|  |
| --- |
| HOGESCHOOL VAN AMSTERDAM |
| Simple shopping List |
| Tutorial for week 6 app 1 |
|  |
| **Hogeschool van Amsterdam** |

|  |
| --- |
| This document contains the tutorial for the first Android application of week 6 |

Table of Content

[Introduction 4](#_Toc411093307)

[Step 1: Create a new project 5](#_Toc411093308)

[Step 2: Creating the list 6](#_Toc411093309)

[2.1: Create the XML layout 6](#_Toc411093310)

[2.2: Get the ListView 6](#_Toc411093311)

[2.2: Create the adapter 7](#_Toc411093312)

[2.2.1: Populate the adapter 9](#_Toc411093313)

[2.3: Handle list clicks 12](#_Toc411093314)

[Step 3: Using the ActionBar 14](#_Toc411093315)

[3.1: Creating an ActionBar menu 14](#_Toc411093316)

[3.3: Handling ActionBar menu clicks 15](#_Toc411093317)

[Step 4: Adding a Context menu 18](#_Toc411093318)

[4.1: Create a context menu 18](#_Toc411093319)

[4.2: Register context menu 19](#_Toc411093320)

[4.3: Create the context menu 20](#_Toc411093321)

[4.4: Handling context menu clicks 20](#_Toc411093322)

[Step 5: Dynamically adding list items 24](#_Toc411093323)

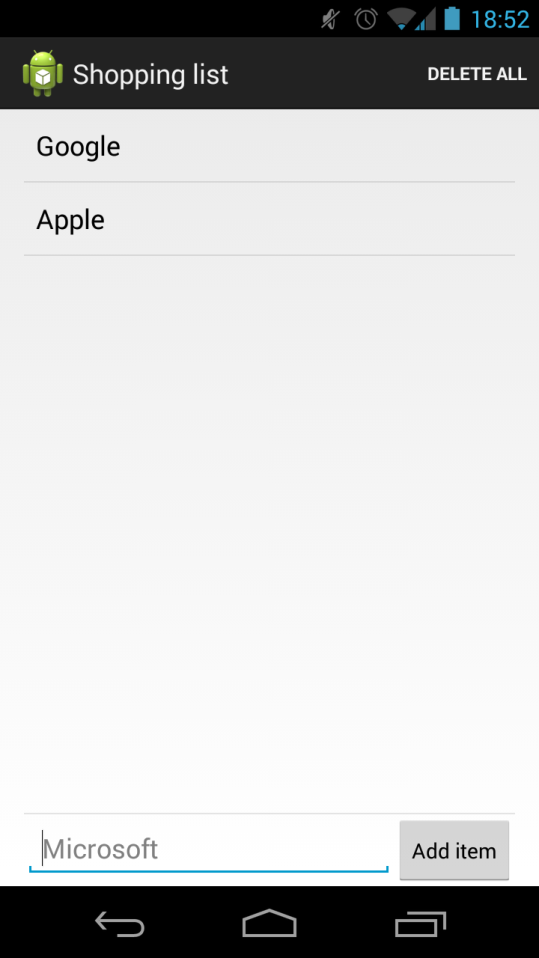
[5.1: Adjusting the layout 24](#_Toc411093324)

[5.2: Add items 25](#_Toc411093325)

# Introduction

In this tutorial we’ll create a simple shopping list without a persistent memory. By doing this tutorial you’ll learn how to work with the ListView widget (adding and deleting items, handling clicks) , how to work with the ActionBar, how to work with a context menu and finally how to dynamically add items to the list.

We’ll end up with an application that looks like this:



# Step 1: Create a new project

We’ll start this tutorial by creating a new Android Studio project. The creation of projects in Android Studio has been explained in detail in the first tutorial of week 4, refer back to this if you have any trouble settings up a project.

Name the application however you like, and select API 14 as the minimum SDK level. This will ensure that all phones that use Android 4.0 and up will be able to run this application.

Finally select the ‘Blank Activity’ option to add an activity to the project. The activity name should be ‘ListActivity’, the layout name should be ‘activity\_list’ and the title can be anything.

# Step 2: Creating the list

In this section we’ll be creating a layout with a list. We’ll start out by adding the ListView widget in our XML layout file. Then we’ll use the added list in our Activity and add some items to it. Finally we’ll have the different list items display a message when the user clicks on it.

## 2.1: Create the XML layout

Open ‘activity\_list.xml’, this is the layout file for ListActivity, the file is located in the layout directory. Open de text tab of the layout editor and add the following code to the Layout of the xml file.

|  |
| --- |
| <ListView  android:id=**"@+id/listView"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"match\_parent"**  android:layout\_weight=**"1"** /> |

This adds a ListView to the layout. The ListView can later be used in the code by referencing to its id ‘listView’. Of course the TextView should be removed and for this view we have set a LinearLayout (see below for full code).

The entire Layout file looks like this:

|  |
| --- |
| <LinearLayout xmlns:android=**"http://schemas.android.com/apk/res/android"**  xmlns:tools=**"http://schemas.android.com/tools"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"match\_parent"**  android:orientation=**"vertical"**  android:paddingLeft=**"@dimen/activity\_horizontal\_margin"**  android:paddingRight=**"@dimen/activity\_horizontal\_margin"**  tools:context=**".ListActivity"**>    <ListView  android:id=**"@+id/listView"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"match\_parent"**  android:layout\_weight=**"1"** />    </LinearLayout> |

## 2.2: Get the ListView

Open ‘ListActivity.java’, this is the Activity that will host the ListView and it will also control what will be shown in it. ‘ListActivity’ should contain the onCreate method, make sure it looks like this:

|  |
| --- |
| @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**  **}** |

The onCreate method is the first method that will run after the Activity is created, you want to add the actions that should happen on the start to the onCreate method. Notice that the ‘setContentView’ method is also executed in the above snippet of code. This method sets the XML layout that we’ve created earlier as the content for the activity, in this specific case that means that the ListView is now loaded into the Activity.

The next step is to bind the ListView to an attribute in our activity so that we can manipulate it, instead of just having it present. Declare a ListView variable name ‘listView’ at the top of the Activity. Then initialize it in the onCreate method via the findViewById method. Your Activity Should now look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    //Declare the listView variable  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the listView  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**  **}**  **}** |

The findViewById method allows you to reference to loaded view by their id. We set the id of the ListView in the earlier section of the tutorial when we created the XML layout. We set the id to ‘listView’ and now we can get the view by its id.

The next step is to create an adapter for the ListView.

## 2.2: Create the adapter

An Adapter is a class that acts a controller for the data that a ListView (or many other views) have to display. Adapters are very versatile and there are many different types of adapters. In this tutorial we’ll be focusing on one of the most basic types of adapters, the ArrayAdapter.

The ArrayAdapter handles a list of objects and couples each object to a view, in the case of a list this view would be a single row. The object can then be used to populate the view and thus showing the values of the object on the screen of the device.

To create the adapter we’ll first need to create it as a variable. We will be using the ArrayAdapter type adapter. We’ll also create an List that holds the String values you can define it like this:

|  |
| --- |
| // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;** |

This creates an ArrayAdapter that’s able to handle String objects. The next step is to initialize the adapter. We’ll be using the standard ArrayAdapter class that comes with the Android SDK. To initialize the adapter we’ll have to pass it two arguments, a context object and a layout resource. A context object is an object that allows access to android system resources. The Layout resource will be the layout for the ListView rows. Each row in the list will get this layout. We’ll be using a standard layout from the Android SDK. Intialize the ArrayAdapter in the ‘onCreate’ method like this:

|  |
| --- |
| //Create the List of items  items **=** **new** ArrayList**<**String**>();**    //Create the Array Adapter, give it a layout and a list of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1, items**);** |

We can use ‘this’ for the context argument because an Activity is a subclass of the Context class. The ‘android.R.layout.simple\_list\_item\_1’ argument refers to the simple\_list\_item\_1 layout of the Android layout directory. Much like our own app android has a layout folder that contain some of the layouts used throughout the Android OS. We can use these layouts freely for our own purposes.

Finally we’ll have to set the newly created adapter as the adapter for our ListView, you can do this by adding the following line at the bottom of the onCreate method:

|  |
| --- |
| //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);** |

The adapter is now able to control which items are displayed in the ListView. If we want to add, remove or edit items in the ListView we’ll have to do it via the adapter.

After this section of the tutorial your activity should look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the views  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**    //Create the List of items  items **=** **new** ArrayList**<**String**>();**    //Create the Array Adapter, give it a layout and a list of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1, items**);**    //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);**  **}**  **}** |

The next step will be to add item to the adapter so the list will have some content.

### 2.2.1: Populate the adapter

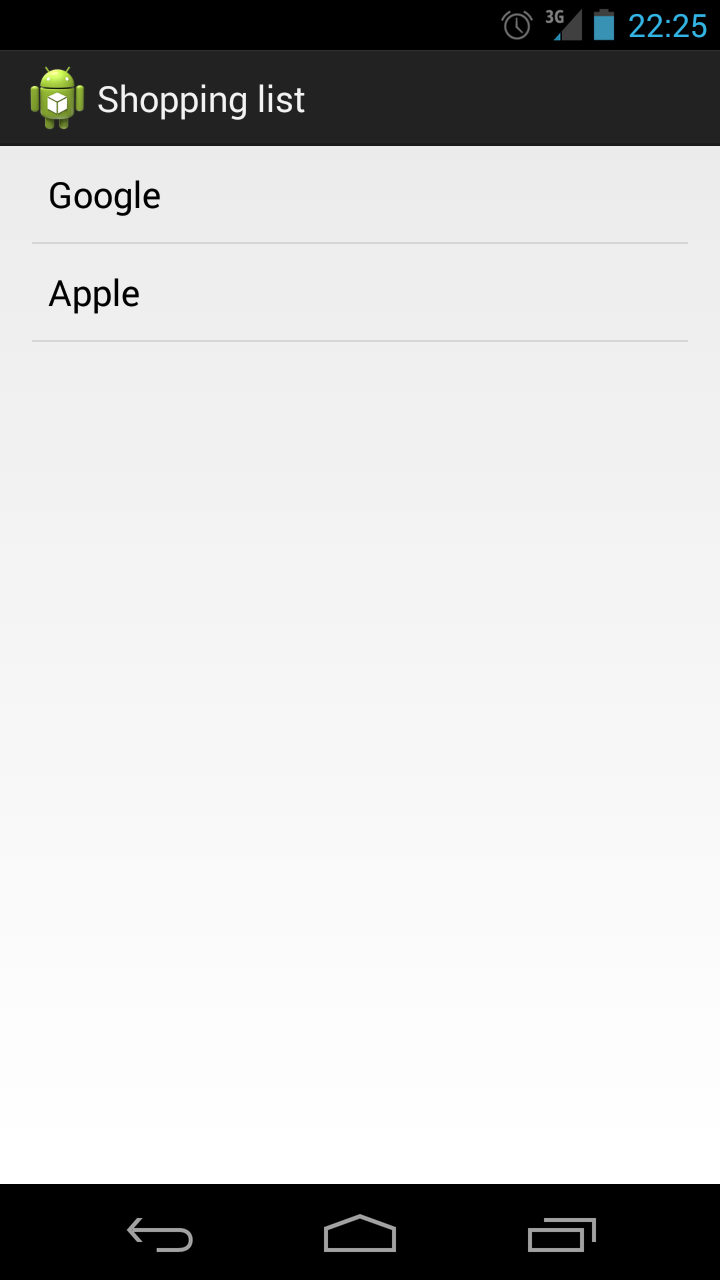
Populating the adapter is fairly straight forward. We’ve created an ArrayAdapter that handles String object, we also gave the adapter a list that hold the Strings. We can simply add a String to the list and notifiy the adapter that the data set has changed. Our ‘items’ list and the list that the adapter uses are the same, so if we change something in our items list we only have to let the adapter know and it’ll update our list. Keeping a reference to our list let us more easily manipulate the items in the adapter.

|  |
| --- |
| //Add items to the adapter  items**.**add**(**"Google"**);**  adapter.notifyDataSetChanged(); |

Now we can run the application for the first time, your activity should look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the views  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**    //Create the List of items  items **=** **new** ArrayList**<**String**>();**    //Create the Array Adapter, give it a layout and a list of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1, items**);**    //Add some items to the  items**.**add**(**"Google"**);**  items**.**add**(**"Apple"**);**    //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);**  **}**  **}** |

When you run this application you app should look like this:



The next step is to learn how to handle clicks on items in the list

## 

## 

## 2.3: Handle list clicks

To handle list clicks we have to set an OnItemClickListener on the ListView. As the name suggests the OnItemClickListener handles clicks on separate rows of the ListView and allows you to execute code when an item is clicked. In our case we’ll be displaying a Toast message when an item is clicked. Add the following lines of code to the onCreate method of the ListActivity:

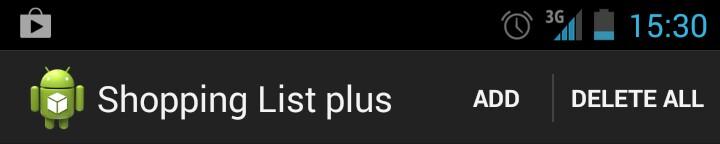
|  |
| --- |
| //Set the listview on item click listener  listView**.**setOnItemClickListener**(new** AdapterView**.**OnItemClickListener**()** **{**  @Override  public void onItemClick**(**AdapterView**<?>** parent**,** View listItem**,** int position**,** long id**)** **{**  //Get the value of the item that the user clicked on  String clickedItem **=** **(**String**)** parent**.**getItemAtPosition**(**position**);**    //Display a Toast message to show the user the item he/she clicked on  Toast**.**makeText**(**ListActivity**.this,** "Clicked: " **+** clickedItem**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **});** |

The complete activity should now look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the items options\_menu  items **=** **new** ArrayList**<**String**>();**    //Add some items to the options\_menu  items**.**add**(**"Google"**);**  items**.**add**(**"Apple"**);**    //Initialize the views  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**    //Create the Array Adapter, give it a layout and a options\_menu of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1**,** items**);**    //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);**    //Set the listview on item click listener  listView**.**setOnItemClickListener**(new** AdapterView**.**OnItemClickListener**()** **{**  @Override  public void onItemClick**(**AdapterView**<?>** parent**,** View listItem**,** int position**,** long id**)** **{**  //Get the value of the item that the user clicked on  String clickedItem **=** **(**String**)** parent**.**getItemAtPosition**(**position**);**    //Display a Toast message to show the user the item he/she clicked on  Toast**.**makeText**(**ListActivity**.this,** "Clicked: " **+** clickedItem**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **});**  **}**  **}** |

# Step 3: Using the ActionBar

The ActionBar is the navigation bar that’s displayed on the top of most apps. The ActionBar provides the user with information about where they are in the application and it shows quick actions that are related to the displayed content on the screen. An example of an ActionBar:



This ActionBar shows the title of the application and two options to manipulate the content of the screen below.

The ActionBar is a standard element found in each user interface since Android version 4.0. Apps will have an ActionBar by default. This segment of the tutorial will be focusing on how to add actions to the actionbar that manipulate the content of the screen. In the example screenshot these actions would be the ‘Add’ and ‘Delete all’ buttons.

## 3.1: Creating an ActionBar menu

While the action bar is added by default, the content specific options have to be added separately. This is done by creating a menu file in the resources directory. Open the ‘menu’ directory of your resources directory. This directory holds all your menu resource files. There should be one menu file present, rename it to ‘action\_bar\_menu.xml’. Open the file and replace it with the following code:

|  |
| --- |
| <menu xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  tools:context=".ListActivity">  </menu> |

This will create a new menu without any items, the next step is to add an item to the menu. Add the following code to the menu:

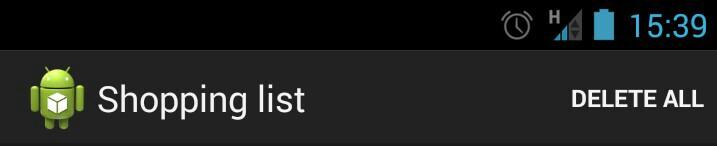
|  |
| --- |
| <item android:id="@+id/action\_bar\_menu\_delete\_all"  android:title="Delete all"  app:showAsAction="ifRoom" /> |

This code represents a simple form of a single menu item. It has an id and a title text. app:showAsAction=”ifRoom”’ means that the item will only be shown on the ActionBar when there’s enough room. If there isn’t enough room it will be pushed into an overflow menu.

You can add as many items to the menu as you’d like, the next step is to add the ActionBar menu to your activity. Your entire menu resource file should look like this:

|  |
| --- |
| <menu xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  tools:context=".ListActivity">  <item android:id="@+id/action\_bar\_menu\_delete\_all"  android:title="Delete all"  app:showAsAction="ifRoom" />  </menu> |

Run the application again and your ActionBar should look like this.



Notice the ‘DELETE ALL’ button that’s been added. The next step is to respond to users clicking on the delete button.

## 

## 3.3: Handling ActionBar menu clicks

To respond to a user clicking on an ActionBar menu item we need to override the ‘onOptionsItemSelected’ method that an Activity has, we can do this as following:

|  |
| --- |
| @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**    **return** **super.**onOptionsItemSelected**(**item**);**  **}** |

The next step is to make sure that it’s our own button that has been clicked. We can to this by checking the ID of the button. When we created the menu file we gave our ‘Delete all’ menu item the id ‘action\_bar\_menu\_delete\_all’. We can now reference this button by this ID. Change the ‘onOptionsItemSelected’ method to this:

|  |
| --- |
| @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**  //Check which menu item has been clicked  **if** **(**item**.**getItemId**()** **==** R**.**id**.**action\_bar\_menu\_delete\_all**)** **{**    **}**  **return** **super.**onOptionsItemSelected**(**item**);**  **}** |

Now we have a filter so that we only execute our code when the delete button is clicked. As the name of the button suggests we want to delete all the items in the list when it’s clicked. Add the following lines to the ‘onOptionsItemSelected’ method:

|  |
| --- |
| @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**  //Check which menu item has been clicked  **if** **(**item**.**getItemId**()** **==** R**.**id**.**action\_bar\_menu\_delete\_all**)** **{**  //Clears the list  items**.**clear**();**  //Tell the adapter that it should reload the data  adapter**.**notifyDataSetChanged**();**  **}**  **return** **super.**onOptionsItemSelected**(**item**);**  **}** |

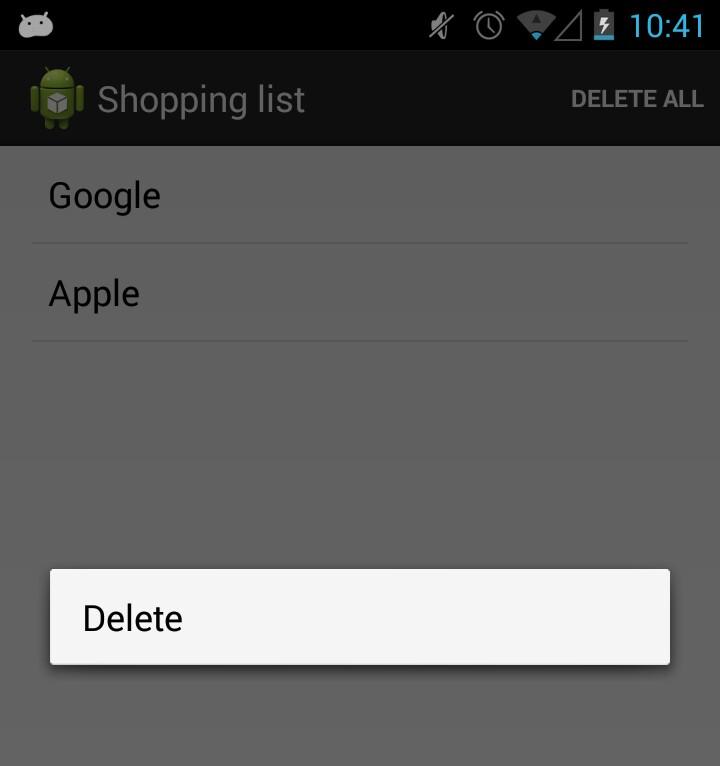
Run the application again and test if the delete button does its job.

The complete ListActivity code should now look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    /\* All the functionality we added to onCreate Earlier \*/  **}**    @Override  public boolean onCreateOptionsMenu**(**Menu menu**)** **{**  //Inflate the action bar menu  getMenuInflater**().**inflate**(**R**.**menu**.**action\_bar\_menu**,** menu**);**    **return** **super.**onCreateOptionsMenu**(**menu**);**  **}**    @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**  //Check which menu item has been clicked  **if** **(**item**.**getItemId**()** **==** R**.**id**.**action\_bar\_menu\_delete\_all**)** **{**  //Clears the list  items**.**clear**();**  //Tell the adapter that it should reload the data  adapter**.**notifyDataSetChanged**();**  **}**  **return** **super.**onOptionsItemSelected**(**item**);**  **}**  **}** |

# Step 4: Adding a Context menu

A context menu is a pop up style menu that is usually only displayed when a user long click an item in a collection (like a list). The context menu shows some actions that can be applied to the specific item that has been selected. An example of a context menu:



The context menu greys out the background content to highlight the different options. In many ways the context menu is similar to the ActionBar menu, it is a lot more specific in use however. In this segment of the tutorial we’ll be creating a context menu that deletes a selected item from out shopping list.

## 4.1: Create a context menu

The first step is to create the context menu in your resources folder. A context menu is created in the exact same way as an ActionBar menu so this part should be familiar. Create a menu file called ‘context\_menu.xml’ in the ‘menu’ directory of your resources directory. Replace the new menu with the following code:

|  |
| --- |
| <menu xmlns:android=**"http://schemas.android.com/apk/res/android"**  xmlns:tools=**"http://schemas.android.com/tools"**>  <item  android:id=**"@+id/context\_menu\_delete\_item"**  android:title=**"Delete"**/>  </menu> |

The next step will be to register the context menu for a View.

## 

## 

## 4.2: Register context menu

A context menu is bound to a View, this means that you have to register a view to act as the trigger for the context menu. In this tutorial we’ll register the ListView as the trigger for the context menu. We can register a view for the context menu with the ‘registerForContextMenu’ method of the Activity class. Add the following line to the ‘onCreate’ method of your ListActivity:

|  |
| --- |
| registerForContextMenu**(**listView**);** |

That’s all it takes, the context menu will now be created when the user long clicks an item in the ListView.

Your complete ‘onCreate’ method should look like this:

|  |
| --- |
| @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the items options\_menu  items **=** **new** ArrayList**<**String**>();**    //Add some items to the options\_menu  items**.**add**(**"Google"**);**  items**.**add**(**"Apple"**);**    //Initialize the views  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**    //Create the Array Adapter, give it a layout and a options\_menu of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1**,** items**);**    //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);**    //Register the the ListView for a context menu  registerForContextMenu**(**listView**);**    //Set the listview on item click listener  listView**.**setOnItemClickListener**(new** AdapterView**.**OnItemClickListener**()** **{**  @Override  public void onItemClick**(**AdapterView**<?>** parent**,** View listItem**,** int position**,** long id**)** **{**  //Get the value of the item that the user clicked on  String clickedItem **=** **(**String**)** parent**.**getItemAtPosition**(**position**);**    //Display a Toast message to show the user the item he/she clicked on  Toast**.**makeText**(**ListActivity**.this,** "Clicked: " **+** clickedItem**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **});**  **}** |

## 4.3: Create the context menu

Because the ListView is now registered for a context menu it will trigger the ‘onCreateContextMenu’ method when a user long clicks an item in the list. In order to show the menu we’ve created earlier we have to override the method and give our own implementation of it. Creating a context menu is nearly identical to the creation of an ActionBar menu which we’ve done earlier. Add the following code to your ListActivity to have the application create the context menu:

|  |
| --- |
| @Override  public void onCreateContextMenu**(**ContextMenu menu**,** View view**,** ContextMenu**.**ContextMenuInfo menuInfo**)** **{**  //Inflate the context menu from the resource file  getMenuInflater**().**inflate**(**R**.**menu**.**context\_menu**,** menu**);**    **super.**onCreateContextMenu**(**menu**,** view**,** menuInfo**);**  **}** |

The example in the bundled source code of the tutorial app is a little more detailed and will add the name of the item in the button so it’s absolutely clear which item the user is deleting.

## 4.4: Handling context menu clicks

Finally we need to respond to context menu item click by the user, this is also very similar to the way the application responds to ActionBar menu clicks. Override the Activity’s ‘onContextItemSelected’ method like so:

|  |
| --- |
| @Override  public boolean onContextItemSelected**(**MenuItem item**)****{**  **if****(**item**.**getItemId**()****==** R**.**id**.**context\_menu\_delete\_item**)****{**  **}**    **return****super.**onContextItemSelected**(**item**);**  **}** |

We have to know the position of the item in the list in order to target it for removal, we can retrieve information about the selected list item by adding the following line to the method:

|  |
| --- |
| @Override  public boolean onContextItemSelected**(**MenuItem item**)** **{**  //Retrieve info about the long pressed list item  AdapterView**.**AdapterContextMenuInfo itemInfo **=** **(**AdapterView**.**AdapterContextMenuInfo**)** item**.**getMenuInfo**();**  **if** **(**item**.**getItemId**()** **==** R**.**id**.**context\_menu\_delete\_item**)** **{**  **}**    **return** **super.**onContextItemSelected**(**item**);**  **}** |

Now that we know the position of the item we can remove it from the item list and update the adapter:

|  |
| --- |
| @Override  public boolean onContextItemSelected**(**MenuItem item**)** **{**  //Retrieve info about the long pressed list item  AdapterView**.**AdapterContextMenuInfo itemInfo **=** **(**AdapterView**.**AdapterContextMenuInfo**)** item**.**getMenuInfo**();**  **if** **(**item**.**getItemId**()** **==** R**.**id**.**context\_menu\_delete\_item**)** **{**  //Remove the item from the list  items**.**remove**(**itemInfo**.**position**);**    //Update the adapter to reflect the list change  adapter**.**notifyDataSetChanged**();**  **return** **true;**  **}**    **return** **super.**onContextItemSelected**(**item**);**  **}** |

We can now add more text to the context menu so the user knows what it’s currently deleting:

|  |
| --- |
| @Override  public void onCreateContextMenu**(**ContextMenu menu**,** View view**,** ContextMenu**.**ContextMenuInfo menuInfo**)** **{**  //Get the clicked item  AdapterView**.**AdapterContextMenuInfo info **=** **(**AdapterView**.**AdapterContextMenuInfo**)** menuInfo**;**    //Get the name of the clicked item  String clickedItem **=** **(**String**)** listView**.**getItemAtPosition**(**info**.**position**);**    //Inflate the context menu from the resource file  getMenuInflater**().**inflate**(**R**.**menu**.**context\_menu**,** menu**);**    //Find the delete MenuItem by its ID  MenuItem deleteButton **=** menu**.**findItem**(**R**.**id**.**context\_menu\_delete\_item**);**    //Get the title from the menu button  String originalTitle **=** deleteButton**.**getTitle**().**toString**();**    //Make a new title combining the original title and the name of the clicked list item  deleteButton**.**setTitle**(**originalTitle **+** " '" **+** clickedItem **+** "'?"**);**    //Let Android do its magic  **super.**onCreateContextMenu**(**menu**,** view**,** menuInfo**);**  **}** |

Run the application to see if it’s now possible to delete a single item by long clicking it.

The entire ListActivity should like like this now:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    /\* All the functionality we added to onCreate Earlier \*/  **}**    @Override  public boolean onCreateOptionsMenu**(**Menu menu**)** **{**  /\* All the functionality we added to onCreateOptionsMenu Earlier \*/  **}**    @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**  /\* All the functionality we added to onOptionsItemSelected Earlier \*/  **}**    @Override  public void onCreateContextMenu**(**ContextMenu menu**,** View view**,** Context**-**Menu**.**ContextMenuInfo menuInfo**)** **{**  //Inflate the context menu from the resource file  getMenuInflater**().**inflate**(**R**.**menu**.**context\_menu**,** menu**);**    **super.**onCreateContextMenu**(**menu**,** view**,** menuInfo**);**  **}**    @Override  public boolean onContextItemSelected**(**MenuItem item**)** **{**  //Retrieve info about the long pressed list item  AdapterView**.**AdapterContextMenuInfo itemInfo **=** **(**AdapterView**.**AdapterContextMenuInfo**)** item**.**getMenuInfo**();**  **if** **(**item**.**getItemId**()** **==** R**.**id**.**context\_menu\_delete\_item**)** **{**  //Remove the item from the list  items**.**remove**(**itemInfo**.**position**);**    //Update the adapter to reflect the list change  adapter**.**notifyDataSetChanged**();**  **return** **true;**  **}**    **return** **super.**onContextItemSelected**(**item**);**  **}**    **}** |

# 

# 

# Step 5: Dynamically adding list items

The final part of this tutorial is to have the ability to add list items dynamically, this means the user will be able to enter some text in a text field and it will be added to the ListView by the click of a button.

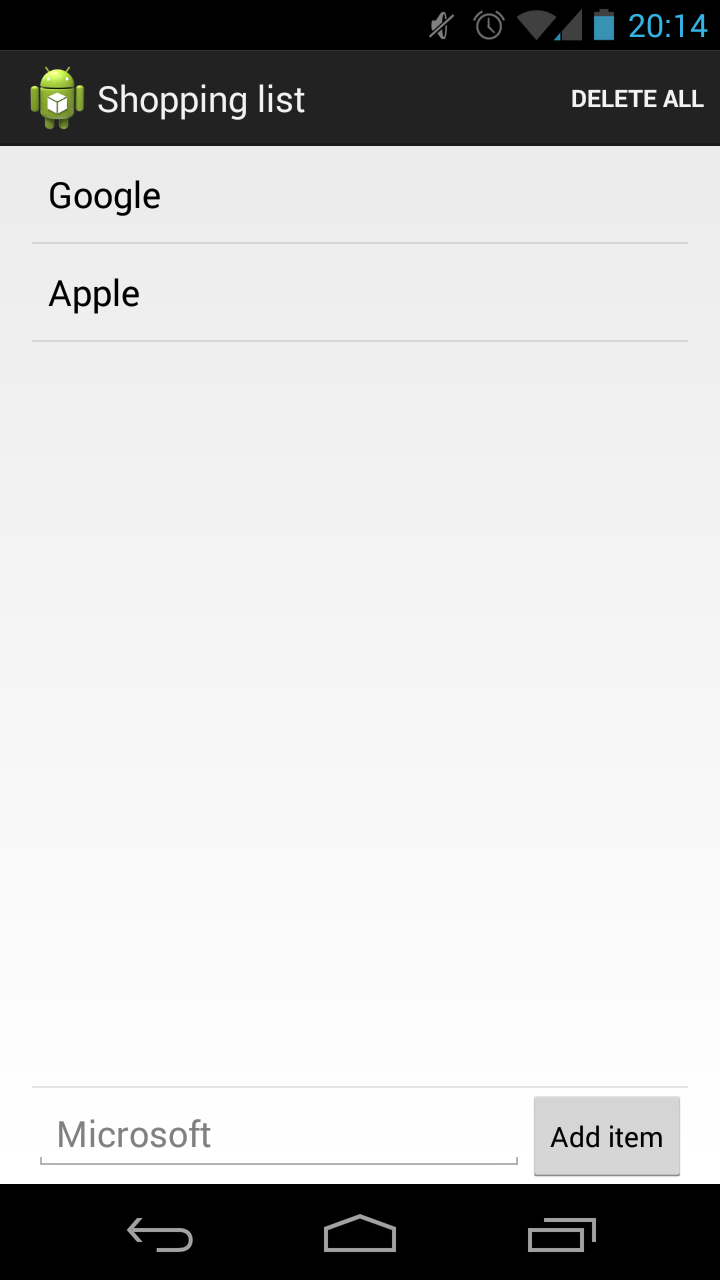
## 5.1: Adjusting the layout

The first step is to adjust our layout file so that it has an EditText and a Button at the bottom of the screen. Open ‘activity\_list.xml’ from your layout resources directory and change the file so it looks like this:

Note the ‘android:divider’ and ‘android:showDividers’ in the top LinearLayout.

|  |
| --- |
| <LinearLayout xmlns:android=**"http://schemas.android.com/apk/res/android"**  xmlns:tools=**"http://schemas.android.com/tools"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"match\_parent"**  android:divider=**"?android:dividerHorizontal"**  android:orientation=**"vertical"**  android:paddingLeft=**"@dimen/activity\_horizontal\_margin"**  android:paddingRight=**"@dimen/activity\_horizontal\_margin"**  android:showDividers=**"middle"**  tools:context=**".ListActivity"**>    <ListView  android:id=**"@+id/listView"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"match\_parent"**  android:layout\_weight=**"1"** />    <LinearLayout  android:layout\_width=**"match\_parent"**  android:layout\_height=**"wrap\_content"**  android:layout\_weight=**"0"**  android:orientation=**"horizontal"**>    <EditText  android:id=**"@+id/editText"**  android:layout\_width=**"match\_parent"**  android:layout\_height=**"wrap\_content"**  android:layout\_weight=**"1"**  android:hint=**"Microsoft"** />    <Button  android:id=**"@+id/button"**  style=**"?android:attr/buttonStyleSmall"**  android:layout\_width=**"wrap\_content"**  android:layout\_height=**"wrap\_content"**  android:layout\_weight=**"0"**  android:text=**"Add item"** />  </LinearLayout>  </LinearLayout> |

Run your application and make sure it looks similar to this:



## 5.2: Add items

Go back to the ListActivity and declare and initialize new variables for the newly created EditText and Button an initialize them using the following code:

addItemEditText **=** **(**EditText**)** findViewById**(**R**.**id**.**editText**);**

addItemButton **=** **(**Button**)** findViewById**(**R**.**id**.**button**);**

Then create a new method called ‘addListItem’ in the ListActivity, this method will gather the text that the user entered and add it to the ListView as a new item.

|  |
| --- |
| private void addListItem**()** **{**  //Get the user text from the textfield  String text **=** addItemEditText**.**getText**().**toString**();**    //Check if some text has been added  **if** **(!(**TextUtils**.**isEmpty**(**text**)))** **{**  //Add the text to the adapter  items**.**add**(**text**);**    //Notify the adapter that the action\_bar\_menu of data has changed and the view should be updated  adapter**.**notifyDataSetChanged**();**    //Clear the EditText for the next item  addItemEditText**.**setText**(**""**);**  **}** **else** **{**  //Show a message to the user if the textfield is empty  Toast**.**makeText**(**ListActivity**.this,** "Please enter some text in the textfield"**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **}** |

And finally, set an on click listener for the Button in the ‘onCreate’ method so it will respond to a clikc on it by the user:

|  |
| --- |
| //Set the add item button on click listener  addItemButton**.**setOnClickListener**(new** View**.**OnClickListener**()** **{**  @Override  public void onClick**(**View v**)** **{**  //Execute this method when the button is clicked  addListItem**();**  **}**  **});** |

Run your application to see if you can dynamically add items.

The final version of your ListActivity should look like this:

|  |
| --- |
| public class ListActivity **extends** ActionBarActivity **{**    // Adapter and ArrayList  private ArrayAdapter**<**String**>** adapter**;**  private List**<**String**>** items**;**    //Views  private ListView listView**;**  private EditText addItemEditText**;**  private Button addItemButton**;**    @Override  protected void onCreate**(**Bundle savedInstanceState**)** **{**  **super.**onCreate**(**savedInstanceState**);**    //Set the ListView Layout as the Activity content  setContentView**(**R**.**layout**.**activity\_list**);**    //Initialize the items action\_bar\_menu  items **=** **new** ArrayList**<**String**>();**    //Add some items to the action\_bar\_menu  items**.**add**(**"Google"**);**  items**.**add**(**"Apple"**);**    //Initialize the views  listView **=** **(**ListView**)** findViewById**(**R**.**id**.**listView**);**  addItemEditText **=** **(**EditText**)** findViewById**(**R**.**id**.**editText**);**  addItemButton **=** **(**Button**)** findViewById**(**R**.**id**.**button**);**    //Create the Array Adapter, give it a layout and a action\_bar\_menu of values  adapter **=** **new** ArrayAdapter**<**String**>(this,** android**.**R**.**layout**.**simple\_list\_item\_1**,** items**);**    //Set the newly crated adapter as the adapter for the listview  listView**.**setAdapter**(**adapter**);**    //Register the the ListView for a context menu  registerForContextMenu**(**listView**);**    //Set the listview on item click listener  listView**.**setOnItemClickListener**(new** AdapterView**.**OnItemClickListener**()** **{**  @Override  public void onItemClick**(**AdapterView**<?>** parent**,** View listItem**,** int position**,** long id**)** **{**  //Get the value of the item that the user clicked on  String clickedItem **=** **(**String**)** parent**.**getItemAtPosition**(**position**);**    //Display a Toast message to show the user the item he/she clicked on  Toast**.**makeText**(**ListActivity**.this,** "Clicked: " **+** clickedItem**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **});**    //Set the add item button on click listener  addItemButton**.**setOnClickListener**(new** View**.**OnClickListener**()** **{**  @Override  public void onClick**(**View v**)** **{**  //Execute this method when the button is clicked  addListItem**();**  **}**  **});**  **}**    @Override  public boolean onCreateOptionsMenu**(**Menu menu**)** **{**  //Inflate the action bar menu  getMenuInflater**().**inflate**(**R**.**menu**.**action\_bar\_menu**,** menu**);**    **return** **super.**onCreateOptionsMenu**(**menu**);**  **}**    @Override  public boolean onOptionsItemSelected**(**MenuItem item**)** **{**    //Handle the menu items  **if** **(**item**.**getItemId**()** **==** R**.**id**.**action\_bar\_menu\_delete\_all**)** **{**  //Clear the list and have the adapter update the listview  items**.**clear**();**  adapter**.**notifyDataSetChanged**();**  **}**  **return** **super.**onOptionsItemSelected**(**item**);**  **}**    @Override  public void onCreateContextMenu**(**ContextMenu menu**,** View view**,** ContextMenu**.**ContextMenuInfo menuInfo**)** **{**  //Get the clicked item  AdapterView**.**AdapterContextMenuInfo info **=** **(**AdapterView**.**AdapterContextMenuInfo**)** menuInfo**;**    //Get the name of the clicked item  String clickedItem **=** **(**String**)** listView**.**getItemAtPosition**(**info**.**position**);**    //Inflate the context menu from the resource file  getMenuInflater**().**inflate**(**R**.**menu**.**context\_menu**,** menu**);**    //Find the delete MenuItem by its ID  MenuItem deleteButton **=** menu**.**findItem**(**R**.**id**.**context\_menu\_delete\_item**);**    //Get the title from the menu button  String originalTitle **=** deleteButton**.**getTitle**().**toString**();**    //Make a new title combining the original title and the name of the clicked list item  deleteButton**.**setTitle**(**originalTitle **+** " '" **+** clickedItem **+** "'?"**);**    //Let Android do its magic  **super.**onCreateContextMenu**(**menu**,** view**,** menuInfo**);**  **}**    @Override  public boolean onContextItemSelected**(**MenuItem item**)** **{**  //Retrieve info about the long pressed list item  AdapterView**.**AdapterContextMenuInfo itemInfo **=** **(**AdapterView**.**AdapterContextMenuInfo**)** item**.**getMenuInfo**();**  **if** **(**item**.**getItemId**()** **==** R**.**id**.**context\_menu\_delete\_item**)** **{**  //Remove the item from the list  items**.**remove**(**itemInfo**.**position**);**    //Update the adapter to reflect the list change  adapter**.**notifyDataSetChanged**();**  **return** **true;**  **}**    **return** **super.**onContextItemSelected**(**item**);**  **}**    private void addListItem**()** **{**  //Get the user text from the textfield  String text **=** addItemEditText**.**getText**().**toString**();**    //Check if some text has been added  **if** **(!(**TextUtils**.**isEmpty**(**text**)))** **{**  //Add the text to the adapter  items**.**add**(**text**);**    //Notify the adapter that the action\_bar\_menu of data has changed and the view should be updated  adapter**.**notifyDataSetChanged**();**    //Clear the EditText for the next item  addItemEditText**.**setText**(**""**);**  **}** **else** **{**  //Show a message to the user if the textfield is empty  Toast**.**makeText**(**ListActivity**.this,** "Please enter some text in the textfield"**,** Toast**.**LENGTH\_LONG**).**show**();**  **}**  **}**  **}** |