



<http://www.android.com/>

More about sensors

Alarms, notifications and remote views

App widget providers

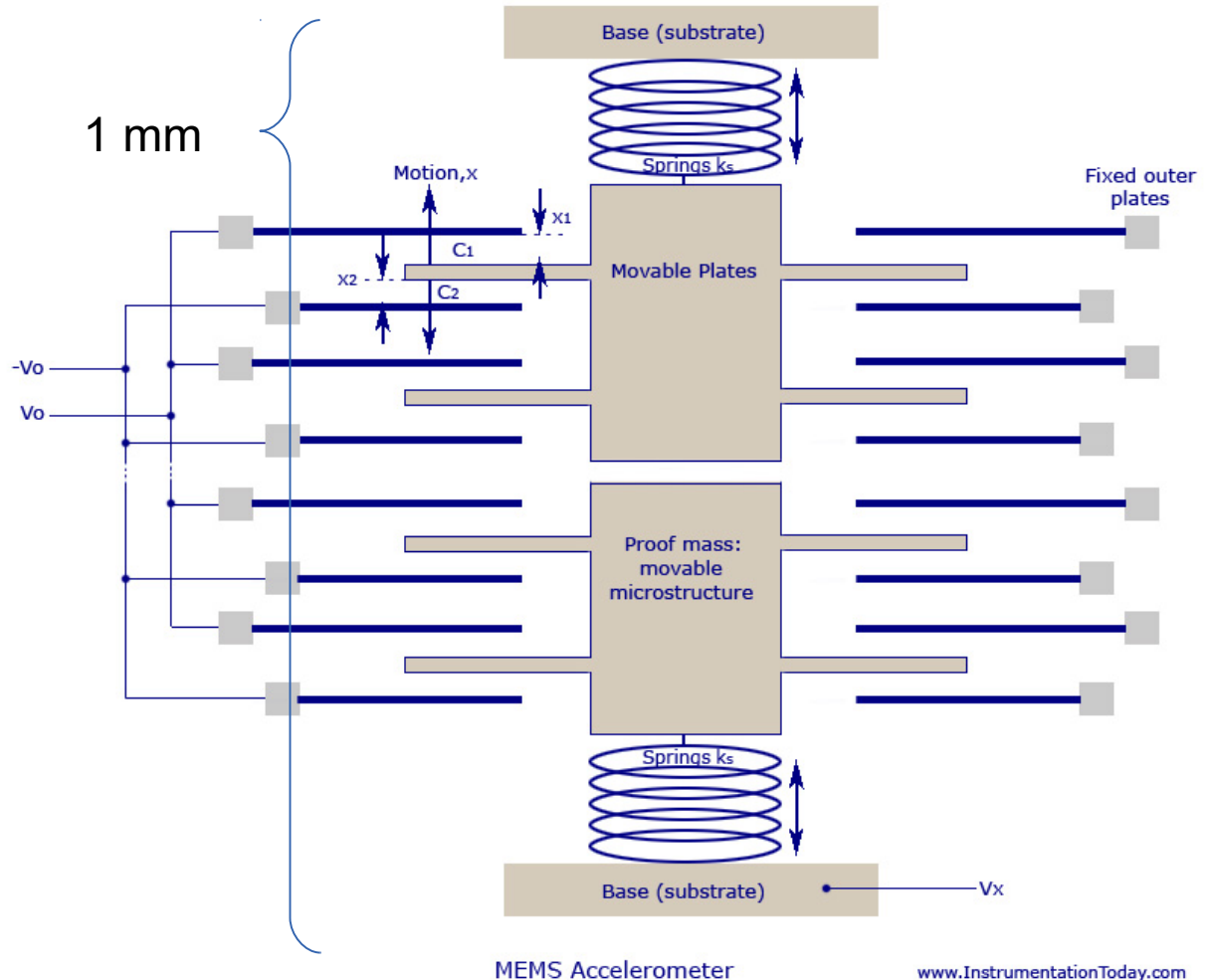
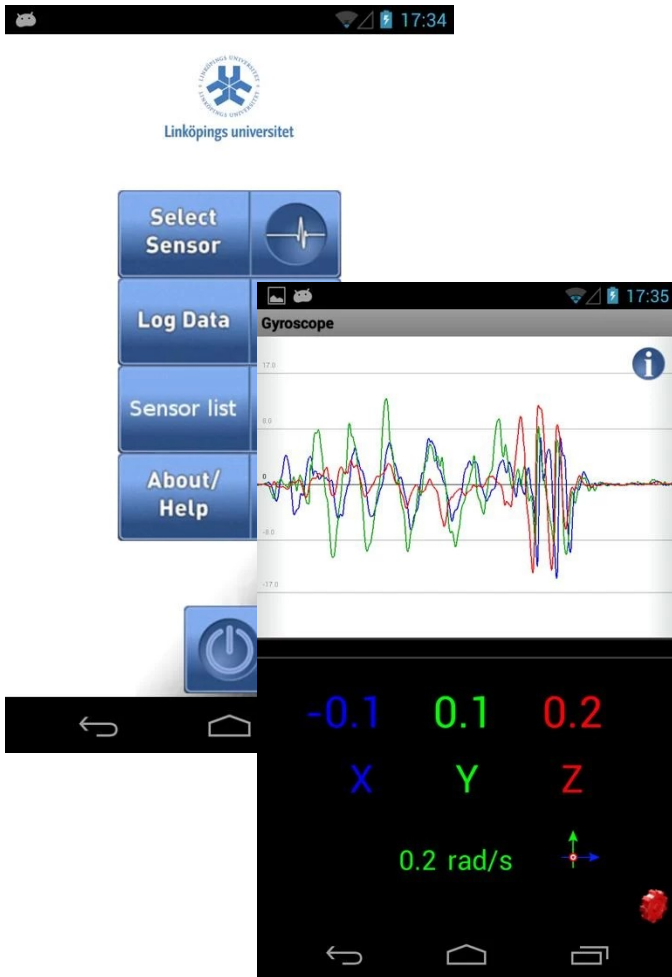
Testing the app and general guidelines

Publish the app

Sensors



- Often small Micro Electro-Mechanical Systems (MEMS)



<https://play.google.com/store/apps/details?id=com.hiq.sensor&hl=sv>

Hardware vs. virtual sensors



- Hardware sensors
 - A real sensor which deliver raw data from the sensor
 - See examples on next slide
- Virtual sensors
 - A single sensor or compund sensors which delivers filtered or mixed data in some way
 - For example the gravity sensor (`Sensor.TYPE_GRAVITY`) delivers accelerometer data which have been filtered with a low pass filter
 - A low-pass filter is a filter that passes low-frequency signals and attenuates (reduces the amplitude of) signals with frequencies higher than the cutoff frequency
- Other virtual sensors
 - `TYPE_LINEAR_ACCELERATION`, `TYPE_ORIENTATION`, `TYPE_ROTATION_VECTOR`
 - Sensor Fusion on Android Devices: A Revolution in Motion Processing
 - <http://www.youtube.com/watch?v=C7JQ7Rpwn2k>

Using sensors 1



- The Android SDK supports many different types of sensor devices (14+ sensors)
 - TYPE_ACCELEROMETER: Measures acceleration in the x-, y-, and z-axes
 - TYPE_LIGHT: Tells you how bright your surrounding area is
 - TYPE_MAGNETIC_FIELD: Returns magnetic attraction in the x-, y-, and z-axes
 - TYPE_GYROSCOPE: Measures the yaw, pitch, and roll of the device
 - TYPE_PRESSURE: Senses the current atmospheric pressure
 - TYPE_PROXIMITY: Provides the distance between the sensor and some object
 - TYPE_AMBIENT_TEMPERATURE: Measures the temperature of the surrounding area
 - TYPE_RELATIVE_HUMIDITY: Measures the humidity of the surrounding area
 - **Full list: <http://developer.android.com/reference/android/hardware/Sensor.html>**

```
private SensorManager mgr;
private final LinkedList<float[]> mFifo;
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mgr.registerListener(mySensorListener, mgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_GAME);
    // ...

    // can also be a whole class which implements SensorEventListener
    private final SensorEventListener mySensorListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event)
            if (event.sensor.getType() != Sensor.TYPE_ACCELEROMETER)
                return;
            float[] val = new float[] { event.values[0], event.values[1],
                event.values[2] };
            mFifo.add(val);
            long nsTime = event.timestamp;
    }
}
```

Remember that do not spend too much time in callback methods or receivers!

Using sensors 2



- Sensor frequency
 - `SensorManager.SENSOR_DELAY_FASTEST`, ($\approx 80 - 500$ Hz)
 - `SENSOR_DELAY_GAME`, `SENSOR_DELAY_NORMAL`,
 - `SENSOR_DELAY_UI` (≈ 5 Hz)
 - Or a specified interval: `int samplingPeriodUs`
- `event.values` - double-array
 - For the number of values and content see the API docs for given sensor type
 - `TYPE_ACCELEROMETER`: (A_x, A_y, A_z), [m/s^2]
 - `TYPE_PROXIMITY`: distance, [cm]
- `event.timestamp` - long
 - The time in nanosecond at which the event happened

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
    //...
}
};
@Override
protected void onPause() {
    sensorManager.unregisterListener(mySensorListener);
    super.onPause();
}
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(mySensorListener, ...);
}
```

Rotation and orientation?

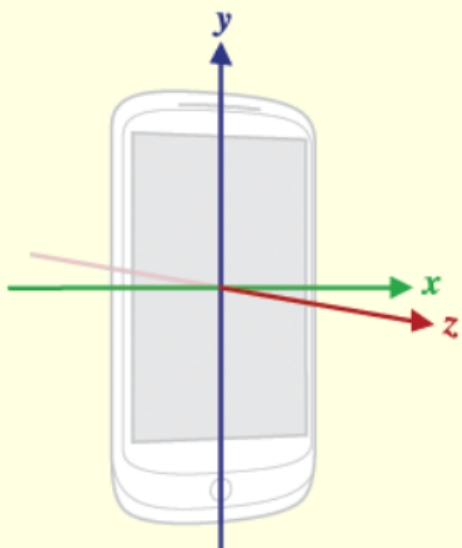


android.hardware.SensorEvent

Definition of the coordinate system used by the SensorEvent API.

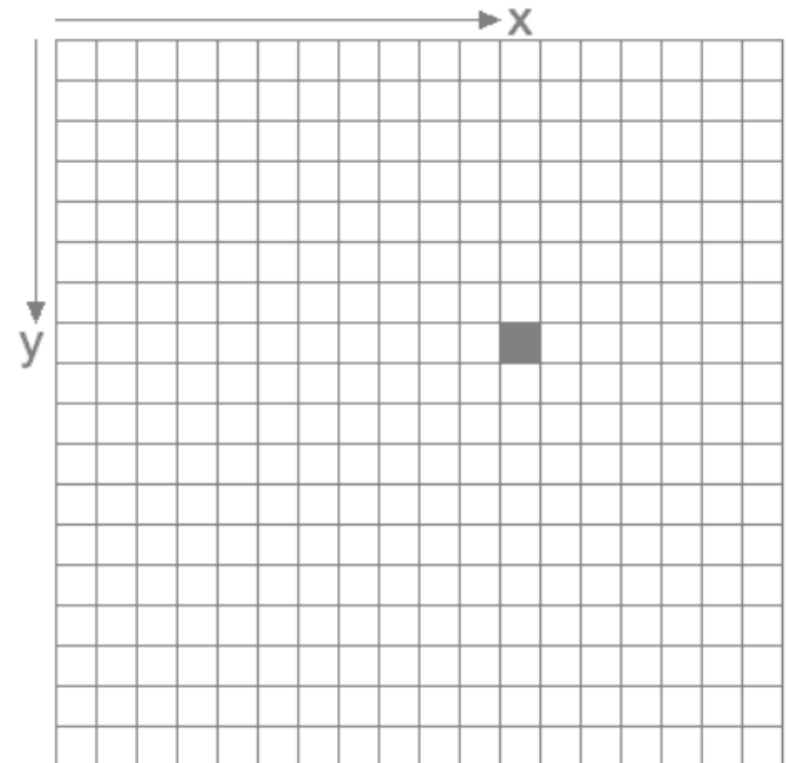
The coordinate-system is defined relative to the screen of the phone in its default orientation. The axes are not swapped when the device's screen orientation changes.

The X axis is horizontal and points to the right, the Y axis is vertical and points up and the Z axis points towards the outside of the front face of the screen. In this system, coordinates behind the screen have negative Z values.



Note: This coordinate system is different from the one used in the Android 2D APIs where the origin is in the top-left corner.

- Coordinates on the screen (2D API) are as usual "pixel" based



Portrait or landscape, which side is up?



- There are some classes that can be used to find out
 - **Display** - Provides information about the size and density of a logical display
 - **Configuration** - This class describes all device configuration information that can impact the resources the application retrieves
 - **Surface** - Handle onto a raw buffer that is being managed by the screen compositor

```
// Overall orientation of the screen
// May be one of Configuration.ORIENTATION_LANDSCAPE, Configuration.ORIENTATION_PORTRAIT.
int orientation = getResources().getConfiguration().orientation; // 1 = portrait, 2 = landscape

if(orientation == Configuration.ORIENTATION_PORTRAIT)
{ ... }
// we use this in accelerometer onSensorChanged(SensorEvent event)
Display display = ((WindowManager) getSystemService(Context.WINDOW_SERVICE)).getDefaultDisplay();
switch (display.getRotation())
{
    case Surface.ROTATION_0:
        x = event.values[0]; y = event.values[1];
        break;
    case Surface.ROTATION_90:
        x = -event.values[1]; y = event.values[0];
        break;
    case Surface.ROTATION_180:
        x = -event.values[0]; y = -event.values[1];
        break;
    case Surface.ROTATION_270:
        x = event.values[1]; y = -event.values[0];
        break;
}
```

Check out the API Demos samples for more sensor examples

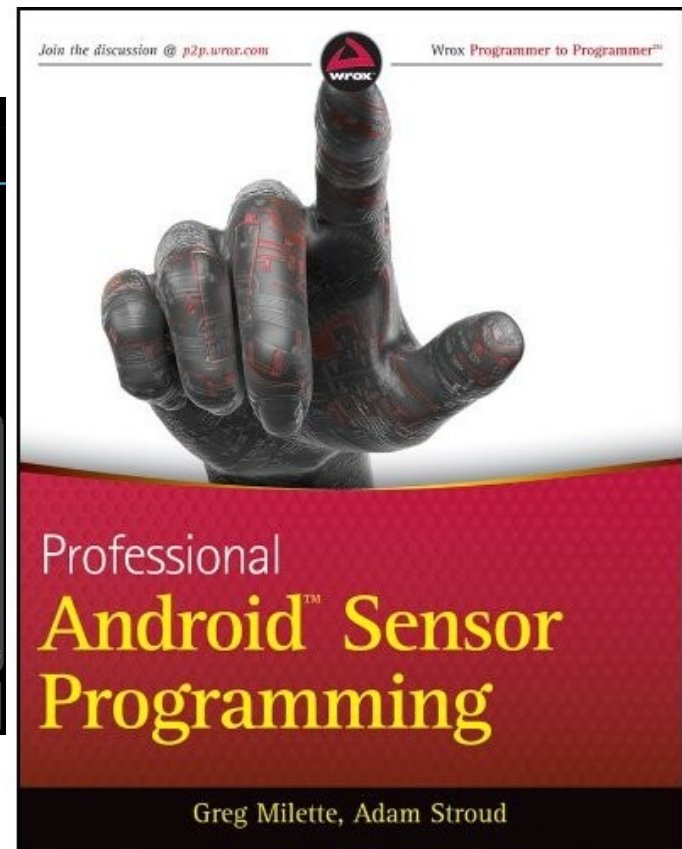
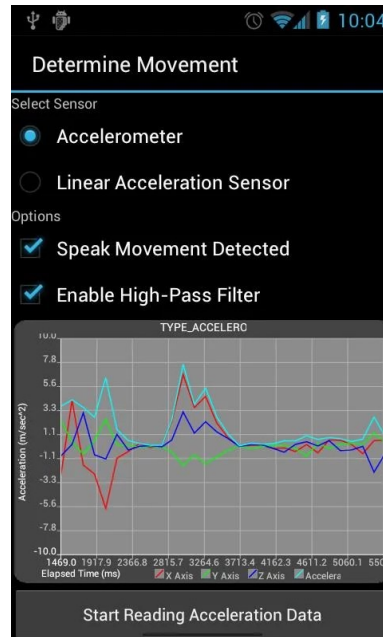
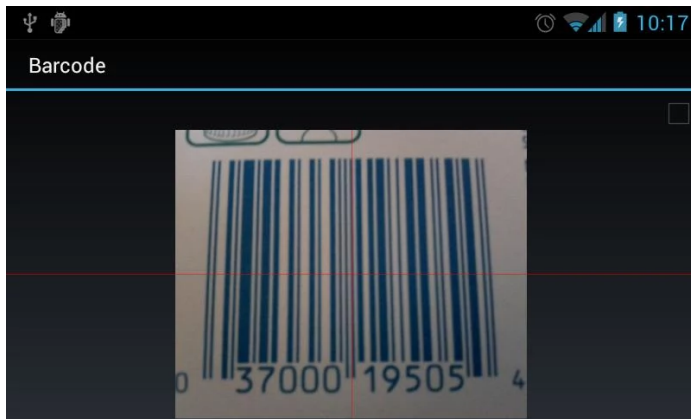
More about sensors

http://developer.android.com/guide/topics/sensors/sensors_overview.html

Sensors book



- <http://www.wrox.com/WileyCDA/WroxTitle/Professional-Android-Sensor-Programming.productCd-1118183487.html>



Source code on:
<https://github.com/gast-lib>

<https://play.google.com/store/apps/details?id=root.gast.playground>

Configuration changes at runtime

- Unless you specify otherwise, a configuration change at runtime will cause your current activity to be destroyed, examples:
 - Change in screen orientation, Language, Input devices, ..., (defined in the `android.content.res.Configuration` class)
- If the activity had been in the foreground or visible to the user, a new instance of the activity will be created, and resource values reloaded
 - Save UI-state in `onPause()` or `onSaveInstanceState()`
- You can handle configuration changes in the code by yourself

```
<!-- This supresses the Activity to be destroyed and restarted for these config changes -->
<activity
    android:label="@string/app_name"
    android:configChanges="orientation|screenSize" />
```

/ Called by the system when the device configuration changes while your activity is running. Note that this will only be called if you have selected configurations you would like to handle with the configChanges attribute in your manifest.*

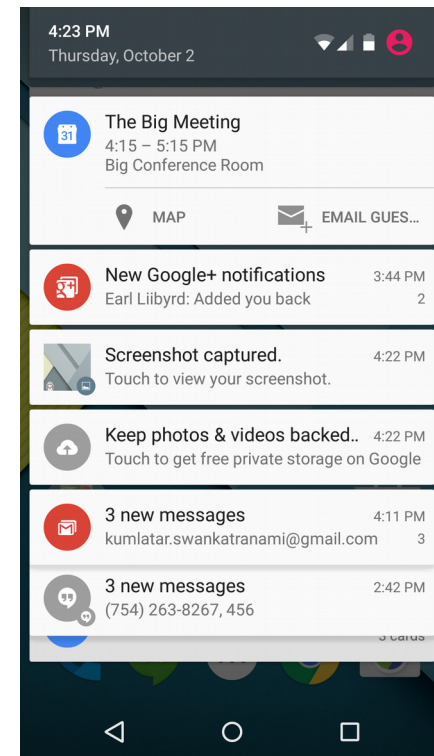
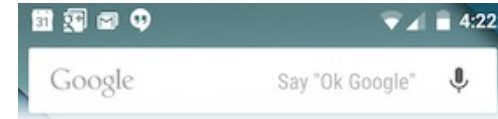
*If any configuration change occurs that is not selected to be reported by that attribute, then instead of reporting it the system will stop and restart the activity (to have it launched with the new configuration). */*

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    // Update all UIs based on resource values, they might have been changed by the new config
    if(newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Log.d("test", newConfig.toString());
    }
}
```

Notifications



- A notification is a message you can display to the user outside of your application's normal UI
 - When you tell the system to issue a notification, it first appears as an icon in the notification area
 - To see the details of the notification, the user opens the notification drawer
 - Both the notification area and the notification drawer are system-controlled areas that the user can view at any time
- A Notification object **must** contain the following
 - A small icon, set by `setSmallIcon()`
 - A title, set by `setContentTitle()`
 - A detail text, set by `setContentText()`
- Notifications design patterns and guide
 - <http://developer.android.com/design/patterns/notifications.html>
 - <http://developer.android.com/guide/topics/ui/notifiers/notifications.html>



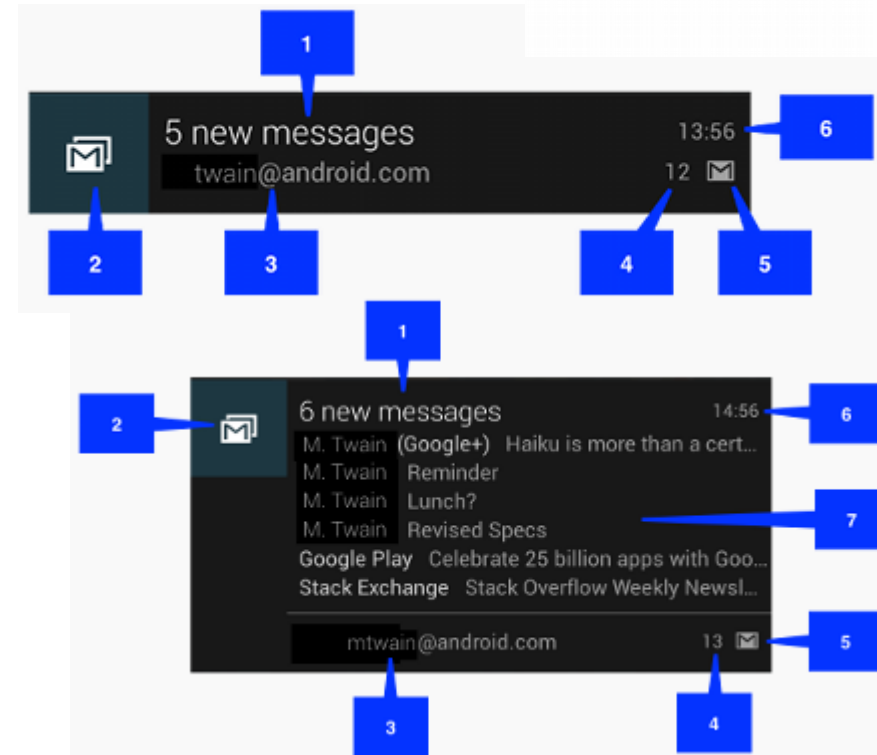
Notifications (API 16-20)



- Normal view and big/expanded view (expanded only \geq API 16)

- 1. Content title
- 2. Large icon
- 3. Content text
- 4. Content info
- 5. Small icon
- 6. Time that the notification was issued
- 7. Details area with a specific style (only big)

- Android \geq 4.1/ API 16 use the Notification.Builder()
 - Uses simpler Builder.setMethods() for the configuration
 - With support library: NotificationCompat.Builder() for older APIs



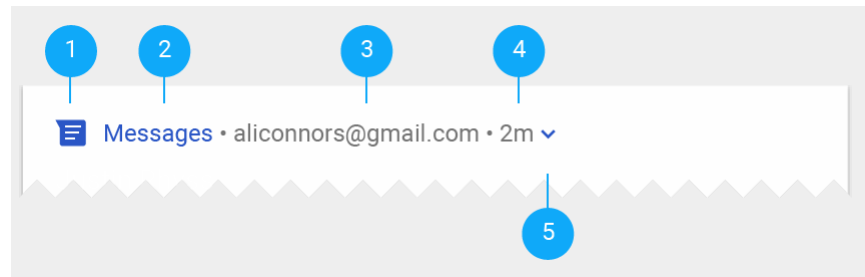
Notifications (API 21- ...)



- Notifications have evolved and changed quite a bit with later API versions ≥ 21 and beyond
 - The methods and syntax have however not changed very much
 - On API ≥ 26 one or more notification channels **must** be used!

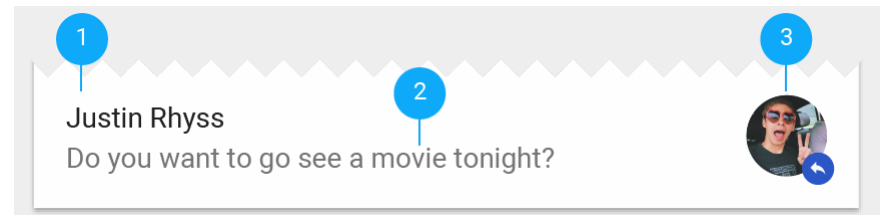
- Header area

- 1. App icon
- 2. App name
- 3. Header text
- 4. Timestamp (optional)
- 5. Expand indicator



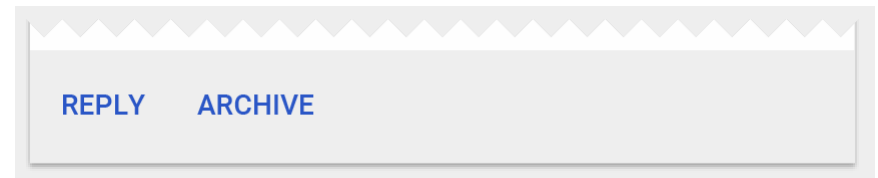
- Content area

- 1. Content title
- 2. Content text
- 3. Large icon (optional)



- Action area

- When expanded up to 3 actions may be seen



Post to a notification channel



- Creating a notification channel and creating a simple notification (API >= 26)

```
NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
String CHANNEL_ID = "my_channel_01"; // The id of the channel
CharSequence name = getString(R.string.channel_name); // The user-visible name of the channel
String description = getString(R.string.channel_description); // The user-visible description of the channel
int importance = NotificationManager.IMPORTANCE_HIGH;
NotificationChannel mChannel = new NotificationChannel(CHANNEL_ID, name, importance);
mChannel.setDescription(description); // Configure the notification channel
mChannel.enableLights(true);
// Sets the notification light color for notifications posted to this channel, if the device supports this feature
mChannel.setLightColor(Color.RED);
mChannel.enableVibration(true);
mChannel.setVibrationPattern(new long[]{100, 200, 300, 400, 500, 400, 300, 200, 400});
mNotificationManager.createNotificationChannel(mChannel);
```

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setChannelId(CHANNEL_ID);
Intent resultIntent = new Intent(this, ResultActivity.class); // Creates an explicit intent for an Activity in your app

// The stack builder object will contain an artificial back stack for the started Activity. This ensures that navigating
// backward from the Activity leads out of your app to the Home screen which preserves the expected navigation!
// See: https://developer.android.com/guide/topics/ui/notifiers/notifications.html#NotificationResponse
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class); // Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addNextIntent(resultIntent); // Adds the Intent that starts the Activity to the top of the stack
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// mNotificationId is a unique integer your app uses to identify the notification. For example,
// to cancel the notification, you can pass its ID number to NotificationManager.cancel()
mNotificationManager.notify(mNotificationId, mBuilder.build());
```

Alarm and notification example 1.1

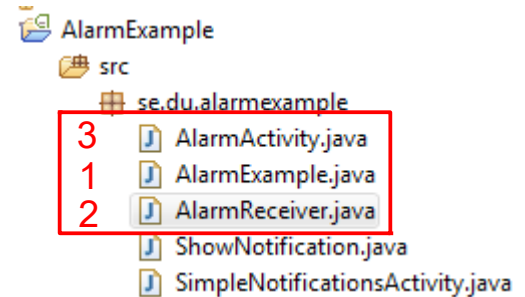


- The Activity AlarmExample executes and use the AlarmManager to set a wakeup intent with a message within 5 seconds (see: AlarmNotificationExample)

1

```
private void alarmTest()
{
    int requestCode = 192837;
    // get a Calendar object with current time
    Calendar cal = Calendar.getInstance();
    // add 5 seconds to the calendar object
    cal.add(Calendar.SECOND, 5);
    Intent intent = new Intent(this, AlarmReceiver.class);
    intent.putExtra("alarm_message", "Android Notifications Rules!");
    // Private request code for the sender (currently not used) according to
    // http://developer.android.com/reference/android/app/PendingIntent.html
    // Retrieve a PendingIntent that will perform a broadcast
    PendingIntent sender = PendingIntent.getBroadcast(this, requestCode,
        intent, PendingIntent.FLAG_UPDATE_CURRENT);

    // Get the AlarmManager service and set the alarm manager
    AlarmManager am = (AlarmManager) getSystemService(ALARM_SERVICE);
    // http://developer.android.com/reference/android/app/AlarmManager.html
    am.set(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(), sender);
}
```



Alarm and notification example 1.2

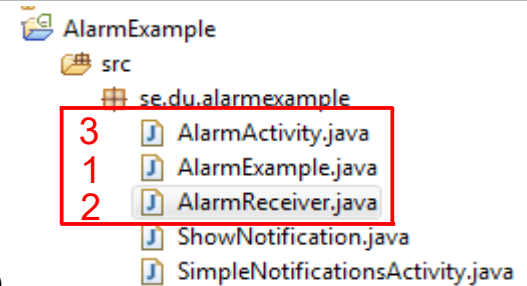


- The AlarmReceiver will get the intent and message and create a new intent/message which in turn will start the AlarmActivity
- `<receiver android:process=":remote" android:name="ALarmReceiver"></receiver>`

2

```
public class AlarmReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent) {
        try {
            Bundle bundle = intent.getExtras();
            String message = bundle.getString("alarm_message");

            Intent newIntent = new Intent(context, AlarmActivity.class);
            newIntent.putExtra("alarm_message", message);
            newIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(newIntent);
        }
        catch (Exception e) {
            Toast.makeText(context, "There was an error somewhere, " +
                "but we still received an alarm", Toast.LENGTH_SHORT).show();
            e.printStackTrace();
        }
    }
}
```



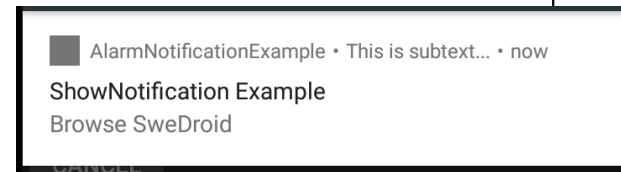
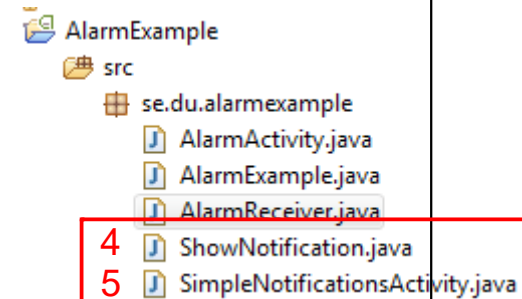
Alarm and notification example 1.3



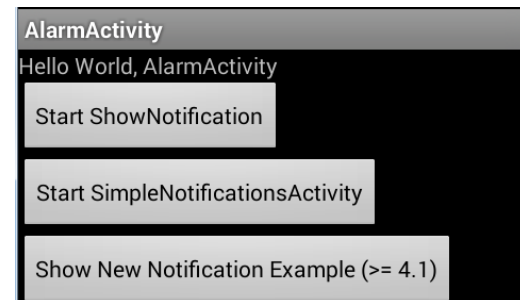
- The AlarmActivity class (3) will finally toast the message and now we can start execute different kinds of notification tests

4

```
private NotificationManager mNManager;
public void onCreate(Bundle savedInstanceState) {
    Button start = (Button)findViewById(R.id.button1);
    Button cancel = (Button)findViewById(R.id.button2);
    mNManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    // start button in the ShowNotification class
    start.setOnClickListener(new OnClickListener() {
        public void onClick(View v)
        {
            // the intents are only necessary to open the browser with the URI
            Intent msgIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://www.swedroid.se"));
            PendingIntent pendingIntent = PendingIntent.getActivity>ShowNotification.this, 0,
                msgIntent, PendingIntent.FLAG_UPDATE_CURRENT);
            // below is the actual notification
            Notification.Builder builder = new Notification.Builder>ShowNotification.this);
            builder.setDefaults(Notification.DEFAULT_SOUND);
            builder.setAutoCancel(true);
            builder.setContentTitle("ShowNotification Example");
            builder.setContentText("Browse SweDroid");
            builder.setSmallIcon(R.drawable.android_32);
            builder.setContentIntent(pendingIntent);
            builder.setSubText("This is subtext..."); //API level 16
            Notification myNotification = builder.build();
            mNManager.notify(NOTIFY_ID, myNotification);
        }
    });
    // cancel button in the ShowNotification class
    cancel.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            mNManager.cancel(NOTIFY_ID);
        }
    });
}
```



3



Notification example 2.1

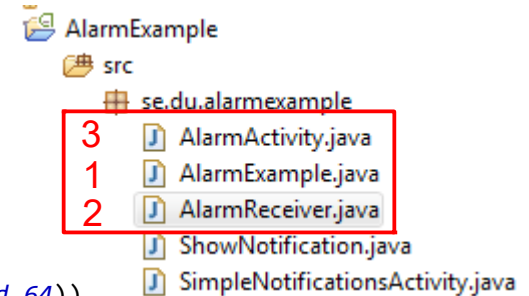


- Pressing button 6 (≥ 4.1), using the Notification.Builder

3

```
/* Creating a Notification
You specify the UI information and actions for a notification in a Notification.Builder object.
To create the notification itself, you call Notification.Builder.build(), which returns a Notification object
containing your specifications. To issue the notification, you pass the Notification object to the system by
calling NotificationManager.notify(). */
mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
private void createNewNotification(){
    /*
    A Notification object must contain the following:
    - A small icon, set by setSmallIcon()
    - A title, set by setContentTitle()
    - Detail text, set by setContentText()
    */
    mBuilder = new Notification.Builder(this)
        .setSmallIcon(R.drawable.android_32)
        .setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.android_64))
        // set defaults
        .setDefaults(Notification.DEFAULT_VIBRATE | Notification.DEFAULT_SOUND)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
    // Creates an explicit intent for an Activity in your app
    Intent resultIntent = new Intent(this, AlarmActivity.class);

    // The stack builder object will contain an artificial back stack for the started Activity.
    // This ensures that navigating backward from the Activity leads out of your application to the Home screen.
    TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
    // Adds the back stack for the Intent (but not the Intent itself)
    stackBuilder.addParentStack(AlarmActivity.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);
    mNotificationManager.notify(mId1, mBuilder.build());
}
```

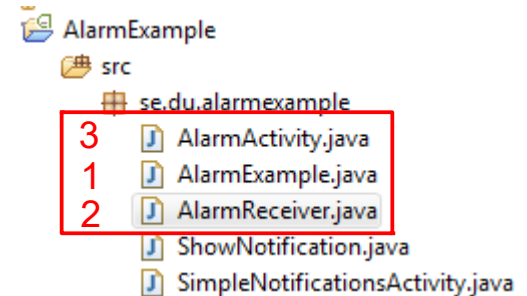


Notification example 2.2



- Pressing button 6 (≥ 4.1), display progress in a notification

```
private void createNewProgressNotification()
{
    final Notification.Builder mBuilder = new Notification.Builder(this);
    mBuilder.setContentTitle("Picture Download")
        .setContentText("Download in progress")
        .setSmallIcon(R.drawable.stat_notify_sync_anim0);
    // Start a lengthy operation in a background thread
    new Thread(new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value, the
                // current completion percentage, and "determinate" state
                mBuilder.setProgress(100, incr, false);
                // Displays the progress bar for the first time.
                mNotificationManager.notify(mId2, mBuilder.build());
                // Sleeps the thread, simulating an operation that takes time
                try {
                    Thread.sleep(1*1000); // Sleep for 1 seconds
                } catch (InterruptedException e) {
                    Log.d(TAG, "sleep failure");
                }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
                .setProgress(0, 0, false); // Removes the progress bar
            mNotificationManager.notify(mId2, mBuilder.build());
        }
    })
    // Starts the thread by calling the run() method in its Runnable
    .start();
}
```



3

Notification example 3.1



- Sometimes you want to perform longer work in the background, you can use an ongoing notification for this with possibly some kind of progress with a **remote view**

```
final int NOTIFY_ID = 434;
int progress = 10;
Context context = getApplicationContext();
Intent intent = new Intent(this, DownloadProgress.class); // configure the intent
final PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent, 0);
// configure the notification
final Notification notification = new Notification(R.drawable.icon, "simulating a download", System.currentTimeMillis());
notification.flags = notification.flags | Notification.FLAG_ONGOING_EVENT;
notification.contentView = new RemoteViews(context.getPackageName(), R.layout.download_progress);
notification.contentIntent = pendingIntent;
notification.contentView.setImageDrawable(R.id.status_icon, R.drawable.ic_menu_save);
notification.contentView.setTextViewText(R.id.status_text, "simulation in progress");
notification.contentView.setProgressBar(R.id.status_progress, 100, progress, false);
final NotificationManager notificationManager =
    (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFY_ID, notification);

// simulate progress
Thread download = new Thread() {
    @Override
    public void run() {
        for (int i = 1; i < 100; i++) {
            progress++;
            notification.contentView.setProgressBar(R.id.status_progress, 100, progress, false);
            notificationManager.notify(NOTIFY_ID, notification); // inform the progress bar of updates in progress
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        notificationManager.cancel(NOTIFY_ID); // remove the notification (we're done)
    }
};
download.run();
```

See the UCdroid example
DownloadProgress.java

Notification example 3.2



- The layout for the remote view in the notification drop-down

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="5dp" >

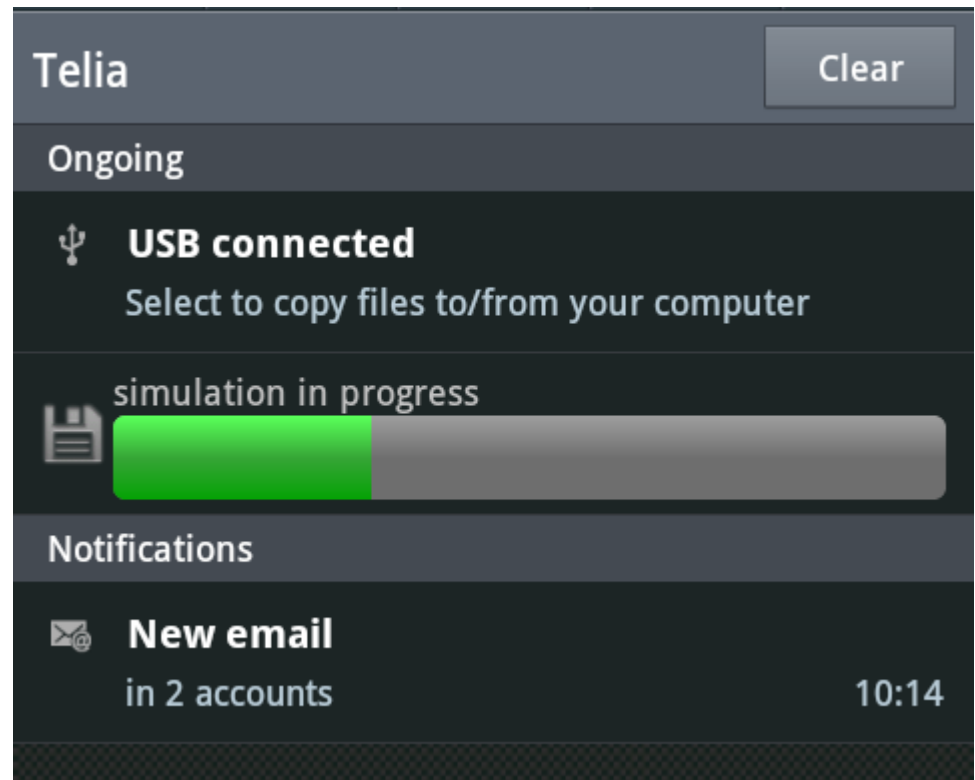
    <ImageView
        android:id="@+id/status_icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true" />

    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_toRightOf="@id/status_icon" >

        <TextView
            android:id="@+id/status_text"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true" />

        <ProgressBar
            android:id="@+id/status_progress"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_below="@id/status_text"
            android:indeterminate="false"
            android:indeterminateOnly="false"
            android:progressDrawable="@android:drawable/progress_horizontal" />

    </RelativeLayout>
</RelativeLayout>
```



NotificationActivityService example 1



- How a service can send data to an activity or receiver with a custom intent filter

```
public class SimpleIntentService extends IntentService {
    private static final String DEBUG_TAG = "SimpleIntentService";
    public static final String ACTION_NOTIFY = "se.du.notification.DATA_RETRIEVED";
    public static final String PARAM_OUT_COUNT = "count";
    private static final int NOTIFY_ID2 = 0x2;
    private Notification.Builder mBuilder;

    public SimpleIntentService() {
        super(DEBUG_TAG);
        Log.d(DEBUG_TAG, "SimpleIntentService created");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        // retrieve the url from the intent
        String url = intent.getStringExtra("url");
        createProgressNotification();
        Log.d(DEBUG_TAG, "Data received");

        int incr;
        // Do the "lengthy" operation 20 times
        for (incr = 0; incr <= 100; incr+=5) {
            try {
                if(incr%50 == 0)
                    sendSimpleBroadcast(incr);
                updateProgressNotification(incr);
                // Sleep for 1 seconds
                Thread.sleep(1*1000);
            } catch (InterruptedException e) {
                Log.d(DEBUG_TAG, "sleep failure");
            }
        }
        finishProgressNotification();
    }
}
```

NotificationActivityService example 2



- A persistent notification is also displayed

```
private void sendSimpleBroadcast(int count) {
    Log.d(DEBUG_TAG, "sendSimpleBroadcast " + count);
    Intent broadcastIntent = new Intent();
    // use same action in receiver
    broadcastIntent.setAction(SimpleIntentService.ACTION_NOTIFY);
    // use same category in receiver
    broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
    broadcastIntent.putExtra(PARAM_OUT_COUNT, count);
    // Broadcast the given intent to all interested BroadcastReceivers
    sendBroadcast(broadcastIntent, CustomPermission.SEND_SIMPLE_NOTIFICATIONS);
} // permission = se.du.notification.Permission.SEND_SIMPLE_NOTIFICATIONS
// class cont.

public class NotificationActivityService extends Activity {

    private static final String DEBUG_TAG = "SimpleNotificationActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_simple_notification);
        Intent serviceIntent = new Intent(NotificationActivityService.this, SimpleIntentService.class);
        serviceIntent.putExtra("url", "http://androidhotel.wordpress.com");
        startService(serviceIntent);
        Log.d(DEBUG_TAG, "onCreate startService");
    }
    // create the receiver
    private BroadcastReceiver mBroadcastReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            NotificationActivityService.this.receivedBroadcast(intent);
        }
    };
};
```

NotificationActivityService example 3



- Handling the received data (both a dynamic and a persistent receiver is present)

```
@Override
public void onResume() {
    super.onResume();
    IntentFilter ifilter = new IntentFilter();
    ifilter.addAction(SimpleIntentService.ACTION_NOTIFY);
    ifilter.addCategory(Intent.CATEGORY_DEFAULT);
    // Put whatever message you want to receive as the action
    this.registerReceiver(this.mBroadcastReceiver, ifilter);
}

@Override
public void onPause() {
    this.unregisterReceiver(this.mBroadcastReceiver);
    super.onPause();
}

private void receivedBroadcast(Intent intent) {
    // Put your receive handling code here
    Log.d(DEBUG_TAG, "receivedBroadcast received");

    int value;
    // retrieves a map of extended data from the intent
    Bundle extras = intent.getExtras();
    if(extras != null){
        // get the parameter from the Bundle out of the Intent
        value = extras.getInt(SimpleIntentService.PARAM_OUT_COUNT);
    }
    else
        value = 0; // default init

    Toast.makeText(this, "NotificationActivityService Value: " + value, Toast.LENGTH_SHORT).show();
}
}
```

NotificationActivityService example 4



```
<!-- Declaring the special permission -->
<permission
    android:name="se.du.notification.Permission.SEND_SIMPLE_NOTIFICATIONS"
    android:label="permission to send simple notifications"
    android:permissionGroup="android.permission-group.PERSONAL_INFO"
    android:protectionLevel="normal" />

<!-- Use the special permission! -->
<uses-permission android:name="se.du.notification.Permission.SEND_SIMPLE_NOTIFICATIONS" />

<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="se.du.notification.NotificationActivityService"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <service
        android:name="se.du.notification.SimpleIntentService"
        android:enabled="true"
        android:exported="false"
        android:label="Android simple Intent service" />
    <receiver
        android:name="se.du.notification.SimpleBroadcastReceiver"
        android:permission="se.du.notification.Permission.SEND_SIMPLE_NOTIFICATIONS" android:enabled="true">
        <intent-filter>
            <action android:name="se.du.notification.DATA_RETRIEVED" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </receiver>
</application>
```


NotificationActivityService example 5



```
public final class PersistentNotification {
    private static final String TAG = PersistentNotification.class.getSimpleName();
    public static void cancelNotification(NotificationManager mNManager, int NOTIFY_ID) {
        mNManager.cancel(NOTIFY_ID);
    }
    public static void createNotification(NotificationManager mNManager, Context ctx, Class cls, int NOTIFY_ID){
        Log.d(TAG, "createNotification()");
        // cancel any current notification
        mNManager.cancel(NOTIFY_ID);
        // set activity flags for calling class
        Intent notifyIntent = new Intent(ctx, cls);
        notifyIntent.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
        // needed to start the notification
        PendingIntent pendIntent = PendingIntent.getActivity(ctx, 0, notifyIntent, 0);
        // build notification
        Notification.Builder builder = new Notification.Builder(ctx);
        builder.setContentIntent(pendIntent)
            .setSmallIcon(R.drawable.ic_launcher)
            /*
            .setLargeIcon(BitmapFactory.decodeResource(res, R.drawable.some_big_img))
            .setTicker(res.getString(R.string.your_ticker))
            .setWhen(System.currentTimeMillis())
            */
            .setContentTitle(ctx.getString(R.string.persistent_notification_title))
            .setOngoing(true)
            .setAutoCancel(false)
            .setContentText(ctx.getString(R.string.persistent_notification_text));
        final Notification notifyMsg = builder.build();
        // Notification setting flags
        // notifyMsg.defaults |= Notification.DEFAULT_SOUND;
        //notifyMsg.flags |= Notification.FLAG_NO_CLEAR;
        // make this notification appear in the 'Ongoing events' section
        //notifyMsg.flags |= Notification.FLAG_ONGOING_EVENT;
        // post the notification
        mNManager.notify(NOTIFY_ID, notifyMsg);
    }
}
```

App Widgets

<http://developer.android.com/guide/topics/appwidgets/index.html>



- App Widgets are usually small icon-like views in an application. They implement a subclass of the broadcast receiver for use in updating this view.
- Called widgets for short, they can be embedded into other applications, such as the home screen. In all, they require the following
 - A view describing the appearance of the widget. This is defined in an XML layout resource file and contains text, background, and other layout parameters.
 - An App Widget provider that receives broadcast events and interfaces to the widget to update it.
 - Detailed information about the App Widget, such as the size and update frequency. Note that the home screen is divided into 4x4 cells and so a widget is often a multiple of a single cell size (which is 80x100dp in Portrait mode and 106x74dp in Landscape mode).
 - http://developer.android.com/guide/practices/ui_guidelines/widget_design.html
 - Optionally, an App Widget configuration activity can be defined to properly set any parameters of the Widget. This activity is launched upon creation of the Widget.



```
<receiver android:name=".SimpleWidgetProvider">
  <intent-filter>
    <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
  </intent-filter>
  <meta-data android:name="android.appwidget.provider"
    android:resource="@xml/widget_info" />
</receiver>
```

xml/widget_info

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
  android:minWidth="146dp" android:minHeight="72dp"
  android:updatePeriodMillis="1800000" android:initialLayout="@Layout/widget_layout">
</appwidget-provider>
```

Simple App widget

```
public class SimpleWidgetProvider extends AppWidgetProvider {
// Note! Updates requested with updatePeriodMillis will not be delivered more than once every 30 minutes
  @Override
  public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
    super.onUpdate(context, appWidgetManager, appWidgetIds);
    Log.v(Consts.TAG, "SimpleWidgetProvider > onUpdate()");
    // Perform this loop procedure for each App Widget that belongs to this provider
    final int N = appWidgetIds.length; // i.e. user have created
    for (int i=0; i<N; i++) {
      int appWidgetId = appWidgetIds[i];
      String titlePrefix = "Time since January 1, 1970 00:00:00 UTC:";
      updateAppWidget(context, appWidgetManager, appWidgetId, titlePrefix);
    }
  }

  static void updateAppWidget(Context context, AppWidgetManager
    appWidgetManager, int appWidgetId, String titlePrefix)
  {
    Long millis = System.currentTimeMillis();
    int seconds = (int) (millis / 1000);
    int minutes = seconds / 60;
    seconds = seconds % 60;
    String text = titlePrefix;
    text += " " + minutes + ":" + String.format("%02d", seconds);
    Log.v(Consts.TAG, "updateAppWidget(): " + text);
    // Construct the RemoteViews object.
    RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget_layout);
    // String implements CharSequence, but CharSequence doesn't implement String
    views.setTextViewText(R.id.widget_example_text, text);
    // Tell the AppWidgetManager to perform an update on the current app widget
    appWidgetManager.updateAppWidget(appWidgetId, views);
  }
}
```

Time since January 1, 1970
00:00:00 UTC: 23006442:18

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/widget_example_text"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:background="#ffffff"
  android:textColor="#ff000000" />
```

Compability



- New Android versions are generally additive and forward compatible at the API level. A device can be called an Android device only if it passes compatibly tests with the Android APIs
 - Do not use internal or unsupported APIs
 - Do not directly manipulate settings without asking the user
 - Do not go overboard with layouts. This is rare, but complicated layouts (more than 10 deep or 30 total) can cause crashes
 - Do not make bad hardware assumptions. Be sure to check for the hardware needed
 - Ensure device orientations do not disrupt the application or result in unpredictable behavior
- Use at least the v13 support library
 - <http://developer.android.com/tools/support-library/features.html>
- Note that backward compatibility is not guaranteed with Android! Use the minimum SDK version

```
<uses-sdk android:minSdkVersion="21" />
```

Robustness



- In the same vein as compatibility support, applications should be designed and tested for robustness.
 - Use the Android libraries before Java libraries. Android libraries are constructed specifically for embedded devices and cover many of the requirements needed in an application.
 - Take care of memory allocation. Initialize variables. Try to reuse objects rather than reallocate. This speeds up application execution and avoids excessive use of garbage collection
 - Utilize the LogCat tool for debugging and check for warnings or errors
 - Test thoroughly, including different environments and devices if possible

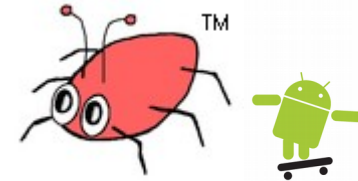
Effective use of Java



- Make good use of static methods and scalar types
- Compare against 0 or null, ex. `for(int i=s.size(); i>=0; i--)`
- Avoid operations on String objects - Use the StringBuilder class for efficient manipulation of strings
- Limit the use of inner classes and floating point types
- Use an obfuscator to reduce class file size and optimize code
- Set object references to null as soon as they are no longer needed
- Avoid unnecessary re-initialization of variables that are automatically set to 0 or null by the VM
- Use synchronization sparingly, it is costly and is only needed in multi-threaded applications
- **Design is most important as usual!**
- Use native bridged code as: `System.arraycopy()`
- Profiling/tracing to reduce bottlenecks...

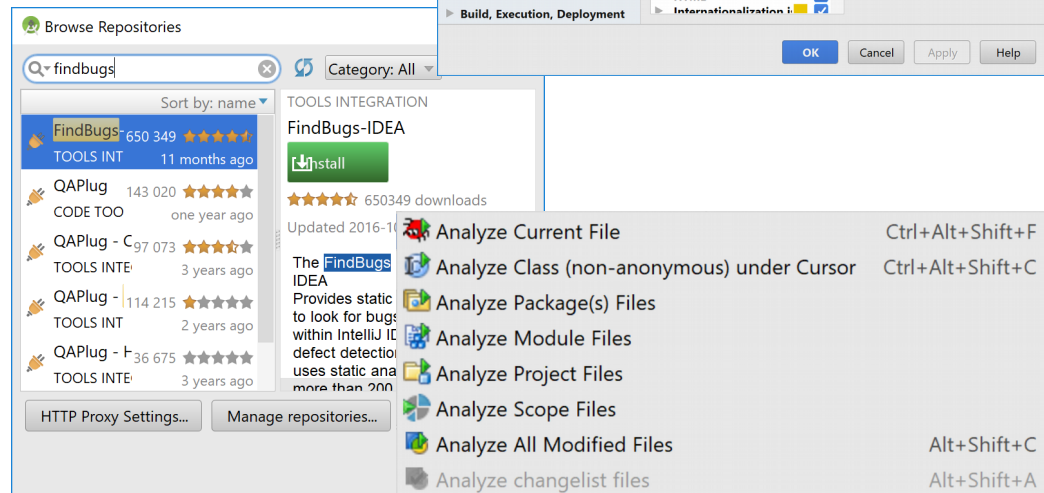
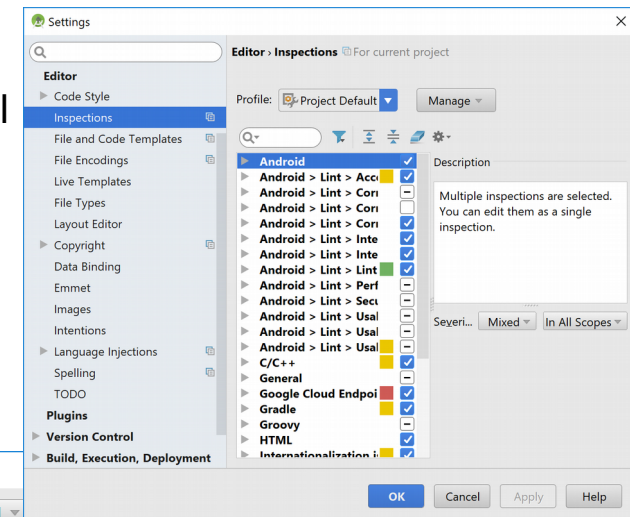


Performance tips & find bugs



- There are two basic rules for writing efficient code
 - Don't do work that you don't need to do
 - Don't allocate memory if you can avoid it
 - Find many more optimizations that can be done at:
 - <http://developer.android.com/training/articles/perf-tips.html>
- The Android Lint inspection tool
 - A code analysis tool that check project source files for potential bugs and optimization improvements
 - <http://developer.android.com/tools/help/lint.html>
- FindBugs-IDEA™
 - A free software program which uses static code analysis to look for bugs in Java code
 - Plugin compatible with AS
 - Right click code > FindBugs

Lint settings

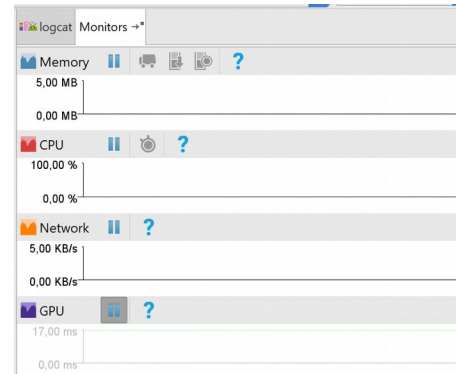


<https://plugins.jetbrains.com/plugin/3847-findbugs-idea>

Using traceview and AM

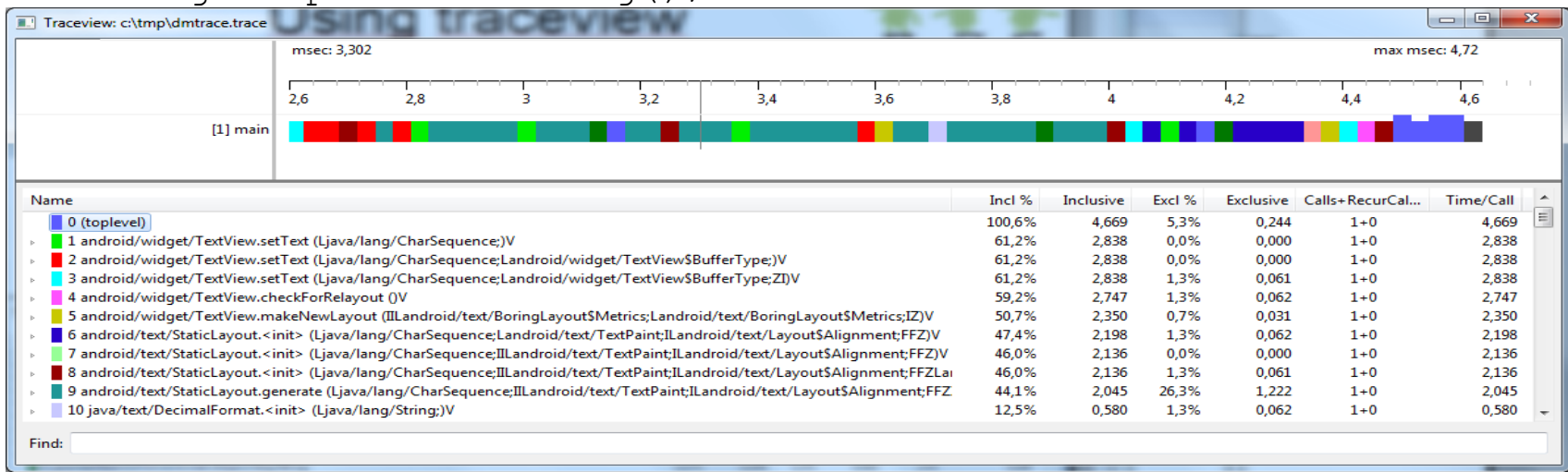


- Android Monitor besides LogCat do
 - Memory
 - CPU,
 - GPU usage and
 - Network traffic (hardware device only)
- TraceView is a tool to optimize performance (profile the program)
- A dmtrace.trace file will be created on the SD-card
- Also possible via DDMS (Start Method Profiling button)



```
Debug.startMethodTracing("tag")  
doHeavyWorkHere();  
Debug.stopMethodTracing();
```

Note! TraceView disables the JIT



Test your app



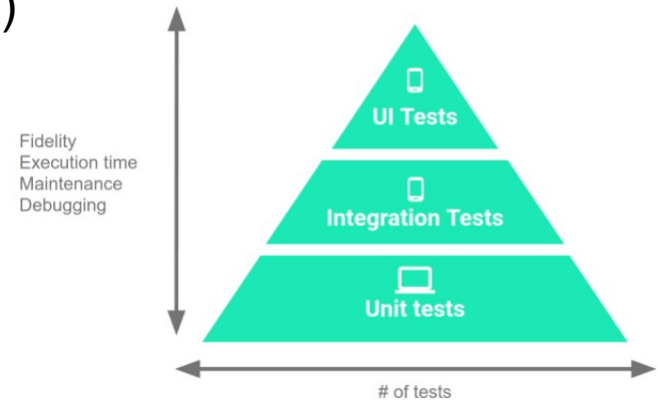
- Test types

- **Local unit tests** - These are tests that run on your machine's local Java Virtual Machine (JVM)
- **Instrumented tests** - These are tests that run on a hardware device or emulator

- Android testing support library

- **Espresso** for UI testing
 - Espresso tests state expectations, interactions, and assertions
- **JUnit** is a tests runner for classes including Espresso and UI Automator test framework
- **Roboelectric** is a extensive Junit alternative
- **Mockito** – third party test framework
- <https://developer.android.com/training/testing/index.html>
- <https://developer.android.com/studio/test/index.html>

- The **Assert** class methods compare values you expect from a test to the actual results and throw an exception if the comparison fails



What to test?



- Change in orientation
 - Is the screen re-drawn correctly? Any custom UI code you have should handle changes in the orientation.
 - Does the application maintain its state?
- Change in configuration
 - Change in the device's configuration, such as a change in the availability of a keyboard or a change in system language.
- Battery life
 - You need to write your application to minimize battery usage, you need to test its battery performance, and you need to test the methods that manage battery usage.
- Dependence on external resources
 - If your application depends on network access, SMS, Bluetooth, or GPS, then you should test what happens when the resource or resources are not available or limited.

App install and publish



- Enable USB debugging for ADB installs (Settings > Developer options)
 - Open a cmd prompt in the project bin folder and execute
`adb -d install -r <application filename>.apk`
- Enable Unknown sources in Settings > Security
 - Install via file/package manager
- Sign and publish application
 - Test, test, test... test, test, test on device...
 - Fix all icons etc. for different display sizes
 - Remove all debug and test logging
 - Ensure you got the different *SdkVersions correct set
 - Update versionCode (int) and versionName (string) in AndroidManifest
 - The sign certificate may be self-signed or issued by a CA
 - Sign all your packages with the same certificate
 - Makes it possible to share data and will not be a new App on Google Play

Remember to enable ProGuard in the proguard-rules.pro file (valid for AS)

<http://developer.android.com/tools/publishing/app-signing.html>

Android Studio 1



- Sign and publish application
 - Build > Generate Signed APK...
 - app-debug.apk vs. app-release.apk
 - Correct Key Alias is important!

Password is "android"
for store and key

- APK signing error : Failed to read key from keystore
- Android 7.0 introduces APK Signature Scheme v2, a new app-signing scheme that offers faster app install times and more protection against unauthorized alterations to APK files. App may however not build properly!
 - https://developer.android.com/about/versions/nougat/android-7.0.html#apk_signature_v2

Generate Signed APK

Key store path: C:\Users\hjo\.android\debug.keystore

Key store password:

Key alias: androiddebugkey

Key password:

Remember passwords

Previous Next Cancel Help

Generate Signed APK

Note: Proguard settings are specified using the Project Structure Dialog

APK Destination Folder: C:\Users\hjo\Desktop

Build Type: release

Flavors: No product flavors defined

Signature Versions: V1 (Jar Signature) V2 (Full APK Signature) [Signature Help](#)

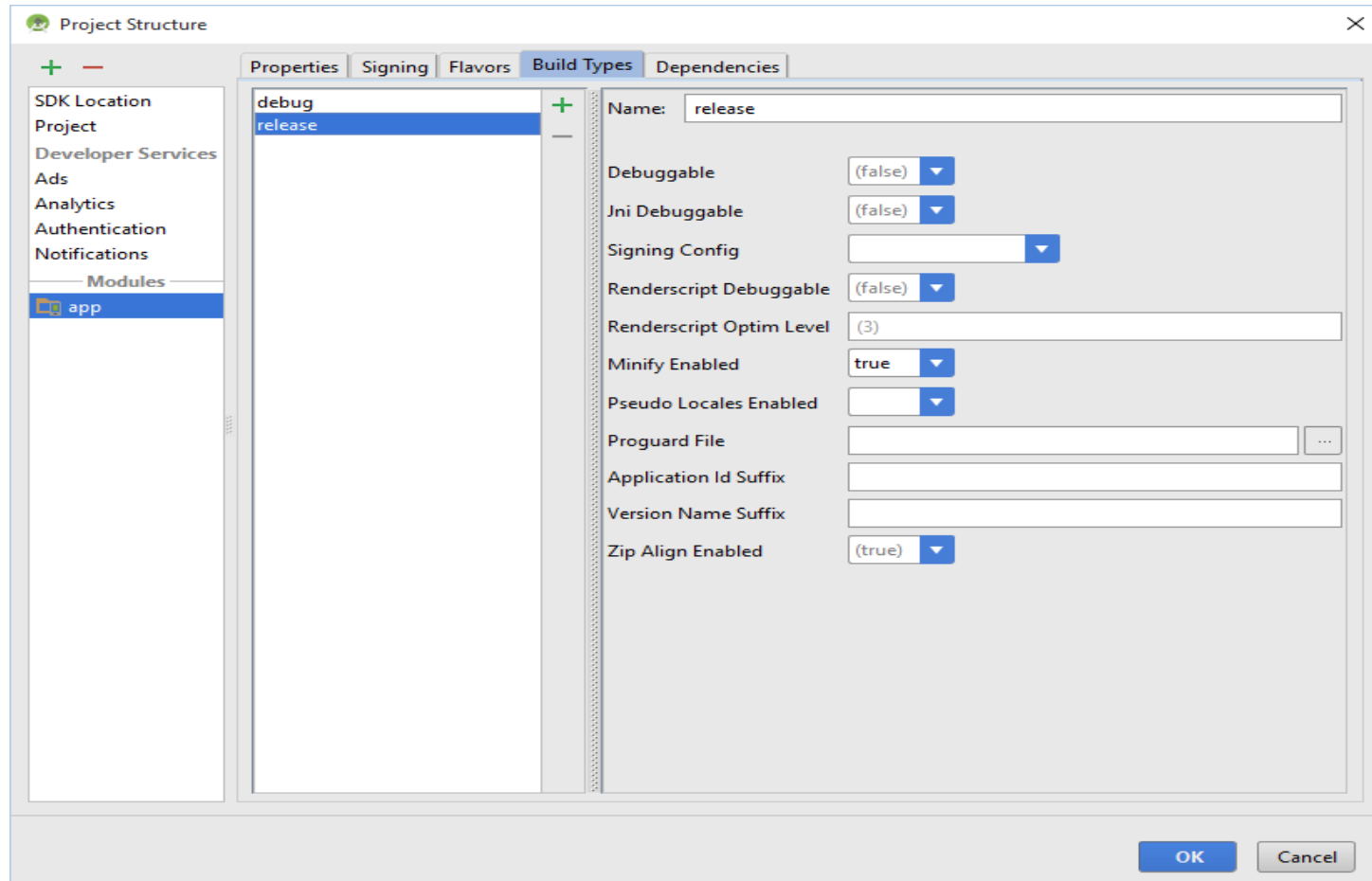
Previous Finish Cancel Help

Android Studio 2

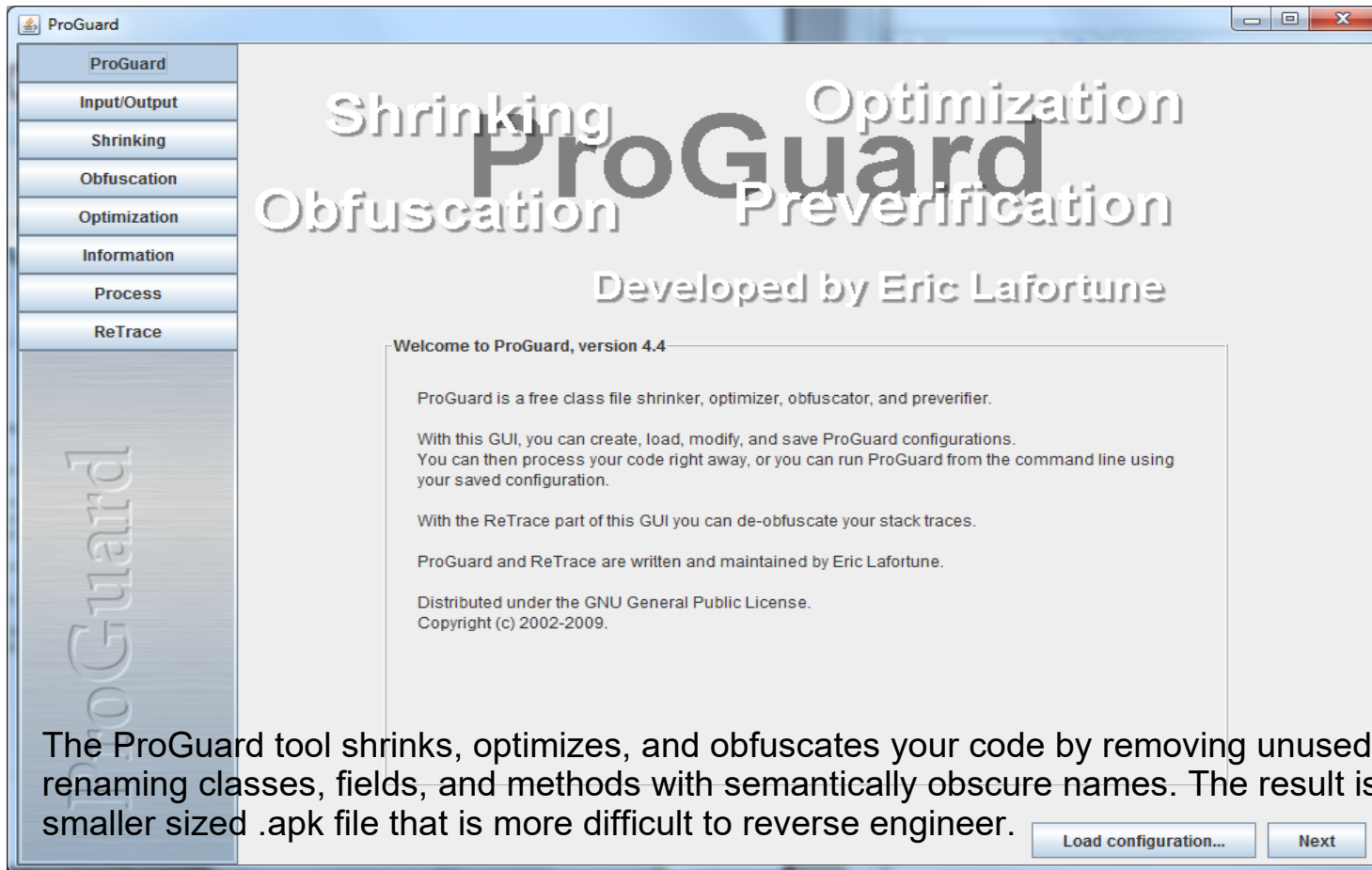


- Project Structure Dialog

- File > Project Structure... > ... > release > Minify Enabled



ProGuard



The ProGuard tool shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names. The result is a smaller sized .apk file that is more difficult to reverse engineer.

Enable ProGuard in the **module** specific build.gradle file:

```
buildTypes {
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
    }
}
```

<http://developer.android.com/guide/developing/tools/proguard.html>

Lab review - Android Lab5



- List with topics you need to understand before next laboration
- You must be able or know how to
 - understand all the previous points from former labs
 - use location
 - use the Google Maps API
 - use services
 - use media APIs
 - use alarms and notifications
 - pinpoint problems and know what to test in an app