AsyncTask & Notifications

In this Lecture, you will learn:

- AsyncTask
- Showing Notifications
- Opening Activity on Notification Tap

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: type of the parameters sent to the task upon execution.

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

Result: 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> { ... }
```

Asynchronous execution goes through 4 steps:

onPreExecute(): invoked on the UI thread before the task is executed. This is used to setup the task, like showing a progress bar.

doInBackground(Params…): invoked on the background thread immediately after onPreExecute() finishes executing. Here background computation is performed that can take a long time. The parameters of the asynchronous task are passed in this method. The result of the computation returned from this method are 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...). 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 computation or task is passed to this step as a parameter.

Cancelling a task

A task can be cancelled at any time by invoking cancel(boolean). After invoking this method, onCancelled(java.lang.Object), instead of onPostExecute(java.lang.Object) will be invoked after doInBackground(java.lang.Object[]) returns.

Threading rules that must be followed for this class to work properly:

The AsyncTask class must be loaded on the UI thread. The task instance must be created on the UI thread. execute(Params...) must be invoked on the UI thread.

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

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

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.

AsyncTask class example:

```
private class DownloadAsyncTask extends AsyncTask<String, String, Bitmap>{
    private ProgressDialog progressDialog;
    protected void onPreExecute() {
        super.onPreExecute();
        progressDialog = new ProgressDialog(MainActivity.this);
        progressDialog.setMessage("Downloading...");
        progressDialog.setCancelable(false);
        progressDialog.show();
    }
    @Override
    protected Bitmap doInBackground(String... strings) {
        InputStream inputStream;
        Bitmap bitmapImage = null;
        try {
            URL ImageUrl = new URL(strings[0]);
            HttpURLConnection conn = (HttpURLConnection)
                                        ImageUrl.openConnection();
            conn.setDoInput(true);
            conn.connect();
            inputStream = conn.getInputStream();
            BitmapFactory.Options options = new BitmapFactory.Options();
            options.inPreferredConfig = Bitmap.Config.RGB_565;
            bitmapImage = BitmapFactory.decodeStream(inputStream, null,
                              options);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return bitmapImage;
    }
    @Override
    protected void onPostExecute(Bitmap bitmap) {
        super.onPostExecute(bitmap);
        progressDialog.hide();
        mImageView.setImageBitmap(bitmap);
        showNotification();
    }
}
```

After defining the class, use the class instance 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 image file to download. It downloads the contents of the file in background and sets the imageView source after image is downloaded.

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());
```