MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 8

Objectives:

- To understand the working of Broadcast Receivers
- To understand usage of Content Providers
- To use UI notifications.
- Practice Activities

OBJECTIVE 1: Broadcast Receivers and its Usage

- A broadcast receiver (receiver) is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.

- For example, applications can register for the **ACTION_BOOT_COMPLETED** system event which is fired once the Android system has completed the boot process.

Registering a Receiver

- A receiver can be registered via the **AndroidManifest.xml** file.
- Alternatively to this static registration, you can also register a receiver dynamically via the **Context.registerReceiver()** method.
- The implementing class for a receiver extends the **BroadcastReceiver** class.
- If the event for which the broadcast receiver has registered happens, the **onReceive()** method of the receiver is called by the Android system

System Broadcasts

- Several system events are defined as final static fields in the Intent class. Other Android system classes also define events, e.g., the TelephonyManager defines events for the change of the phone state

Table 1. System Events

Event	Description
Intent.ACTION_BOOT_COMPLETED	Boot completed. Requires the `android.permission.RECEIVE_BOOT_COMPLETED` permission
Intent.ACTION_POWER_CONNECTED	Power got connected to the device.
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device.
Intent.ACTION_BATTERY_LOW	Triggered on low battery. Typically used to reduce activities in your app which consume power.
Intent.ACTION_BATTERY_OKAY	Battery status good again.

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 8

1. **Static Broadcast Receiver**
   A static broadcast receiver is something which is registered inside the XML code of Manifest.XML file - for example:

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".ServiceConsumerActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name="MyScheduleReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
    <receiver android:name="MyStartServiceReceiver" >
    </receiver>
</application>
```

2. **Dynamic Broadcast Receiver**
   A broadcast receiver which is registered inside the java or kotlin code, which can be registered via **Context.registerReceiver()** method.

```
BroadcastReceiver br = new MyBroadcastReceiver();
```

```
IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY_ACTION);
    filter.addAction(Intent.ACTION_AIRPLANE_MODE_CHANGED);
    this.registerReceiver(br, filter);
```

OBJECTIVE 2: Understand the working of Content Providers

- Find the document provided

OBJECTIVE 3: Send UI Notifications

- A notification is a message that Android displays outside your app's UI to provide the user with reminders, communication from other people, or other timely information from your app. Users can tap the notification to open your app or take an action directly from the notification.

Users can drag down on a notification in the drawer to reveal the expanded view, which shows additional content and action buttons, if provided.

A notification remains visible in the notification drawer until dismissed by the app or the user.

Heads-up notification

Beginning with Android 5.0, notifications can briefly appear in a floating window called a *heads-up notification*. This behavior is normally for important notifications that the user should know about immediately, and it appears only if the device is unlocked.

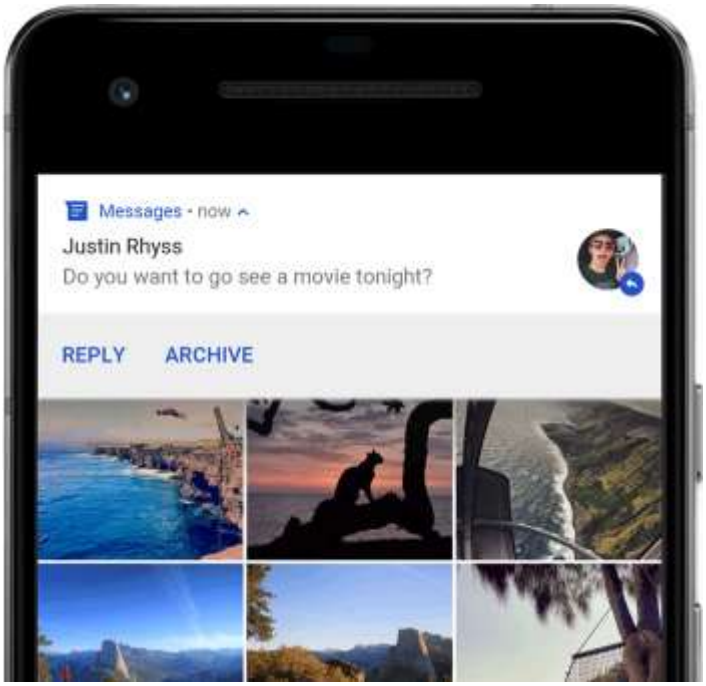


Figure 3. A heads-up notification appears in front of the foreground app

The heads-up notification appears the moment your app issues the notification and it disappears after a moment, but remains visible in the notification drawer as usual.

Example conditions that might trigger heads-up notifications include the following:

- The user's activity is in fullscreen mode (the app uses fullScreenIntent).
- The notification has high priority and uses ringtones or vibrations on devices running Android 7.1 (API level 25) and lower.
- The notification channel has high importance on devices running Android 8.0 (API level 26) and higher.

Lock screen

Beginning with Android 5.0, notifications can appear on the lock screen.

You can programmatically set the level of detail visible in notifications posted by your app on a secure lock screen, or even whether the notification will show on the lock screen at all.

Users can use the system settings to choose the level of detail visible in lock screen notifications, including the option to disable all lock screen notifications. Starting with Android 8.0, users can choose to disable or enable lock screen notifications for each notification channel.

Figure 4. Notifications on the lock screen with sensitive content hidden

To learn more, see how to Set lock screen visibility.

App icon badge

In supported launchers on devices running Android 8.0 (API level 26) and higher, app icons indicate new notifications with a colored "badge" (also known as a "notification dot") on the corresponding app launcher icon.

Users can long-press on an app icon to see the notifications for that app. Users can then dismiss or act on notifications from that menu, similar to the notification drawer.

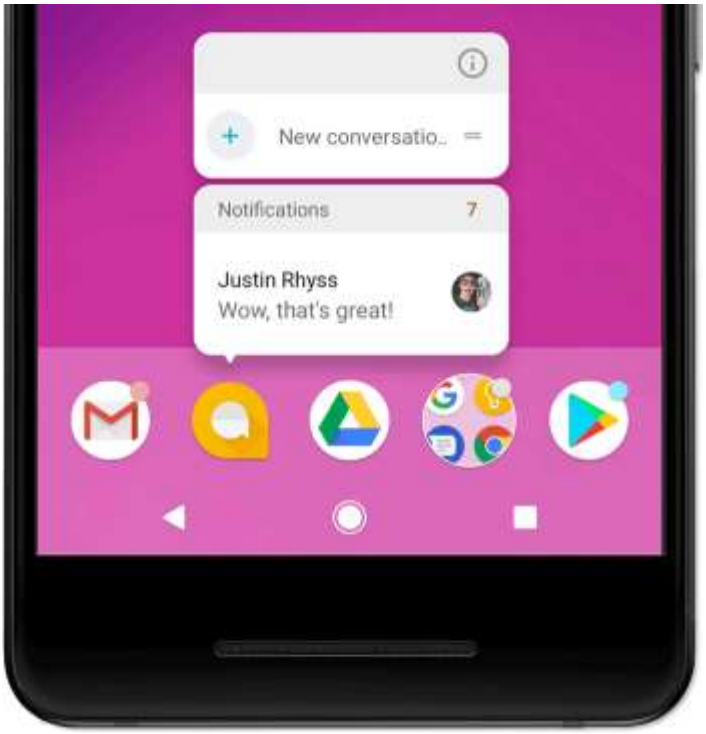


Figure 5. Notification badges and the long-press menu

To learn more about how badges work, read Notification badges.

Wear OS devices

If the user has a paired Wear OS device, all your notifications appear there automatically, including expandable detail and action buttons.

You can also enhance the experience by customizing some appearances for the notification on wearables and provide different actions, including suggested replies and voice input replies. For more information, see how to add wearable-specific features to your notification.

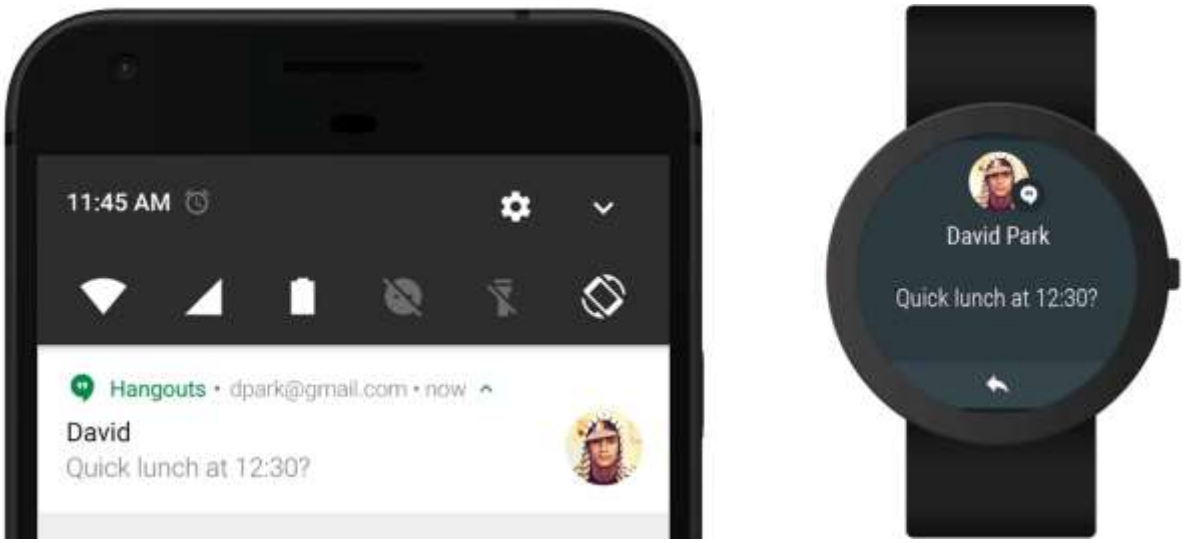


Figure 6. Notifications automatically appear on a paired Wear OS device

Notification anatomy

The design of a notification is determined by system templates—your app simply defines the contents for each portion of the template. Some details of the notification appear only in the expanded view.

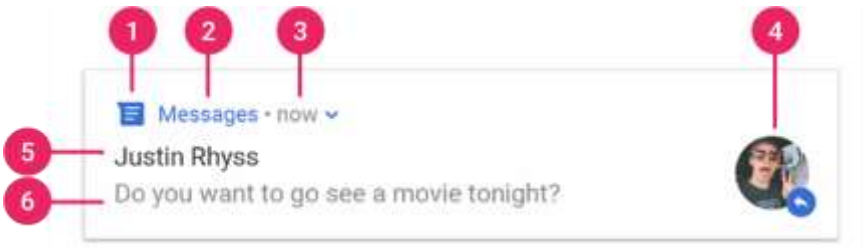


Figure 7. A notification with basic details

The most common parts of a notification are indicated in figure 7 as follows:

1. Small icon: This is required and set with setSmallIcon().
2. App name: This is provided by the system.
3. Time stamp: This is provided by the system but you can override with setWhen() or hide it with setShowWhen(false).

4. Large icon: This is optional (usually used only for contact photos; do not use it for your app icon) and set with setLargeIcon().
5. Title: This is optional and set with setContentTitle().
6. Text: This is optional and set with setContentText().

For more information about how to create a notification with these features and more, read Create a Notification.

Notification actions

Although it's not required, every notification should open an appropriate app activity when tapped. In addition to this default notification action, you can add action buttons that complete an app-related task from the notification (often without opening an activity), as shown in figure 9.

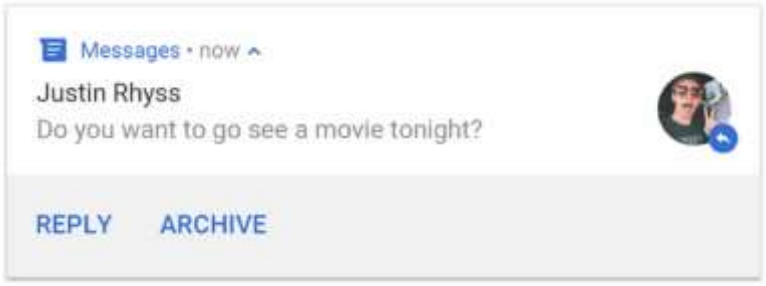


Figure 9. A notification with action buttons

Starting in Android 7.0 (API level 24), you can also add an action to reply to messages or enter other text directly from the notification.

Starting in Android 10 (API level 29), the platform can automatically generate action buttons with suggested intent-based actions.

Adding action buttons is explained further in Create a Notification.

Expandable notification

By default, the notification's text content is truncated to fit on one line. If you want your notification to be longer, you can enable a larger text area that's expandable by applying an additional template, as shown in figure 8.

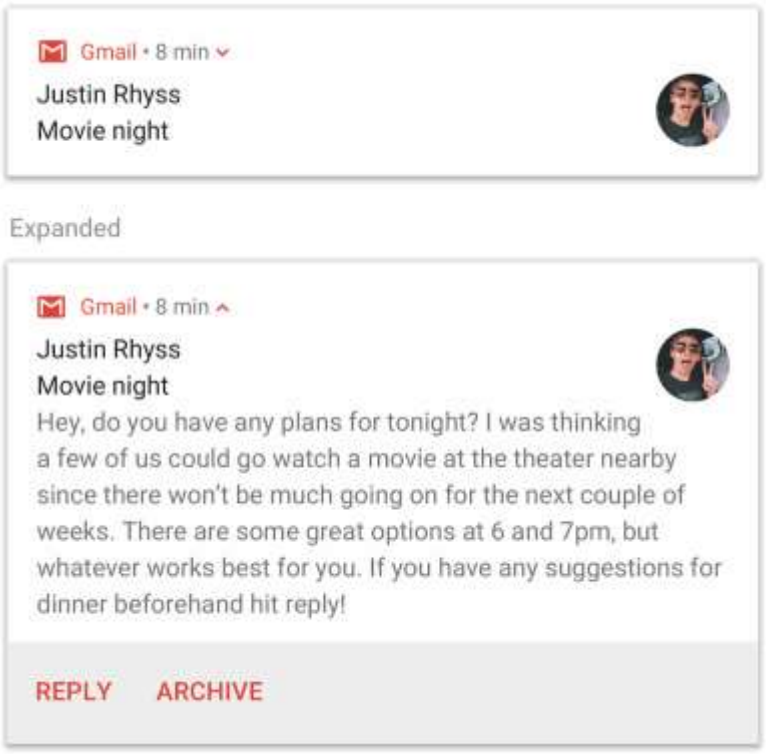


Figure 8. An expandable notification for large text

You can also create an expandable notification with an image, in inbox style, a chat conversation, or media playback controls. For more information, read Create an Expandable Notification.

And although we recommend you always use these templates to ensure proper design compatibility on all devices, if necessary, you can also create a custom notification layout.

Notification updates and groups

To avoid bombarding your users with multiple or redundant notifications when you have additional updates, you should consider updating an existing notification rather than issuing a new one, or consider using the inbox-style notification to show conversation updates.

However, if it's necessary to deliver multiple notifications, you should consider grouping those separate notifications into a group (available on Android 7.0 and higher). A notification group allows you to collapse multiple notifications into just one post in the notification drawer, with a summary. The user can then expand the notification to reveal the details for each individual notification.

The user can progressively expand the notification group and each notification within it for more details.

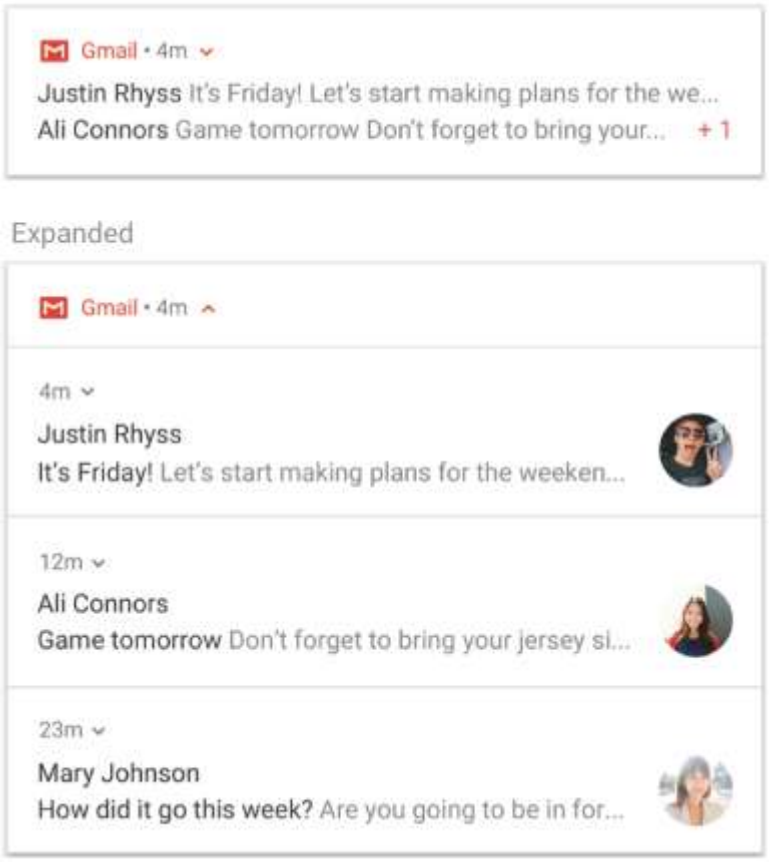


Figure 10. A collapsed and expanded notification group

To learn how to add notifications to a group, see Create a Group of Notifications.

Note: If the same app sends four or more notifications and does not specify a grouping, the system automatically groups them together.

Notification channels

Starting in Android 8.0 (API level 26), all notifications must be assigned to a channel or it will not appear. By categorizing notifications into channels, users can disable specific notification channels for your app (instead of disabling *all* your notifications), and users can control the visual and auditory options for each channel—all from the Android system settings (figure 11). Users can also long-press a notification to change behaviors for the associated channel.

On devices running Android 7.1 (API level 25) and lower, users can manage notifications on a per-app basis only (effectively each app only has one channel on Android 7.1 and lower).

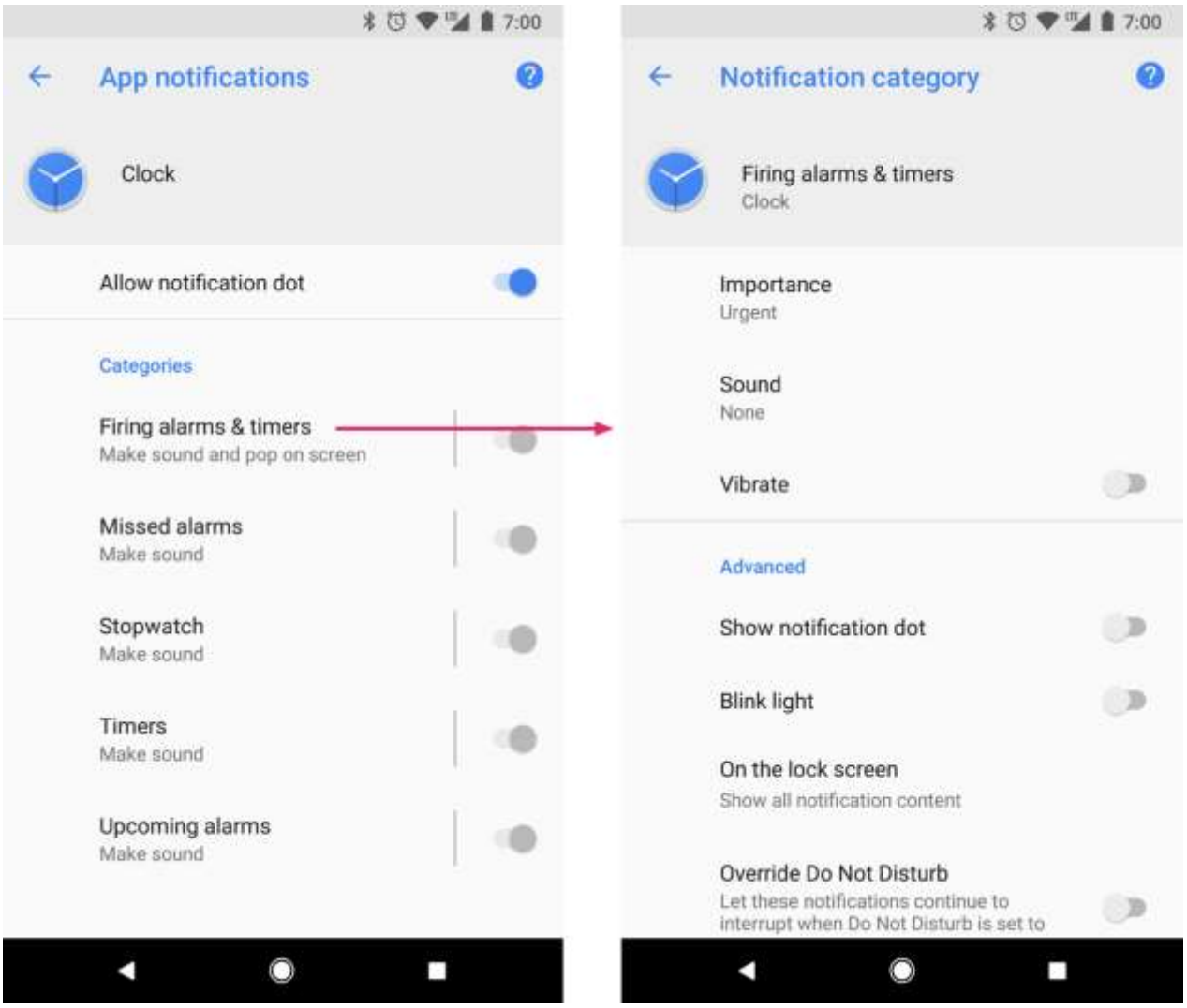


Figure 11. Notification settings for Clock app and one of its channels

How to setup a Notification:

Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this)
```

Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification
Detail!");
```

Step 3 : Attach Actions

This is an optional part and required 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 or do further work

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification

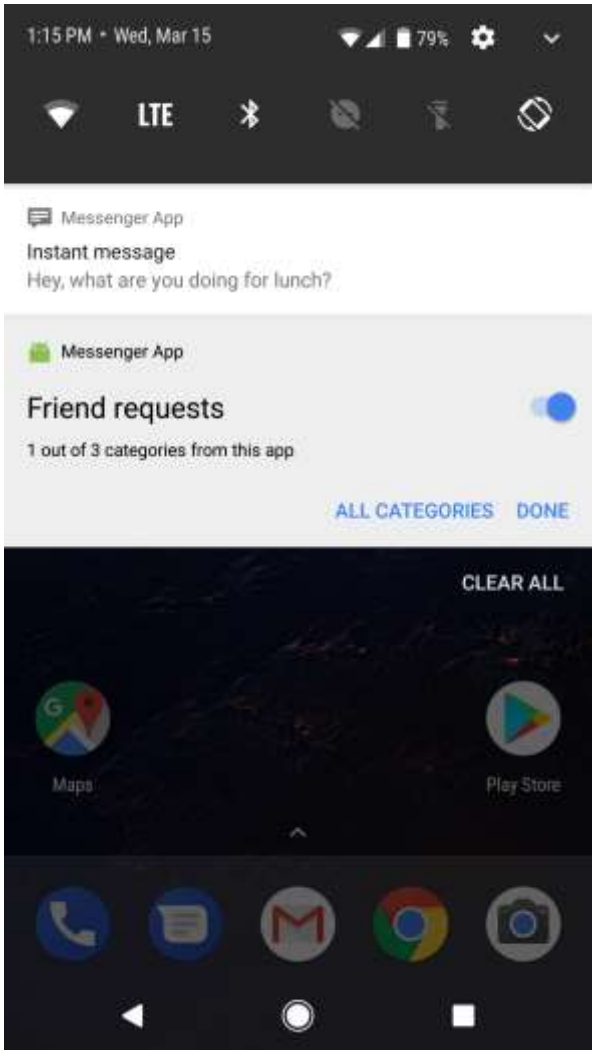
```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

MOBILE APPLICATION DEVELOPMENT (MAD) – LAB 8

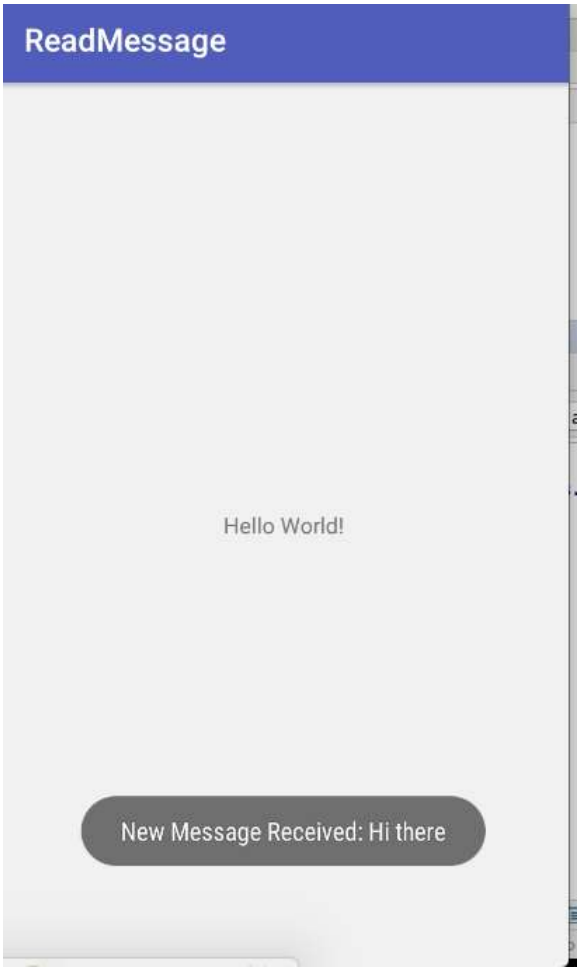
Practice Activities:

Activity 1: Create a notification shown like attached picture



Activity 2: Create an SMS reader app.

- Which receives all the incoming messages and shown them on screen in either Toast or A label along with the sender's information



Activity 3: Create a user dictionary and make it available so that other applications can read and update the words in the dictionary.

- Database to store the data
- Search button
- Menu option to add a new word.
- Content Provider to share the stored words with other apps.