

MODULE 12

ASYNCHRONOUS TASK

TABLE OF CONTENT

- Asynchronous Task
- AsyncTask Class
- AsyncTask Steps
- AsyncTask Generic Types
- Using AsyncTask Class
- Cancelling an Asynchronous Task
- Webography

Asynchronous Task

- ▶ To perform background operations
 - Computation that runs on a background thread
- ▶ And then to publish results on the UI thread

AsyncTask Class

- ▶ Used to perform asynchronous work
 - Performs the blocking operations in a worker thread
 - Then publishes the results on the UI thread
 - Without requiring to handle threads and/or handlers
- ▶ Enables proper and easy use of the UI thread
 - The **UI thread is not blocked** waiting for the long run task to be finished
- ▶ Should ideally be used for short operations (a few seconds)

AsyncTask Steps

- ▶ Step 1: **onPreExecute ()**
 - Invoked on the **UI thread** before the task is executed
 - Normally used to setup the task
 - E.g, showing a progress bar in the user interface

AsyncTask Steps

- ▶ Step 2 : **doInBackground (Params...)**
 - Invoked on the **background thread**
 - Immediately after *onPreExecute()* finishes executing
 - To perform 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**
 - **And will be passed back to *onPostExecute(Result)***
 - 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

AsyncTask Steps

- ▶ Steps 3: **onProgressUpdate(Progress...)**
 - Invoked on the **UI thread**
 - After a call to *publishProgress(Progress...)*
 - The timing of the execution is undefined
 - Used to display any form of progress in the user interface
 - While the background computation is still executing
 - E.g, to animate a progress bar or to show logs in a text field

AsyncTask Steps

- ▶ Steps 4: **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

AsyncTask Steps

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

AsyncTask Generic Types

- ▶ 3 generic types
 - **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 used by an asynchronous task
 - To mark a type as unused: use the type **Void**

Using AsyncTask Class

- ▶ Create a subclass of AsyncTask class
 - Override at least **doInBackground(Params...)** method
 - Runs in a pool of background threads
 - **Result** will be automatically passed to onPostExecute method
 - Override most often **onPostExecute(**Result**)** method
 - Delivers the result from doInBackground()
 - Runs in the UI thread ⇒ to update UI

Using AsyncTask Class

- ▶ Run the task
 - Create an instance of the AsyncTask subclass in the UI thread
 - Call **execute(Params...)** on this instance from the UI thread
 - The task can be executed only once
 - An exception will be thrown if a second execution is attempted

Using AsyncTask Class

► E.g,

```
public class MainActivity extends Activity {  
  
    private TextView text;  
    private Button button;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        text = (TextView) this.findViewById(R.id.editTextID);  
        button = (Button) this.findViewById(R.id.buttonID);  
        button.setOnClickListener(new OnClickListener()  
        { public void onClick (View arg0)  
        {  
            String url1 = "...";  
            String url2 = "...";  
            String url3 = "...";  
            new MyAsyncTask().execute(url1, url2, url3);  
        }  
        });  
    }  
}
```

UI to be updated after async task is finished

Run the AsyncTask

Variable number of arguments

Using AsyncTask Class

Type of params Type of result

```
private class MyAsyncTask extends AsyncTask<String, Void, String> {  
    protected String doInBackground(String... urls) {  
        int count = urls.length;  
        int totalLength = 0;  
        try { for (int i = 0; i < count; i++)  
            { URL url = new URL(urls[i]);  
              URLConnection connection = url.openConnection();  
              connection.connect();  
              totalLength = connection.getContentLength();  
            }  
        }  
        catch (Exception e)  
        { Log.i("Exception: ", e.getMessage());  
        }  
        return "Total length: " + totalLength;  
    }  
    protected void onPostExecute(String result) {  
        text.setText(result);  
    }  
}
```

Using AsyncTask Class

```
private class MyAsyncTask extends AsyncTask<String, Void, String> {  
  
    protected String doInBackground(String... urls) {  
        int count = urls.length;  
        int totalLength = 0;  
        try { for (int i = 0; i < count; i++)  
            { URL url = new URL(urls[i]);  
              URLConnection connection = url.openConnection();  
              connection.connect();  
              totalLength = connection.getContentLength();  
            }  
        }  
        catch (Exception e)  
        { Log.i("Exception: ", e.getMessage());  
        }  
        return "Total length: " + totalLength;  
    }  
  
    protected void onPostExecute(String result) {  
        text.setText(result);  
    }  
}
```

Variable arguments

Update of the UI

Using AsyncTask Class

- ▶ All callback calls are synchronized

↳ The following operations are safe

- Set member fields in the *constructor* or *onPreExecute()* and refer to them in *doInBackground(Params...)*
- Set member fields in *doInBackground(Params...)* and refer to them in
onProgressUpdate(Progress...)
onPostExecute(Result)

Cancelling an Asynchronous Task

- ▶ By invoking **cancel(boolean)**
- ▶ Will cause subsequent calls to **isCancelled()** to return true
- ▶ After *doInBackground* returns
 - **onCancelled(Object)** will be invoked
 - instead of *onPostExecute(Object)*

Webography

- ▶ <http://developer.android.com/reference/android/os/AsyncTask.html>
- ▶ <http://developer.android.com/guide/components/processes-and-threads.html>