Example 5:

Develop an application that makes use of Notification Manager.

Notification

Android Toast class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

Android notification message fills up the void in such situations. Android notification is a message that we can display to the user outside of our application's normal UI. Notifications in android are built using **NotificationCompat** library.

Creating Android Notification

A Notification is created using the **NotificationManager** class as shown below:

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

The **Notification.Builder** provides an builder interface to create an Notification object as shown below:

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

Android Notification Methods

We can set the notification properties on this builder object. Some of the frequently used methods and there descriptions are given below.

1. **Notification build()** : Combines all of the options that have been set and returns a new Notification object
2. **NotificationCompat.Builder setAutoCancel (boolean autoCancel)** : Setting this flag will make it such that the notification is automatically canceled when the user clicks it in the panel
3. **NotificationCompat.Builder setContent (RemoteViews views)** : Supplies a custom RemoteViews to use instead of the standard one

4. **NotificationCompat.Builder setContentInfo (CharSequence info)** : Sets the large text at the right-hand side of the notification
5. **NotificationCompat.Builder setContentIntent (PendingIntent intent)** : Supplies a PendingIntent to send when the notification is clicked
6. **NotificationCompat.Builder setContentText (CharSequence text)** : Sets the text (second row) of the notification, in a standard notification
7. **NotificationCompat.Builder setContentTitle (CharSequence title)** : Sets the text (first row) of the notification, in a standard notification
8. **NotificationCompat.Builder setDefaults (int defaults)** : Sets the default notification options that will be used.

   An example is;

   ```
   mBuilder.setDefaults(Notification.DEFAULT_LIGHTS |
   Notification.DEFAULT_SOUND)
   ```
9. **NotificationCompat.Builder setLargeIcon (Bitmap icon) :** Sets the large icon that is shown in the ticker and notification
10. **NotificationCompat.Builder setNumber (int number)** : Sets the large number at the right-hand side of the notification
11. **NotificationCompat.Builder setOngoing (boolean ongoing)** : Sets whether this is an ongoing notification
12. **NotificationCompat.Builder setSmallIcon (int icon)** : Sets the small icon to use in the notification layouts
13. **NotificationCompat.Builder setStyle (NotificationCompat.Style style)** : Adds a rich notification style to be applied at build time
14. **NotificationCompat.Builder setTicker (CharSequence tickerText)** : Sets the text that is displayed in the status bar when the notification first arrives
15. **NotificationCompat.Builder setVibrate (long[] pattern)** : Sets the vibration pattern to use
16. **NotificationCompat.Builder setWhen (long when)** : Sets the time that the event occurred. Notifications in the panel are sorted by this time.

PendingIntent

Android **PendingIntent** is an object that wraps up an intent object and it specifies an action to be taken place in future. In other words, **PendingIntent** lets us pass a future Intent to another application and allow that application to execute that Intent as if it had the same permissions as our application, whether or not our application is still around when the Intent is eventually invoked.

A **PendingIntent** is generally used in cases were an **AlarmManager** needs to be executed or for Notification (that we'll implement later in this tutorial). A **PendingIntent** provides a means for applications to work, even after their process exits.

For security reasons, the base Intent that is supplied to the **PendingIntent** must have the component name explicitly set to ensure it is ultimately sent there and nowhere else. Each explicit intent is supposed to be handled by a specific app component like Activity, **BroadcastReceiver** or a Service. Hence **PendingIntent** uses the following methods to handle the different types of intents:

1. **PendingIntent.getActivity()** : Retrieve a PendingIntent to start an Activity
2. **PendingIntent.getBroadcast()** : Retrieve a PendingIntent to perform a Broadcast
3. **PendingIntent.getService()** : Retrieve a PendingIntent to start a Service

An example implementation of **PendingIntent** is given below.

```
Intent intent = new Intent(this, SomeActivity.class);

// Creating a pending intent and wrapping our intent

PendingIntent pendingIntent = PendingIntent.getActivity(this, 1, intent,
PendingIntent.FLAG_UPDATE_CURRENT);

try {
   // Perform the operation associated with our pendingIntent
   pendingIntent.send();
} catch (PendingIntent.CanceledException e) {
   e.printStackTrace();
}
```

The operation associated with the **pendingIntent** is executed using the send() method.

The parameters inside the **getActivity()** method and there usages are described below :

- **this (context)** : This is the context in which the **PendingIntent** starts the activity
- **requestCode** : "1" is the private request code for the sender used in the above example. Using it later with the same method again will get back the same pending intent. Then we can do various things like cancelling the pending intent with **cancel()**, etc.
- **intent :** Explicit intent object of the activity to be launched
- **flag** : One of the **PendingIntent** flag that we've used in the above example is FLAG_UPDATE_CURRENT. This one states that if a previous **PendingIntent** already exists, then the current one will update it with the latest intent. There are many other flags like FLAG_CANCEL_CURRENT etc.

Cancelling Android Notification

We can also call the **cancel()** for a specific notification ID on the **NotificationManager**.
The **cancelAll()** method call removes all of the notifications you previously issued.