ANDROID

Tarefas Assíncronas

Conteúdo

- Main Thread
- Threads
- AsyncTask

Main Thread

- A aplicação roda em uma thread principal
  - *Main Thread*, *UI Thread*
  - Gerencia eventos
  - Atualiza a interface gráfica
- Se uma tarefa demorada precisa ser executada
  - Ex.: Web Service, Consulta ao BD, Acesso à lista de contatos, leitura de arquivo
  - A interface para de responder eventos (ex, toques do usuário)
  - Se demorar mais de 5s, recebe um erro ANR – *Application Not Responding*
    - Alerta com Force Close
    - Sistema fecha a aplicação
- Atualmente, operações de I/O devem obrigatoriamente ser efetuadas em outras threads
- Para executar uma tarefa demorada
  - Deve-se utilizar outras threads

Threads

Threads

- Para executar um trecho de código em outra thread usa-se Thread
- Exemplo:

```
new Thread() {
   public void run() {
      // Aqui código em segundo plano
   };
}.start();
```

Threads – Atualizar UI

- O Android não permite que outra thread atualize a interface gráfica
- Usa-se **Handler** para tal
- Exemplo:

```
final Handler handler = new Handler();
new Thread() {
   public void run() {
      // Aqui código em segundo plano
      handler.post(new Runnable() {
         public void run() {
            // Atualização da interface gráfica
         }
      });
   };
}.start();
```

Threads – Atualizar UI

- Também pode-se utilizar o método `runOnUIThread()`
- Exemplo:

```
new Thread() {
    public void run() {
        // Aqui código em segundo plano
        runOnUiThread(new Runnable() {
            public void run() {
                // Atualização da interface gráfica
            }
        });
    };
}.start();
```

Agendamento de Tarefas

- Com o Handler pode-se agendar uma tarefa para rodar em um determinado tempo
- Métodos
  - `postDelayed(Runnable, milisegundos)`
  - Runnable contém o método `run()` com o código a ser executado
  - Milisegundos é o tempo de espera para o código rodar

```
handler.postDelayed(
      new Runnable() {
           public void run() {
                // Código a ser executado
           }
      }, 2000);  // Espera 2 segundos
```

Splash Screen

- Uma tela que aparece no início da aplicação
- Depois de um determinado tempo, a tela desaparece e a aplicação é apresentada
- A aplicação pode fazer alguma tarefa durante este tempo
- Para implementar:
  - Implementar a Splash (XML e Activity)
  - Fazer a Splash ser a primeira tela a ser apresentada
  - Depois de um determinado tempo (agendamento de tarefa)
    - Apresenta a tela principal
    - Finaliza a Splash

Tela - activity_splash_screen.xml

```
<FrameLayout
   xmlns:android="http://schemas.android.con/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <ImageView
      android:src="@drawable/imagem"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"/>
</FrameLayout>
```

Tela - SplashScreenActivity

```
public class SplashScreenActivity extends Activity {
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_splash_screen);
      // Em 1 segundo: Inicia MainActivity e Finaliza Splash
      new Handler().postDelayed(new Runnable() {
          public void run() {
               // Inicia a MainActivity
               startActivity(new Intent(getBaseContext(), MainActivity.class));
               finish(); // Fecha Splash
          }
      }, 1000); // Espera 1 segundo (1000 ms)
   }
}
```

Tela - Manifest

- Splash é a primeira Activity, a MainActivity deve ser colocada como uma activity normal

```
<manifest ...>
   <uses-permission android:nane="android.permission.INTERNET" />
   <application ... >
      <activity android:nane=".SplashScreenActivity" >
         <intent-filter>
            <action android:nane="android.intent.action.HAIN" />
            <category android:nane="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".MainActivity" android:label="@string/app_nane" />
       ...
   </application>
</manifest>
```

AsyncTask

AsyncTask

- Maneira mais fácil de implementar tarefas concorrentes
- Link: https://developer.android.com/reference/android/os/AsyncTask.html
- AsyncTask :
  - Interface simples
  - Gerencia as Threads e Handlers necessários
  - Uma tarefa pode ser cancelada
  - Métodos para atualizar o progresso da tarefa
  - Pool de threads para executar em série ou paralelo

AsyncTask

- Classe que herda de **`AsyncTask<Params, Progresso, Resultado>`**
  - No nosso exemplo, uma tarefa que faz download de arquivs
  - **Param** : Ex: URL: tipo dos parâmetros passados para a tarefa
  - **Progresso**: Ex: Integer: tipo das unidades de progresso
  - **Resultado**: Ex: Long: tipo do resultado final da computação
- 4 passos, mapeados em métodos
  - **`onPreExecute()`** : invocado pela Main Thread antes da execução da tarefa
  - **`doInBackground(Params...)`** : invocado em segundo plano, logo depois de **`onPreExecute()`**. Pode invocar **`publishProgress(Progresso...)`** para atualizar a visão na Main Thread
  - **`onProgressUpdate(Progresso...)`** : invocado na Main Thread logo depois de **`publishProgress(Progresso...)`**.
  - **`onPostExecute(Resultado)`** : invocado na Main Thread depois que o processamento em segundo plano termina

AsyncTask - Exemplo

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected void onPreExecute() {
        // Executa na Main Thread
        // Executa antes do processo em segundo plano
    }
    protected Long doInBackground(URL... urls) {
        // Executa em segundo plano
        // O retorno do tipo "Long" é passado ao método onPostExecute()
        // Pode chamar publishProgress(Integer) para atualizar o progresso
        return 1L;
    }
    protected void onProgressUpdate(Integer... progress) {
        // Pode atualizar o progresso da tarefa
        // Valor recebido da chamada do método publishProgress(int)
    }
    protected void onPostExecute(Long result) {
        // Recebe o resultado do método doInBackground()
        // Executa na Main Thread e pode atualizar a view
    }
}
```

AsyncTask – Invocação

- Duas formas de invocar
  - `new DownloadFilesTask().execute(url);`
    - Inicia a tarefa passando uma URL como parâmetro
  - `new DownloadFilesTask().execute(url1, url2, url3);`
    - Inicia a tarefa passando várias URLs como parâmetro
- A implementação de `doInBackgroud(URL...)` deve tratar os vários parâmetros

AsyncTask – Implementação

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            // Atualiza o progresso do download em porcentagem
            publishProgress((int) ((i / (float) count) * 100));
            // Se cancel() foi chamado, termina a tarefa
            if (isCancelled())
                break;
        }
        return totalSize;
    }
    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

AsyncTask – Invocação

- Pode-se deixar parâmetros como **Void**, indicando que não são usados

```
private class DownloadTask extends AsyncTask<Void, Void, Bitmap> {
```

  - Indica que não há parâmetros a serem passados na criação da tarefa:

```
new DownloadTask().execute();
```

  - Indica que não há atualização de progresso
  - Indica que o retorno é um Bitmap

AsyncTask – Implementação

```
private class DownloadTask extends AsyncTask<Void,Void,Bitmap> {
   protected void onPreExecute() {
      super.onPreExecute();
      progress.setVisibility(View.VISIBLE);
   }
   protected Bitmap doInBackground(Void... params) {
      // Faz o download em uma thread e retorna o bitmap.
      try {
          bitmap = Download.downloadBitmap(URL);
      }
      catch (Exception e) {
          Log.e("Download ", e.getMessage(), e);
      }
      return bitmap;
   }
   protected void onPostExecute(Bitmap bitmap) {
      if (bitmap != null) {
         // Atualiza a imagem na UI Thread
         imgview.setImageBitmap(bitmap);
         // Esconde o progress
         progress.setVisibility(View.INVISIBLE);
      }
   }
}
```