Creating a List UI with Android

Michele Schimd - 2013

ListActivity

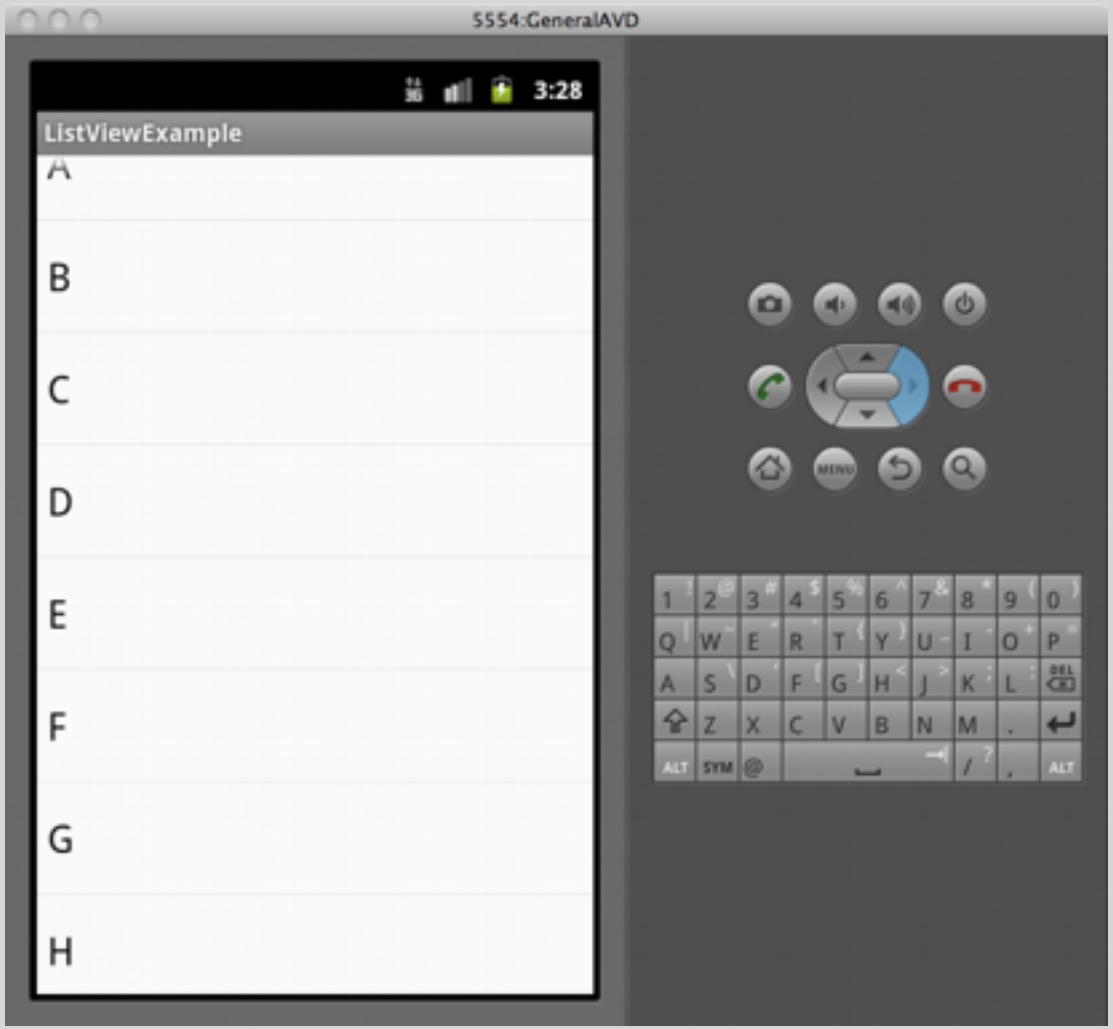
- Direct subclass of `Activity`
- By default a `ListView` instance is already created and rendered as the layout of the activity

  ➡ `myListActivit.getListView();`
- It defines the appearance of the activity (*i.e.* no need for layout file)
- Added in API Level 1 (*i.e.* compatible with all android versions)
  http://developer.android.com/reference/android/app/ListActivity.html

ListActivity - Example

```
public class MainActivity extends ListActivity {
   @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
      // ATTENTION: no setContentView

      String[] myStringArray = fillStringArrayWithData();
      ArrayAdapter<String> arrayAdapter =
            new ArrayAdapter<String>
                  (this,
                  android.R.layout.simple_list_item_1,
                  myStringArray);
      this.getListView().setAdapter(arrayAdapter);
   }
```



The ListViewExample on the virtual device and on a real device (both running *Android 2.3.3 Gingerbread*)

ListView and Adapter

- `ListView` contains an `AdapterView` (which usually you don't manage directly)
- `AdapterView` can be associated with a data structures (e.g. arrays, lists, ...) by means of `Adapter` objects
- In our example we use `ArrayAdapter` which is a implementation of the `Adapter` interface
- Then we set the `Adapter` for the list view contained in the `ListActivity` class using the `setAdapter` method:
  ```
  this.getListView().setAdapter(arrAdapter);
  ```
  or (even simpler)
  ```
  this.setListAdapter(arrAdapter)
  ```

More on ArrayAdapter

- The constructor we used takes 3 arguments
  1. An application context
  2. The id of the resource where list items will be rendered (*e.g.* `TextView`) for which we used a predifined android layout `simple_list_item_1`
  3. The array containing the objects. The text views will be filled with the result of `toString()` method called to these objects.

Interacting with the list

- To manage click events use the method

  `setOnItemClickListener(OnItemClickListener listener)`

  of the `ListView` object and pass a reference to an implementation of the `OnItemClickListener` interface (or create an anonymous class).

Interaction - Example

```
@Override
public void onItemClick(AdapterView<?> parent,
                        View v, int position, long id) {

    Object clickedObj = parent.getItemAtPosition(position);
    Log.i("[onItemClick]", clickedObj.toString());

}
```

- The method to be implemented is the `onItemClick` defined on the `OnItemClickListener` interface.
- `getItemAtPosition(pos)` gives you the `Object` (i.e. the `String` in our previous example, at position given (for which we can use the `position` parameter).

Customization of items

- If you need to customize the layout of items of the `ListView` you can't use the default views provided by the library like `simple_list_item_1`
- You need to create a `View` resource (*i.e.* define an XML for your custom layout)
- You need to tell the `Adapter` how to arrange data into your custom layout by overriding the `getView` by extending the `Adapter` class you're using (e.g. `ArrayAdapter`).

A Simple Problem

- We want to create a phonebook-like activity to display name and image of our contacts in a scrollable list. Each contact has
  - A `String` for his/her name
  - A `Bitmap` as for profile picture
- This is a "toy" example it is never a good idea to create a new activity when some default app does the same task (probably much better than us).

The Model Class

```
public class Contact {
   private String name;
   private Bitmap image;

   public Contact(String aName, Bitmap anImage) {
      this.name = aName;
      this.image = anImage;
   }

   // here the methods to get and set properties
}
```

The Custom Layout

- Create a new *Android Layout XML File*
- with root element `LinearLayout`
- and file name `custom_list_item_layout`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip"
    android:orientation="horizontal" >
    <ImageView
        android:id="@+id/pic"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_margin="6dip"/>
    <TextView
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="40dip"
        android:gravity="center_vertical"/>
</LinearLayout>
```

Custom Adapter

```
public class CustomArrayAdapter extends ArrayAdapter<Contact> {
    // we keep track of model objects in a local variable for our convenience
    private ArrayList<Contact> items;

    public CustomArrayAdapter(Context context, int textViewResourceId,
            ArrayList<Contact> items) {
        super(context, textViewResourceId, items);
        this.items = items;
    }
}
```

- We extend the `ArrayAdapter` class which is a *parametrized class*, we give as parameter a custom type `Contact`.
- We keep the list of *model objects* as a private variable `items`

Overriding `getView`

- The `getView` method of `ListAdapter` is called by the `ListView` when an item needs to be rendered
- By overriding such method we can define the custom appearance of every single item based on the data

getView(int position, View convertView, ViewGroup parent)

The getView method

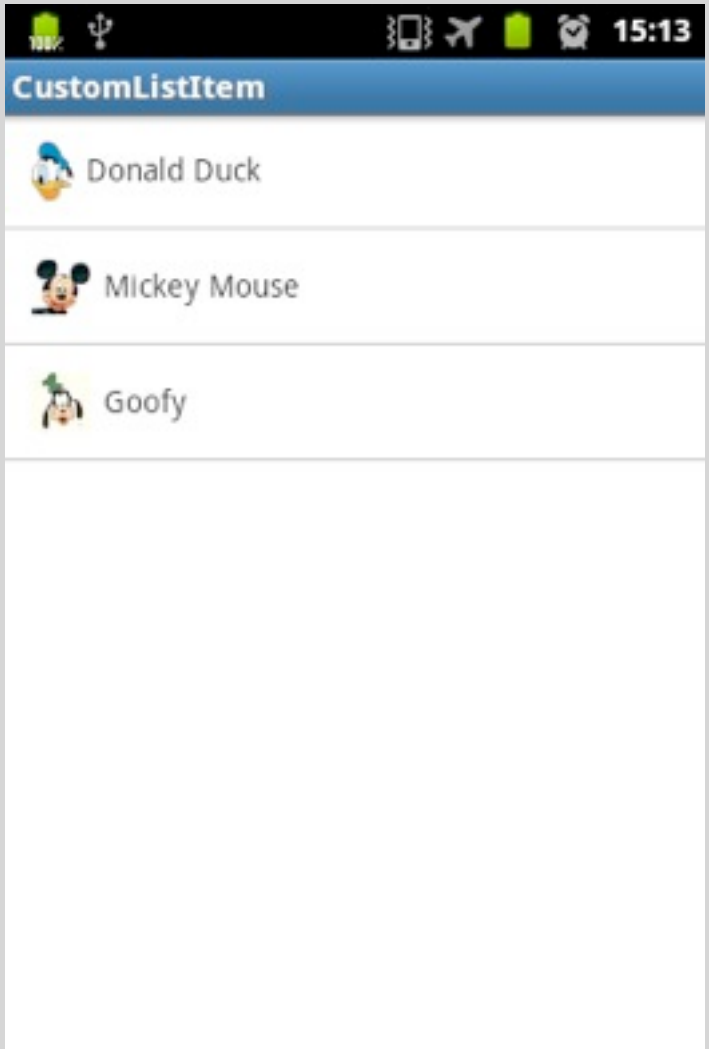
```
@Override
    public View getView(int position, View convertView, ViewGroup parent)
{
        // performance issue (see Google I/O video in references)
        View v = convertView;
        if (v == null) {
            LayoutInflater vi = (LayoutInflater) context
                    .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.custom_list_item_layout, null);
        }
        // position contains the index of the array for
        // the associated item so we retrieve the Contact
        Contact c = this.items.get(position);
        if (c != null) {
            TextView nameTextView = (TextView) v.findViewById(R.id.name);
            nameTextView.setText(c.getName());
            ImageView picImageView = (ImageView) v.findViewById(R.id.pic);
            picImageView.setImageBitmap(c.getImage());
        }
        return v;
    }
```

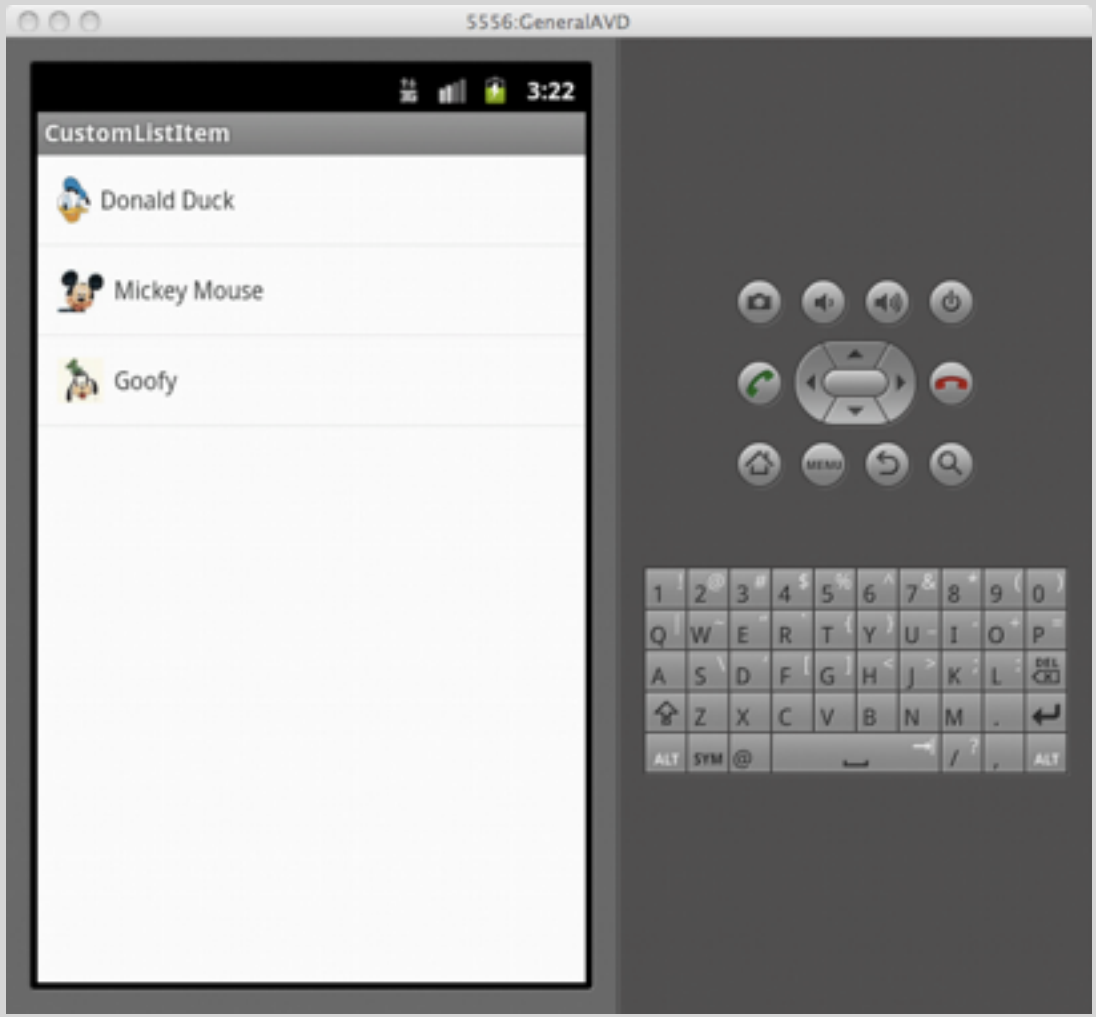
The onCreate method

```
public class MainActivity extends ListActivity {

    // The 'raw' data (loaded from db or from the net or ...)
    private ArrayList<Contact> contacts = null;
    // our custom adapter
    private CustomArrayAdapter adapter = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // load the data into our contacts ArrayList
        this.fillData();
        // create the CustomArrayAdatper
        this.adapter = new CustomArrayAdapter(this,
                R.layout.custom_list_item_layout, this.contacts);
         // set the adapter
        this.setListAdapter(this.adapter);
    }
}
```





The CustomListItem on the virtual device and on a real device (both running *Android 2.3.3 Gingerbread*)

Some remarks

- If you have thousands of contacts each with its own picture it is not a good idea to load them in a bulk, there ways to improve performance for lists (see Google I/O video in the references)
- As you change the model data list does not reflect such changes until the `ListView` is rendered another time
- `ListActivity` may not be the right choice if you have other elements among with the `ListView` in your activity (simply define a `ListView` in your layout and obtain a reference to it with the method `findViewById`).

Fragments

- Since Android 3.0 (API 11) UI design can use Fragments. They are available at lower API levels using *android support library*.
- Fragments are *behaviour portion of an activity*
  - ➡ An activity may contain multiple framgemnt
  - ➡ A fragment can be contained into multiple activity
  - ➡ A *fragment lifecycle is determined by the activity lifecycle*
  - ➡ Fragments receive their own events
- Fragments are useful when
  - ➡ Designing *UI for multiple devices* (e.g. Tablet vs Smarphone)
  - ➡ The *same UI must be used in several different activities* (even in different apps or in widgets)

Fragment creation

- Fragments are created by extending (directly or indirectly) the `Fragment` class.
- Android API provides an *off-the-shelf* `ListFragment` class that
  - ➡ Extends `Fragment`
  - ➡ Hosts a `ListView` object
  - ➡ Exposes *event handlers for list interaction*

MyListFragment

```
public class MyListFragment extends ListFragment {
    ...
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
        super.onActivityCreated(savedInstanceState);
        String[] values = this.fillStringArrayWithData();
        ArrayAdapter<String> adapter = new ArrayAdapter<String>
            (getActivity(),android.R.layout.simple_list_item_1, values);
        setListAdapter(adapter);
    }
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        super.onListItemClick(l, v, position, id);
        Object clickedObj = l.getAdapter().getItem(position);
        Log.i("onItemClick", clickedObj.toString());
    }
}
```

MainActivity and layout

```
public class MainActivity extends FragmentActivity {

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);
	}

}
```

Note: you must extend `FragmentActivity` class

```
<RelativeLayout 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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".MainActivity" >

    <fragment android:name="it.unipd.dei.fragmentlistexample.MyListFragment"
        android:id="@+id/my_list_fragment"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
</RelativeLayout>
```

References

- Android projects for the proposed examples on the Wiki of the course (*Course Material* Section)
- AdapterViews guide from the Android Developer reference
  http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews
- Tutorial on custom list item
  http://www.softwarepassion.com/android-series-custom-listview-items-and-adapters/
- Comprehensive ListView tutorial (Vogella Blog)
  http://www.vogella.com/articles/AndroidListView/article.html
- ListView performance considerations at Google I/O 2010
  http://www.youtube.com/watch?v=wDBM6wVEO70
- Fragment programming from Android Developer reference
  http://developer.android.com/guide/components/fragments.html