KCL Tech Build X: Android

Lecture Eight: Notifications and Wrap-up

- Creating Notifications
- Adding Actions to Notifications
- Receiving Push Notifications
- Next Steps

Creating Notifications

All apps can **create their own notifications**, which can be used to tell the user about an event (a message arriving, a task being due, etc.) or an ongoing process (a file uploading, music being played, etc.).

We'll dive right into coding: the first task is to **create the notification object** that we're going to send to the user interface. Creating a notification exploits the **builder pattern**, which allows different parameters to be set one at a time, instead of creating them with a huge constructor, or user-implementable interface (both of which would be more complex and harder to code).

We start be creating a builder object, from the `NotificationCompat` class:

```
NotificationCompat.Builder builder =
        new NotificationCompat.Builder(<< activity name >>.this);
```

Now that we have the `builder` object, we can use it to set individual parameters, one at a time:

```
NotificationCompat.Builder builder =
        new NotificationCompat.Builder(<< activity name >>.this);

builder.setContentTitle("This is the notification title");
builder.setContentText("This is the notification text");
builder.setSmallIcon(R.drawable.icon_name);
builder.setLargeIcon(
    BitmapFactory.decodeResource(getResources(), R.drawable.icon_name)
);
builder.setAutoCancel(true);
builder.setVibrate(new long[]{0, 500, 200});
builder.setLights(0xff00ff00, 500, 500);
```

Most of these are **optional**, and there are many more properties that can be set to further customise the notification that is sent to users. Note that to make the device vibrate, your app needs the `VIBRATE` permission; look in `AndroidManifest.xml` to see how to get it.

Now that we've created our notification, we need an instance of Android's `NotificationManager` to send it to users:

```
NotificationManager manager =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
```

With access to this manager, we are finally ready to start sending notifications! Each notification requires a **unique ID** (unique to *your app*, not across the whole system). We're going to cheat a little and just create an integer at the top of our class, and increment with each notification.

```
/* at the top of the class */
int notificationId = 0;
```

```
/* after we create our notification */

// increment the notification ID
++notificationId;

// get the notification manager (from last step)
NotificationManager manager =
        (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

// launch the notification
manager.notify(notificationId, builder.build());
```

Done! That notification will appear to users as soon as `manager.notify(...)` is called!

Adding Actions to Notifications

When we last looked at actions, we used an `Intent` to go to a new activity when a user clicked a button. We can do almost the same when they click on a notification. There is just **one key difference**: an `Intent` is designed to be executed immediately, whereas a notification may be clicked any time after it is created (the app that created the notification may no longer be open, and it may not have been open in the first place). Because of this potential delay, we need to use a `PendingIntent`, which "wraps around" a regular `Intent`.

We start by making the regular `Intent`. In this case, we're going to forward the user to a new activity, and send a message along in the intent extras:

```
// make the intent
Intent intent =
   new Intent (<< activity name >>.this, NotificationClickedActivity.class);

// add the message
intent.putExtras("message", "You clicked notification #" + notificationId);
```

Now, we need to "wrap" this inside a `PendingIntent`:

```
// make the pending intent
PendingIntent pendingIntent = PendingIntent.getActivity(
        << activity name >>.this,
        notificationId,
        intent,   // <--- the intent we created earlier
        PendingIntent.FLAG_CANCEL_CURRENT
);
```

Finally, we have to set this as the action for our notification, which we can do using the builder we defined before:

```
builder.setContentIntent(pendingIntent);
```

Try now, and you'll see that clicking your notification takes you to the other activity. This will work even if the original activity has been closed.

Receiving Push Notifications

This section will be covered as an interactive example, rather than a straightforward tutorial.
All code will be posted online after the lecture.

When your receive a notification about a Facebook post, or a new email, or pretty much anything from some online resource, it was probably powered by a **push notification**. Push notifications allow services to **"push" data to your app immediately**, instead of having your app periodically "pull" data from an API on a regular basis.

We will be using the **Google Cloud Messaging (GCM) framework**, because it's easy to use, an industry standard, and free! Receiving push notifications is as **simple** as registering a few services with the right parameters, then specifying the code that should run when a notification is received!

> **Note: Sending Push Notifications**
> Sending push notifications is (in most cases) a server/backend programming concern, and *outside* the scope of this course. All you need to know is that your service needs to maintain a list of GCM "addresses", and then use the GCM API to send a notification to one or more of these "addresses".

The first step of receiving push notifications is **registering for a GCM "address"**, which is a unique identifier for the device that is running your app. This is how the GCM framework knows where to send your notification. Registering for an address is very straightforward, standard code, so it is already included within the example app code. This address needs to be **stored in your backend system** (ideally via an API), so that you can use it in the future to send notifications to the device.

Once you have registered for an ID and sent it to your backend, all that's left to do is tell your app to **listen for notifications** and respond to them when they arrive. This is a matter of implementing a class that extends `GcmListenerService` and overrides `onMessageReceived(...)`.

Next Steps

There is a plethora of Android resources online, and I strongly recommend exploring them to continue building your skills. Some great resources are listed here (but as always, GIYF).

- http://developer.android.com
  The de facto resource for Android, including guides, examples and API documentation.

- https://github.com - search for `android type:issue state:open`
  Look at the issues for public Android repositories, and consider attempting to fix them and submitting a pull request. This is an excellent way to learn how to use Git as well.

- https://youtube.com/user/derekbanas
  This man has an excellent series of tutorials on Android, and other languages as well. This is where I first learned Android!
    - Other good YouTube channels: Android, Android Authority, slidenerd, Google Developers

- http://tutorialspoint.com/android

- Hackathons
  These are an amazing way to learn and practise new skills. I got started in Android at a hackathon, and I've been to around 20 hackathons since. They are great environ ments for learning, and the vast majority have absolutely no required skill level for entry.

- https://github.com/markormesher/KCLTechAndroidCourse
  The notes, slides and code samples from this course will always be available online, here.

- http://kcltechhq.slack.com/messages/android-programming
  The Slack channel for this course will remain open indefinitely.

Feedback

Please fill out our course feedback questionnaire - it only takes 60 seconds or so! We really value your feedback, and we'll use every word to improve the course next term.

Questionnaire: http://tiny.cc/3fby6x