Embedded Systems

CS379 Embedded Systems

ListViews and Adapters

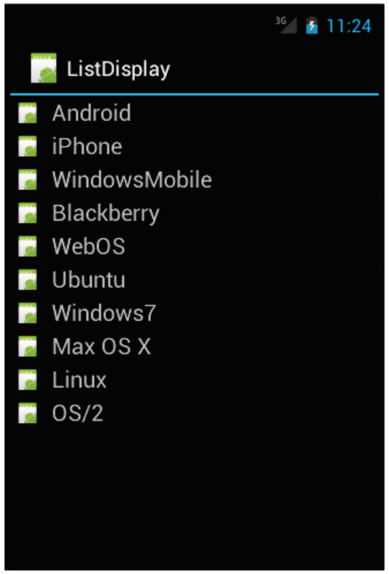
C.-Z. Yang
http://syslab.cse.yzu.edu.tw/~czyang

4.7 ListViews and Adapters

- A ListView is similar to a ScrollView.
- A ScrollView is an extension of the FrameLayout, and is suitable for holding a single control element.
- It provides the user with the scroll mechanism to reveal more content than can be displayed on the screen at once.
- A LinearLayout, containing multiple View items, can be placed within a ScrollView.

ListViews

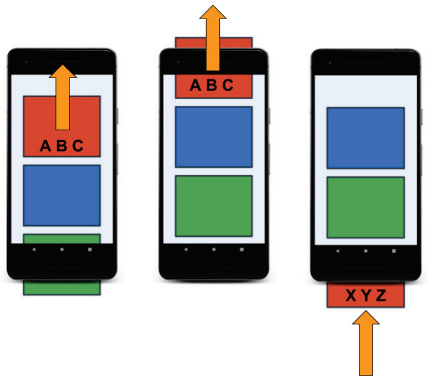
- A ListView is a specialized control that is optimized for displaying long lists of items
- When the data content for the layout is dynamic or not pre-determined, it is possible to use a layout that subclasses an AdapterView to populate the layout with views at runtime



http://www.vogella.com/tutorials/AndroidListView/img/xlistlayout10.png.pagespeed.ic.AySLgw_ZrH.png

RecyclerView

- For a more modern, flexible, and performant approach to displaying lists, use RecyclerView.
  - A flexible view for providing a limited window into a large data set.
- The items in your RecyclerView are arranged by a LayoutManager class.
  - LinearLayoutManager arranges the items in a one-dimensional list.
  - GridLayoutManager arranges all items in a two-dimensional grid.
  - StaggeredGridLayoutManager is similar to GridLayoutManager, but it does not require that items in a row have the same height (for vertical grids) or items in the same column have the same width (for horizontal grids).



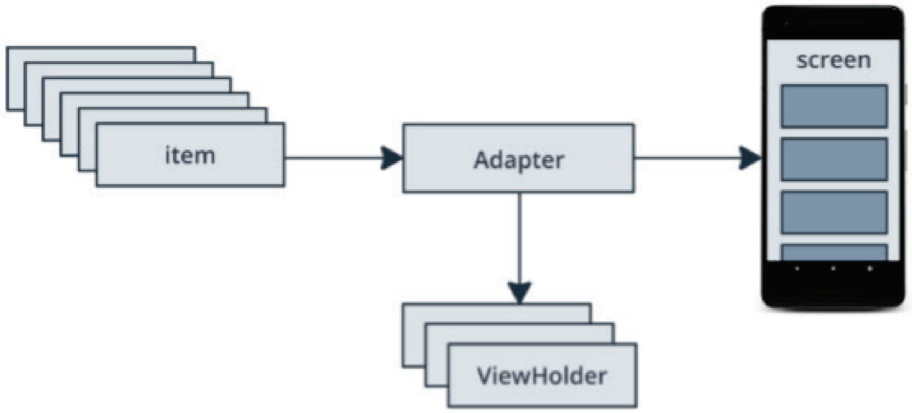
https://developer.android.com/codelabs/basic-android-kotlin-training-recyclerview-scrollable-list

RecyclerView

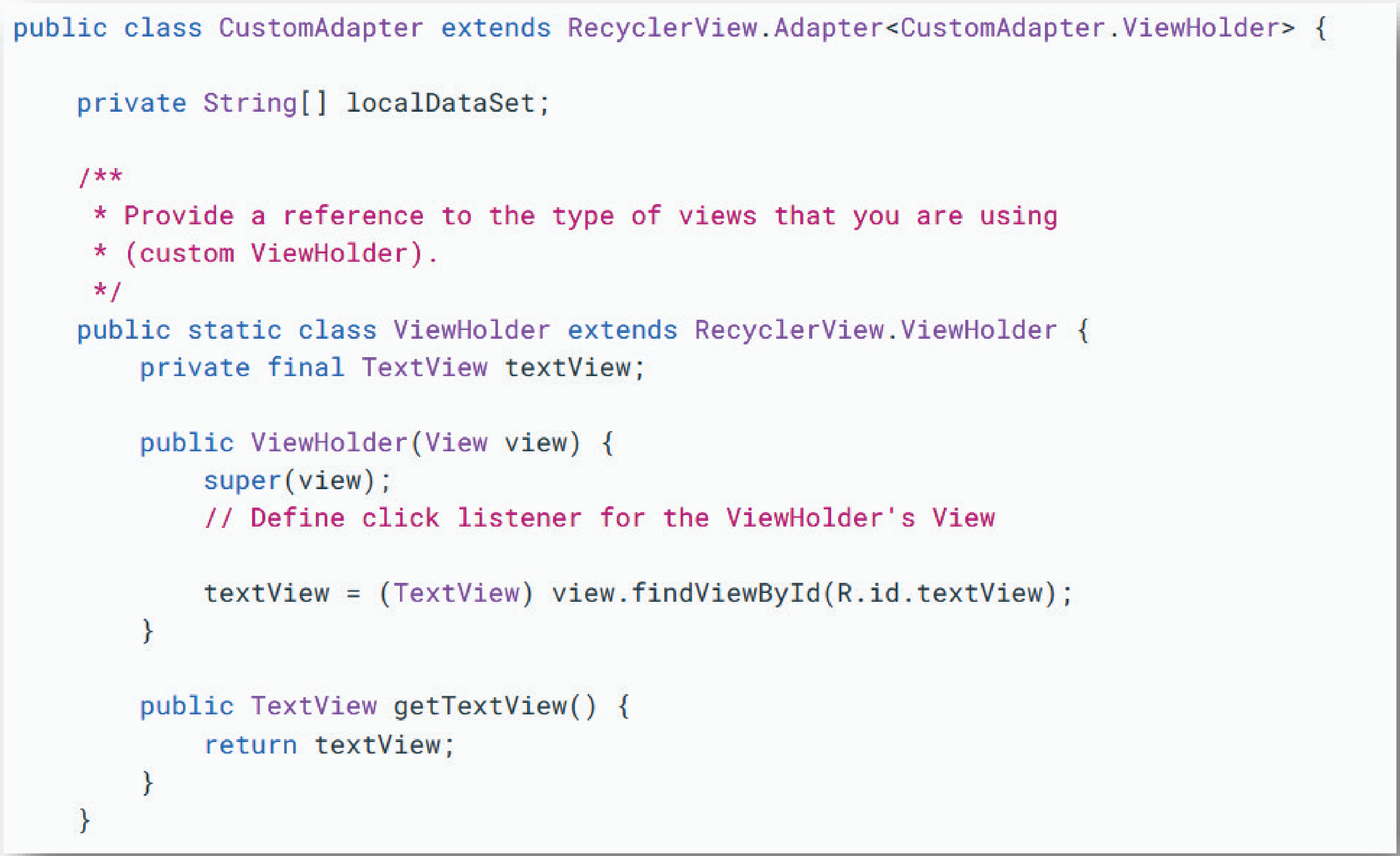
- Once you've determined your layout, you need to implement your Adapter and ViewHolder.
  - These two classes work together to define how your data is displayed.
- When you define your adapter, you need to override three key methods:
  - onCreateViewHolder(): RecyclerView calls this method whenever it needs to create a new ViewHolder.
  - onBindViewHolder(): RecyclerView calls this method to associate a ViewHolder with data.
  - getItemCount(): RecyclerView calls this method to get the size of the data set.

Example

- Here's a typical example of a simple adapter with a nested ViewHolder that displays a list of data.
- In this case, the RecyclerView displays a simple list of text elements.
- The adapter is passed an array of strings, containing the text for the ViewHolder elements.

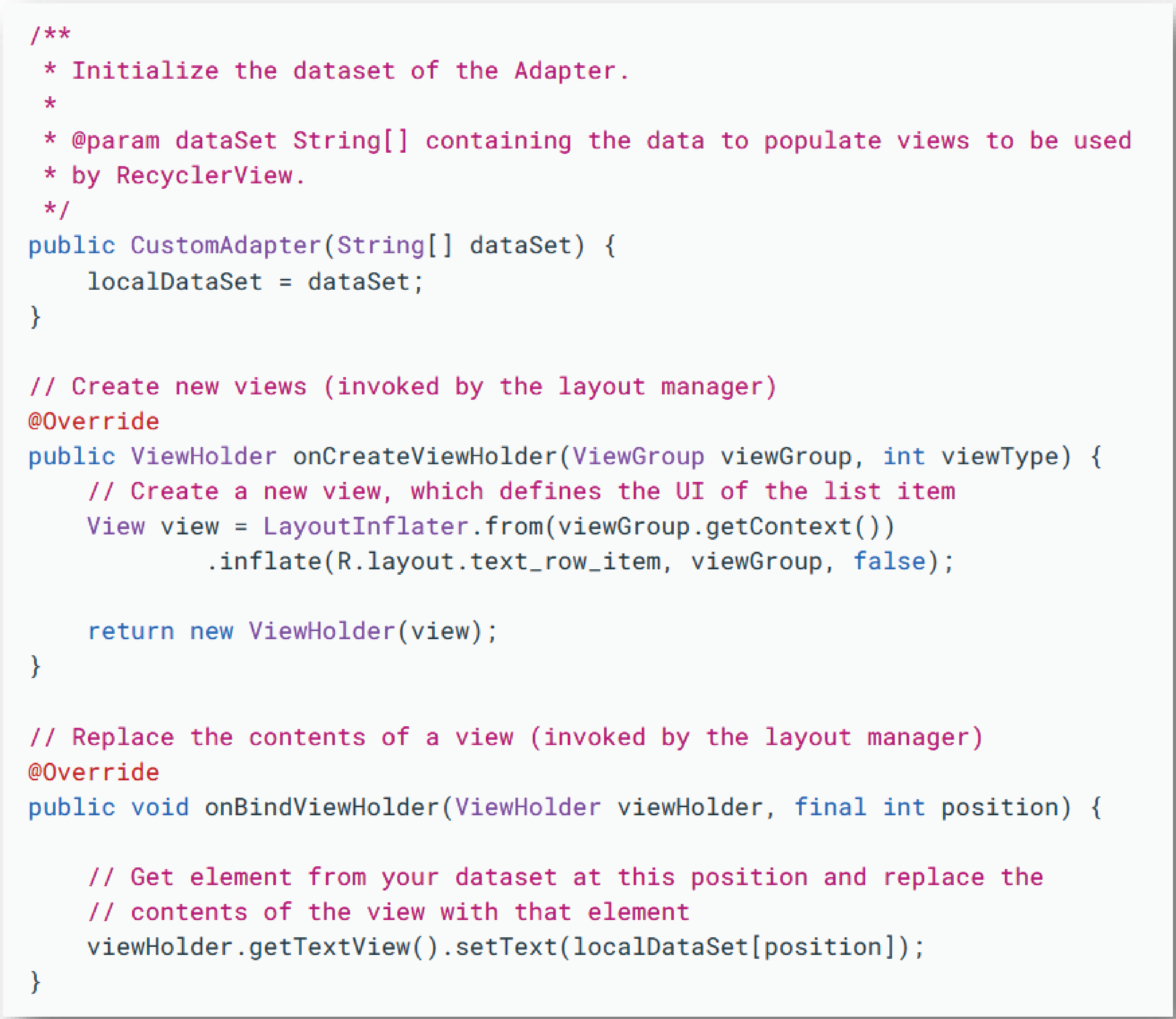


https://developer.android.com/codelabs/basic-android-kotlin-training-recyclerview-scrollable-list



```java
public class CustomAdapter extends RecyclerView.Adapter<CustomAdapter.ViewHolder> {

    private String[] localDataSet;

    /**
     * Provide a reference to the type of views that you are using
     * (custom ViewHolder).
     */
    public static class ViewHolder extends RecyclerView.ViewHolder {
        private final TextView textView;

        public ViewHolder(View view) {
            super(view);
            // Define click listener for the ViewHolder's View

            textView = (TextView) view.findViewById(R.id.textView);
        }

        public TextView getTextView() {
            return textView;
        }
    }
```

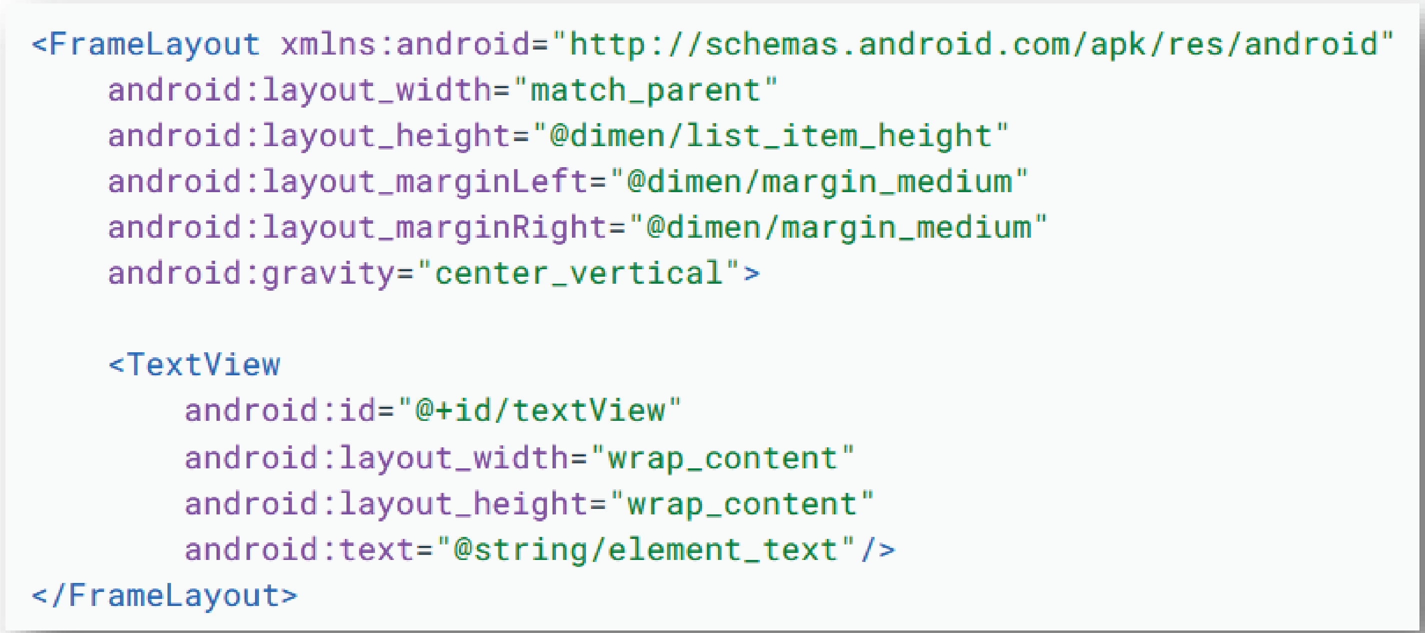
https://developer.android.com/guide/topics/ui/layout/recyclerview



```
/**
 * Initialize the dataset of the Adapter.
 *
 * @param dataSet String[] containing the data to populate views to be used
 * by RecyclerView.
 */
public CustomAdapter(String[] dataSet) {
    localDataSet = dataSet;
}

// Create new views (invoked by the layout manager)
@Override
public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
    // Create a new view, which defines the UI of the list item
    View view = LayoutInflater.from(viewGroup.getContext())
            .inflate(R.layout.text_row_item, viewGroup, false);

    return new ViewHolder(view);
}

// Replace the contents of a view (invoked by the layout manager)
@Override
public void onBindViewHolder(ViewHolder viewHolder, final int position) {

    // Get element from your dataset at this position and replace the
    // contents of the view with that element
    viewHolder.getTextView().setText(localDataSet[position]);
}
```

Layout

- The layout for the each view item is defined in an XML layout file, as usual. In this case, the app has a text_row_item.xml file like this:



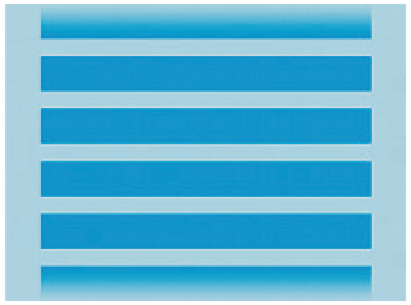
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="@dimen/list_item_height"
    android:layout_marginLeft="@dimen/margin_medium"
    android:layout_marginRight="@dimen/margin_medium"
    android:gravity="center_vertical">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/element_text"/>
</FrameLayout>
```

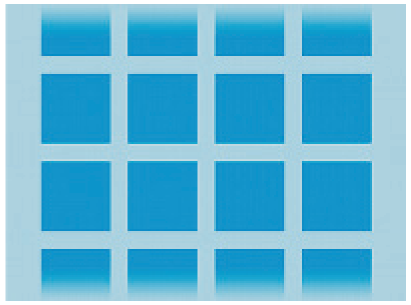
Adapters

- Adapters are used to collect the data content from a source, such as an array or database query, and convert each item result into a view that is placed into the list.
  - An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
  - The Adapter provides access to the data items.
  - The Adapter is also responsible for making a View for each item in the data set.

AdapterViews

- An AdapterView can be populated with data in a ListView or a GridView.
  - An AdpaterView, such as a ListView, can be filled with data by binding the AdapterView instance to an Adapter.





ArrayAdapters

- An ArraryAdpter is a common adapter.
- It is used when the data source is an array.
  - By default, ArraryAdapter creates a view for each array item, by calling toString() on each item and placing the contents in a TextView.

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this,
        android.R.layout.simple_list_item_1,
        myStringArray);

ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

If you have an array of strings you want to display in a ListView, initialize a new ArrayAdapter using a constructor to specify the layout for each string and the string array

4.8 Handling Click Events in a ListView

- A ListView might be populated with items that need to respond to a click event
- You can respond to click events on an item in an AdapterView by implementing the AdapterView.OnItemClickListener interface
- The onItemClick() callback method will always be invoked when an item in the AdapterView has been clicked.

Example code

- onItemClick()

```
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener() {
    public void onItemClick(AdapterView parent, View v, int position, long id)
    {
        // Display a messagebox.
        Toast.makeText(mContext,"You've got an event",Toast.LENGTH_SHORT).show();
    }
};

// Now hook into our object and set its onItemClickListener member
// to our class handler object.
mHistoryView = (ListView)findViewById(R.id.history);
mHistoryView.setOnItemClickListener(mMessageClickedHandler);
```

onItemClick()

- parent
  - AdapterView: The AdapterView where the click happened.
- view
  - View: The view within the AdapterView that was clicked (this will be a view provided by the adapter)
- position
  - int: The position of the view in the adapter.
- id
  - long: The row id of the item that was clicked.

- This anonymous instance defines the onItemClick() callback method to show a Toast that displays the index position (zero-based) of the selected item

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   GridView gridview = (GridView) findViewById(R.id.gridview);
   gridview.setAdapter(new ImageAdapter(this));

   gridview.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View v,
            int position, long id) {
         Toast.makeText(HelloGridView.this, "" + position,
               Toast.LENGTH_SHORT).show();
      }
   });
}
```

- getItem (int position)
  - Get the data item associated with the specified position in the data set.

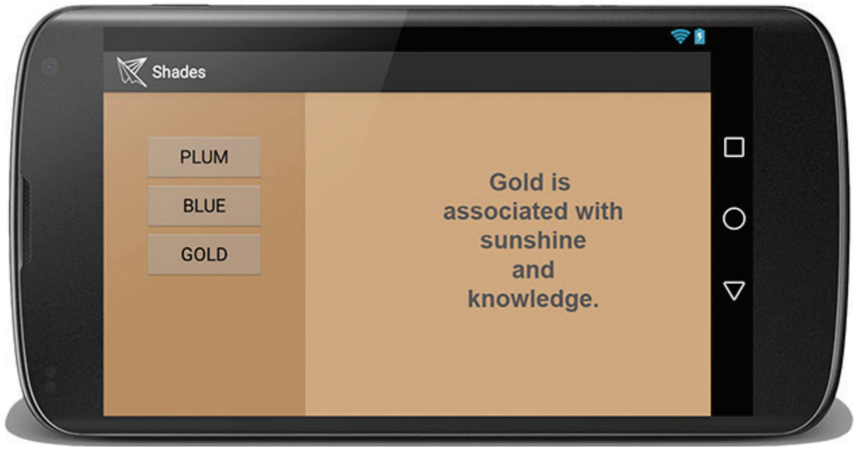
```
listView.setOnItemClickListener(new OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> parent, View view,
                int position, long id) {
            String string=adapter.getItem(position);
            Log.d("**********", string);

        }
    });
```

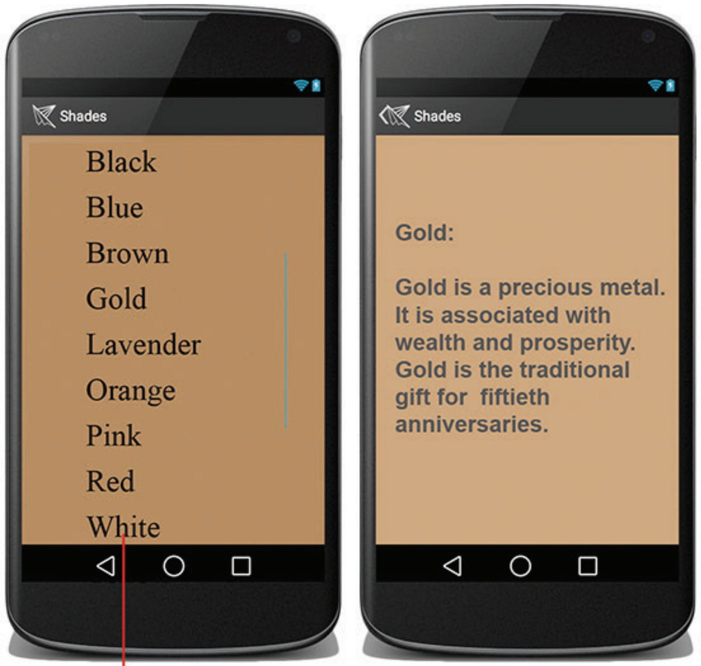
http://stackoverflow.com/questions/18405299/onitemclicklistener-using-arrayadapter-for-listview

Lab example 4-6: Shades II

- In this lab, the app Shades is improved by employing a ListView for the display of items and an Adapter for populating the master list with items from a dummy source.
- Shades I

Horizontal orientation-
Selecting a list item
updates the fragment on the right.

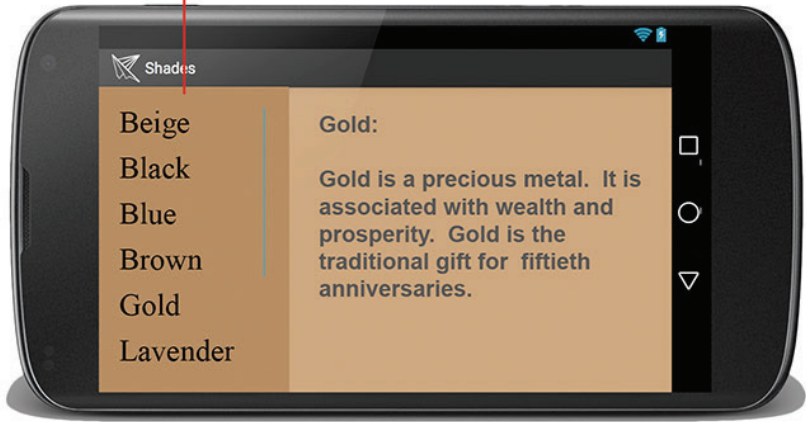




Gold:

Gold is a precious metal. It is associated with wealth and prosperity. Gold is the traditional gift for fiftieth anniversaries.

Scrollable, clickable, items in a `ListView`



Gold:

Gold is a precious metal. It is associated with wealth and prosperity. Gold is the traditional gift for fiftieth anniversaries.

The Layout design

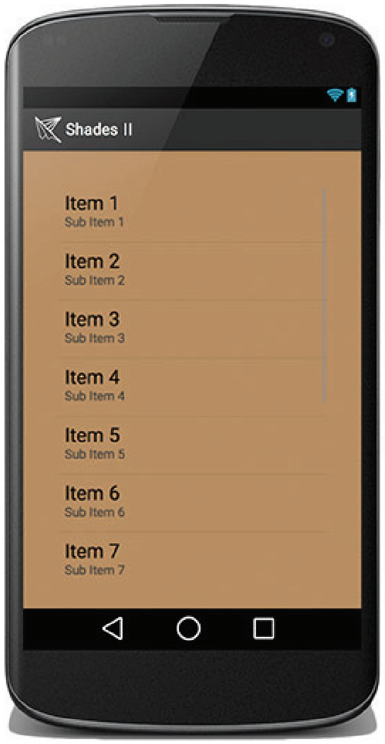
- The goal is to add a more robust list than the set of three color shades provided in Shades I.
- The blueprint of the View object

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

<TextView xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:minHeight="?android:attr/listPreferredItemHeight"
   android:gravity="center_vertical"
   android:id="@+id/list_item_shade_textview"
   android:textSize="40sp">

</TextView>
```

Restructure list_fragment.xml

- The layout for the master list

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:background="@color/caramel"
   android:orientation="vertical"
   android:padding="40dp" >

   <ListView
      android:id="@+id/listview_shades"
      android:layout_width="match_parent"
      android:layout_height="match_parent" />

</LinearLayout>
```



Defining the data source

- Color data is provided by items stored in the Java file DummyData.java in two arrays of data.
  - The first array contains the names of the colors.
  - The second array contains detailed information about each color.

```
package com.cornez.shadesii;

public class DummyData {
    // Some dummy data for the ListView.
    // Here's a sample of shades and their meaning
    public static String[] shade_name = {
            "Tan", "Black", "Blue", "Brown", "Gold",
            "Lavender", "Orange", "Pink", "Red",
            "White", "Yellow"};

    public static String[] shade_detail = {
            "Tan is a neutral color with a bit of the warmth of brown and the " +
                    "crisp, coolness of white. It can be exciting  when  " +
                    "coupled with other colors. It can be a relaxing " +
                    "color for a room.",
            "Black is the absence of color. In clothing, black is visually slimming. " +
                    "Black, like other dark colors, can make a room appear to shrink " +
                    "in size and even a well-lit room looks dark with a lot of black. " +
                    "Black can make other colors appear brighter.",
            "Blue is calming. It can be strong and steadfast or light and friendly. " +
                    "Almost everyone likes some shade of the color blue.",
        …
    };
}
```

Recoding MyListFragment.java

- It represents the master list in the master-detail flow pattern.
- Data source
  - The set of shade names, from the data source, is placed in the ordered list named shadelisting.
  - The shade detail information is placed in sadeNameDetail.

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {

     shadelisting = new ArrayList<String>(Arrays.asList(DummyData.shade_name));
     shadeNameDetail = new ArrayList<String>(Arrays.asList(DummyData.shade_detail));
```

Recoding MyListFragment.java

- An ArraryAdapter is created to take shadelisting data and use it to populate the ListView it is attached to.

```
// Now that we have some dummy shade data, create an ArrayAdapter.
// The ArrayAdapter will take data from a source (like our dummy shades) and
// use it to populate the ListView it's attached to.
final ArrayAdapter<String> mShadeAdapter =
        new ArrayAdapter<String>(
                getActivity(), // The current context (this activity)
                R.layout.list_item_shade, // The name of the layout ID.
                R.id.list_item_shade_textview, // The ID of the textview to populate.
                shadelisting);
```

Recoding MyListFragment.java

- The object listView is assigned the reference to the ListView on the layout.
- The ArrayAdapter mShadeAdapter is attached to the ListView, via the reference.

```
// Get a reference to the ListView, and attach this adapter to it.
ListView listView = (ListView) rootView.findViewById(R.id.listview_shades);
listView.setAdapter(mShadeAdapter);
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        String shadeIndexString = mShadeAdapter.getItem(i);
        String information = shadeIndexString + "\n\n\n" + shadeNameDetail.get(i);
        updateDetail(information);
    }

});
```

Concluding Remarks