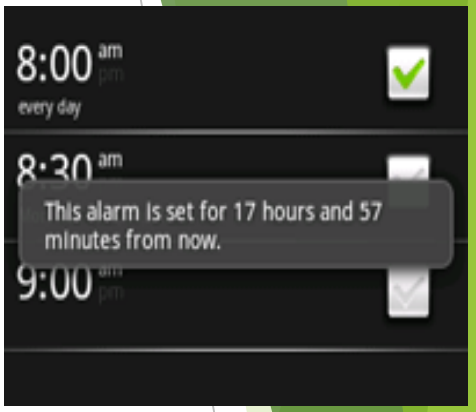
Notifications

Notifications

- Notification tasks can be achieved using a different technique:
  - A Toast Notification, for brief messages that come from the background.
  - A Status Bar Notification, for persistent reminders that come from the background and request the user's response.
  - A Dialog Notification, for Activity-related notifications.

Toast Notifications

- A toast notification is a message that pops up on the surface of the window.
- It only fills the amount of space required for the message and the user's current activity remains visible and interactive.
- The notification automatically fades in and out, and does not accept interaction events.
- A toast can be created and displayed from an Activity or Service.
- Because a toast can be created from a background Service, it appears even if the application isn't visible.



Creating Toast

Basics - Example

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Positioning your Toast

- A standard toast notification appears near the bottom of the screen, centered horizontally.
- You can change this position with the setGravity(int, int, int) method.
- Example: toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);

Creating a Custom Toast View

Creating XML Layout

To create a custom layout, define a View layout, in XML or in your application code, and pass the root View object to the setView(View) method.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:id="@+id/toast_layout_root"
              android:orientation="horizontal"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:padding="10dp"
              android:background="#DAAA"
              >
        <ImageView android:id="@+id/image"
                   android:layout_width="wrap_content"
                   android:layout_height="fill_parent"
                   android:layout_marginRight="10dp"
                   />
        <TextView android:id="@+id/text"
                  android:layout_width="wrap_content"
                  android:layout_height="fill_parent"
                  android:textColor="#FFF"
                  />
    </LinearLayout>
```

Creating a Custom Toast View

Inflating Layout from XML

To create a custom layout, define a View layout, in XML or in your application code, and pass the root View object to the setView(View) method.

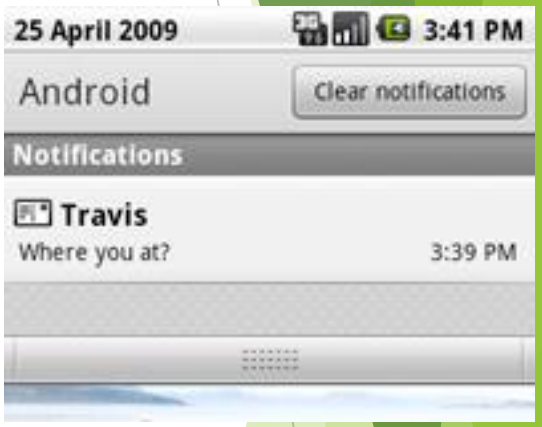
Note: Do not use the public constructor for a Toast unless you are going to define the layout with setView(View).

Custom Toast View - Example

```
LayoutInflater inflater = getLayoutInflater();
   View layout = inflater.inflate(R.layout.toast_layout,
                          (ViewGroup) findViewById(R.id.toast_layout_root));
   ImageView image = (ImageView) layout.findViewById(R.id.image);
   image.setImageResource(R.drawable.android);
   TextView text = (TextView) layout.findViewById(R.id.text);
   text.setText("Hello! This is a custom toast!");
   Toast toast = new Toast(getApplicationContext());
   toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
   toast.setDuration(Toast.LENGTH_LONG);
   toast.setView(layout);
   toast.show();
```

Creating Status Bar Notifications

- A status bar notification adds an icon to the system's status bar (with an optional ticker-text message) and an expanded message in the "Notifications" window.
- When the user selects the expanded message, Android fires an Intent that is defined by the notification (usually to launch an Activity).
- You can also configure the notification to alert the user with a sound, a vibration, and flashing lights on the device.
- Notifications are mainly used in context with services.
  A service should not launch an avtivity on it’s own to have user interaction, instead it should use a notification.



Creating Status Bar Notifications

- An Activity or Service can initiate a status bar notification.
- To create a notification, you must use two classes: Notification and NotificationManager.
- Notification class is used to define the properties of your status bar notification.
- The NotificationManager is an Android system service that executes and manages all Notifications.
- NotificationManager can not be instantiated, however we can obtain reference to it using getSystemService() and then invoke the notify() method by passing Notification object.

Managing your Notifications

- The NotificationManager is a system service that manages all notifications. You must retrieve a reference to it with the getSystemService() method.

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

- notify(int, Notification) method can be used to set status bar notification.
- The first parameter is the unique ID for the Notification and the second is the Notification object.
- This is necessary if you need to update the Notification or select the appropriate action when the user returns to your application via the Intent defined in the Notification.

Managing your Notifications

- To clear the status bar notifictions FLAG_AUTO_CANCEL flag can be set for the Notification object.
- cancel(int) method can be used to clear the Notification manually.
- cancelAll() method can be used to cancel all the notifications.

Creating a Notification

- A status bar notification *requires* all of the following:
  - An icon for the status bar
  - A title and expanded message for the expanded view
  - A PendingIntent, to be fired when the notification is selected
- Optional settings for the status bar notification include:
  - A ticker-text message for the status bar
  - An alert sound
  - A vibrate setting
  - A flashing LED setting

Creating Notification

- Notification(int, CharSequence, long) constructor and the setLatestEventInfo(Context, CharSequence, CharSequence, PendingIntent) methods define all the required settings for a Notification.

Notification Creation

```
int icon = R.drawable.notification_icon;    // icon from resources
CharSequence tickerText = "Hello";          // ticker-text
long when = System.currentTimeMillis();       // notification time
Context context = getApplicationContext();   // application Context
CharSequence contentTitle = "My notification";  // expanded message title
CharSequence contentText = "Hello World!";     // expanded message text
    Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
    notificationIntent, 0);
// the next two lines initialize the Notification, using the configurations
    above
Notification notification = new Notification(icon, tickerText, when);
notification.setLatestEventInfo(context, contentTitle, contentText,
    contentIntent);
```

Updating the notification

- You can update the information in your status bar notification.
- Each notification is uniquely identified by the NotificationManager with an integer ID.
- setLatestEventInfo() with new values can be used to update the notification.
  Change some field values of the Notification, and then call notify() .

Adding a sound

- You can alert the user with the default notification sound (which is defined by the user) or with a sound specified by your application.
- To use the user's default sound, add "DEFAULT_SOUND" to the *defaults* field:

```
notification.defaults |= Notification.DEFAULT_SOUND;
```

- Using different sound with notification:

```
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
```

- the audio file is chosen from the internal MediaStore's ContentProvider:

```
notification.sound =
    Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

Adding vibration

- To use the default pattern, add "DEFAULT_VIBRATE" to the *defaults* field:

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
```

- To Define your own vibration pattern, pass an array of *long* values to the *vibrate* field:

```
long[] vibrate = {0,100,200,300};
    notification.vibrate = vibrate;
```

- The long array defines the alternating pattern for the length of vibration off and on (in milliseconds).
- The pattern can be as long as you like, but it can't be set to repeat.

Adding flashing lights

- To use the default light setting, add "DEFAULT_LIGHTS" to the *defaults* field:

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

More Features

- "FLAG_AUTO_CANCEL" flag
  - Add this to the *flags* field to automatically cancel the notification after it is selected from the Notifications window.
- "FLAG_INSISTENT" flag
  - Add this to the *flags* field to repeat the audio until the user responds.
- "FLAG_ONGOING_EVENT" flag
  - Add this to the *flags* field to group the notification under the "Ongoing" title in the Notifications window.
- "FLAG_NO_CLEAR" flag
  - Add this to the *flags* field to indicate that the notification should *not* be cleared by the "Clear notifications" button.