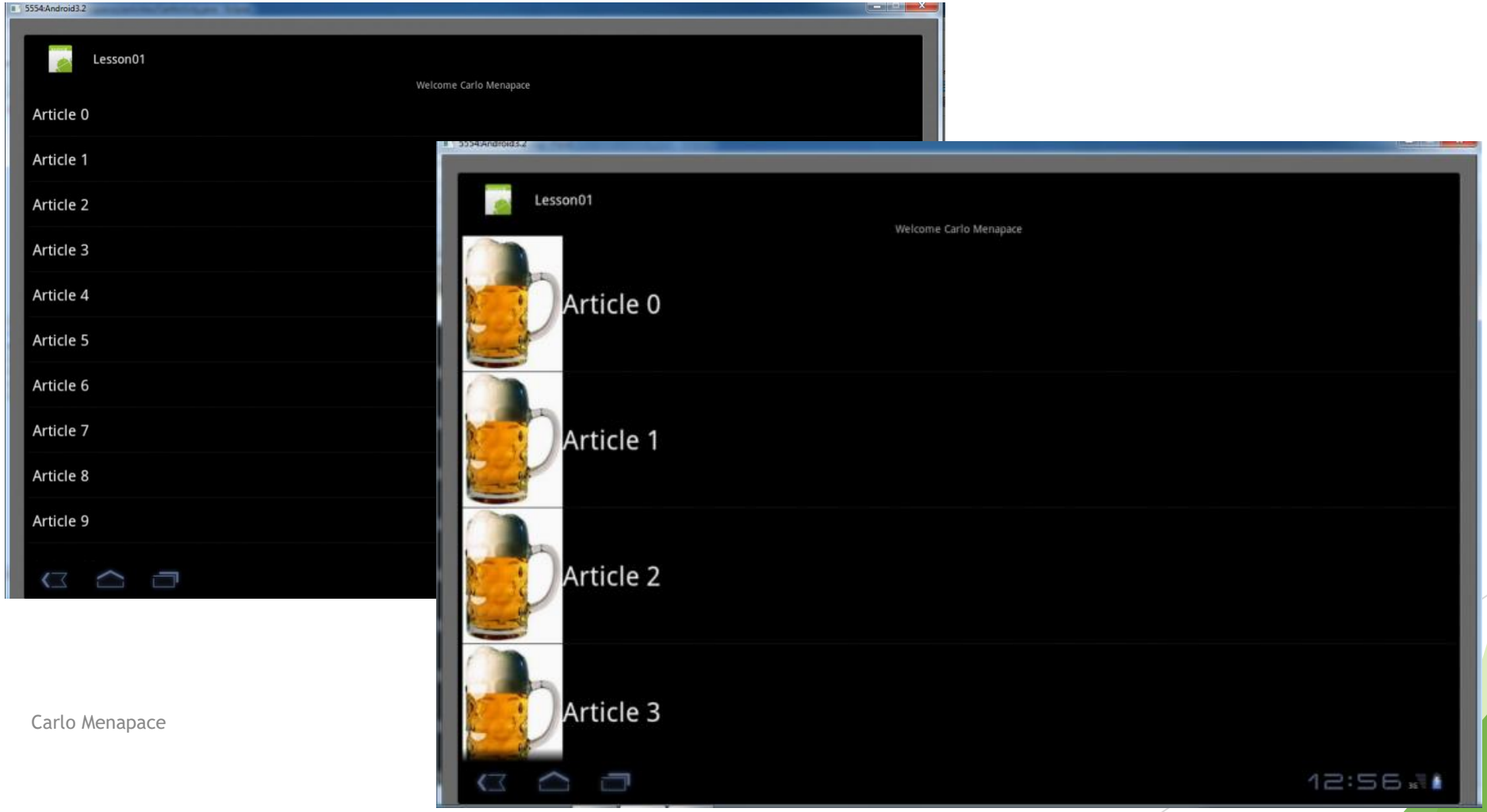
Programmazione di sistemi mobile e tablet

Android development

Carlo Menapace

TODAY'S ROADMAP



INTENT

You can start another activity by calling startActivity(), passing it an Intent that describes the activity you want to start. The intent specifies either the exact activity you want to start or describes the type of action you want to perform.

```
Intent intent = new Intent(this, NewActivity.class);
startActivity(intent);
```

An intent can also carry a small amounts of data to be used by the new activity.

```
Intent.putExtra(parameterIdentifier, parameter);
```

Best paractices tell us that the parameter identifier **must** be composed by

PACKAGE NAME + OUR PARAMETER IDENTIFIER

INTENT

In order to use your new activity, you have to declare it in the **manifest file** so that it could be accessible to the system. To declare your activity, open your manifest file and add an **<activity>** element as a child of the <application> element. For example:

```
<activity android:name="ActivityName" />
```

Once done this, your application should work correctly (If there aren't errors in it ☺)

LISTVIEW

A **ListView** is a ViewGroup that creates a list of scrollable items.

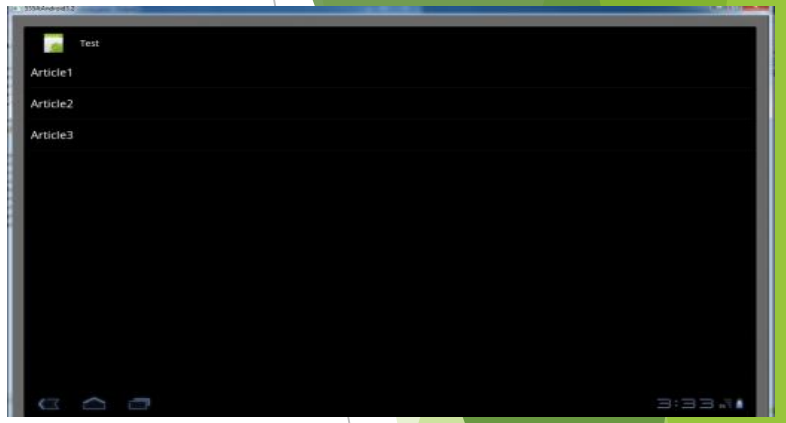
In order to insert elements into the list we have to use a ListAdapter. Based on the item's detail that we have to show, it is possible to use different Adapters:

- Extends our activity with ListActivity instead of Activity. In this way we are going to use a ready to use layout; each row of this layout is composed by a TextView that we can access using android.R.id.text1.

- Create a customized layout and programmatically implement what is needed. For this it is possibile to use:
  - SimpleAdapter (simple is just the name, we have to create a List<Map<String,String>>)
  - ArrayAdapter<ObjectOfItemsToInsertInOurList>

In the next slides we are going to see how does those Adapters works.

LISTVIEW – Extension approach




```
Public class Lesson01ListActivity extends ListActivity{
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);

                String[] arrayOfResources = {"Article1", "Article2", "Article3"};
                setListAdapter(new ArrayAdapter<String>(getApplicationContext(),
        android.R.layout.simple_list_item_1, arrayOfResources));
        }
}
```

Note that adopting this approach we can not customize the Activity layout. We can however customize the list row layout so that it is possibile to visualize every object we want.

LISTVIEW – SimpleAdapter

```
public class TestActivity extends Activity {

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

          ListView lv= (ListView)findViewById(R.id.listView);

          //create the grid item mapping
          String[] from = {"text"};
          int[] to =  {android.R.id.text1};

          // prepare the list of all records
          List<HashMap<String, String>> fillMaps = new ArrayList<HashMap<String, String>>();
          for(int i = 0; i < 100; i++){
         HashMap<String, String> map = new HashMap<String, String>();
         map.put("text","Articolo"+i );
         fillMaps.add(map);
           }
           SimpleAdapter adapter = new SimpleAdapter(this, fillMaps, android.R.layout.simplelist_item_1, from, to);
           lv.setAdapter(adapter);
   }
}
```

This approach is quite complex since, in order to prepare a list we have to set up a HashMap. For each item we want to use we have to create a map so that we can associate the resource with the object we want to add.

LISTVIEW – ArrayAdapter

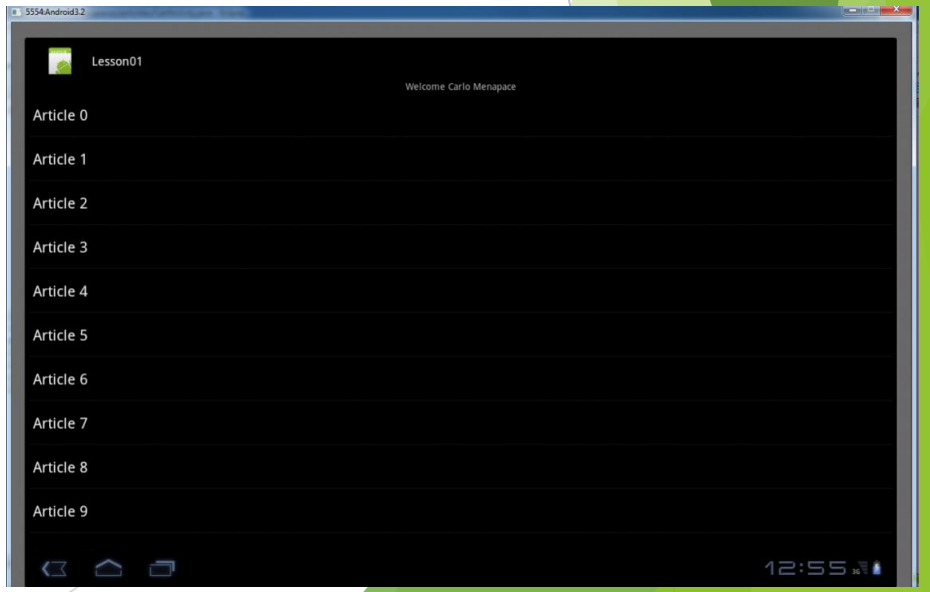
```
list.setAdapter(new ItemsComplexAdapter (getApplicationContext(), R.layout.listRowLayout, listOfArticles));

private class ItemsComplexAdapter extends ArrayAdapter<Article> {

private ArrayList<Article> mArticles;
private Context  mContext = null;
private int mlistRowLayout;

public ItemsComplexAdapter(Context context, int layoutId, ArrayList<Article> articles) {
        super(context, layoutId, articles);
        mArticles = articles;
        mContext = context;
        mlistRowLayout = layoutId;

        }
@Override
public View getView(int position, View view, ViewGroup parent) {
        LayoutInflater vi = (LayoutInflater)mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        view = vi.inflate(mlistRowLayout , null);

        Article article = mArticles .get(position);
}}
```


TODAY'S ROADMAP

- HANDS ON!

  You have to create an Android Application composed by **2 Activities** that interact passing between them some parameters. The first activity (let's call it A) accepts in input a **String** and an **Integer.** When we start the second Activity (let's call it B) we are going to use those parameters in order to:

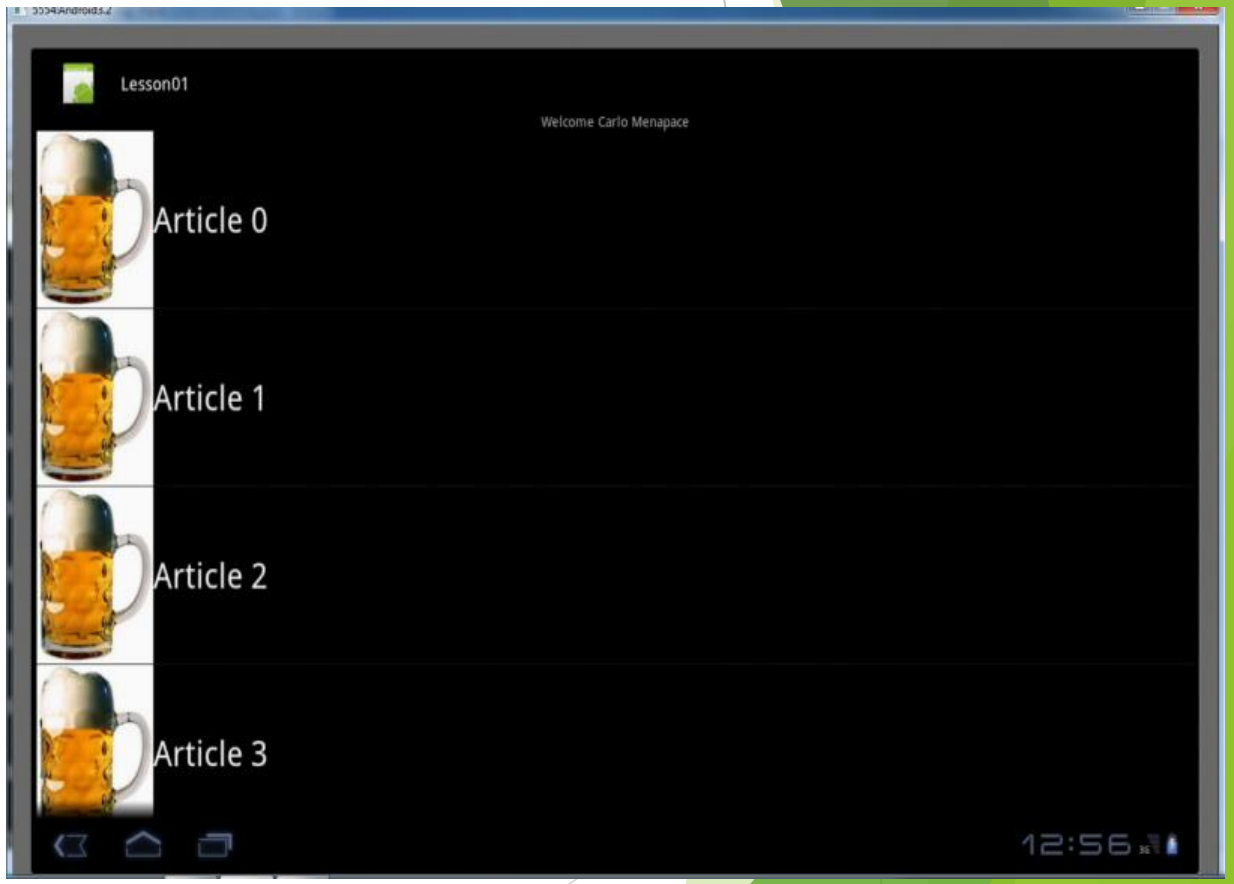
  - **Say Welcome ☺**
  - Create a list of **Items** with the given number.



ADVANCED EXERCISE

- HANDS ON +1

  Instead of creating a list of strings, create a list of **Objects** composed by an Image and a **String**.



<u>Suggestion</u>: Create a class named Article with attributes image and description.

WOULD YOU LIKE A TOAST?

A toast notification is a message that pops up on the surface of the window. It only fills the amount of space required for the message and the user's current activity remains visible and interactive. The notification automatically fades in and out, and does not accept interaction events.

We can print on our device whatever (STRING) we want in this way:

```
Toast.makeText(getApplicationContext(), stringWeWantToPrint, Toast.LENGHT_LONG).show();
```

The result of a Toast is something like this: