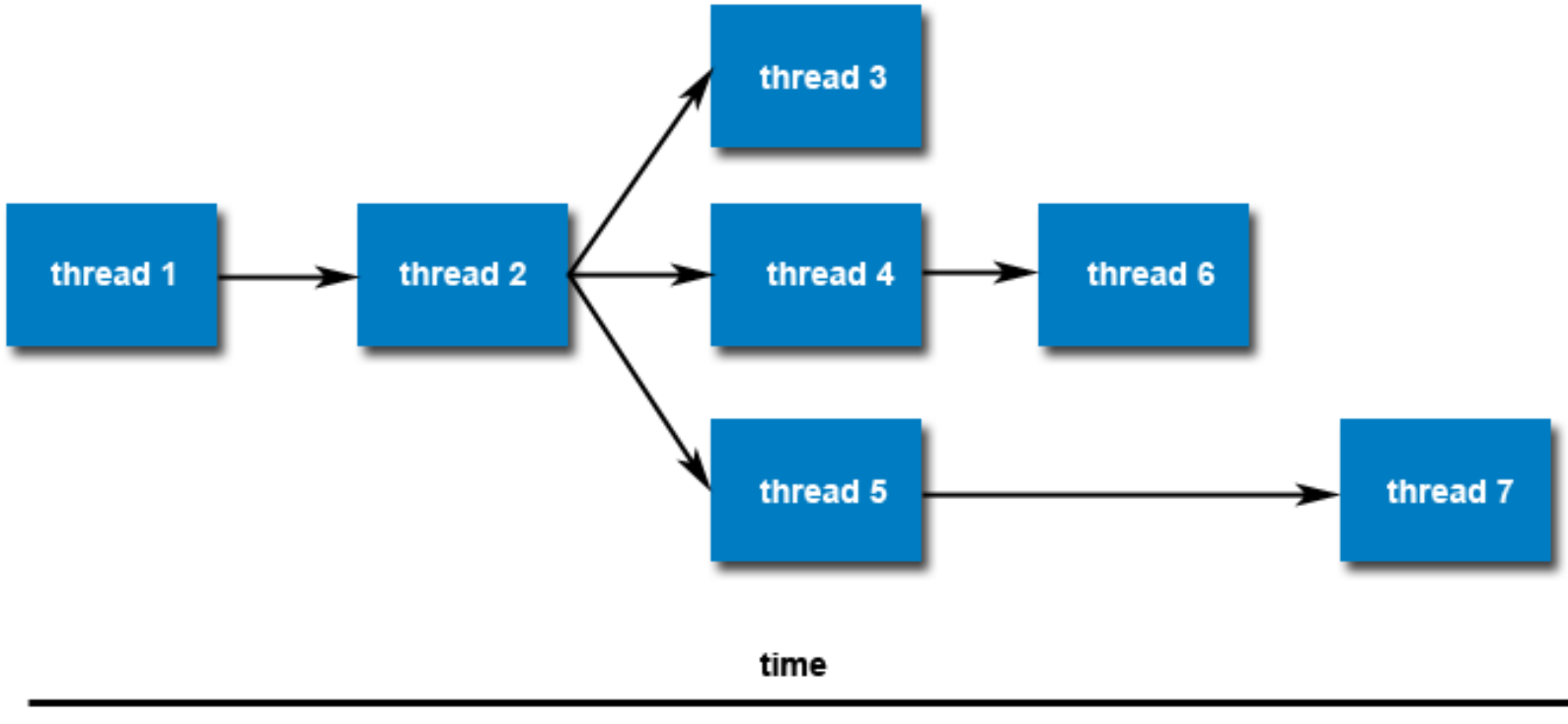
PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

THREADS

O que são Threads?

- Representam uma “divisão” na linha de execução do programa, criando uma linha de execução separada;
- Esta separação na execução do programa permite que dois trechos de códigos sejam executados em “paralelo”.

O que são Threads?



Threads no Android

- No Android, os aplicativos tem uma thread principal, a de interface gráfica da atividade;
- Para executar uma tarefa em segundo plano, é necessário criar uma thread separada
- Por quê?

Thread Principal do Android

- Ao executar uma requisição na thread principal, a interface “trava” até a execução terminar (para o usuário, parece que a aplicação “travou”);
- Para evitar que o usuário fique preso na atividade, o Android impede esse tipo de execução (“X não está respondendo”).

Diálogo “X Não Está Respondendo”

Exemplos de threads

- Exemplos de execuções que precisam de thread?
  - **Comunicação web;**
  - Processamento de dados pesados;
  - Gravação ou leitura de arquivos de texto ou binários com grande volume (arquivos grandes).

Criando uma Thread

- No Android, é possível definir uma “thread” com um componente do sistema chamado **“AsyncTask”** (ou “tarefa assíncrona”);
- Podemos criar uma nova thread através da definição de uma classe que estenda de **“android.os.AsyncTask”.**

AsyncTasks

- As AsyncTasks permitem, então, declarar uma classe que executará uma tarefa (como baixar conteúdo de um serviço da web) em background, sem que a UX seja comprometida;
- É necessário criar uma classe de AsyncTask para cada tarefa que se deseja executar em background.

AsyncTasks

- Normalmente, definimos AsyncTasks como classes privadas de uma classe pública (semelhante a classes anônimas);
- Cada processo que precisa rodar em background terá sua definição de classe privada e sua chamada de execução.

AsyncTasks

- Isto permite que as classes das AsyncTasks possam acessar os elementos da interface da atividade ou do fragmento (o que torna possível, por exemplo, exibir uma barra de progresso enquanto o download é realizado e manipular a interface diretamente quando este for concluído).

Estendendo AsyncTask

- Os métodos mais importantes de AsyncTask a serem implementados são:
  - ***onPreExecute [opcional]****: realiza ações necessárias antes da execução*
  - **doInBackground**: realiza a tarefa em segundo plano
  - **onProgressUpdate**: permite atualizar a interface com o progresso da tarefa
  - **onPostExecute**: realiza as ações necessárias após a execução

Exemplo de thread

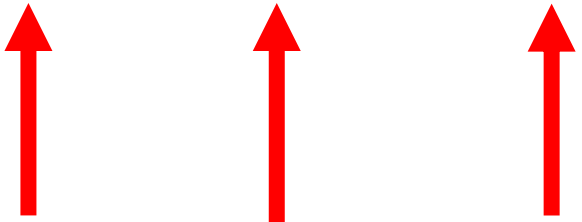
```
/** Task para execução de tarefa em segundo plano */
private class AsyncTaskExample extends AsyncTask<Void, Void, String> {

     /* Realiza uma tarefa em segundo plano.
      * Por exemplo, faz uma requisição web, que pode
      * retornar o conteúdo de uma página, iamgem ou web service. */
     protected Long doInBackground(Void... params) {

          return "RESULTADO";
     }

     /** Callback da task. Chamado sempre que houver atualização de progresso */
     protected void onProgressUpdate(Void... progress) {

     }

     /** Callback da task, chamado quando a tarefa é concluída ou encerrada */
     protected void onPostExecute(String result) {
          /* Aqui, o retorno do método doInBackground é passado como parâmetro.
           * Pode-se realizar tarefas com o resultado, como atualizar um
           * componente visual da tela, por exemplo */
     }
}
```

Especificação de Tipos de Dados

- A classe AsyncTask permite especificar 3 tipos de dados:
- O tipo de dado dos parâmetros de entrada
- O tipo de dado usado para informar o progresso
- O tipo de dado de retorno

Especificação de Tipos de Dados

```
private class AsyncTaskExample extends
      AsyncTask<Void, Void, String> {

}
```



Dado de entrada

Dado de progresso

Dado de retorno

Tipos de Dados do Exemplo

- **Dado de entrada: “Void”**, pois não eram necessários dados de entrada
- **Dado de progresso: “Void”**, pois o progresso também não será informado ao usuário
- **Dado de retorno: “String”**, pois o retorno do método de execução (“doInBackground”) retornará dados neste formado

CARREGANDO IMAGENS DA WEB

Carregando Imagens com AsyncTasks

- Vamos utilizar uma AsyncTask para criar um elemento comum da programação Android utilizado para download de recursos (imagens, texto, JSON de Web Services e etc), chamado de “Network Call”;
- O Network Call é uma AsyncTask comum, tem esta nomenclatura apenas por convenção.

Carregando Imagens com AsyncTasks

- Criaremos um projeto com um atividade;
- Na atividade, haverá também uma classe privada para o Network Call, responsável por baixar uma imagem da internet;
- Ao clicar em um botão, a atividade iniciará o Network Call (iniciando a AsyncTask);
- Quando concluir, o Network Call atualizará a tela com a imagem baixada.

Projeto para Download de Imagens

- Crie um projeto no Android Studio chamado **“ImageDownloader”**;
- O projeto deve seguir o padrão de projetos que criamos até agora:
  - Para celulares e tablets (**“Phone and Tablet”**);
  - Na versão do Android utilizada em seu aparelho (**“6.0 Marshmallow”** caso utilize o Genymotion);
  - Sem nenhuma atividade (**“Add no Activity”**).

Atividade para Download de Imagens

- Crie uma atividade para o projeto (**“Botão Direito” > “New” > “Activity” > “Empty Activity”**);
- Chame-a de **“MainActivity”**;
- Marque as caixas de seleção para criação de layout (deixe-o com o nome padrão, **“activity_main”**) e de atividade da tela inicial (**“Launcher activity”**);

Layout da Atividade

- Altere o layout da atividade para que fique conforme o exemplo a seguir;
- Nomeie os elementos de acordo (ids);

Layout da Atividade

Componentes	IDs
ProgressBar	progressBar
ImageView	image
Button	button

Layout da Atividade

```
...

    <ProgressBar
        android:id="@+id/progressBar"
style="@style/Widget.AppCompat.ProgressBar.Horizontal"
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" android:indeterminate="false" />

    <ImageView
        android:id="@+id/image" android:layout_width="match_parent"
        android:layout_height="wrap_content" android:layout_alignParentStart="true"
        android:layout_below="@+id/progressBar" android:adjustViewBounds="true"
        android:scaleType="fitCenter" app:srcCompat="@drawable/noimage" />

    <Button
        android:id="@+id/button" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true" android:layout_alignParentStart="true"
        android:text="Baixar Imagem" />

...
```

Declarando e Associando Views

- Vamos alterar a classe Java da atividade (“MainActivity”) para que **declare os componentes da interface** necessários (nos atributos da classe) e faça os **bindings** destes com os elementos do XML (dentro do método “onCreate”).

Declarando e Associando Views

```
public class MainActivity extends AppCompatActivity {
    //Declaração dos componentes visuais
    private ProgressBar progressBar;
    private Button button;
    private ImageView image;

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

        //Binding dos componentes visuais
        progressBar = (ProgressBar) findViewById(
                R.id.progressBar);
        button = (Button) findViewById(R.id.button);
        image = (ImageView) findViewById(R.id.image);
    }
}
```

Método Auxiliar para Barra de Progresso

- Crie um método auxiliar para atualizar o valor da barra de progresso (fora do método “onCreate”, mas dentro da atividade “MainActivity”).

Método Auxiliar para Barra de Progresso

```
...
        //Chamado pela AsyncTask
        private void setProgressPercent(int val) {
                //Configura o percentual na barra
                progressBar.setProgress(val);
        }
...
```

Método Auxiliar para Exibição de Diálogo

- Crie um método auxiliar para exibição de uma mensagem por uma caixa de diálogo, conforme exemplo a seguir:

Método Auxiliar para Exibição de Diálogo

```
...
//Método facilitador para exibição de uma mensagem em diálogo
private void showDialog(String message, String title) {

    //Cria uma fábrica de diálogos utilizando a atividade corrente
    AlertDialog.Builder builder = new
        AlertDialog.Builder(MainActivity.this);

    //Configura a mensagem do diálogo como o parâmetro fornecido
    builder.setMessage(message);
    //Configura o título da mensagem
    builder.setTitle(title);
    //Impede o fecamento do diálogo sem o clique em um de seus botões
    builder.setCancelable(false);
    //Configura um botão de OK no diálogo, junto ao seu listener de clique
    builder.setPositiveButton("OK", null);

    //Solicita a criação de uma instância de diálogo à fábrica
    AlertDialog dialog = builder.create();
    //Exibe o diálogo
    dialog.show();
}
...
```

Declaração de um Network Call (AsyncTask)

- Declare uma classe privada no interior da classe “MainActivity”;
- Esta classe será a classe de “Network Call” (uma async task), responsável por executar as tarefas de download em paralelo;
- A classe deve ficar fora de todos os métodos, mas ainda dentro da classe da atividade.

Declaração de um Network Call (AsyncTask)

```
...
//Task para execução de tarefa em segundo plano
private class ImageDownloadNetworkCall extends AsyncTask<URL, Integer, Long> {
    //Cria um atributo que representa a imagem baixada. Será utilizado
    //para configuração no componente imageViewer ao fim do download
    Bitmap bmp = null;

    //Método principal da AsyncTask, executa tarefas em background
    protected Long doInBackground(URL... urls) {
       //Código executado em background aqui
       return null;
    }

    //Método de callback da task. Executado para atualização de progresso
    protected void onProgressUpdate(Integer... progress) {
       //Código de atualização do progresso aqui
    }

    //Método de callback para quando a tarefa é concluída ou encerrada
    protected void onPostExecute(Long result) {
        //Código para quando a tarefa acabar aqui
    }
}
...
```

Implementação do Network Call

- No método “doInBackground”, faremos o download da imagem;
- Este método será executado em paralelo, até que o download da imagem acabe;
- Quando encerrar, o método “onPostExecute” será chamado;
- O Android pode chamar o método “onProgressUpdate”, para atualizar o usuário (interface) do progresso.

Bitmap

- Em primeiro lugar, na classe “Network Call”, declare um elemento do tipo “Bitmap” (imagem);
- Este elemento será utilizado para salvar a imagem no método “doInBackground” posteriormente.

Bitmap

```
...
//Task para execução de tarefa em segundo plano
private class ImageDownloadNetworkCall extends AsyncTask<URL, Integer, Long> {
    //Cria um atributo que representa a imagem baixada. Será utilizado
    //para configuração no componente imageViewer ao fim do download
    Bitmap bmp = null;

    //Método principal da AsyncTask, executa tarefas em background
    protected Long doInBackground(URL... urls) {
       //Código executado em background aqui
       return null;
    }

    //Método de callback da task. Executado para atualização de progresso
    protected void onProgressUpdate(Integer... progress) {
       //Código de atualização do progresso aqui
    }

    //Método de callback para quando a tarefa é concluída ou encerrada
    protected void onPostExecute(Long result) {
        //Código para quando a tarefa acabar aqui
    }
}
...
```

Bytes Baixados e Bytes da Imagem

- Dentro do método “doInBackground”, declare e inicialize dois elementos, um inteiro (“totalSize”) para armazenar a quantidade de bytes baixados e um elemento do tipo “InputStream”, responsável por armazenar os bytes da imagem baixados do servidor.

Bytes Baixados e Bytes da Imagem

```
//Método principal da AsyncTask, executa tarefas em background
protected Long doInBackground(URL... urls) {

    //Inicializa o total de bytes baixados com 0
    long totalSize = 0;

    //Declara um InputStream para receber os bits da imagem
    InputStream is = null;

    return null;
}
```

Baixando a Imagem

- Faça o download da imagem, através da abertura da conexão da primeira URL fornecida como parâmetro;
- É necessário utilizar um try/catch para tratar possíveis erros (operação de I/O);
- Coloque o código logo abaixo da declaração de variáveis de “doInBackground”.

Baixando a Imagem

```
...
//try/catch é uma exigência do método openStream da URL
try {
    //Obtém um elemento de conexão com a URL da imagem
    //(sempre o primeiro parâmetro de URL, temos apenas 1 imagem)
    URLConnection connection = urls[0].openConnection();

    //Especifica um timeout caso não consiga baixar o conteúdo
    connection.setConnectTimeout(4000);

    //EFETUA O DOWNLOAD DA IMAGEM
    is = connection.getInputStream();

} catch (Exception e) {
    //Exibe mensagem de erro no console caso algum erro ocorra
    e.printStackTrace();
}
...
```

Convertendo Bytes em Imagem

- Após a obtenção dos bytes da imagem, vamos transformá-los em uma imagem propriamente dita (objeto da classe Bitmap);
- Faça conforme o exemplo a seguir:

Convertendo Bytes em Imagem

```
...
//EFETUA O DOWNLOAD DA IMAGEM
in = connection.getInputStream();

//Transforma os bytes em imagem
bmp = BitmapFactory.decodeStream(is);
//Obtém o tamanho (em bytes) da imagem
totalSize += bmp.getByteCount();

//Atualiza o progresso para 100%
publishProgress(100);
...
```

Retornando a Imagem

- Por fim, retorne o tamanho total da imagem (retorno do método) para que este valor seja enviado para o método “onPostExecute”;
- Como salvamos o valor do Bitmap numa variável da classe, não é necessário retornar este valor (estará disponível para o método “onPostExecute”).

Retornando a Imagem

```
...
//Retorna o tamanho da imagem (em bytes)
return totalSize;
...
```

Informando o Progresso

- No método “onProgressUpdate”, chame o método auxiliar criado na atividade (“setProgressPercent”), para atualizar a barra de progresso com o valor adequado.

Informando o Progresso

```
//Método de callback da task. Executado sempre que houver
atualização de progresso
protected void onProgressUpdate(Integer... progress) {

    //Chama o método da atividade para atualizar o progresso
    setProgressPercent(progress[0]);

}
```

Tratando o Resultado da Execução

- No método “onPostExecute”, vamos tratar a conclusão do Network Call;
- Ou seja, vamos obter o bitmap e atribuí-lo ao ImageView, exibindo uma mensagem em seguida.

Tratando o Resultado da Execução

```
//Método de callback da task, chamado quando a tarefa é concluída ou
encerrada
protected void onPostExecute(Long result) {

    //Verifica se a imagem foi baixada
    if (bmp != null) {

        //Configura a imagem para visualização no componente image viewer
        image.setImageBitmap(bmp);

        //Mostra um diálogo com o tamanho total do download
        showDialog(result + " bytes foram baixados", "Tarefa Concluída");

    }
    else {

        //Mostra um diálogo de erro caso a imagem não tenha sido baixada
        showDialog("Não foi possível baixar a imagem", "Erro");

    }
}
```

Iniciando a AsyncTask de Network Call

- Por fim, logo abaixo dos bindings dos componentes (método “onCreate”) vamos inicializar a barra de progresso e, em seguida, declarar um listener para o botão capaz de iniciar a Network Call (como uma AsyncTask);
- Especifique uma URL de imagem válida para efetuar o download e exibi-la na tela.

Iniciando a AsyncTask de Network Call

```
//Configuração inicial da barra de progresso
progressBar.setMax(100);
progressBar.setProgress(0);

//Tratamento do evento de clique no botão
button.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {

        //Cria uma nova instância da thread
        ImageDownloadNetworkCall downloadTask = new
                ImageDownloadNetworkCall();

        //try/catch é uma exigência do comando de criação da URL
        try {
            //Reinicializa a barra de progresso (caso já tenha completado)
            progressBar.setProgress(0);
            //Executa a tarefa (thread)
            downloadTask.execute(new URL("http://www.sp.senac.br/moldura/
                images/senac_logo.png"));
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
});
```

Permissão de Acesso a Internet

- Para finalizar e permitir o download da imagem, é necessário solicitar permissão de acesso a internet ao usuário;
- Adicione a configuração a seguir no arquivo AndroidManifest.xml

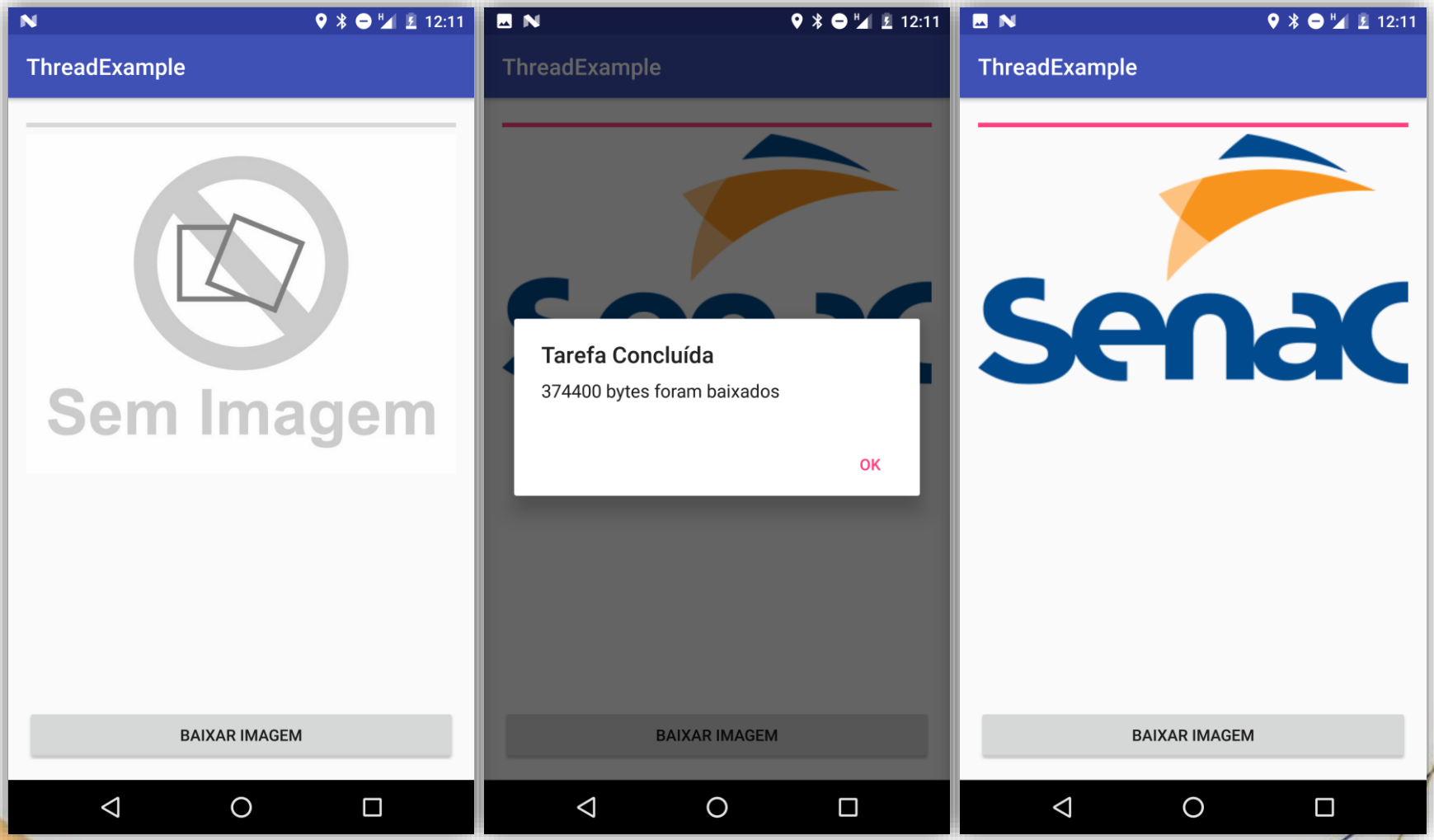
Arquivo AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.example.fabio.threadexample">

    <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>

    <!-- ADICIONE ISSO AQUI! -->
    <uses-permission android:name="android.permission.INTERNET"/>

</manifest>
```

Execute a Aplicação

- Execute a Aplicação para ver o resultado.

Execute a Aplicação



EXERCÍCIO

Exercício 01

- Modifique o projeto de download de imagens para que seja capaz de salvar 3 imagens, todas com URLs fornecidas pelo usuário através de campos de texto.

ADO 01

http://hipsters.tech/desenvolvimento-android/

Ouvir o podcast (~1 hora) e responder um questionário no início da próxima aula (12/04/2018)


"That's all Folks!"