Andhra Pradesh State Skill Development Corporation

eSkill AP
Learn Anytime Anywhere
Andhra Pradesh State Skill Development Corporation

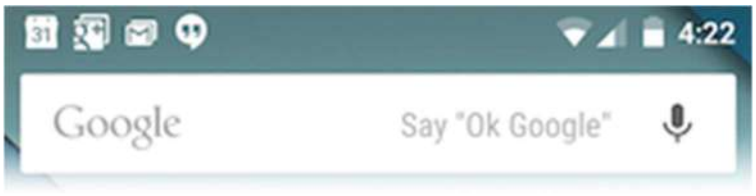
ANDROID APPLICATION DEVELOPMENT

NOTIFICATIONS

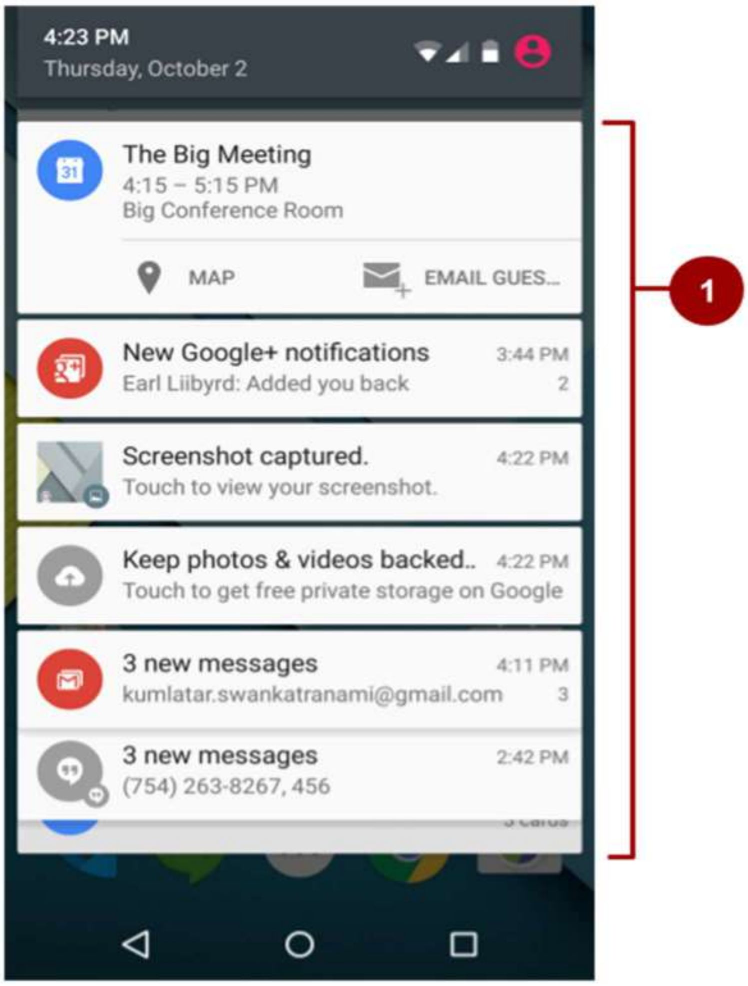
Notifications

What is a notification?

A notification is a message your app displays to the user outside your app's normal UI. When your app tells the system to issue a notification, the notification appears to the user as an icon in the notification area, on the left side of the status bar.



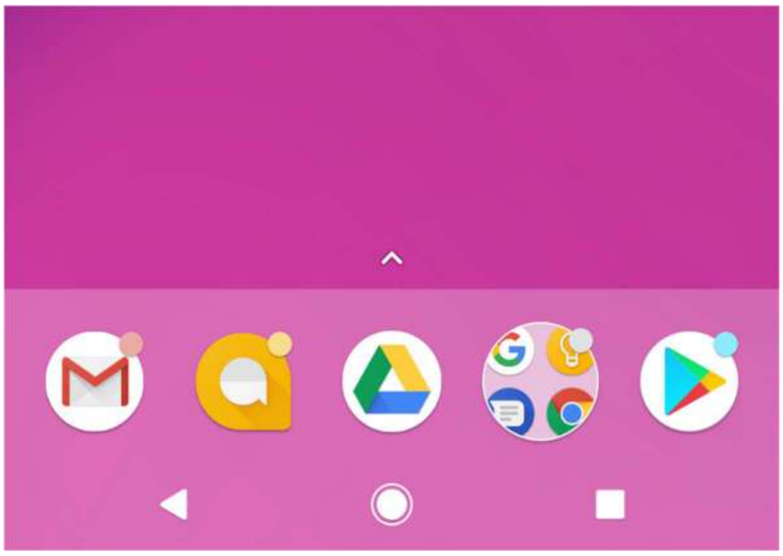
If the device is unlocked, the user opens the notification drawer to see the details of the notification. If the device is locked, the user views the notification on the lock screen. The notification area, lock screen, and notification drawer are system-controlled areas that the user can view at any time.


4:23 PM
Thursday, October 2
4:22
The Big Meeting
4:15 – 5:15 PM
Big Conference Room
MAP
EMAIL GUES...
New Google+ notifications 3:44 PM
Earl Liibyrd: Added you back 2
Screenshot captured. 4:22 PM
Touch to view your screenshot.
Keep photos & videos backed.. 4:22 PM
Touch to get free private storage on Google
3 new messages 4:11 PM
kumlatar.swankatranami@gmail.com 3
3 new messages 2:42 PM
(754) 263-8267, 456
1

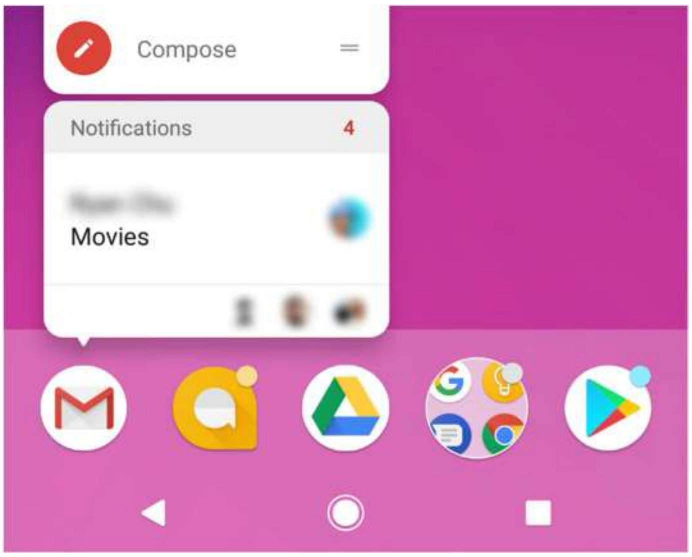
1. An "open" notification drawer. The status bar isn't visible in this screenshot, because the notification drawer is open.

App icon badge

In supported launchers on devices running Android 8.0 (API level 26) and higher, an app icon changes its appearance slightly when the app has a new notification to show to the user. The app icon shows a colored badge, also known as a notification dot, as shown on four of the five app icons in the screenshot below.



To see the notification for an app with a notification dot, the user long-presses the app icon. The notification menu appears, as shown below, and the user dismisses the notification or acts on it from the menu. This is similar to the way the user interacts with a notification in the notification drawer.

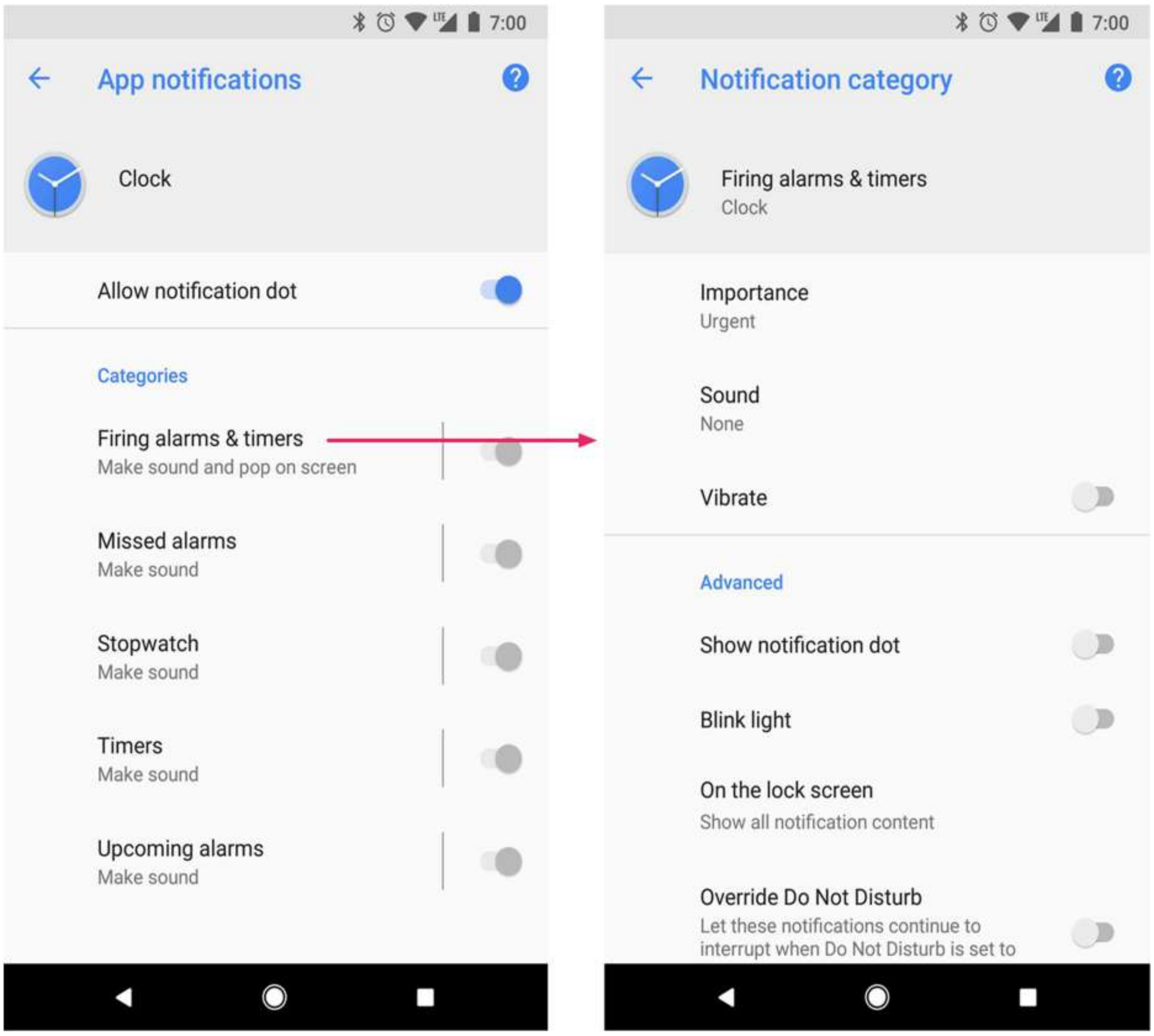


Notification channels

In the Settings app on an Android-powered device, users can adjust the notifications they receive. Starting with Android 8.0 (API level 26), you can assign each of your app's notifications to a notification channel. Each notification channel represents a type of notification, and you can group several notifications in each channel.

If you target only lower-end devices, you don't need to implement notification channels to display notifications, but it's good practice to target the latest available SDK, then check the device's SDK version before building the notification channel.

Notification channels are called Categories in the user-visible Settings app. For example, the screenshot on the left shows the notification settings for the Clock app, which has five notification channels. The screenshot on the right shows the settings for the "Firing alarms & timers" notification channel.


7:00

App notifications

Clock

Allow notification dot

Categories

Firing alarms & timers
Make sound and pop on screen

Missed alarms
Make sound

Stopwatch
Make sound

Timers
Make sound

Upcoming alarms
Make sound

7:00

Notification category

Firing alarms & timers
Clock

Importance
Urgent

Sound
None

Vibrate

Advanced

Show notification dot

Blink light

On the lock screen
Show all notification content

Override Do Not Disturb
Let these notifications continue to interrupt when Do Not Disturb is set to

When you create a notification channel in your code, you set behavior for that channel, and the behavior is applied to all of the notifications in the channel. For example, your app might set the notifications in a channel to play a sound, blink a light, or vibrate. Whatever behavior you set for a notification channel, the user can change it, and they can turn off notifications from your app altogether.

Note: - If your app targets Android 8.0 (API level 26) or higher, you must implement one or more notification channels. - If your targetSdkVersion is set to 25 or lower, - when your app runs on Android 8.0 (API level 26) or higher, - it behaves the same as it would on devices running Android 7.1 (API level 25) or lower.

Creating a notification channel

To create a notification channel instance, use the NotificationChannel constructor. Specify an ID that's unique within your package, a user-visible channel name, and an importance for the channel:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
   NotificationChannel notificationChannel =
       new NotificationChannel(CHANNEL_ID, "Mascot Notification",
       NotificationManager.IMPORTANCE_DEFAULT);
}
```

Important: - Before you use the notification-channel APIs, check the SDK version. - Notification channels are only available in Android 8.0 (API level 26) and higher. - Notification channels are not available in the Android Support Library.

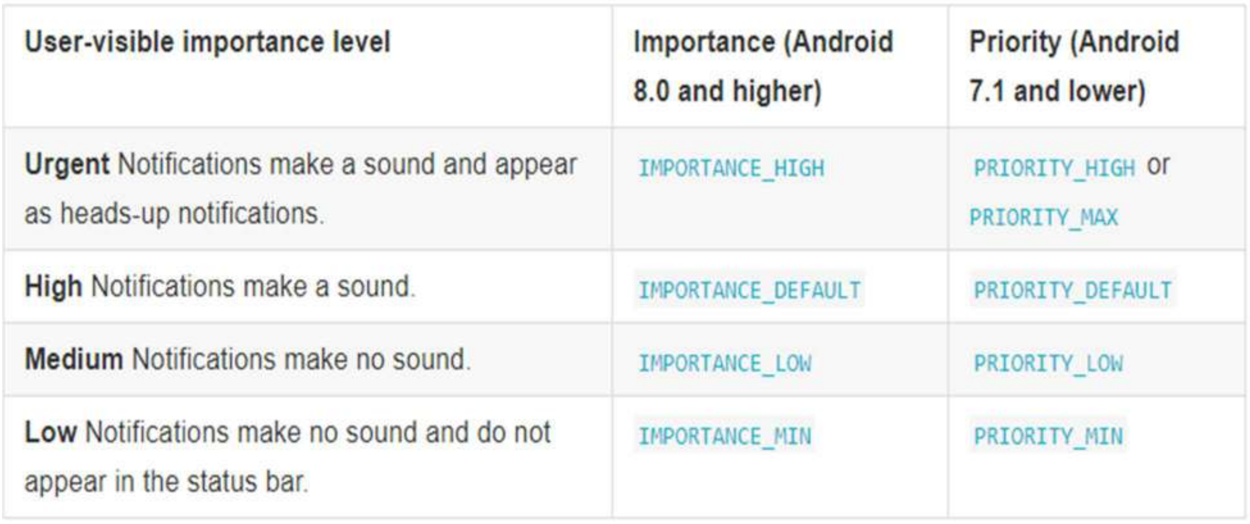
Set the importance level

The NotificationChannel constructor, which is available in Android 8.0 (API level 26) and higher, requires an importance level. The channel's importance determines the instrusiveness of the notifications posted in that channel. For example, notifications with a higher importance might make sound and show up in more places than notifications with a lower importance. There are five importance levels, ranging from IMPORTANCE_NONE(0) to IMPORTANCE_HIGH(4).

To support Android 7.1 (API level 25) or lower, you must also set a priority for each notification. To set a priority, use the setPriority() method with a priority constant from the NotificationCompat class.

```
mBuilder.setPriority(NotificationCompat.PRIORITY_HIGH);
```

On devices running Android 8.0 and higher, all notifications, regardless of priority and importance level, appear in the notification drawer and as app icon badges. After a notification is created and delivered, the user can change the notification channel's importance level in the Android Settings app. The following table shows how the user-visible importance level maps to the notification-channel importance level and the priority constants.



User-visible importance level	Importance (Android 8.0 and higher)	Priority (Android 7.1 and lower)
Urgent Notifications make a sound and appear as heads-up notifications.	`IMPORTANCE_HIGH`	`PRIORITY_HIGH` or `PRIORITY_MAX`
High Notifications make a sound.	`IMPORTANCE_DEFAULT`	`PRIORITY_DEFAULT`
Medium Notifications make no sound.	`IMPORTANCE_LOW`	`PRIORITY_LOW`
Low Notifications make no sound and do not appear in the status bar.	`IMPORTANCE_MIN`	`PRIORITY_MIN`

Configure the initial settings

Configure the notification channel object with initial settings such as an alert sound, a notification light color, and an optional user-visible description.

```
notificationChannel.enableLights(true);
notificationChannel.setLightColor(Color.RED);
notificationChannel.enableVibration(true);
notificationChannel.setDescription("Notification from Mascot");
```

Starting from Android 8.1 (API Level 27), apps can only make a notification alert sound once per second. Alert sounds that exceed this rate aren't queued and are lost. This change doesn't affect other aspects of notification behavior, and notification messages still post as expected.

Create the notification channel

To create the notification channel, pass the instance of NotificationChannel to the createNotificationChannel() method from the NotificationManager class.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
NotificationManager mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotifyManager.createNotificationChannel(notificationChannel);
}
```

Check the SDK version before using createNotificationChannel(), because this API is only available on Android 8.0 and higher and in support packages.

Once you create a notification and notification channel and submit it to the NotificationManager, you cannot change the importance level using code. However, the user can change their preferences for your app's channels using the Android Settings app on their device.

Creating notifications

You create a notification using the NotificationCompat.Builder class. (NotificationCompat from the Android Support Library provides compatibility back to Android 4.0, API level 14. For more information, see Notification compatibility, below.) The builder classes simplify the creation of complex objects.

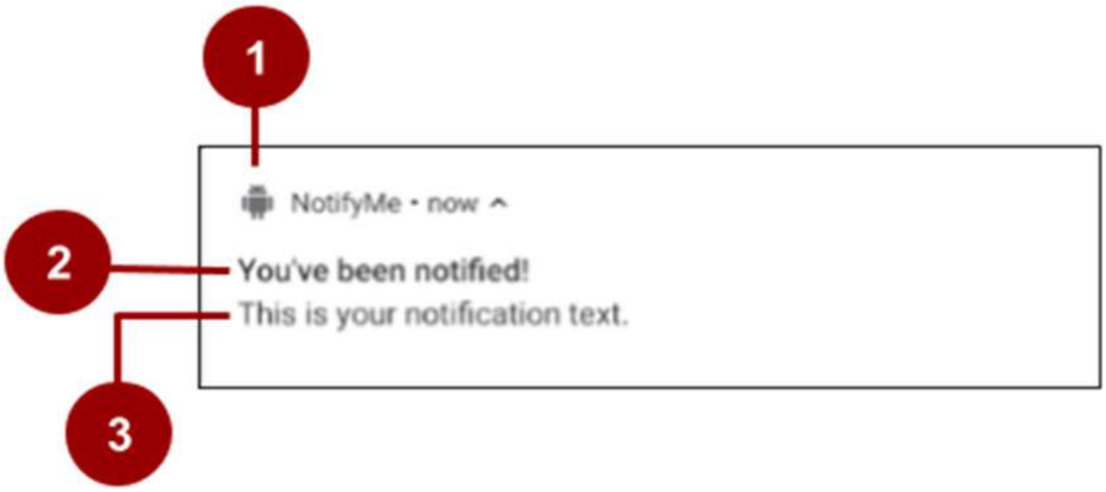
To create a NotificationCompat.Builder, pass the application context and notification channel ID to the constructor:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this, CHANNEL_ID);
```

The NotificationCompat.Builder constructor takes the notification channel ID as one of its parameters. This parameter is only used by Android 8.0 (API level 26) and higher. Lower versions of Android ignore it.

Set notification contents

You can assign components to the notification like a small icon, a title, and the notification message.



In the screenshot above:

1. A small icon, set by setSmallIcon(). This is the only content that's required.
2. A title, set by setContentTitle().
3. The body text, set by setContentText(). This text must be fewer than 40 characters, and it should not repeat what is in the title.

```
NotificationCompat.Builder mBuilder =
new NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.android_icon)
        .setContentTitle("You've been notified!")
        .setContentText("This is your notification text.");
```

Set the intent for the notification's tap action

Every notification must respond when it is tapped, usually by launching an Activity in your app. To launch an Activity in your app, set a content intent using the setContentIntent() method, passing in the Intent wrapped in a PendingIntent object. When your app uses a PendingIntent, the system can launch the Activity in your app on your behalf.

To instantiate a PendingIntent, use one of the following methods, depending on how you want the contained Intent to be delivered:

- To launch an Activity when a user taps the notification, use PendingIntent.getActivity(). Pass in an explicit Intent for the Activity you want to launch. The getActivity() method corresponds to an Intent delivered using startActivity().
- For an Intent passed into startService(), for example a service to download a file, use PendingIntent.getService().
- For a broadcast Intent delivered with sendBroadcast(), use PendingIntent.getBroadcast().

Each of these PendingIntent methods takes the following arguments:

- The application context.
- A request code, which is a constant integer ID for the PendingIntent.
- The Intent to be delivered.

- A PendingIntent flag that determines how the system handles multiple PendingIntent objects from the same app.

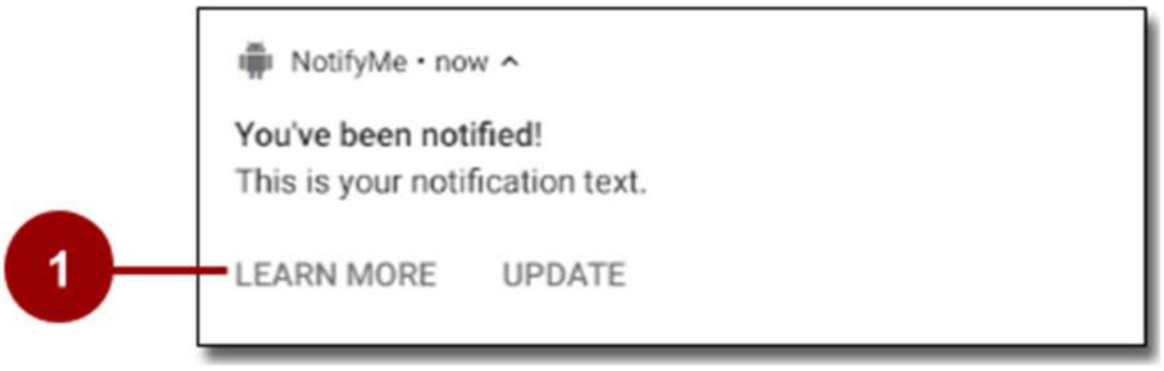
The following snippet shows how to create a basic Intent to open an Activity when the user taps the notification:

```
// Create an explicit intent for an Activity in your app
Intent contentIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingContentIntent = PendingIntent.getActivity(this, 0,
                    contentIntent, PendingIntent.FLAG_UPDATE_CURRENT);

// Set the intent that will fire when the user taps the notification
mBuilder.setContentIntent(pendingContentIntent);
```

Add notification action buttons

Notification action buttons allow the user to perform an app-related task without launching the app. The system typically displays action buttons adjacent to the notification content. A notification can have up to three notification action buttons.



1. "Learn more" and "Update" action buttons

Using action buttons, you can let the user perform a variety of actions, beyond just launching an Activity after the user taps the notification itself. For example, you can use action buttons to let the user start a background task to upload a file, place a phone call, snooze an alarm, or play music. For Android 7.0 (API level 24) and higher, you can use an action button to let the user reply to a message directly from a notification.

Adding an action button is similar to setting up the notification's default tap action: pass a PendingIntent to the addAction() method in the NotificationCompat.Builder class. But this action should not replicate what happens when the user taps the notification itself.

The following code shows how to add an action button using the addAction() method with the NotificationCompat.Builder object, passing in the icon, the title string for the label, and the PendingIntent to trigger when the user taps the action button.

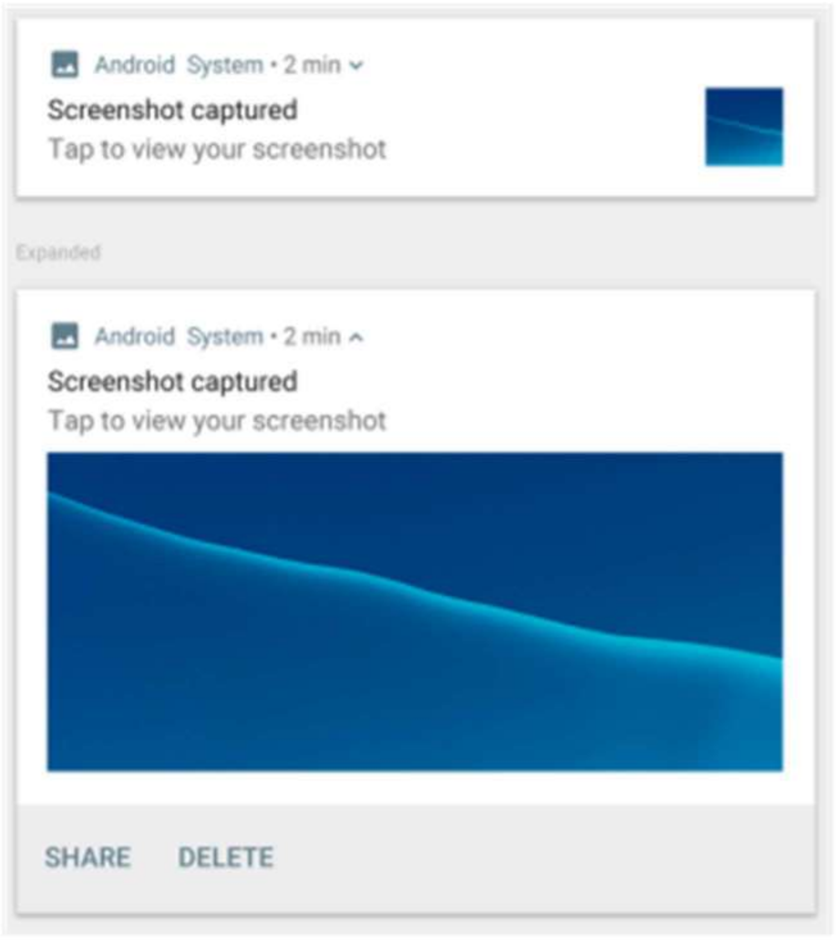
```
mBuilder.addAction(R.drawable.car, "Get Directions", mapPendingIntent);
```

Starting from Android 7.0, icons are not displayed in notifications. Instead, more room is provided for the notification labels themselves. But notification action icons are still required, and they are used on older versions of Android and on devices such as Android Wear.

Expandable notifications

Notifications in the notification drawer appear in two main layouts, normal view (which is the default) and expanded view. Expanded view notifications were introduced in Android 4.1. Use them sparingly, because they take up more space and attention than normal view layouts.

To create notifications that appear in an expanded layout, use one of these helper classes to set the style object to the setStyle() method: - Use NotificationCompat.BigTextStyle for large-format notifications that include a lot of text. - Use NotificationCompat.InboxStyle for displaying a list of summary lines, for example for incoming messages or emails. - Use NotificationCompat.MediaStyle for media playback notifications. - Use NotificationCompat.MessagingStyle to display sequential messages in an ongoing conversation. This style currently applies only on devices running Android 7.0 and higher. On lower devices, these notifications are displayed in the supported style. - Use NotificationCompat.BigPictureStyle for large-format notifications that include large image attachments, as shown in the screenshot below.



For example, here's how you'd set the BigPictureStyle on a notification: java NotificationCompat notif = new NotificationCompat.Builder(mContext, channelId) .setContentTitle("New photo from " + sender.toString()) .setContentText(subject) .setSmallIcon(R.drawable.new_post) .setLargeIcon(aBitmap) .setStyle(new NotificationCompat.BigPictureStyle() .bigPicture(aBigBitmap) .setBigContentTitle("Large Notification Title")) .build();

Ongoing notifications

Ongoing notifications are notifications that the user can't dismiss. Use ongoing notifications for background tasks that the user actively engages with, for example playing music. You can also use ongoing notifications to show tasks that are occupying the device, for example file downloads, sync operations, and active network connections.

Ongoing notifications can be a nuisance to your users, because users can't cancel them, so use them sparingly.

To make a notification ongoing, set setOngoing() to true.
Your app must explicitly cancel ongoing notifications by calling cancel() or cancelAll().

Delivering notifications

Use the NotificationManager class to deliver notifications:

- To create an instance of NotificationManager, call getSystemService(), passing in the NOTIFICATION_SERVICE constant.
- To deliver the notification, call notify().

Pass these two values in the notify() method:

- A notification ID, which is used to update or cancel the notification.
- The NotificationCompat object that you created using the NotificationCompat.Builder object.

The following example creates a NotificationManager instance, then builds and delivers a notification:

```java
mNotifyManager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
//Builds the notification with all the parameters NotificationCompat.Builder notifyBuilder = new
NotificationCompat.Builder(this, PRIMARY_CHANNEL)
.setContentTitle(getString(R.string.notification_title))
.setContentText(getString(R.string.notification_text)) .setSmallIcon(R.drawable.ic_android)
.setContentIntent(notificationPendingIntent) .setPriority(NotificationCompat.PRIORITY_HIGH)
java .setDefaults(NotificationCompat.DEFAULT_ALL);
//Delivers the notification mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());
```

Updating and reusing notifications

Sometimes you need to issue a notification multiple times for the same type of event. In this situation, you can update a previous notification by changing some of the notification's values, adding to the notification, or both.

To reuse an existing notification:

- Update a NotificationCompat.Builder object and build a Notification object from it, as when you first created and built the notification.
- Deliver the notification with the same ID you used previously.

Important: If the previous notification is still visible, the system updates it from the contents of the Notification object. If the previous notification has been dismissed, a new notification is created and displayed.

Clearing notifications

Notifications remain visible until one of the following happens:

- If the notification can be cleared, it disappears when the user dismisses it by swiping it or by using "Clear All".
- If you called setAutoCancel() when you created the notification, the notification cancels itself automatically. When the user taps the notification, the notification is removed from the status bar.
- If you call cancel() on the Notification object for a specific notification ID, the notification is removed from the status bar.
- If you call cancelAll() on the Notification object, all the notifications you've issued are removed from the status bar.

Because the user can't cancel ongoing notifications, your app must cancel them by calling cancel() or cancelAll() on the Notification object.

Codelab For Notification

1. In Android Studio, create a new project called "Notify Me!" Accept the default options, and use the Empty Activity template.
2. In your activity_main.xml layout file, replace the default TextView with a button that has the following attributes:

```
<Button
    android:id="@+id/notify"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Notify Me!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Do the following steps in the MainActivity.java file:

- Create a member variable for the Notify Me! button:

```
private Button button_notify;
```

- Create a method stub for the sendNotification() method:

```
public void sendNotification() {}
```

- In the onCreate() method, initialize the Notify Me! button and create an onClickListener for it. Call sendNotification() from the onClick method:

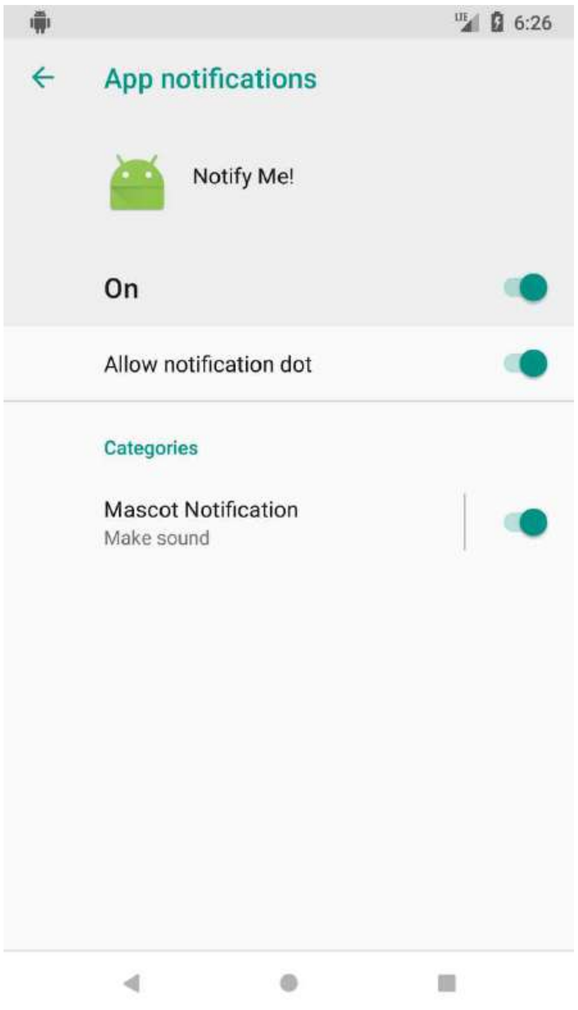
```
button_notify = findViewById(R.id.notify);
button_notify.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
        sendNotification();
    }
});
```

Create a notification channel

In the Settings app on an Android-powered device, users can adjust the notifications they receive. Starting with Android 8.0 (API level 26), your code can assign each of your app's notifications to a user-customizable notification channel:

- Each notification channel represents a type of notification.
- In your code, you can group several notifications in each notification channel.
- For each notification channel, your app sets behavior for the channel, and the behavior is applied to all the notifications in the channel. For example, your app might set the notifications in a channel to play a sound, blink a light, or vibrate.
- Whatever behavior your app sets for a notification channel, the user can change that behavior, and the user can turn off your app's notifications altogether. On Android-powered devices running Android 8.0 (API level 26) or higher, notification channels that you create in your app appear as Categories under App notifications in the device Settings app.

For example, in the screenshot below of a device running Android 8.0, the Notify Me! app has one notification channel, Mascot Notification.



When your app targets Android 8.0 (API level 26), to display notifications to your users you must implement at least one notification channel. To display notifications on lower-end devices, you're not required to implement notification channels. However, it's good practice to always do the following:

- Target the latest available SDK.
- Check the device's SDK version in your code. If the SDK version is 26 or higher, build notification channels.

If your targetSdkVersion is set to 25 or lower, when your app runs on Android 8.0 (API level 26) or higher, it behaves the same as it would on devices running Android 7.1 (API level 25) or lower.

Create a notification channel:

1. In MainActivity, create a constant for the notification channel ID. Every notification channel must be associated with an ID that is unique within your package. You use this channel ID later, to post your notifications.
```
private static final String PRIMARY_CHANNEL_ID = "primary_notification_channel";
```
2. The Android system uses the NotificationManager class to deliver notifications to the user. In MainActivity.java, create a member variable to store the NotificationManager object.
```
private NotificationManager mNotifyManager;
```
3. In MainActivity.java, create a createNotificationChannel() method and instantiate the NotificationManager inside the method.
```
public void createNotificationChannel()
{
    mNotifyManager = (NotificationManager)
          getSystemService(NOTIFICATION_SERVICE);
}
```
4. Create a notification channel in the createNotificationChannel() method. Because notification channels are only available in API 26 and higher, add a condition to check for the device's API version.
```
public void createNotificationChannel() {
mNotifyManager = (NotificationManager)
      getSystemService(NOTIFICATION_SERVICE);
    if (android.os.Build.VERSION.SDK_INT >=
                        android.os.Build.VERSION_CODES.O) {
    // Create a NotificationChannel
    }
}
```
5. Inside the if statement, construct a NotificationChannel object and use PRIMARY_CHANNEL_ID as the channel id.
6. Set the channel name. The name is displayed under notification Categories in the device's user-visible Settings app.
7. Set the importance to IMPORTANCE_HIGH. (For the complete list of notification importance constants, see the NotificationManager documentation.)
```
// Create a NotificationChannel
NotificationChannel notificationChannel = new NotificationChannel(PRIMARY_CHANNE
L_ID,
"Mascot Notification", NotificationManager
.IMPORTANCE_HIGH);
```

8. In createNotificationChannel(), inside the if statement, configure the notificationChannel object's initial settings. For example, you can set the notification light color, enable vibration, and set a description that's displayed in the device's Settings app. You can also configure a notification alert sound.

```
notificationChannel.enableLights(true);
notificationChannel.setLightColor(Color.RED);
notificationChannel.enableVibration(true);
notificationChannel.setDescription("Notification from Mascot");
mNotifyManager.createNotificationChannel(notificationChannel);
```

Build your first notification Notifications are created using the NotificationCompat.Builder class, which allows you to set the content and behavior of the notification. A notification can contain the following elements:

- Icon (required), which you set in your code using the setSmallIcon() method.
- Title (optional), which you set using setContentTitle().
- Detail text (optional), which you set using setContentText().

To create the required notification icon:

1. In Android Studio, go to File > New > Image Asset.
2. From the Icon Type drop-down list, select Notification Icons.
3. Click the icon next to the Clip Art item to select a Material Design icon for your notification. For this app, use the Android icon.
4. Rename the resource ic_android and click Next and Finish. This creates drawable files with different resolutions for different API levels.

To build your notification and display it:

1. You need to associate the notification with a notification ID so that your code can update or cancel the notification in the future. In MainActivity.java, create a constant for the notification ID:

```
private static final int NOTIFICATION_ID = 0;
```

2. In MainActivity.java, at the end of the onCreate() method, call sendNotification().

```
public void sendNotification(){}
```

3. Inside the sendNotification() method,create and instantiate the notification builder. For the notification channel ID, use PRIMARY_CHANNEL_ID. If a popup error is displayed, make sure that the NotificationCompat class is imported from the v4 Support Library.

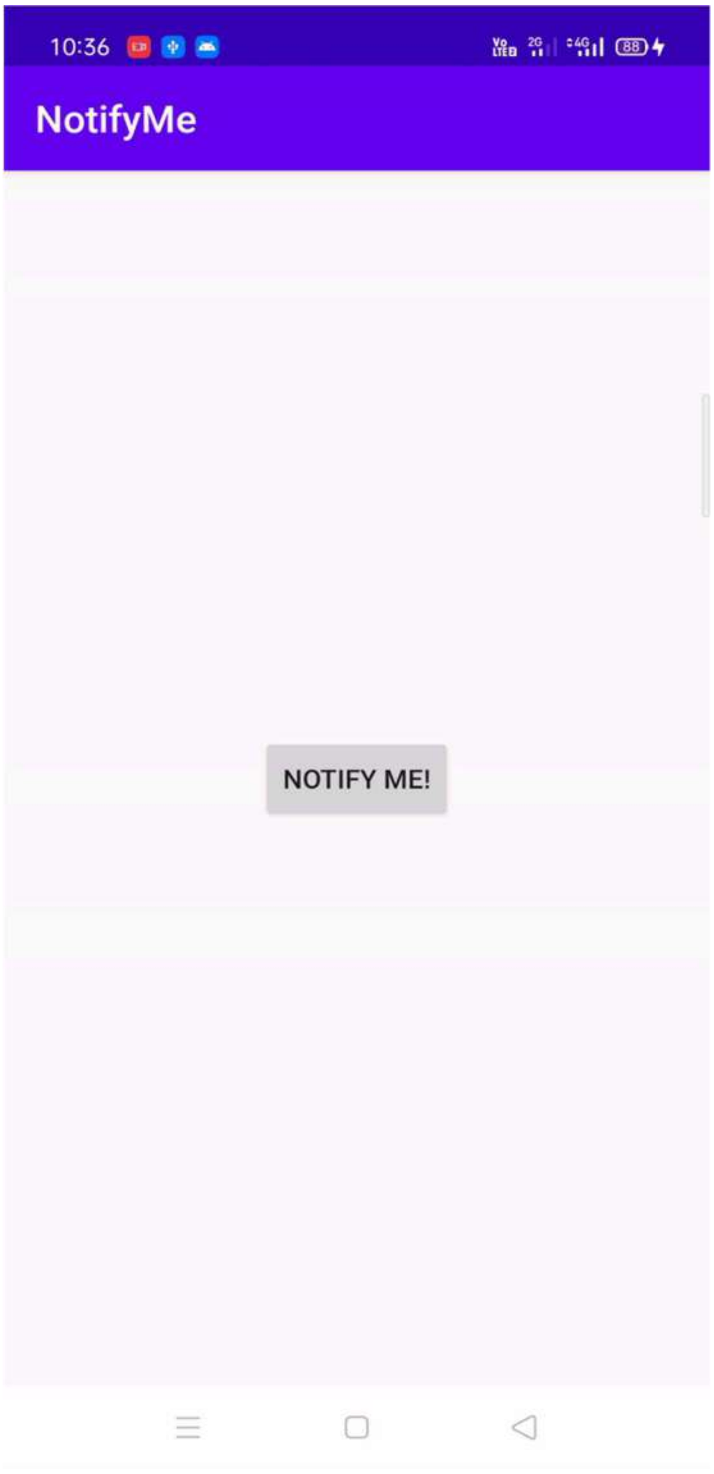
```
NotificationCompat.Builder notifyBuilder = new NotificationCompat.Builder(this, PRIMA
RY_CHANNEL_ID);
```

4. Inside the sendNotification() method, add the title, text, and icon to the builder, as shown below.

```
notifyBuilder.setContentTitle("You've been notified!");
notifyBuilder.setContentText("This is your notification text.");
notifyBuilder.setSmallIcon(R.drawable.ic_android);
```

5. In MainActivity.java inside the oncreate method you should have to call the createNotificationChannel() method.
   button_notify = findViewById(R.id.notify); createNotificationChannel();
6. Call notify() on the NotificationManager:
   mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());

OutPut


10:36
NotifyMe
NOTIFY ME!