Mobile Application Lab 12

Instructors: Legand Burge, Asad Ullah Naweed
Date: 10/04/2018
Topics: Services and Notifications

Be sure to click on the links in this doc to view more information about certain topics. These may or may not help you in your lab, but can always provide some extra context and information.

Initial Setup

You'll be creating a new project for this lab. This new project should have the package name `com.techexchange.mobileapps.lab12`. In this lab, we'll be exploring three different concepts: Services, Handlers and Notifications.

Services

IntentService

Right-click on the main package in your project, and create a new IntentService called `NotifierService`. This should do two things:

1. Generate a new Java file called NotifierService.java.
2. Add an entry for this service in your application's manifest.

Once this is done, remove the extra code from `NotifierService`, until only this is left:

```
public class NotifierService extends IntentService {
  private static final String TAG = "NotifierService";

  public NotifierService() {
    super(TAG);
  }

  @Override
  protected void onHandleIntent(Intent intent) {
    if (intent == null)
      return;
    Log.d(TAG, "Received Intent!");
  }
}
```

Now, add a static method to this class that returns a new intent for starting this service, similar to what we covered in Lecture 11.

```
static Intent newIntent(Context context) {
  return new Intent(context, NotifierService.class);
}
```

Now, start the service from your `MainActivity` in the `onCreate` method. Build and run the app, and view Logcat output. Make sure that you can see the message from `NotifierService`.

A Sticky Service

An `IntentService` is the simplest kind of service that can do most background tasks, but we want to create a service that runs continuously in the background, regardless of the Activity's lifecycle. To make that happen, we will make `NotifierService` extend from `Service` instead of `IntentService`.

```
public class NotifierService extends ~~IntentService~~ Service {

~~  public NotifierService() {~~
~~    super(TAG);~~
~~  }~~

~~  @Override~~
~~  protected void onHandleIntent(Intent intent) {~~
~~    if (intent == null)~~
~~      return;~~
~~    Log.d(TAG, "Received Intent!");~~
~~  }~~

  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    return super.onStartCommand(intent, flags, startId);
  }

  @Override
  public IBinder onBind(Intent intent) {
    return null;
  }
}
```

Ignore the `onBind` method for now. It's something that must be implemented, but returning null should do just fine for now. This means that this service cannot be "bound". We'll not be covering bound services in this lab.

Now, we want to modify the `onStartCommand` method in two major ways. In here, we want to:

1. Set up a background task that periodically prints out a Logcat message.
2. Make sure the service continues running even when the Activity is paused/killed.

The HandlerThread

We'll be setting up a `HandlerThread` in the service. This is a thread that will create a thread, a `Looper` and a Message queue. Each message will be tied to a single `Handler` which will contain the necessary code for processing each `Message`.

Set up a custom `HandlerThread` as a new class. Create a new class in your package, and call it `NotifierThread`. Make this class extend `HandlerThread` and override the `onLooperPrepared` method.

```
final class NotifierThread extends HandlerThread {
  private static final String TAG = "NotifierThread";

  private Handler handler;
  private Context context;

  NotifierThread(Context context) {
    super(TAG);
    this.context = context;
  }

  @Override
  protected void onLooperPrepared() {
    super.onLooperPrepared();
  }
}
```

Take a look at the documentation for HandlerThread, and see when the `onLooperPrepared` method is called, and by what thread.

There is a `Handler` in the NotifierThread, but we have not initialized it anywhere. The `Handler` is only valid when it has a valid message queue and Looper attached to it, so we will initialize it in the `onLooperPrepared` method, when we know that a Looper has been created.

While `Handler` itself is a concrete class and can be created by itself, it does absolutely nothing when it receives a `Message`. We will create a new subclass of `Handler` and override the necessary methods to make it do something with the messages it receives.

The NotificationHandler

Create a new Java class called `NotificationHandler`, and make it extend `Handler`. Override the `handleMessage` method, as follows:

```
final class NotificationHandler extends Handler {
  private static final String TAG = "NotificationHandler";

  private Context context;

  NotificationHandler(Context context) {
    this.context = context;
  }

  @Override
  public void handleMessage(Message msg) {
    Log.d(TAG, "Received message!");
  }
}
```

Coming back to your `NotifierThread` class, add the following to your `onLooperPrepared` method:

```
@Override
protected void onLooperPrepared() {
  super.onLooperPrepared();
  handler = new NotificationHandler(context);
  handler.obtainMessage(0, null).sendToTarget();
}
```

Finally, in your `NotifierService` class, add the following in `onStartCommand`:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  NotifierThread thread = new NotifierThread(this);
  thread.start();
  thread.getLooper();
  return START_NOT_STICKY;
}
```

Before proceeding further, build and run your app. You should see the Logcat output from the `NotificationHandler` class.

Notifications

We will now make your Service display a notification instead of simply printing out Logcat. Before proceeding, please download the notification icon and save it to your res/drawable directory.

Starting with Android O, you should create a `NotificationChannel` before creating any notifications. Each notification should be posted to a known notification channel. These channels allow the user to have more fine-grained controls over what notifications they want to see, and which ones to block.

To start off, create a new private method in your `NotifierService` class as follows:

```
static final String CHANNEL_ID = NotifierService.class.getName() + "pingchannel";
private static final String CHANNEL_NAME = "30-sec ping";

private void createNotificationChannel() {
  NotificationChannel channel = new NotificationChannel(CHANNEL_ID, CHANNEL_NAME,
      NotificationManager.IMPORTANCE_DEFAULT);
  channel.setDescription("This is the main channel for the thirty-second timer.");
  NotificationManager notificationManager =
      getSystemService(NotificationManager.class);
  notificationManager.createNotificationChannel(channel);
}
```

Now, invoke this method from the Service's `onStartCommand` method. This should be the first thing you do.

We will now use this channel in the `NotificationHandler` class to create a Notification instead of simply generating Logcat output:

```
@Override
public void handleMessage(Message msg) {
  Notification notification = new NotificationCompat.Builder(context,
      NotifierService.CHANNEL_ID)
      .setSmallIcon(R.drawable.notification_icon)
```

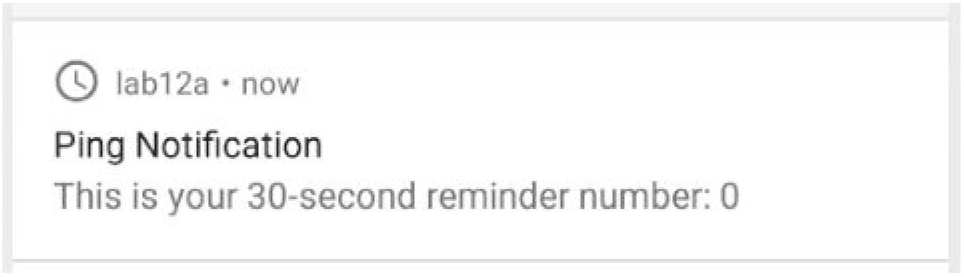
```
        .setContentTitle("Ping Notification")
        .setContentText("This is your 30-second reminder")
        .build();
    NotificationManagerCompat notificationManager =
        NotificationManagerCompat.from(context);
    notificationManager.notify(0, notification);
}
```

Now, build and run your app. You should see a notification pop up instead of a Logcat message.

Running Forever

Now that you have the basic framework for Services and Handlers set up, the following three tasks are left as exercises:

1. Make the service and its threads stick around even when the Activity is killed.
2. Make the Handler post another (delayed) message onto its own queue once it has displayed a notification.
3. Include some kind of counter in the Handler that shows something like this:



lab12a • now

Ping Notification

This is your 30-second reminder number: 0

4. Making sure that the counter does not get reset when the app is restarted.