**onlongclick()**

Add an OnItemLongClickListener too your ListView for handling the visibility of your ImageView

Add an OnItemClickListener too your ListView for handling the your wanted functionality there.

Add an OnClickListener too your ImageView and handle the the deletion of an item there. You can make use of a custom Adapter and override the getView-method

arrayadapter

An adapter is something that knows about the list items and how to represent or draw each list item on the screen.

Linear layout

A Layout that arranges its children in a single column or a single row

First argument is Context to access system services and resources ( you need **layout inflater to create list item view** )

Second argument defines the layout of the list that defines how the list item appears in listview. Here layout android.R.layout.simple\_list\_item\_1 which is defined by framework is used.

Third argument is the information about the list item, typically this information is used to create view for the list item.

Finally created **Adapter is given to the ListView .**

Toast class

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.

Action bar

The **action bar** is an important design element, usually at the top of each screen in an app, that provides a consistent familiar look between **Android** apps. It is used to provide better user interaction and experience by supporting easy navigation through tabs and drop-down lists.

@Override

public boolean onCreateOptionsMenu(Menu menu) {

MenuInflater inflater = getMenuInflater();

inflater.inflate(R.menu.main\_activity\_bar, menu);

return super.onCreateOptionsMenu(menu);

}

<?xml version="1.0" encoding="utf-8"?>

<resources>

<string name="app\_name">ActionBar</string>

<string name="action\_settings">Settings</string>

<string name="action\_search">Search</string>

<string name="action\_record">Record Video</string>

<string name="action\_save">Save</string>

<string name="action\_label">Add Label</string>

<string name="action\_play">Play Video</string>

<string name="hello\_world">Hello world!</string>

</resources>

Pager

swipable multi-touch interface. The most obvious example of this is the pager view, which allows a user to swipe through pages of information intuitively. The best known example of this is the home screen of iOS,

**Fragments**

UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

# Inflater

1 .Inflating means to read the XML file that describes a layout (or GUI element) and to create the actual objects that correspond to it, and thus make the object visible within an Android app.

2**.** [LayoutInflater](http://developer.android.com/intl/de/reference/android/view/LayoutInflater.html) class is used to instantiate layout XML file into its corresponding View objects.

In other words, **it takes as input an XML file and builds the View objects from it.**

3. XML layout, it will be inflated by the Android OS which basically means that it will be rendered.

You can also inflate views explicitly by using the LayoutInflater. In that case you have to:

1. Get an instance of the LayoutInflater
2. Specify the XML to inflate
3. Use the returned View

For instance:

LayoutInflater inflater = LayoutInflater.from(YourActivity.this); // 1

View theInflatedView = inflater.inflate(R.layout.your\_layout, null); // 2 and 3

# [What is the purpose of requestWindowFeature()?](http://stackoverflow.com/questions/12680055/what-is-the-purpose-of-requestwindowfeature)

Each activity has an associated window (on which the whole UI draws). This window has default way of being drawn (whether toolbar, **actionbar** , icons are drawn or not).

#### **public void setFillViewport (boolean fillViewport)**

Indicates this **ScrollView** whether it should stretch its content height to fill the viewport or not.

**ContentProviderResult**

# [get contact id after insert](http://stackoverflow.com/questions/23521963/android-get-contact-id-after-insert)

The [ContentResolver.applyBatch()](http://developer.android.com/reference/android/content/ContentResolver.html#applyBatch%28java.lang.String,%20java.util.ArrayList%3Candroid.content.ContentProviderOperation%3E%29) method returns an array of [ContentProviderResult](http://developer.android.com/reference/android/content/ContentProviderResult.html) objects, one for each operation**. Each of these has the uri of the inserted contact** (in the format content://com.android.contacts/raw\_contacts/<contact\_id>).

So to get the contact's id you just have to parse this uri, i.e.

ContentProviderResult[] results = getContentResolver().applyBatch(ContactsContract.AUTHORITY, ops);

int contactId = Integer.parseInt(results[0].uri.getLastPathSegment());

**log**

API for sending log output.

Generally, use the Log.v() Log.d() Log.i() Log.w() and Log.e() methods.

The order in terms of verbosity, from least to most is ERROR, WARN, INFO, DEBUG, VERBOSE. Verbose should never be compiled into an application except during development. Debug logs are compiled in but stripped at runtime. Error, warning and info logs are always kept.

**Tip:** A good convention is to declare a TAG constant in your class:

private static final String TAG = "MyActivity";

and use that in subsequent calls to the log methods.

**Tip:** Don't forget that when you make a call like

Log.v(TAG, "index=" + i);

**getTopics()**

1.listener interested in.

2. public java.lang.String[] getTopics()

**ViewHolder**

Your code might call [findViewById()](http://developer.android.com/reference/android/app/Activity.html#findViewById%28int%29) frequently during the scrolling of [ListView](http://developer.android.com/reference/android/widget/ListView.html), which can slow down performance. Even when the [Adapter](http://developer.android.com/reference/android/widget/Adapter.html) returns an inflated view for recycling, you still need to look up the elements and update them. A way around repeated use of [findViewById()](http://developer.android.com/reference/android/app/Activity.html#findViewById%28int%29) is to use the "view holder" design pattern.

A ViewHolder object stores each of the component views inside the tag field of the Layout, so you can immediately access them without the need to look them up repeatedly. First, you need to create a class to hold your exact set of views. For example:

static class ViewHolder {  
  TextView text;  
  TextView timestamp;  
  ImageView icon;  
  ProgressBar progress;  
  int position;  
}

Then populate the ViewHolder and store it inside the layout.

ViewHolder holder = new ViewHolder();  
holder.icon = (ImageView) convertView.findViewById(R.id.listitem\_image);  
holder.text = (TextView) convertView.findViewById(R.id.listitem\_text);  
holder.timestamp = (TextView) convertView.findViewById(R.id.listitem\_timestamp);  
holder.progress = (ProgressBar) convertView.findViewById(R.id.progress\_spinner);  
convertView.setTag(holder);

Now you can easily access each view without the need for the look-up, saving valuable processor cycles.

**Comparator**

Comparator can be used to obtain a sorted Collection which is totally ordered.

**Bundle**

Bundle is generally used for **passing data between various activities** of android