**PRACTICAL 15**

**OBJECT:**

Working with AsyncTask and AsyncTaskLoader

**THEORY:**

To perform a background operation on a thread in Android, you can use AsyncTask class. AsyncTask facilitates you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask needs to be subclassed to be implemented. The subclass will override the required methods to perform the background tasks.

Listing 1:

*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]);*

*publishProgress((int) ((i / (float) count) \* 100));*

*// Escape early if cancel() is called*

*if (isCancelled()) break;*

*}*

*return totalSize;*

*}*

*protected void onProgressUpdate(Integer... progress) {*

*setProgressPercent(progress[0]);*

*}*

*protected void onPostExecute(Long result) {*

*showDialog("Downloaded " + result + " bytes");*

*}*

*}*

To executed the task:

*new DownloadFilesTask().execute(url1, url2, url3);*

**AsyncTask's generic types**

The three types used by an asynchronous task are the following:

Params, the type of the parameters sent to the task upon execution.

Progress, the type of the progress units published during the background computation.

Result, the type of the result of the background computation.

Not all types are used by an asynchronous task. To mark a type as unused, simply use the type Void:

private class MyTask extends AsyncTask<Void, Void, Void> { ... }

When an asynchronous task is executed, the task goes through 4 steps:

onPreExecute(), invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.

doInBackground(Params...), invoked on the background thread immediately after onPreExecute() finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use publishProgress(Progress...) to publish one or more units of progress. These values are published on the UI thread, in the onProgressUpdate(Progress...) step.

onProgressUpdate(Progress...), invoked on the UI thread after a call to publishProgress(Progress...). The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.

onPostExecute(Result), invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Cancelling a task

A task can be cancelled at any time by invoking cancel(boolean). Invoking this method will cause subsequent calls to isCancelled() to return true. After invoking this method, onCancelled(java.lang.Object), instead of onPostExecute(java.lang.Object) will be invoked after doInBackground(java.lang.Object[]) returns. To ensure that a task is cancelled as quickly as possible, you should always check the return value of isCancelled() periodically from doInBackground(java.lang.Object[]), if possible (inside a loop for instance.)

Threading rules

There are a few threading rules that must be followed for this class to work properly:

The AsyncTask class must be loaded on the UI thread. This is done automatically as of Build.VERSION\_CODES.JELLY\_BEAN.

The task instance must be created on the UI thread.

execute(Params...) must be invoked on the UI thread.

Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually.

The task can be executed only once (an exception will be thrown if a second execution is attempted.)

Memory observability

AsyncTask guarantees that all callback calls are synchronized to ensure the following without explicit synchronizations.

The memory effects of onPreExecute(), and anything else executed before the call to execute(Params...), including the construction of the AsyncTask object, are visible to doInBackground(Params...).

The memory effects of doInBackground(Params...) are visible to onPostExecute(Result).

Any memory effects of doInBackground(Params...) preceding a call to publishProgress(Progress...) are visible to the corresponding onProgressUpdate(Progress...) call. (But doInBackground(Params...) continues to run, and care needs to be taken that later updates in doInBackground(Params...) do not interfere with an in-progress onProgressUpdate(Progress...) call.)

Any memory effects preceding a call to cancel(boolean) are visible after a call to isCancelled() that returns true as a result, or during and after a resulting call to onCancelled().

Order of execution

When first introduced, AsyncTasks were executed serially on a single background thread. Starting with Build.VERSION\_CODES.DONUT, this was changed to a pool of threads allowing multiple tasks to operate in parallel. Starting with Build.VERSION\_CODES.HONEYCOMB, tasks are executed on a single thread to avoid common application errors caused by parallel execution.

If you truly want parallel execution, you can invoke executeOnExecutor(java.util.concurrent.Executor, java.lang.Object[]) with THREAD\_POOL\_EXECUTOR.

Full Example:

To download an image file in background, we will use AsyncTask:

Step 1 − Create a new project in Android Studio, go to File ⇒ New Project and fill all required details to create a new project.

Step 2 − Add the following code to res/layout/activity\_main.xml.

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

*<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"*

*xmlns:tools = "http://schemas.android.com/tools"*

*android:id = "@+id/rootview"*

*android:layout\_width = "match\_parent"*

*android:layout\_height = "match\_parent"*

*android:orientation = "vertical"*

*android:background = "#c1c1c1"*

*android:gravity = "center\_horizontal"*

*tools:context = ".MainActivity">*

*<Button*

*android:id = "@+id/asyncTask"*

*android:text = "Download"*

*android:layout\_width = "wrap\_content"*

*android:layout\_height = "wrap\_content" />*

*<ImageView*

*android:id = "@+id/image"*

*android:layout\_width = "300dp"*

*android:layout\_height = "300dp" />*

*</LinearLayout>*

In the above xml we have created a button, when user click on the button it going to download image and append image to imageview.

Step 3 − Add the following code to src/MainActivity.java

*import android.app.ProgressDialog;*

*import android.graphics.Bitmap;*

*import android.graphics.BitmapFactory;*

*import android.os.AsyncTask;*

*import android.os.Bundle;*

*import android.support.v7.app.AppCompatActivity;*

*import android.view.View;*

*import android.widget.Button;*

*import android.widget.ImageView;*

*import java.io.IOException;*

*import java.io.InputStream;*

*import java.net.HttpURLConnection;*

*import java.net.URL;*

*public class MainActivity extends AppCompatActivity {*

*URL ImageUrl = null;*

*InputStream is = null;*

*Bitmap bmImg = null;*

*ImageView imageView= null;*

*ProgressDialog p;*

*@Override*

*protected void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*setContentView(R.layout.activity\_main);*

*Button button=findViewById(R.id.asyncTask);*

*imageView=findViewById(R.id.image);*

*button.setOnClickListener(new View.OnClickListener() {*

*@Override*

*public void onClick(View v) {*

*AsyncTaskExample asyncTask=new AsyncTaskExample();*

*asyncTask.execute("https://www.tutorialspoint.com/images/tp-logo-diamond.png");*

*}*

*});*

*}*

*private class AsyncTaskExample extends AsyncTask<String, String, Bitmap> {*

*@Override*

*protected void onPreExecute() {*

*super.onPreExecute();*

*p = new ProgressDialog(MainActivity.this);*

*p.setMessage("Please wait...It is downloading");*

*p.setIndeterminate(false);*

*p.setCancelable(false);*

*p.show();*

*}*

*@Override*

*protected Bitmap doInBackground(String... strings) {*

*try {*

*ImageUrl = new URL(strings[0]);*

*HttpURLConnection conn = (HttpURLConnection) ImageUrl.openConnection();*

*conn.setDoInput(true);*

*conn.connect();*

*is = conn.getInputStream();*

*BitmapFactory.Options options = new BitmapFactory.Options();*

*options.inPreferredConfig = Bitmap.Config.RGB\_565;*

*bmImg = BitmapFactory.decodeStream(is, null, options);*

*} catch (IOException e) {*

*e.printStackTrace();*

*}*

*return bmImg;*

*}*

*@Override*

*protected void onPostExecute(Bitmap bitmap) {*

*super.onPostExecute(bitmap);*

*if(imageView!=null) {*

*p.hide();*

*imageView.setImageBitmap(bitmap);*

*}else {*

*p.show();*

*}*

*}*

*}*

*}*

In the above code we are downloading image using asyncTask and appending image to imageview.

Step 4 − Add the following code to manifest.xml

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

*<manifest xmlns:android = "http://schemas.android.com/apk/res/android"*

*package = "com.example.andy.myapplication">*

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

*<application*

*android:allowBackup = "true"*

*android:icon = "@mipmap/ic\_launcher"*

*android:label = "@string/app\_name"*

*android:roundIcon = "@mipmap/ic\_launcher\_round"*

*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>*

In the above AndroidManifest.xml file we have added internet permission to access internet to download image.

Run the application, press the download button and wait for the background task to complete downloading file.

**ACTIVITIES**

**Activity 1**

Create

**REVIEW QUESTIONS**

1. What
2. What
3. Which
4. Why