



Trường Đại học Bách Khoa Hà Nội
Hanoi University of Science and Technology

Chapter 4. Graphical User Interfaces

Hanoi University of Science and Technology

Nguyen Hong Quang
19/8/2012

© HUST 2012





Content

- 4.3. Advanced Adapter
 - ListView Widget/ ArrayList
 - Custom Adapter
 - TabHost
 - WebKit

TUTORIAL 5. Making Our List Be Fancy

Objective

3G 8:09 AM

LunchList

Foo Bar

Sir Lunch-A-Lot's

Name: That Other Place

Address: 1313 Mockingbird Lane

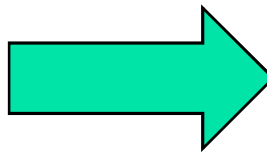
Type:

☐ Take-Out

☐ Sit-Down

☒ Delivery

Save



☒ T Nha an 1/5
Ta Quang Buu

☒ S Khach san Kim Lien
Kim Lien

☒ S Cafe Bach Khoa
Tran Dai Nghia

☒ D Bach my Bach Khoa
Cong Ky tuc xa

Name: Bach my Bach Khoa

Address: Cong Ky tuc xa

Type:

☐ Take-Out

☐ Sit-Down

☒ Delivery

Save



row.xml

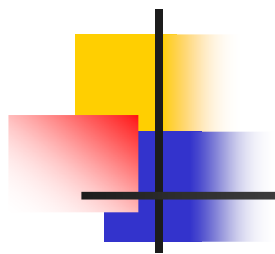
```
<LinearLayout>
    android:orientation="horizontal"
    <ImageView id="@+id/icon" />
    <LinearLayout>
        <TextView id="@+id/title" />
        <TextView android:id="@+id/address" />
    </LinearLayout>
</LinearLayout>
```



Basic ListView

```
class RestaurantAdapter extends  
    ArrayAdapter<Restaurant> {  
    RestaurantAdapter() {  
        super(LunchList.this,  
            android.R.layout.simple_list_item_1,  
            model);  
    }  
}
```

Dynamic ListView → override getView()



Using getView()



Dynamic ListView

- The `getView()` method of an Adapter is what an AdapterView (like ListView or Spinner) calls when it needs the View associated with a given piece of data the Adapter is managing.
- In the case of an ArrayAdapter, `getView()` is called as needed for each position in the array – "get me the View for the first row", "get me the View for the second row", etc.



Method getView()

- View row=super.**getView(position, convertView, parent);**
- It chains to the superclass' implementation of getView(), which returns to us an instance of our row View, as prepared by ArrayAdapter.
- In particular, our word has already been put into the TextView, since ArrayAdapter does that normally.



Method getView()

```
View row=super.getView(position, convertView,  
parent);
```

```
Restaurant r = model.get(position);  
((TextView)row.findViewById(R.id.title)).setText(r.get  
Name());  
((TextView)row.findViewById(R.id.address)).setText(r.  
getAddress());
```

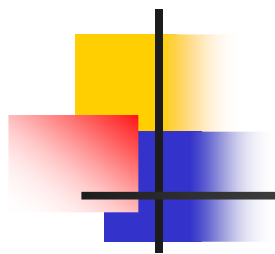
Method getView() – change icon

```
ImageView icon=(ImageView)row.findViewById(R.id.icon);  
if (r.getType().equals("sit_down")) {  
    icon.setImageResource(R.drawable.ball_red);  
}  
else if (r.getType().equals("take_out")) {  
    icon.setImageResource(R.drawable.ball_yellow);  
}  
else {  
    icon.setImageResource(R.drawable.ball_green);  
}  
return(row);
```



Method getView() – change icon

- It finds our ImageView and applies a business rule to set which icon should be used, referencing one of two drawable resources (R.drawable.ok and R.drawable.delete).



Inflating Rows



Inflating Rows Ourselves

- The solution shown in this version of the DynamicDemo works fine.
- However, there will be times when ArrayAdapter cannot even be used for setting up the basics of our row.
- In that case, we may need to do all of the work ourselves, starting with inflating our rows.



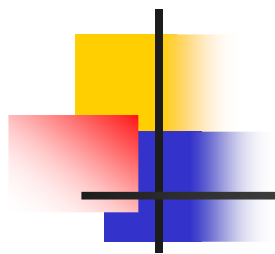
Inflating Rows Ourselves

- LayoutInflater
inflater=**getLayoutInflater();**
- View row=inflater.**inflate(R.layout.row,
parent, false);**



Comments

- The getView() implementation is inefficient. Every time the user scrolls, we have to create a bunch of new View objects to accommodate the newly-shown rows.
 - the list appears to be sluggish
 - it will be bad due to battery usage – every bit of CPU that is used eats up the battery. This is compounded by the extra work the garbage collector needs to do to get rid of all those extra objects you create.



Using convertView()



Using convertView

- Sometimes, convertView will be null. In those cases, you have to create a new row View from scratch (e.g., via inflation), just as we did before.
- However, if convertView is not null, then it is actually one of your previously-created View objects!
- This will happen primarily when the user scrolls the ListView – as new rows appear, Android will attempt to recycle the views of the rows that scrolled off the other end of the list, to save you having to rebuild them from scratch.



Method getView()

```
View row=convertView;
```

```
if (row==null) {
```

```
    LayoutInflater inflater=getLayoutInflater();
```

```
    row=inflater.inflate(R.layout.row, null);
```

```
}
```



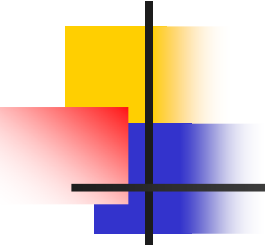
Using convertView - faster

- The advantage is that we avoid the potentially-expensive inflation step.
- In fact, according to statistics cited by Google at the 2010 Google I/O conference, a ListView that uses a recycling ListAdapter will perform 150% faster than one that does not.



Using convertView - it uses much less memory

- Each widget or container – in other words, each subclass of View – holds onto up to 2KB of data, not counting things like images in ImageView widgets.
- Each of our rows, therefore, might be as big as 6KB.
- For our list of 25 nonsense words, consuming as much as 150KB for a non-recycling list (25 rows at 6KB each) would be inefficient but not a huge problem.
- A list of 1,000 nonsense words, though, consuming as much as 6MB of RAM,



Using the Holder Pattern



Using the Holder Pattern

- Another somewhat expensive operation we do a lot with fancy views is call `findViewById()`.
 - This dives into our inflated row and pulls out widgets by their assigned identifiers
- Since `findViewById()` can find widgets anywhere in the tree of children of the row's root View, this could take a fair number of instructions to execute



ViewHolder

- All View objects have `getTag()` and `setTag()` methods.
- These allow to associate an arbitrary object with the widget.
 - Use that "tag" to hold an object that, in turn, holds each of the child widgets of interest.



ViewHolder

- By attaching that holder to the row View, every time we use the row, we already have access to the child widgets we care about, without having to call `findViewById()` again.



RestaurantHolder

```
static class RestaurantHolder {  
    private TextView name=null;  
    private TextView address=null;  
    private ImageView icon=null;
```

```
    RestaurantHolder(View row) {  
        name=(TextView)row.findViewById(R.id.title);  
        address=(TextView)row.findViewById(R.id.address);  
        icon=(ImageView)row.findViewById(R.id.icon);  
    }
```



RestaurantHolder

```
void populateFrom(Restaurant r) {  
    name.setText(r.getName());  
    address.setText(r.getAddress());  
    if (r.getType().equals("sit_down")) {  
        icon.setImageResource(R.drawable.ball_red);  
    }  
    else if (r.getType().equals("take_out")) {  
        icon.setImageResource(R.drawable.ball_yellow);  
    }  
    else {  
        icon.setImageResource(R.drawable.ball_green);  
    }  
}
```




RestaurantHolder

- ViewHolder holds onto the child widgets, initialized via `findViewById()` in its constructor.



Using RestaurantHolder

- Using ViewHolder is a matter of creating an instance whenever we inflate a row and attaching said instance to the row View via `setTag()`



```
public View getView(int position, View convertView, ViewGroup
parent) {
    View row=convertView;
    RestaurantHolder holder=null;
    if (row==null) {
        LayoutInflater inflater=getLayoutInflater();
        row=inflater.inflate(R.layout.row, parent, false);
        holder=new RestaurantHolder(row);
        row.setTag(holder);
    }
    else {
        holder=(RestaurantHolder)row.getTag();
    }
    holder.populateFrom(model.get(position));
    return(row);
}
```

New getView()



Performance

- Using a holder helps performance, but the effect is not as dramatic.
- Whereas recycling can give you a 150% performance improvement, adding in a holder increases the improvement to 175%.



Trường Đại học Bách Khoa Hà Nội

Hanoi University of Science and Technology



