

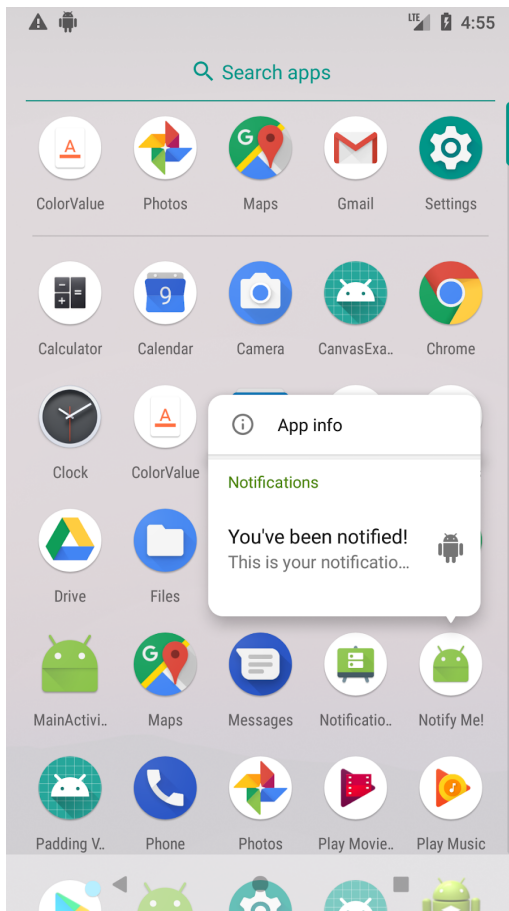
Lesson 8.1: Notifications

Introduction

Sometimes you want your app to show information to users even when the app isn't running in the foreground. For example, you might want to let users know that new content is available, or that their favorite sports team just scored a goal in a game. The Android [notification framework](#) provides a way for your app to notify users even when the app is not in the foreground.

A *notification* is a message that your app displays to the user outside of your app's normal UI. Notifications appear as icons in the device's notification area, which is in the status bar. To see the details of a notification, the user opens the *notification drawer*, for example by swiping down on the status bar. The notification area and the notification drawer are system-controlled areas that the user can view at any time.

On devices running Android 8.0 and higher, when your app has a new notification to show to the user, your app icon automatically shows a *badge*. (Badges are also called *notification dots*). When the user long-presses the app icon, the notification appears above the app icon, as shown in the screenshot below.



In this practical you create an app that triggers a notification when the user taps a button in your app. The user can update the notification or cancel it.

What you should already know

You should be able to:

- Implement the `onClick()` method for buttons.

*This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).
This PDF is a one-time snapshot. See developer.android.com/courses/fundamentals-training/toc-v2
for the latest updates.*

- Create implicit intents.
- Send custom broadcasts.
- Use broadcast receivers.

What you'll learn

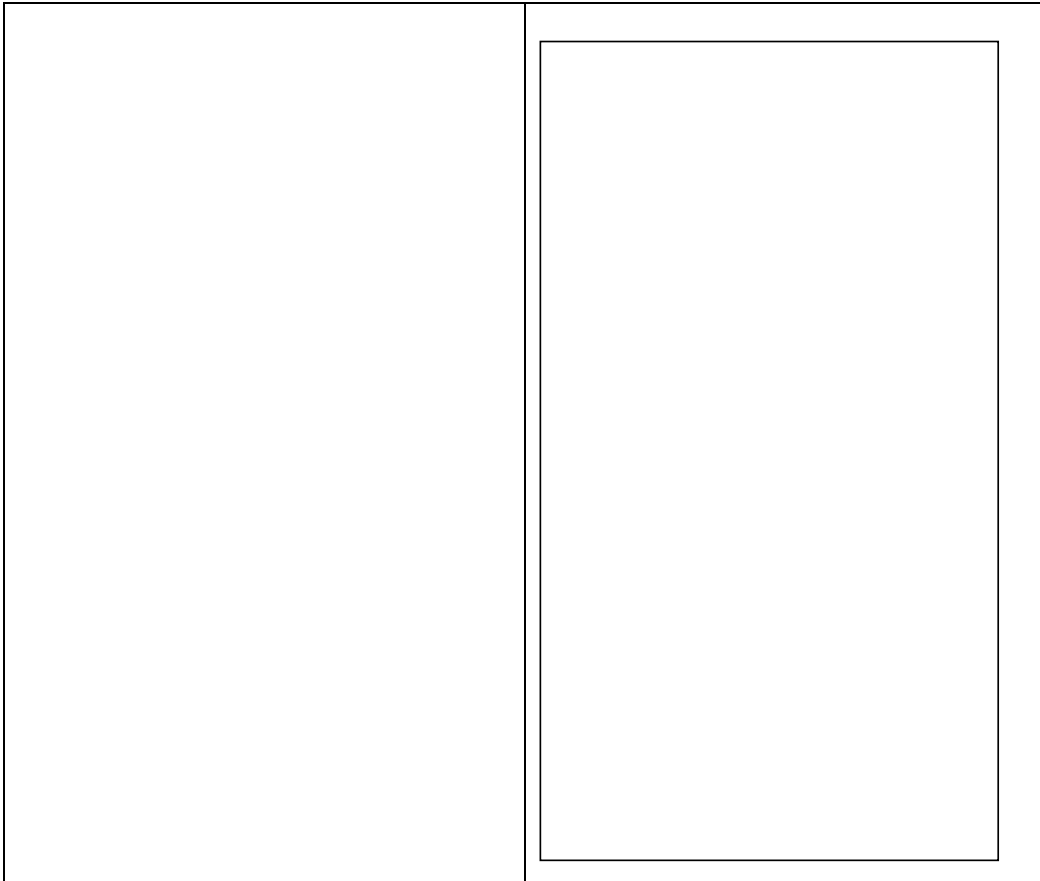
- How to create a notification using the notification builder.
- How to use pending intents to respond to notification actions.
- How to update or cancel existing notifications.

What you'll do

- Create an app that sends a notification when the user taps a button in the app.
- Update the notification from a button in your app, and from an action button that's inside the notification.

App overview

Notify Me! is an app that lets the user trigger, update, and cancel a notification using the three buttons shown in the screenshots below. While you create the app, you'll experiment with notification styles, actions, and priorities.



Task 1: Create a basic notification

1.1 Create the project

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:

1. Create a member variable for the **Notify Me!** button:

```
private Button button_notify;
```

1. Create a method stub for the `sendNotification()` method:

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

2. In the `onCreate()` method, initialize the **Notify Me!** button and create an `onClick` listener 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();
    }
});
```

```
}  
});
```

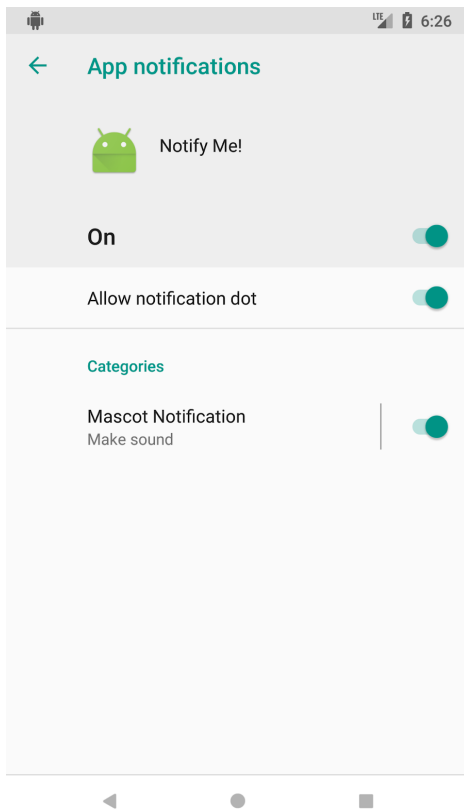
1.2 Create a notification channel

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_CHANNEL_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);
```

1.3 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 `createNotificationChannel()`. If you miss this step, your app crashes!
3. In `MainActivity.java`, create a helper method called `getNotificationBuilder()`. You use `getNotificationBuilder()` later, in the `NotificationCompat.Builder` object. Android Studio will show an error about the missing return statement, but you'll fix that soon.

```
private NotificationCompat.Builder getNotificationBuilder() {}
```

4. Inside the `getNotificationBuilder()` 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, PRIMARY_CHANNEL_ID);
```

5. Inside the `getNotificationBuilder()` method, add the title, text, and icon to the builder, as shown below. At the end, return the `Builder` object.

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

Now you can finish the `sendNotification()` method that sends the notification:

1. In `MainActivity.java`, inside the `sendNotification()` method, use `getNotificationBuilder()` to get the `Builder` object.
2. Call `notify()` on the `NotificationManager`:

```
NotificationCompat.Builder notifyBuilder = getNotificationBuilder();
mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());
```

1. Run your app. The **Notify Me!** button issues a notification, and the icon appears in the status bar. However, the notification is missing an essential feature: nothing happens when you tap it. You add that functionality in the next task.

1.4 Add a content intent and dismiss the notification

Content intents for notifications are similar to the intents you've used throughout this course. Content intents can be explicit intents to launch an activity, implicit intents to perform an action, or broadcast intents to notify the system of a system event or custom event.

The major difference with an `Intent` that's used for a notification is that the `Intent` must be wrapped in a [PendingIntent](#). The `PendingIntent` allows the Android notification system to perform the assigned action on behalf of your code.

In this step you update your app so that when the user taps the notification, your app sends a content intent that launches the `MainActivity`. (If the app is open and active, tapping the notification will not have any effect.)

1. In `MainActivity.java`, in the beginning of the `getNotificationBuilder()`, create an explicit intent method to launch the `MainActivity`:

```
Intent notificationIntent = new Intent(this, MainActivity.class);
```

2. Inside `getNotificationBuilder()`, after the `notificationIntent` declaration, use the [getActivity\(\)](#) method to get a `PendingIntent`. Pass in the notification ID constant for the `requestCode` and use the [FLAG_UPDATE_CURRENT](#) flag.

By using a `PendingIntent` to communicate with another app, you are telling that app to execute some predefined code at some point in the future. It's like the other app can perform an action on behalf of your app.

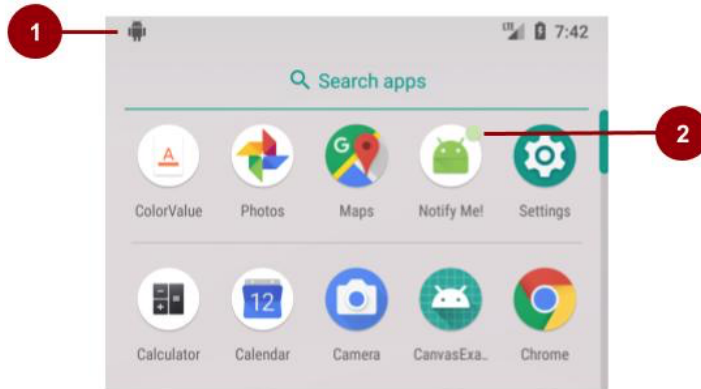
```
PendingIntent notificationPendingIntent =
PendingIntent.getActivity(this,
    NOTIFICATION_ID, notificationIntent,
    PendingIntent.FLAG_UPDATE_CURRENT);
```

3. Use the [setContentIntent\(\)](#) method from the `NotificationCompat.Builder` class to set the content intent. Inside `getNotificationBuilder()`, call `setContentIntent()` in the code that's building the notification. Also set auto-cancel to `true`:

```
.setContentIntent(notificationPendingIntent)
.setAutoCancel(true)
```

Setting auto-cancel to `true` closes the notification when user taps on it.

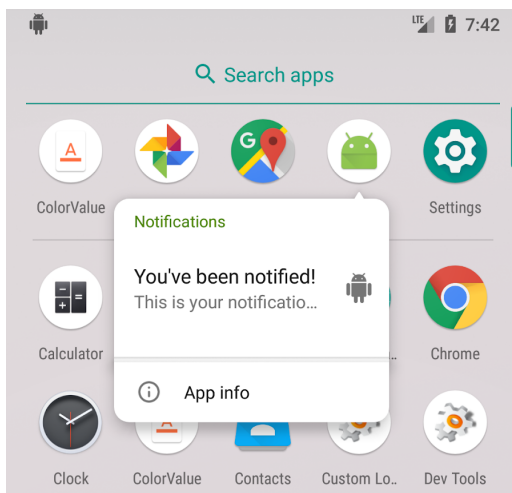
1. Run the app. Tap the **Notify Me!** button to send the notification. Tap the home button. Now view the notification and tap it. Notice the app opens back at the `MainActivity`.
2. If you are running the app on a device or emulator with API 26 or higher, press the Home button and open the app launcher. Notice the badge (the notification dot) on the app icon.



In the screenshot above:

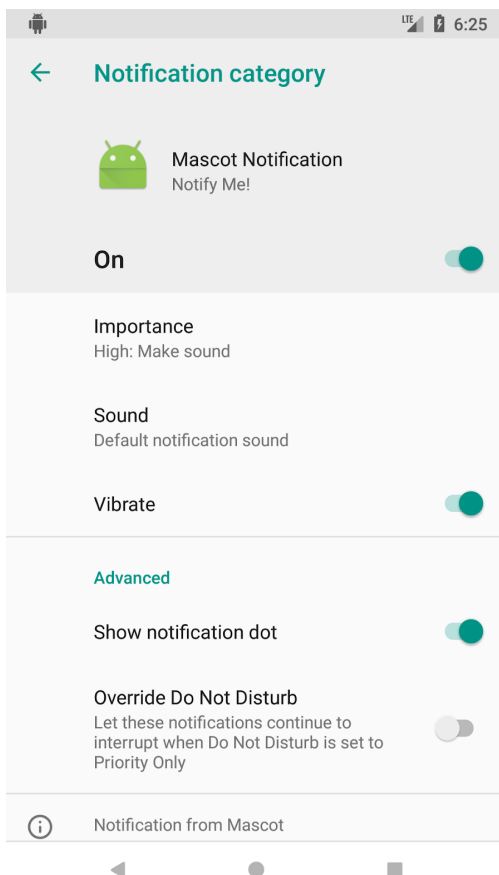
1. Notification in the status bar
2. Notification dot on the app icon (only in API 26 or higher)

When the user touches and holds the app icon, a popup shows notifications along with the icon.



If you're running on a device or emulator with API 26 or higher, here's how to view the notification channel that you created:

1. Open the device's Settings app.
2. In the search bar, enter your app name, "Notify Me!"
3. Open **Notify Me! > App Notifications > Mascot Notifications**. Use this setting to customize the notification channel. The notification channel's description is displayed at the bottom of the screen.



1.5 Add priority and defaults to your notification for backward compatibility

Note: This task is required for devices running Android 7.1 or lower, which is most Android-powered devices. For devices running Android 8.0 and higher, you use notification channels to add priority and defaults to notifications, but it's a best practice to provide backward compatibility and support for lower-end devices.

When the user taps the **Notify Me!** button in your app, the notification is issued, but the only visual that the user sees is the icon in the notification bar. To catch the user's attention, set notification default options.

Priority is an integer value from [PRIORITY_MIN](#) (-2) to [PRIORITY_MAX](#) (2). Notifications with a higher priority are sorted above lower priority ones in the notification drawer. `HIGH` or `MAX` priority notifications are delivered as "heads up" notifications, which drop down on top of the user's active screen. It's not a good practice to set all your notifications to `MAX` priority, so use `MAX` sparingly.

1. Inside the `getNotificationBuilder()` method, set the priority of the notification to `HIGH` by adding the following line to the notification builder object:

```
.setPriority(NotificationCompat.PRIORITY_HIGH)
```

1. Set the sound, vibration, and LED-color pattern for your notification (if the user's device has an LED indicator) to the default values.

Inside `getNotificationBuilder()`, add the following line to your `notifyBuilder` object:

```
.setDefaults(NotificationCompat.DEFAULT_ALL)
```

1. To see the changes, quit the app and run it again from Android Studio. If you are unable to see your changes, uninstall the app and install it again.

Note: The high-priority notification will not drop down in front of the active screen unless both the priority and the defaults are set. Setting the priority alone is not enough.

Task 2: Update or cancel the notification

After your app issues a notification, it's useful for your app to be able to update or cancel the notification if the information changes or becomes irrelevant.

In this task, you learn how to update and cancel a notification.

2.1 Add an update button and a cancel button

1. In your `activity_main.xml` layout file, create two copies of the **Notify Me!** button. In the design editor, constrain the new buttons to each other and to their parent, so that they don't overlap each other.
2. Change the `android:text` attribute in the new buttons to "Update Me!" and "Cancel Me!"
3. Change the `android:id` attributes for the buttons to `update` and `cancel`.

```
<Button
    android:id="@+id/notify"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Notify Me!"
    app:layout_constraintBottom_toTopOf="@+id/update"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/update"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Update Me!"
    app:layout_constraintBottom_toTopOf="@+id/cancel"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
```



```
        app:layout_constraintTop_toBottomOf="@+id/notify" />

<Button
    android:id="@+id/cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Cancel Me!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/update" />
```

1. Extract all the text strings to `strings.xml`.

Do the following steps in the `MainActivity.java` file:

1. Add a member variable for each of the new buttons.

```
private Button button_cancel;
private Button button_update;
```

1. At the end of `onCreate()` method, initialize the button variables and set their `onClick` listeners. If Android Studio throws an error, rebuild your project

```
button_update = findViewById(R.id.update);
button_update.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Update the notification
    }
});

button_cancel = findViewById(R.id.cancel);
button_cancel.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Cancel the notification
    }
});
```

2. Create methods for updating and canceling the notification. The methods take no parameters and return `void`:

```
public void updateNotification() {}  
public void cancelNotification() {}
```

3. In the `onCreate()` method, call `updateNotification()` in the update button's `onClick` method. In the cancel button's `onClick` method, call `cancelNotification()`.

2.2 Implement the cancel and update notification methods

To cancel a notification, call `cancel()` on the `NotificationManager`, passing in the notification ID.

1. In `MainActivity.java`, inside the `cancelNotification()` method, add the following line:

```
mNotificationManager.cancel(NOTIFICATION_ID);
```

2. Run the app.
3. Tap the **Notify Me!** button to send the notification. Notice that the notification icon appears in the status bar.
4. Tap the **Cancel Me!** button. The notification should be canceled.

Updating a notification is more complex than canceling a notification. Android notifications come with styles that can condense information. For example, the Gmail app uses `InboxStyle` notifications if the user has more than one unread message, which condenses the information into a single notification.

In this example, you update your notification to use [BigPictureStyle](#), which allows you to include an image in the notification.

1. Download [this image](#) to use in your notification and rename it to `mascot_1`. If you use your own image, make sure that its aspect ratio is 2:1 and its width is 450 dp or less.
2. Put the `mascot_1` image in the `res/drawable` folder.

3. In `MainActivity.java`, inside the `updateNotification()` method, convert your drawable into a bitmap.

```
Bitmap androidImage = BitmapFactory
    .decodeResource(getResources(), R.drawable.mascot_1);
```

4. Inside `updateNotification()`, use the `getNotificationBuilder()` method to get the `NotificationCompat.Builder` object.

```
NotificationCompat.Builder notifyBuilder = getNotificationBuilder();
```

5. Inside `updateNotification()`, after the `notifyBuilder` declaration, change the style of your notification and set the image and the title:

```
notifyBuilder.setStyle(new NotificationCompat.BigPictureStyle()
    .bigPicture(androidImage)
    .setBigContentTitle("Notification Updated!"));
```

Note: `BigPictureStyle` is a subclass of [NotificationCompat.Style](#) which provides alternative layouts for notifications. For other defined subclasses, see the documentation.

6. Inside `updateNotification()`, after setting the notification style, build the notification and call `notify()` on the `NotificationManager`. Pass in the same notification ID as before.

```
mNotifyManager.notify(NOTIFICATION_ID, notifyBuilder.build());
```

7. Run your app. Tap the update button and check the notification again—the notification now has the image and the updated title! To shrink back to the regular notification style, pinch the extended notification.

2.3 Toggle the button state

In this app, the user can get confused because the state of the notification is not tracked inside the activity. For example, the user might tap **Cancel Me!** when no notification is showing.

You can fix this by enabling and disabling the buttons depending on the state of the notification:

- When the app is first run, the **Notify Me!** button should be the only button enabled, because there is no notification yet to update or cancel.
- After a notification is sent, the cancel and update buttons should be enabled, and the notification button should be disabled, because the notification has been delivered.
- After the notification is updated, the update and notify buttons should be disabled, leaving only the cancel button enabled.
- If the notification is canceled, the buttons should return to their initial states, with only the notify button enabled.

To toggle the button state for all the buttons, do the following steps in `MainActivity.java`:

1. Add a utility method called `setNotificationButtonState()` to toggle the button states:

```
void setNotificationButtonState(Boolean isNotifyEnabled,
                                Boolean isUpdateEnabled,
                                Boolean isCancelEnabled) {
    button_notify.setEnabled(isNotifyEnabled);
    button_update.setEnabled(isUpdateEnabled);
    button_cancel.setEnabled(isCancelEnabled);
}
```

1. At the end of each of the relevant methods, add a call to `setNotificationButtonState()` to enable and disable the buttons as appropriate.

`onCreate()`:

```
setNotificationButtonState(true, false, false);
```

```
sendNotification():
```

```
setNotificationButtonState(false, true, true);
```

```
updateNotification():
```

```
setNotificationButtonState(false, false, true);
```

```
cancelNotification():
```

```
setNotificationButtonState(true, false, false);
```

Task 3: Add a notification action button

Sometimes, a notification requires interaction from the user. For example, the user might snooze an alarm or reply to a text message. When these types of notifications occur, the user might tap the notification to respond to the event. Android then loads the relevant activity in your app.

To avoid opening your app, the notification framework lets you embed a notification action button directly in the notification itself.

An action button needs the following components:

- An icon, to be placed in the notification.
- A label string, to be placed next to the icon.
- A `PendingIntent`, to be sent when the user taps the notification action.

In this task, you add an action button to your notification. The action button lets the user update the notification from within the notification, without opening the app. This **Update Notification** action works whether your app is running in the foreground or the background.

3.1 Implement a broadcast receiver that calls updateNotification()

In this step you implement a broadcast receiver that calls the `updateNotification()` method when the user taps an **Update Notification** action button inside the notification.

1. In `MainActivity.java`, add a subclass of `BroadcastReceiver` as an inner class. Override the `onReceive()` method. Don't forget to include an empty constructor:

```
public class NotificationReceiver extends BroadcastReceiver {

    public NotificationReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // Update the notification
    }
}
```

2. In the `onReceive()` method of the `NotificationReceiver`, call `updateNotification()`.
3. In `MainActivity.java`, create a unique constant member variable to represent the update notification action for your broadcast. Make sure to prefix the variable value with your app's package name to ensure its uniqueness:

```
private static final String ACTION_UPDATE_NOTIFICATION =
    "com.example.android.notifyme.ACTION_UPDATE_NOTIFICATION";
```

4. In `MainActivity.java`, create a member variable for your receiver and initialize it using the default constructor.

```
private NotificationReceiver mReceiver = new NotificationReceiver();
```

5. To receive the `ACTION_UPDATE_NOTIFICATION` intent, register your broadcast receiver in the `onCreate()` method:

This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/). This PDF is a one-time snapshot. See developer.android.com/courses/fundamentals-training/toc-v2 for the latest updates.

```
registerReceiver(mReceiver, new IntentFilter(ACTION_UPDATE_NOTIFICATION));
```


6. To unregister your receiver, override the `onDestroy()` method of your Activity:

```
@Override
protected void onDestroy() {
    unregisterReceiver(mReceiver);
    super.onDestroy();
}
```

Note: It may seem as if the broadcast sent by the notification only concerns your app and should be delivered with a `LocalBroadcastManager`. However, using a `PendingIntent` delegates the responsibility of delivering the notification to the Android framework. Because the Android runtime handles the broadcast, you cannot use `LocalBroadcastManager`.

3.2 Create an icon for the update action

To create an icon for the update-action button:

1. In Android Studio, select **File > New > Image Asset**.
2. In the **Icon Type** drop-down list, select **Action Bar and Tab Icons**.
3. Click the **Clip Art** icon.
4. Select the update icon  and click **OK**.
5. In the **Name** field, name the icon `ic_update`.

*This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).
This PDF is a one-time snapshot. See developer.android.com/courses/fundamentals-training/toc-v2
for the latest updates.*

6. Click **Next**, then **Finish**.

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

3.3 Add the update action to the notification

In `MainActivity.java`, inside the `sendNotification()` method, implement the following steps:

1. At the beginning of the method, create an `Intent` using the custom update action `ACTION_UPDATE_NOTIFICATION`.
2. Use `getBroadcast()` to get a `PendingIntent`. To make sure that this pending intent is sent and used only once, set [FLAG_ONE_SHOT](#).

```
Intent updateIntent = new Intent(ACTION_UPDATE_NOTIFICATION);
PendingIntent updatePendingIntent = PendingIntent.getBroadcast
    (this, NOTIFICATION_ID, updateIntent,
    PendingIntent.FLAG_ONE_SHOT);
```

3. Use the [addAction\(\)](#) method to add an action to the `NotificationCompat.Builder` object, after the `notifyBuilder` definition. Pass in the icon, the label text, and the `PendingIntent`.

```
notifyBuilder.addAction(R.drawable.ic_update, "Update Notification",
    updatePendingIntent);
```

4. Run your app. Tap the **Notify Me!** button, then press the Home button. Open the notification and tap on **Update Notification** button. The notification is updated.

The user can now update the notification without opening the app!