AsyncTask & Notifications

In this Lecture, you will learn:

- Showing Notifications
- Opening Activity on Notification Tap
- Notifications

AsyncTask

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.

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.)

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.

AsyncTask class example:

```
private class DownloadAsyncTask extends AsyncTask<String,
String, Bitmap> {
    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        p = new ProgressDialog(MainActivity.this);
        p.setMessage("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);
        notifyDownload();
        if(imageView!=null) {
            p.hide();
            imageView.setImageBitmap(bitmap);
        }else {
            p.show();
        }
    }
}
```

Use the class as:

```
DownloadAsyncTask asyncTask=new DownloadAsyncTask();
asyncTask.execute("https://homepages.cae.wisc.edu/~ece533/
images/fruits.png");
```

Here the inherited AsyncTask class is instantiated. The execute method takes the url of file to download. It downloads the contents of the file in background.

Notifications

Notifications show short information about events in your app while it may not be in use. A notification is a message for the user that Android displays outside your app's UI. Users can tap the notification to open your app or take an action directly from the notification.

Notifications appear to users in different locations and formats, including an icon in the status bar, a more detailed entry in the notification drawer.

We will use NotificationCompat APIs from the Android support library.

Create and Use Notifications

1) Create Notification Builder

```
NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this, "11");
```

2) Setting Notification Properties

```
builder.setSmallIcon(R.drawable.baseline_notifications_
active_black_18dp)
      .setContentTitle("Demo Notification")
      .setContentText("This is demo message for
notification");
```

3) Attach Actions

Optional part if you want to attach an action with the notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events.

```
Intent notificationIntent = new Intent(MainActivity.this,
NotificationViewActivity.class);
```

```
notificationIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
//notification message will get at NotificationView
notificationIntent.putExtra("message", "This is a notification
message");
PendingIntent pendingIntent = PendingIntent.getActivity(this,
0, notificationIntent,
      PendingIntent.FLAG_UPDATE_CURRENT);
builder.setContentIntent(pendingIntent);
```

4) Issue the notification

Start the notification by calling NotificationManager.notify() to send your notification. Make sure you call NotificationCompat.Builder.build() method on builder object before notifying it. This method combines all of the options that have been set and return a new Notification object.

```
mNotificationManager.notify(0, builder.build());
```