualizações da interface do usuário (UI) são feitas. Tudo que força uma atualização da interface do usuário deve acontecer na UIThread.

À medida que um aplicativo é desenvolvido e novas funcionalidades são adicionadas a ele, essa linha de processamento tende a tornar-se mais complexa. Adicione, ainda, acesso a componentes que exijam grande capacidade de processamento ou acesso a funções com bloqueio – como funções de rede – e seu aplicativo se tornará lento, chegando até mesmo a travar. Se a UIThread for bloqueada por mais de alguns segundos (cerca de 5 segundos), o usuário receberá a mensagem de “Aplicativo não respondendo”. Para que isso não ocorra, devemos tratar processos concorrentes em paralelo, criando novas linhas de processamento.

Para tarefas que consumam muito tempo ou uso intensivo da CPU, temos basicamente duas maneiras de criar isso:

- Threads Java
- AsyncTasks

Uma não é necessariamente melhor do que a outra, mas conhecer o potencial de cada uma trará benefícios à performance:

Use AsyncTasks para:

✓ Operações simples de rede que não requeiram o *download* de muitos dados.

✓ Tarefas de disco que não ocupem mais do que poucos milissegundos.

Use Threads Java para:

✓ Operações de rede que envolvam quantidades maiores de dados (tanto para *upload* quando para *download*).

✓ Tarefas de alto consumo de CPU que necessitem ser executadas em segundo plano.

Quando criamos novos processos, devemos ter duas regras básicas em mente:

1. Não podemos bloquear o processamento da UIThread.
2. Não podemos acessar os componentes de interface do usuário do android e fora da UIThread.

As AsyncTasks possibilitam um modo fácil e correto da UIThread. Essa classe permite a execução de operações em segundo plano e publica seu resultado na UIThread sem a necessidade de manipulação de threads ou handlers.

Para que possamos utilizar uma Asynctask, devemos criar uma nova classe que a tenha como base e sobrescrever ao menos um de seus métodos: doInBackground(), mas a maior parte de seu uso também sobrescreverá o método onPostExecute().

Uma classe Asynctask requer a criação de três tipos de dados a serem utilizados pela tarefa assíncrona:

```
private class MinhaTarefa extends AsyncTask<String, Void, Bitmap>
```

1. Parâmetros: tipo de argumento passado à tarefa no momento de sua execução.
2. Progresso: informação a respeito do progresso a ser publicado durante a execução em segundo plano.
3. Resultado: tipo de dado resultante da operação em segundo plano.

Nem todos os tipos são sempre utilizados por uma AsyncTask. Para marcar um tipo não utilizado, simplesmente utilize o tipo Void.

```
private class MinhaTarefa extends AsyncTask<Void, Void, Void>
```

Quando uma AsyncTask é executada, ela passa por 4 passos:

1. **onPreExecute**() – executado antes do início da tarefa;
2. **doInBackground**() – invocado em segundo plano imediatamente após o onPreExecute();
3. **onProgressUpdate**() – invocado na UIThread após a chamada de publishProgress();
4. **onPostExecute**() – invocado na UIThread após o processamento em segundo plano ser finalizado.

Regras a respeito de AsyncTask

- A classe AsyncTask deve ser carregada na UIThread.
- A classe deve ser instanciada na UIThread.
- Não executar as operações onPreExecute, onPostExecute, doInBackground ou onProgressUpdate manualmente.
- A tarefa pode ser executada apenas uma vez.

Para testar esse conceito, criaremos um aplicativo que efetua o *download* de uma imagem via internet e a mostra em nossa interface principal.

Inicie um novo projeto no Android Studio, chamando-o de UtilizandoAsyncTask.

Esse projeto será composto de uma Activity contendo uma ImageView e

um Button.

Configure o Button com as seguintes propriedades:

id: btnDownload

text: CARREGAR IMAGEM

Configure a ImageView com as seguintes propriedades:

Id: imgCarregada

Para sua referência, aqui está o código fonte completo de nosso arquivo activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.utilizandoasynctask.MainActivity
">

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Carregar Imagem"
      android:id="@+id/btnDownload"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true" />
```

```
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/imgCarregada"
        android:layout_centerVertical="true"
        android:layout_alignStart="@+id/btnDownload" />
</RelativeLayout>
```

Em nossa classe MainActivity.java, devemos criar os objetos que representam os componentes de nossa interface:

```
ImageView imgCarregada;
Button btnDownload;
```

E os instancie corretamente:

```
imgCarregada = (ImageView) findViewById(R.id.imgCarregada);
btnDownload = (Button) findViewById(R.id.btnDownload);
```

Agora devemos redefinir o listener onClickListener de nosso objeto btnDownload:

```
btnDownload.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
    }
});
```

Crie agora uma nova classe chamada MinhaAsyncTask, dentro de nossa classe MainActivity, passando como tipos genéricos String, Void e Bitmap. Você será solicitado a implementar o método doInBackground():

```
private class MinhaAsyncTask extends AsyncTask<String, Void, Bitmap>{
    @Override
    protected Bitmap doInBackground(String... strings) {
        return null;
    }
}
```

Para efetuar o *download* da imagem, utilizaremos a classe URL para definir o endereço de onde a imagem deve ser baixada:

```
URL url = new URL(strings[0]);
```

E utilizaremos o endereço armazenado na url para baixar o arquivo, empregando a factory BitmapFactory através de seu método decodeStream():

```
Bitmap b =
BitmapFactory.decodeStream(url.openConnection().getInputStream());
```

E então retornamos o objeto b como retorno de nossa função.

Não esqueça de encapsular esses comandos em uma estrutura de try catch, uma vez que exceções podem ocorrer (url incorreta, servidor indisponível etc.).

```
@Override
protected Bitmap doInBackground(String... strings) {
   try {
      URL url = new URL(strings[0]);
      Bitmap b =
BitmapFactory.decodeStream(url.openConnection().getInputStream());
      return b;
   } catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}
```

Após finalizarmos o método doInBackground(), que apenas definiu o que deve ser feito em segundo plano, devemos atualizar a interface do usuário com a imagem recém-baixada. Para isso, utilizaremos o método que ocorre após a execução do doInBrackground(), no caso onPostExecute().

Nesse método (que recebe o retorno de doInBackground(), atualizaremos o componente imgCarregada com o conteúdo do Bitmap baixado. Não se esqueça que, em caso de falha no laço try catch o

componente bitmap pode vir null, portanto, trate esse evento de acordo:

```
@Override
protected void onPostExecute(Bitmap bitmap) {
    if (null != bitmap)
        imgCarregada.setImageBitmap(bitmap);
}
```

Para sua conferência, aqui está a classe MinhaAsyncTask completa:

```
private class MinhaAsyncTask extends AsyncTask<String, Void, Bitmap>{

    @Override
    protected Bitmap doInBackground(String... strings) {
        try {
            URL url = new URL(strings[0]);
            Bitmap b =
BitmapFactory.decodeStream(url.openConnection().getInputStream());
            return b;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @Override
    protected void onPostExecute(Bitmap bitmap) {
        if (null != bitmap)
            imgCarregada.setImageBitmap(bitmap);
    }
}
```

Retornando agora à classe MainActivity (lembre-se que as duas classes residem no mesmo arquivo, MainActivity.java), vamos redefinir o método

onClick de btnDownload para que execute a chamada de nossa Asynctask:

```
new MinhaAsyncTask().execute("https://developer.android.com/samples/CustomNotifications/res/drawable-xhdpi/robot_expanded.png");
```

O último passo a ser tomado é solicitar ao sistema operacional permissão para que possamos executar operações de acesso à Internet. Em seu AndroidManifest.xml, adicione a seguinte linha:

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

Para sua referência, aqui está o código completo de AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.grupouninter.aula.utilizandoasynctask">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>
```

```
</manifest>
```

Execute agora seu aplicativo. Clicando-se no botão, a imagem do robô Android é carregada na ImageView.

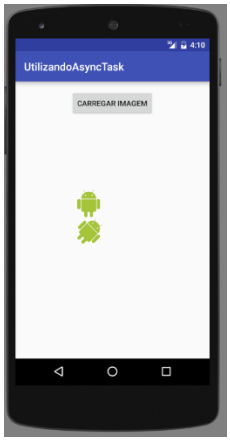

Figura 1 – AsyncTask em execução

Um ponto importante a destacarmos neste aplicativo é que o mesmo considera que o dispositivo possui acesso à internet disponível. Como se trata de um dispositivo com recursos limitados (acesso à internet é um recurso limitado), devemos considerar verificar se possuímos acesso à internet antes de solicitarmos o *download* do arquivo. Podemos verificar o *status* de rede utilizando o Objeto ConnectivityManager e NetWorkInfo. No entanto, deveremos solicitar permissão especial para isso em nosso AndroidManifest:

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

Para sua referência, aqui está o arquivo AndroidManifest.xml após essa alteração:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.com.grupouninter.aula.utilizandoasynctask">
   <uses-permission android:name="android.permission.INTERNET" />
   <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
```

```
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Antes de mais nada, instanciamos o objeto ConnectivityManager:

```
ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(view.getContext().CONNECTIVITY_SERVICE);
```

E então o objeto NetworkInfo, a partir do objeto ConnectionManager:

```
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
```

Caso o objeto networkInfo não seja nulo, e esteja conectado, executamos o nosso código.

```
if (networkInfo != null && networkInfo.isConnected()) {
```

Para sua referência, aqui está o código completo do arquivo MainActivity.java:

```
package br.com.grupouninter.aula.utilizandoasynctask;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
```

```
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.Toast;

import java.net.URL;

public class MainActivity extends AppCompatActivity {

    ImageView imgCarregada;
    Button btnDownload;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        imgCarregada = (ImageView) findViewById(R.id.imgCarregada);
        btnDownload = (Button) findViewById(R.id.btnDownload);

        btnDownload.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                ConnectivityManager connMgr = (ConnectivityManager)
```

```
getSystemService(view.getContext().CONNECTIVITY_SERVICE);
            NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
            if (networkInfo != null && networkInfo.isConnected()) {

                new
MinhaAsyncTask().execute("https://developer.android.com/samples/Custo
mNotifications/res/drawable-xhdpi/robot_expanded.png");
            } else{
                Toast.makeText(view.getContext(), "Sem acesso à rede",
Toast.LENGTH_SHORT).show();
            }
        }
    });
  }

  private class MinhaAsyncTask extends AsyncTask<String, Void, Bitmap>{

    @Override
    protected Bitmap doInBackground(String... strings) {
      try {
        URL url = new URL(strings[0]);
        Bitmap b =
BitmapFactory.decodeStream(url.openConnection().getInputStream());
        return b;
      } catch (Exception e) {
        e.printStackTrace();
      }
      return null;
    }
```

```
        @Override
        protected void onPostExecute(Bitmap bitmap) {
            if (null != bitmap)
                imgCarregada.setImageBitmap(bitmap);
        }
    }
}
```

É importante ressaltar, no entanto, que a verificação de acesso à rede não identifica conectividade com um determinado servidor. Ou seja, ter acesso à rede não necessariamente indica que você tem acesso à internet ou ao servidor de imagem onde seu arquivo está hospedado.

O professor Marcelo aborda os pormenores deste tema no vídeo. Acompanhe!

Tema 2: Notificações ao Usuário

O sistema Android permite que sejam colocadas notificações na barra de título de sua aplicação, que podem ser expandidas pelo usuário e também iniciar uma nova atividade. Portanto, uma notificação é uma mensagem que pode ser exibida ao usuário fora da UI (User Interface) normal do aplicativo.

Ao ser disparada, uma notificação passa por duas etapas: primeiro, ela aparece como um ícone na área de notificação. Quando o usuário clica nessa área, a gaveta de notificação é aberta, onde estão disponíveis as notificações que foram emitidas para aquele dispositivo.

Notificações podem ser percebidas através do ícone na barra de notificações, aparecendo na tela de trava do dispositivo, piscando o LED, tocando um som, vibrando ou aparecendo momentaneamente na tela atual.

As notificações são representadas pela classe Notification. Para criar uma notificação você pode utilizar a classe NotificationManager, que pode ser recebida de um Context.

As ações e informações da interface de usuário da notificação são

criadas no objeto NotificationCompat.Builder. Para criar a própria notificação, chama-se o método NotificationCompat.Builder.build(), que retorna um objeto do tipo Notification contendo suas especificações.

Para se emitir a notificação, passamos o objeto Notification ao sistema chamando o método NotificationManager.notify();

Todo objeto Notification deve conter o seguinte:

- um ícone pequeno, definido por setSmallIcon();
- um título, definido por setContentTitle();
- um texto de detalhes, definido por setContentText().

Crie um novo projeto no Android Studio chamado MinhaNotificacao. Em sua activity_main, adicione apenas um Button com as seguintes propriedades:

id: btnNotifica

text: Notifica

Para sua referência, aqui está o código completo de nosso arquivo activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.minhanotificacao.MainActivity">

   <Button
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Notifica"
        android:id="@+id/btnNotifica"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Em nosso arquivo MainActivity.java, devemos agora criar o objeto que representará o Button em nossa interface:

```
Button btnNotifica;
```

E instanciá-lo corretamente:

```
btnNotifica = (Button) findViewById(R.id.btnNotifica);
```

Redefinimos agora o comportamento de nosso listener onClickListener

```
btnNotifica.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {


    }
});
```

O primeiro passo é então criarmos o Builder de notificação, passando os três métodos obrigatórios – setSmallIcon, setContentTitle e setContentText.

```
new NotificationCompat.Builder(view.getContext())
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Minha notificacao")
        .setContentText("Olá, mundo!");
```

Criamos, então, o NotificationManager:

```
NotificationManager mNotificationManager =
        (NotificationManager)
getSystemService(view.getContext().NOTIFICATION_SERVICE);
```

E solicitamos ao mesmo que execute a notificação

```
mNotificationManager.notify(1, mBuilder.build());
```

Para sua referência, aqui está o código completo de nosso MainActivity.java:

```
package br.com.grupouninter.aula.minhanotificacao;

import android.app.Activity;
import android.app.NotificationManager;
import android.os.Bundle;
import android.support.v7.app.NotificationCompat;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

    Button btnNotifica;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnNotifica = (Button) findViewById(R.id.btnNotifica);

        btnNotifica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                android.support.v4.app.NotificationCompat.Builder mBuilder =
                        new NotificationCompat.Builder(view.getContext())
                                .setSmallIcon(R.mipmap.ic_launcher)
                                .setContentTitle("Minha notificacao")
```

```
                .setContentText("Olá, mundo!");
        NotificationManager mNotificationManager =
                (NotificationManager)
getSystemService(view.getContext().NOTIFICATION_SERVICE);
        mNotificationManager.notify(1, mBuilder.build());
    }
  });
 }
}
```

Execute seu aplicativo e clique no botão NOTIFICA.

Figura 2 – Notificação enviada à barra de notificação.

Como você pode perceber, o ícone de seu aplicativo é enviado à barra de notificação. Se o usuário clicar no mesmo, a gaveta de notificações é mostrada, apresentando maiores detalhes a respeito da notificação recebida pelo dispositivo:

Figura 3 – Notificação enviada à barra de notificação.

Nosso aplicativo tem agora a capacidade de notificar ao usuário a respeito de algum evento que necessite sua atenção, mas não fornece nenhum tipo de ação que possibilite acesso direto ao evento em si.

Quando uma notificação é criada, é importante associarmos ao menos uma ação à mesma. Geralmente, a ação padrão em uma notificação é abrir uma Activity em seu aplicativo. A execução dessa Activity é passada à notificação através de uma Intent.

Como as Activities trabalham no formato de pilha (ou seja, as Activities vão sendo sobrepostas conforme são chamadas), devemos utilizar o objeto TaskStackBuilder para criar uma pilha virtual que será passada ao sistema operacional quando uma Activity for iniciada a partir de uma Notificação.

Após a criação da pilha virtual, informamos ao Builder qual será a Intent disparada (ou colocada no topo da pilha), através do objeto PendingIntent.

Vamos alterar nossa notificação agora para que, ao ser clicada, inicie uma segunda Activity em nosso aplicativo.

Crie uma segunda Activity chamada SegundaActivity e, em sua interface, (arquivo activity_segunda.xml) deixe o *layout* uma TextView informando que esta é a Segunda Activity.

Para sua comodidade, aqui está o código completo da Segunda Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.minhanotificacao.SegundaActivi
ty">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:text="Segunda Activity"
      android:id="@+id/textView"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Não implementaremos nenhum método na classe SegundaActivity.java, já que esta tem somente o propósito de mostrar a interface com o usuário, sem qualquer tipo de interação.

Aqui está o código completo da SegundaActivity.java, para sua referência:

```
package br.com.grupouninter.aula.minhanotificacao;
```

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class SegundaActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_segunda);
    }
}
```

De volta para a MainActivity, devemos agora criar a Intent que armazenará a Activity a ser enviada ao sistema operacional quando o usuário clicar na Activity.

```
Intent intent = new Intent(view.getContext(), SegundaActivity.class);
```

Agora criamos o objeto TaskStackBuilder para gerar a pilha virtual a ser utilizada pelo sistema operacional ao criarmos a Activity.

```
TaskStackBuilder stackBuilder = TaskStackBuilder.create(view.getContext());
```

Adicionamos, então, à stackBuilder a Activity que desejamos incluir na Stack, no caso, a SegundaAcivity.class.

```
stackBuilder.addParentStack(SegundaActivity.class);
```

E, então, adicionamos à stack a Intent que desejamos que seja mostrada ao usuário quando este clicar na notificação.

```
stackBuilder.addNextIntent(intent);
```

Criamos agora o objeto PendingIntent, que informará a nosso builder qual Intent está aguardando para se tornar o topo da pilha.

```
PendingIntent pendingIntent = stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);
```

A FLAG_UPDATE_CURRENT indica que caso a PendingIntent já exista, mantenha o objeto mas substitua seus dados com o valor da nova Intent.

Adicionamos agora a nosso Builder a pendingIntent:

```
mBuilder.setContentIntent(pendingIntent);
```

Para sua conveniência, aqui está o código completo de nossa MainActivity.java:

```
package br.com.grupouninter.aula.minhanotificacao;

import android.app.Activity;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.TaskStackBuilder;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.NotificationCompat;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

    Button btnNotifica;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnNotifica = (Button) findViewById(R.id.btnNotifica);

        btnNotifica.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
```

```
            android.support.v4.app.NotificationCompat.Builder mBuilder =
                    new NotificationCompat.Builder(view.getContext())
                            .setSmallIcon(R.mipmap.ic_launcher)
                            .setContentTitle("Minha notificacao")
                            .setContentText("Olá, mundo!");

            Intent intent = new Intent(view.getContext(), SegundaActivity.class);

            TaskStackBuilder stackBuilder =
TaskStackBuilder.create(view.getContext());
            stackBuilder.addParentStack(SegundaActivity.class);
            stackBuilder.addNextIntent(intent);

            PendingIntent pendingIntent = stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);

            mBuilder.setContentIntent(pendingIntent);



            NotificationManager mNotificationManager =
                    (NotificationManager)
getSystemService(view.getContext().NOTIFICATION_SERVICE);
            mNotificationManager.notify(1, mBuilder.build());
        }
    });
  }
}
```

Execute o aplicativo. Quando pressionamos agora o botão, a Notificação é enviada. O que acontece quando seu usuário clica na Notificação? E o que acontece quando ele pressiona no botão de retornar do dispositivo Android?

Acompanhe a videoaula.

Tema 3: BroadCast Receivers

Um BroadcastReceiver é o componente Android que possibilita ao seu aplicativo registrar-se para o recebimento de eventos de aplicativos ou de sistema. Todos os receptores registrados para um evento são notificados pela plataforma Android quando este evento ocorre.

Por exemplo, aplicativos podem registrar-se para o evento ACTION_BOOT_COMPLETE, que é disparado sempre que o dispositivo Android termina o processo de boot.

O ciclo de vida básico de um BroadcastReceiver é composto do método chamado onReceive(). Assim que esse método é executado, o sistema considera o objeto finalizado e não mais ativo.

Um BroadcastReceiver pode ser registrado via o manifesto AndroidManifest.xml. Esse tipo de registro, via AndroidManifest.xml, é conhecido como registro estático. No entanto, é possível também registrar-se dinamicamente, utilizando-se o método Context.registerReceiver().

A classe que implementa esse objeto deve ser derivada da classe BroadcastReceiver. Se o evento para qual essa classe é disparada, então, o método onReceive() é chamado automaticamente pelo sistema Android.

Antes da API 11 do Android, você não podia executar operações assíncronas dentro do método onReceive, já que este tem que ser finalizado para que o sistema Android possa reciclar o componente. Caso você tenha um método que demora, deve considerar a inicialização de um Service. Lembre-se de que, uma vez que seu sistema recebe a mensagem para a qual está registrado, você deve finalizar o objeto assim que possível.

Para testarmos esse conceito, criaremos dois aplicativos. Um que se registrará para um determinado evento e, caso este evento ocorra, criará uma notificação para o usuário na barra de notificações. O segundo aplicativo se encarregará apenas de disparar ao sistema operacional um evento. Vamos

iniciar, então, com o projeto MeuBroadcastReceiver.

Este é composto apenas de uma Activity, a mesma com um TextView com as seguintes propriedades:

id: txtRecebido

text: BROADCAST RECEBIDO

Para sua conveniência, aqui está o código completo de nosso arquivo activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.meubroadcastreceiver.MainActiv
ity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:text="BROADCAST RECEBIDO"
      android:id="@+id/txtRecebido"
      android:layout_centerVertical="true"
      android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Criaremos agora, nesse aplicativo, uma nova classe derivada da classe BroadcastReiver, chamada MeuReceiver.

```
package br.com.grupouninter.aula.meubroadcastreceiver;


/**
 * Created by Marcelo on 21/08/2016.
 */
public class MeuReceiver {
}
```

Devemos, agora, informar que essa classe é derivada de BroadcastReceiver e implementar o método necessário onReceive().

```
public class MeuReceiver extends BroadcastReceiver {


    @Override
    public void onReceive(Context context, Intent intent) {


    }
}
```

Quando recebermos esse BroadCast, desejamos que o usuário seja notificado pelo sistema de notificação.

```
android.support.v4.app.NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(context)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentTitle("Mensagem recebida")
                .setContentText("Seu broadcast chegou!");
```

Criamos agora a nova Intent que armazenará os dados da classe a ser utilizada pelo sistema operacional como base para a nova Activity.

```
Intent nintent = new Intent(context, MainActivity.class);
```

Adicionamos uma pilha virtual para nossa Activity:

```
TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);
```

```
stackBuilder.addParentStack(MainActivity.class);
```

Adicionamos nossa Intent como próxima Intent a ser executada:

```
stackBuilder.addNextIntent(nintent);
```

Adicionamos nossa pilha virtual à classe PendingIntent:

```
PendingIntent pendingIntent = stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);
```

Adicionamos a nosso builder nossa PendingIntent:

```
mBuilder.setContentIntent(pendingIntent);
```

E, finalmente, pelo NotificationManager disparamos a mensagem desejada:

```
NotificationManager mNotificationManager =
    (NotificationManager)
context.getSystemService(context.NOTIFICATION_SERVICE);
mNotificationManager.notify(1, mBuilder.build());
```

Agora nosso BroadcastReceiver está pronto para, no momento de seu disparo, apresentar ao nosso usuário uma notificação que terá como ação-padrão a criação de uma nova Activity baseada em MainActiviy.

Para sua conveniência, aqui está o código completo de nosso BroadcastReceiver:

```
package br.com.grupouninter.aula.meubroadcastreceiver;

import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.TaskStackBuilder;
import android.support.v7.app.NotificationCompat;

/**
```

```
 * Created by Marcelo on 21/08/2016.
 */
public class MeuReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {

        android.support.v4.app.NotificationCompat.Builder mBuilder =
                new NotificationCompat.Builder(context)
                        .setSmallIcon(R.mipmap.ic_launcher)
                        .setContentTitle("Mensagem recebida")
                        .setContentText("Seu broadcast chegou!");

        Intent nintent = new Intent(context, MainActivity.class);

        TaskStackBuilder stackBuilder = TaskStackBuilder.create(context);
        stackBuilder.addParentStack(MainActivity.class);
        stackBuilder.addNextIntent(nintent);

        PendingIntent pendingIntent = stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);

        mBuilder.setContentIntent(pendingIntent);


        NotificationManager mNotificationManager =
                (NotificationManager)
context.getSystemService(context.NOTIFICATION_SERVICE);
        mNotificationManager.notify(1, mBuilder.build());
    }
```

}

O que nos resta agora é registrar uma mensagem a ser ouvida por nosso BroadcastReceiver. Iremos criar o vínculo estático para essa mensagem através do AndroidManifest.xml:

```
<receiver android:name=".MeuReceiver" >
   <intent-filter>
      <action
android:name="br.com.grupouninter.aula.meubroadcastreceiver.MEUBR
OADCAST">
      </action>
   </intent-filter>
</receiver>
```

Para sua comodidade, aqui está o código completo de nosso AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="br.com.grupouninter.aula.meubroadcastreceiver">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
      <activity android:name=".MainActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER"
/>
```

```
        </intent-filter>
    </activity>

    <receiver android:name=".MeuReceiver" >
        <intent-filter>
            <action
android:name="br.com.grupouninter.aula.meubroadcastreceiver.MEUBR
OADCAST">
            </action>
        </intent-filter>
    </receiver>
  </application>

</manifest>
```

Execute seu aplicativo e encerre-o. Apenas desejamos que ele seja registrado no sistema Android. Agora criaremos um novo Aplicativo, não ligado a este, que enviará mensagens ao sistema Android para serem capturadas por nosso BroadcastReceiver.

Crie um novo projeto em seu Android Studio chamado BroadcastSender. Esse novo projeto é composto de uma Activity com um botão com as seguintes propriedades:

id: btnEnvia

text: Envia Broadcast

Para sua comodidade, aqui está o código completo de nosso activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.broadcastsender.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Envia Broadcast"
        android:id="@+id/btnEnvia"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

No arquivo MainActivity.java, crie o objeto que representará o componente do layout:

```
Button btnEnvia;
```

Fazemos, então, sua inicialização correta e a redefinição do listener onClickListener:

```
btnEnvia.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
    }
});
```

Nesse aplicativo, não desejamos registrar um novo Broadcast Receiver, mas, sim, enviar uma mensagem ao sistema. Isso é feito criando-se uma Intent.

No registro que criamos em nosso BroadcastReceiver, registramos a

mensagem:

```
br.com.grupouninter.aula.meubroadcastreceiver.MEUBROADCAST
```

Portanto, devemos utilizá-la para o envio de nossa mensagem. O envio é feito pelo comando sendBroadcast():

```
Intent intent = new
Intent("br.com.grupouninter.aula.meubroadcastreceiver.MEUBROADCAST
");
sendBroadcast(intent);
```

Para sua referência, aqui está o código completo de nossa MainActivity.java:

```
package br.com.grupouninter.aula.broadcastsender;

import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    Button btnEnvia;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnEnvia = (Button) findViewById(R.id.btnEnvia);
```

```
        btnEnvia.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new
Intent("br.com.grupouninter.aula.meubroadcastreceiver.MEUBROADCAST
");
                sendBroadcast(intent);
            }
        });
    }
}
```

Execute o aplicativo. O que acontece quando o usuário clica no botão Enviar Broadcast?

Lembre-se que você está enviando uma Intent através de um Broadcast. Portanto, pode aproveitar-se disso para enviar objetos ou mensagens personalizas a seu Broadcast Receiver, para que este as execute de acordo com o tipo de informação recebida.

Não deixe de assistir à videoaula com o professor Marcelo em que ele explica mais sobre BroadCast Receivers.

Tema 4: Criando um Leitor RSS

Um dos serviços mais comuns disponíveis na Internet são os feed RSS. RSS é um recurso em XML que permite a websites divulgarem notícias em um formato predefinido.

Criaremos agora um leitor de notícias que se utilizará do feed gerado pela parte de tecnologia do website UOL (clique no *link* a seguir), colocando em uso vários dos conceitos que aprendemos até agora.

http://tecnologia.uol.com.br/ultnot/index.xml

Nosso aplicativo será composto basicamente de uma Activity, que

conterá uma ListView para exibição das notícias. Essa ListView será alimentada por um adaptador responsável por transformar uma lista de objetos em uma interface utilizável pelo usuário. Quando a View for criada por nosso adaptador, ela se utilizará de um *layout* que mostrará apenas uma TextView contendo o título da notícia a ser apresentada.

Se o usuário optar por pressionar um dos itens disponíveis na ListView, será disparado então uma Intent implícita, solicitando ao sistema operacional a realização de uma ação de visualização (ACTION_VIEW), passando como dado o *link* a ser acessado. Dessa maneira, quando a Intent implícita for executada, o sistema operacional se responsabilizará por localizar o leitor html a ser utilizado entre os disponíveis no sistema operacional, nos livrando, assim, das obrigações de criarmos um WebBrowser completamente do início.

Crie um novo aplicativo no Android Studio, chamado AndroidXMLParser. Esse novo aplicativo terá, como já foi dito, apenas uma Activity – chamada MainActivity.

Como nosso aplicativo utilizará a Internet para receber esse feed de notícias, nosso primeiro passo é justamente solicitar permissão ao sistema Android para utilização da Internet.

Em seu AndroidManifest.xml, adicione a seguinte linha:

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

Para sua referência, aqui está o arquivo completo de nosso AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.com.grupouninter.aula.androidxmlparser">
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
```

```
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Abra agora o arquivo activity_main.xml e adicione a ele um componente do tipo ListView com as seguintes propriedades:

id: listRss

Para sua referência, aqui está o código completo de nosso arquivo activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

tools:context="br.com.grupouninter.aula.androidxmlparser.MainActivity">
```

```
    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/listRss"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

No arquivo MainActivity.java, devemos agora criar o objeto que representará o componente ListView:

```
ListView listRss;
```

E, então, instanciá-lo corretamente:

```
listRss = (ListView) findViewById(R.id.listRss);
```

Antes de darmos prosseguimento, vamos entender a estrutura do RSS de notícias do site UOL:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<rss version="2.0">
<channel>
<title>UOL - O melhor conteúdo: Tecnologia</title>
<link><![CDATA[http://www.uol.com.br/]]></link>
<description>UOL - O melhor conteúdo: Tecnologia</description>
<language>pt-br</language>
<category>Notícias</category>
<copyright>UOL - O melhor conteúdo. Todos os direitos reservados.</copyright>
<image>
<title>
<![CDATA[UOL Tecnologia - Últimas Notícias]]>
</title>
<url>http://rss.i.uol.com.br/uol_rss.gif</url>
<link><![CDATA[http://musica.uol.com.br/ultnot/]]></link>
```

</image>
<item>
<title>
<![CDATA[Todos contra a Apple: veja propagandas em que concorrentes provocam a empresa]]>
</title>
<link>
<![CDATA[http://tecnologia.uol.com.br/album/2012/10/03/todos-contra-a-apple-concorrentes-lancam-propagandas-provocando-a-empresa-da-maca-veja.htm]]>
</link>
<description></description>
<pubDate>Tue, 16 Sep 2014 06:01:00 -0300</pubDate>
</item>
</channel>
</rss>

Após um cabeçalho, os itens que nos interessam – ou seja, as notícias – são armazenadas no formato:

```
<item>
<title> Uma noticia </title>
<link> Um titulo </link>
</item>
```

Ou seja, devemos providenciar uma estrutura de dados que se adeque a essa necessidade. Crie, então, uma nova classe chamada RssUOL e a defina com as seguintes propriedades:

```
private String titulo;
private String link;
```

E crie os getters e setters necessários para sua alimentação:

```
public void setTitulo(String titulo) {
   this.titulo = titulo;
```

```
}
public String getLink() {
    return link;
}
public void setLink(String link) {
    this.link = link;
}
```

Para sua referência, aqui está o código fonte completo de nosso arquivo RssUOL.java:

```
package br.com.grupouninter.aula.androidxmlparser;
/**
 * Created by Marcelo on 21/08/2016.
 */
public class RssUOL {

    private String titulo;
    private String link;

    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public String getLink() {
        return link;
    }
    public void setLink(String link) {
        this.link = link;
```

```
    }
}
```

Essa estrutura representa apenas uma notícia. Idealmente nosso RSS trará várias notícias, então, devemos criar um ArrayList que as comporte.

Voltando ao arquivo MainActivity.java, crie a ArrayList que armazenará os objetos referentes à cada notícia:

```
ArrayList<RssUOL> rssUol;
```

Agora que temos a estrutura pronta, devemos buscá-la na Internet. Como se trata de um serviço de rede, de tempo indeterminado, devemos criar uma AsyncTask para executar essa tarefa.

No arquivo MainActivity.java, crie a classe MinhaAsyncTask, que receberá como dados genéricos Void, Void e ArrayList<RssUOL>:

```
private class MinhaAsyncTask extends AsyncTask<Void, Void,
ArrayList<RssUOL>> {
```

Devemos, agora, implementar o método obrigatório de uma AsyncTask:

```
@Override
protected ArrayList<RssUOL> doInBackground(Void... voids) {
        }
```

Dentro de nosso método doInBackground, nosso primeiro passo será definir uma URL de acesso. Fazemos isso criando um objeto do tipo URL e passando a ele o endereço desejado. Não esqueça de encapsular com try catch, pois imprevistos podem ocorrer (url incorreta, servidor indisponível etc.).

```
try {
URL url = new URL("http://tecnologia.uol.com.br/ultnot/index.xml");

} catch (Exception e){
    e.printStackTrace();

}
```

Agora devemos definir uma forma de leitura de nosso arquivo XML.

Felizmente a linguagem Java nos fornece ferramentas para isso.

Instanciaremos um objeto do tipo Document, que buscará as informações em XML vindas da URL e as processará (ou seja, as converterá a um formato entendido pelo objeto) de forma que seus dados possam ser acessados posteriormente:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(new InputSource(url.openStream()));
doc.getDocumentElement().normalize();
```

Após a normalização (ou seja, processamento de nosso XML), estamos prontos para navegar por seus nós:

O primeiro passo é localizar o nó que armazena a informação desejada, no caso, o nó com nome “item”:

```
NodeList nodeList = doc.getElementsByTagName("item");
```

Após sua localização, instanciaremos uma ArrayList que conterá objetos do tipo RssUOL. Para cada item encontrado, instanciaremos um novo objeto que será preenchido com o valor localizado e então armazenado nesta ArrayList

```
ArrayList<RssUOL> rssUol = new ArrayList<>();
```

Nosso objeto nodeList agora contém uma listagem de objetos (array) onde abriga todos os registros XML com o valor “item” e seus filhos. Tudo que precisamos fazer agora é iterar por eles, buscando seus valores e armazenando na arrayList.

```
for (int i = 0; i < nodeList.getLength(); i++ ){
    Node node = nodeList.item(i);
    Element elemento = (Element) node;

    NodeList noTitulo = elemento.getElementsByTagName("title");
    Element eleTitulo = (Element) noTitulo.item(0);
    noTitulo = eleTitulo.getChildNodes();
```

```
    NodeList noLink = elemento.getElementsByTagName("link");
    Element eleLink = (Element) noLink.item(0);
    noLink = eleLink.getChildNodes();
    RssUOL item = new RssUOL();
    item.setTitulo(((Node) noTitulo.item(1)).getNodeValue());
    item.setLink(((Node) noLink.item(1)).getNodeValue());
    rssUol.add(item);
}
```

Ao finalizarmos a leitura de todos os objetos, retornamos ao nosso método onPostExecute o objeto rssUOL, com todos os objetos nele contidos:

```
return rssUol;
```

No método onPostExecute de nossa AsyncTast, retornaremos a nossa MainActivy a ArrayList preenchida com todos os dados localizados.

```
@Override
protected void onPostExecute(ArrayList<RssUOL> rssUol) {
    retornaTask(rssUol);
}
```

Como você pode perceber, executamos um método chamado retornaTask, que ainda não foi implementado na classe MainActivity. Esse método será responsável por receber a ArrayList enviada por nossa AsyncTask e passá-la ao adaptador.

```
public void retornaTask(ArrayList<RssUOL> _rssUol){
        }
```

Devemos agora criar nosso adaptador, que se encarregará de criar uma interface para cada registro localizado em nossa ArrayList.

Antes de criarmos esse adaptador, lembre-se de criar o arquivo de *layout* que será utilizado pelo método getView para renderizar notícia a notícia.

Para sua conveniência, aqui está o código do arquivo layout_rss.xml:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
  android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="10dp">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textAppearance="?android:attr/textAppearanceSmall"
      android:text="Texto Rss"
      android:id="@+id/txtRss" />
</LinearLayout>
```

Crie uma nova classe chamada AdaptadorRSS e a defina como extensão da classe BaseAdapter:

```
public class AdaptadorRSS extends BaseAdapter {
        }
```

Tal qual nosso primeiro modelo de adaptador, devemos criar uma ArrayList que armazene todos os objetos do tipo RssUOL, um objeto que armazene seu contexto e um objeto responsável pelo LayoutInflater.

```
private List<RssUOL> rssUOL;
Context context = null;
private LayoutInflater inflater;
```

Crie agora o construtor para essa classe, solicitando como argumentos na criação da mesma um contexto e um ArrayList do tipo RssUol:

```
public AdaptadorRSS(Context context, List<RssUOL> rssUOL){
   this.rssUOL = rssUOL;
   this.inflater = LayoutInflater.from(context);
```

```
    this.context = context;
}
```

Lembre-se que, como nossa classe tem como base a classe BaseAdapter, os métodos getCount, getItem, getItemId e getView devem ser implementados.

Já discutimos anteriormente como implementar esses métodos, portanto, aqui estão eles:

```
@Override
public int getCount() {
    return this.rssUOL.size();
}

@Override
public Object getItem(int i) {
    return this.rssUOL.get(i);
}

@Override
public long getItemId(int i) {
    return i;
}
```

No método getView, criaremos a interface que permitirá ao usuário interagir com as notícias enviadas à nossa ArrayList.

Portanto, antes de mais nada, devemos buscar a notícia que estamos renderizando:

```
final RssUOL rss = (RssUOL) getItem(i);
```

Verificamos se nossa view não possui um *layout* atribuído e, caso não possua, atribuímos um *layout* com base no arquivo de layout layout_rss:

```
if (null == view){
    view = inflater.inflate(R.layout.layout_rss, null);
```

```
}
```

Crie então o objeto que representa a TextView no arquivo layout_rss:

```
TextView txtRss = (TextView) view.findViewById(R.id.txtRss);
```

E atribua à mesma o valor da propriedade título que obtemos de nosso objeto rss:

```
txtRss.setText(rss.getTitulo());
```

A ideia nesse projeto é que, ao usuário clicar em qualquer parte da view, e não somente em cima da TextView, uma Intent implícita seja chamada, solicitando ao sistema operacional que selecione uma Activity que tenha a funcionalidade de WebBrowser e passando a ela a URL localizada:

```
view.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse(rss.getLink()));
        context.startActivity(intent);
    }
});
```

E, finalmente, retornarmos a view finalizada.

```
return view;
```

Para sua referência, aqui está o código fonte completo do arquivo AdaptadorRSS.java:

```
package br.com.grupouninter.aula.androidxmlparser;

import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.TextView;
import java.util.List;

/**
 * Created by Marcelo on 21/08/2016.
 */
public class AdaptadorRSS extends BaseAdapter {

    private List<RssUOL> rssUOL;
    Context context = null;
    private LayoutInflater inflater;

    public AdaptadorRSS(Context context, List<RssUOL> rssUOL){
        this.rssUOL = rssUOL;
        this.inflater = LayoutInflater.from(context);
        this.context = context;
    }
    @Override
    public int getCount() {
        return this.rssUOL.size();
    }
    @Override
    public Object getItem(int i) {
        return this.rssUOL.get(i);
    }

    @Override
    public long getItemId(int i) {
```

```
        return i;
    }

    @Override
    public View getView(int i, View view, ViewGroup viewGroup) {

        final RssUOL rss = (RssUOL) getItem(i);

        if (null == view){
            view = inflater.inflate(R.layout.layout_rss, null);
        }
        TextView txtRss = (TextView) view.findViewById(R.id.txtRss);

        txtRss.setText(rss.getTitulo());

        view.setOnClickListener(new View.OnClickListener() {

@Override
            public void onClick(View view) {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(Uri.parse(rss.getLink()));
                context.startActivity(intent);
            }
        });
        return view;
    }
}
```

Voltando ao MainActivity, devemos agora implementar o método retornaTask. Esse método se utiliza de nosso novo adaptador, então, nosso primeiro passo é instanciá-lo corretamente:

```
AdaptadorRSS rssAdaptador;
```

Agora em retornaTast, atribuímos ao nosso Adaptador uma nova instância de nossa classe AdaptadorRSS, passando à mesma o contexto de nossa Activity e a lista de dados encontrada em nossa ArrayList rssUol. Finalmente, atribuímos à nossa ListView, o adaptador resultante dessa operação:

```
public void retornaTask(ArrayList<RssUOL> _rssUol){
   rssUol = _rssUol;
   if (rssUol != null){
      rssAdaptador = new AdaptadorRSS(this, rssUol);
      listRss.setAdapter(rssAdaptador);
   }
}
```

Tudo que nos resta agora é executar nossa AsyncTask, para que esta busque o feed de notícias e o retorne ao nosso método retyornaTask.

```
new MinhaAsyncTask().execute();
```

Para sua comodidade, aqui está o código fonte completo de nosso arquivo MainActivity.java:

```
package br.com.grupouninter.aula.androidxmlparser;

import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.ListView;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
```

```
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

public class MainActivity extends AppCompatActivity {

    ArrayList<RssUOL> rssUol;
    AdaptadorRSS rssAdaptador;
    ListView listRss;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listRss = (ListView) findViewById(R.id.listRss);

        new MinhaAsyncTask().execute();
    }

    public void retornaTask(ArrayList<RssUOL> _rssUol){

        rssUol = _rssUol;
        if (rssUol != null){
            rssAdaptador = new AdaptadorRSS(this, rssUol);
            listRss.setAdapter(rssAdaptador);
        }
```

```
    }

    private class MinhaAsyncTask extends AsyncTask<Void, Void,
ArrayList<RssUOL>> {

        @Override
        protected ArrayList<RssUOL> doInBackground(Void... voids) {
            try {

                URL url = new
URL("http://tecnologia.uol.com.br/ultnot/index.xml");
                DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
                DocumentBuilder db = dbf.newDocumentBuilder();
                Document doc = db.parse(new InputSource(url.openStream()));
                doc.getDocumentElement().normalize();

                NodeList nodeList = doc.getElementsByTagName("item");

                ArrayList<RssUOL> rssUol = new ArrayList<>();

                for (int i = 0; i < nodeList.getLength(); i++ ){
                    Node node = nodeList.item(i);
                    Element elemento = (Element) node;

                    NodeList noTitulo = elemento.getElementsByTagName("title");
                    Element eleTitulo = (Element) noTitulo.item(0);
                    noTitulo = eleTitulo.getChildNodes();
```

```
                NodeList noLink = elemento.getElementsByTagName("link");
                Element eleLink = (Element) noLink.item(0);
                noLink = eleLink.getChildNodes();


                RssUOL item = new RssUOL();
                item.setTitulo(((Node) noTitulo.item(1)).getNodeValue());
                item.setLink(((Node) noLink.item(1)).getNodeValue());

                rssUol.add(item);
            }
            return rssUol;

        } catch (Exception e){
            e.printStackTrace();
        }
        return null;
    }
    @Override
    protected void onPostExecute(ArrayList<RssUOL> rssUol) {
        retornaTask(rssUol);
    }
  }
}
```

Execute seu aplicativo. O que acontece quando ele é carregado?

Figura 4 – Feed de notícias em execução.

O que acontece quando você clica em uma das notícias disponíveis?

Figura 5 – Intent implícito em execução.

Veja mais sobre esse assunto na videoaula com o professor Marcelo.

Na prática

Que tal algumas atividades?

- Altere o projeto UtilizandoAsyncTask para que o usuário possa baixar o arquivo a partir de um endereço digitado por ele. Lembre-se de validar o endereço antes de passá-lo à classe de *download*.
- Altere nosso projeto MinhaNotificacao para que este mostre uma stack completa de uso normal do aplicativo, de maneira que quando o usuário pressionar o botão de retorno o aplicativo não feche, mas, em vez disso, seja enviado a Activity anterior, no caso, MainActivity.
- Altere nosso projeto de BroadcastReceiver para que seja transportado um objeto contendo a data e hora de envio da mensagem. Na Activity iniciada pelo Broadcastreceiver, mostre a data e hora em que a notificação foi enviada.
- Altere nosso projeto de Feed de Notícias para que seja possível buscar outra fonte de notícias além da Uol Tecnologias.
- Crie um serviço que busque alterações nas notícias, atualizando nossa interface automaticamente.

Mãos à obra!

Síntese

Estamos finalizando as aulas de **Tópicos Avançados de Programação**!

Neste módulo, entendemos o conceito da UIThread, suas funções e também a necessidade de se criar linhas de processamento paralelo para evitar seu sobrecarregamento. Trabalhamos o conceito de conectividade buscando imagens na Internet e atribuindo-a a um componente em nossa UIThread. Aprendemos a criar notificações aos usuários utilizando-se do NotificationManager, bem como ouvir e disparar eventos ao sistema operacional Android. Finalmente, criamos um aplicativo de leitura de notícias

completamente funcional, utilizando vários dos conceitos vistos nesta e em aulas anteriores.

Veja as considerações finais com o professor Marcelo.

Referências

DEITEL, Paul; DEITEL, Harvey; OSWALD, Alexander. **Android 6 para Programadores**: uma Abordagem Baseada em Aplicativos. Boockman: 2015.

NOTIFICAÇÃO. Disponível em: <https://developer.android.com/guide/topics/ui/notifiers/notifications.html>. Acesso em: 21 ago. 2016.

BROADCASTRECEIVE. Disponível em: <https://developer.android.com/reference/android/content/BroadcastReceiver.html>. Acesso em: 21 ago. 2016.