Andorid Programming

Week 4

Mina Jung

EECS, Syracuse University

Spring 2017

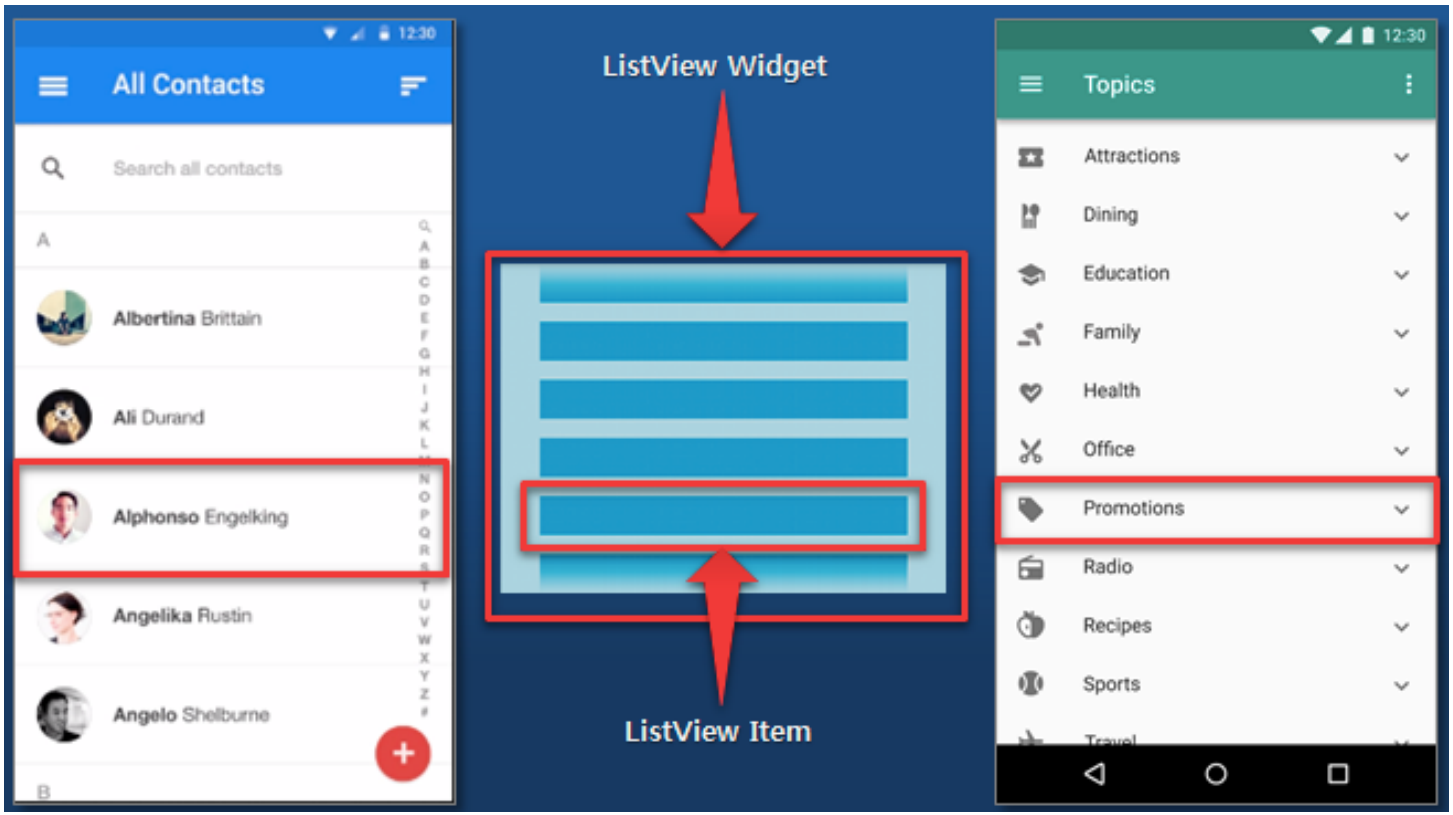
Part I

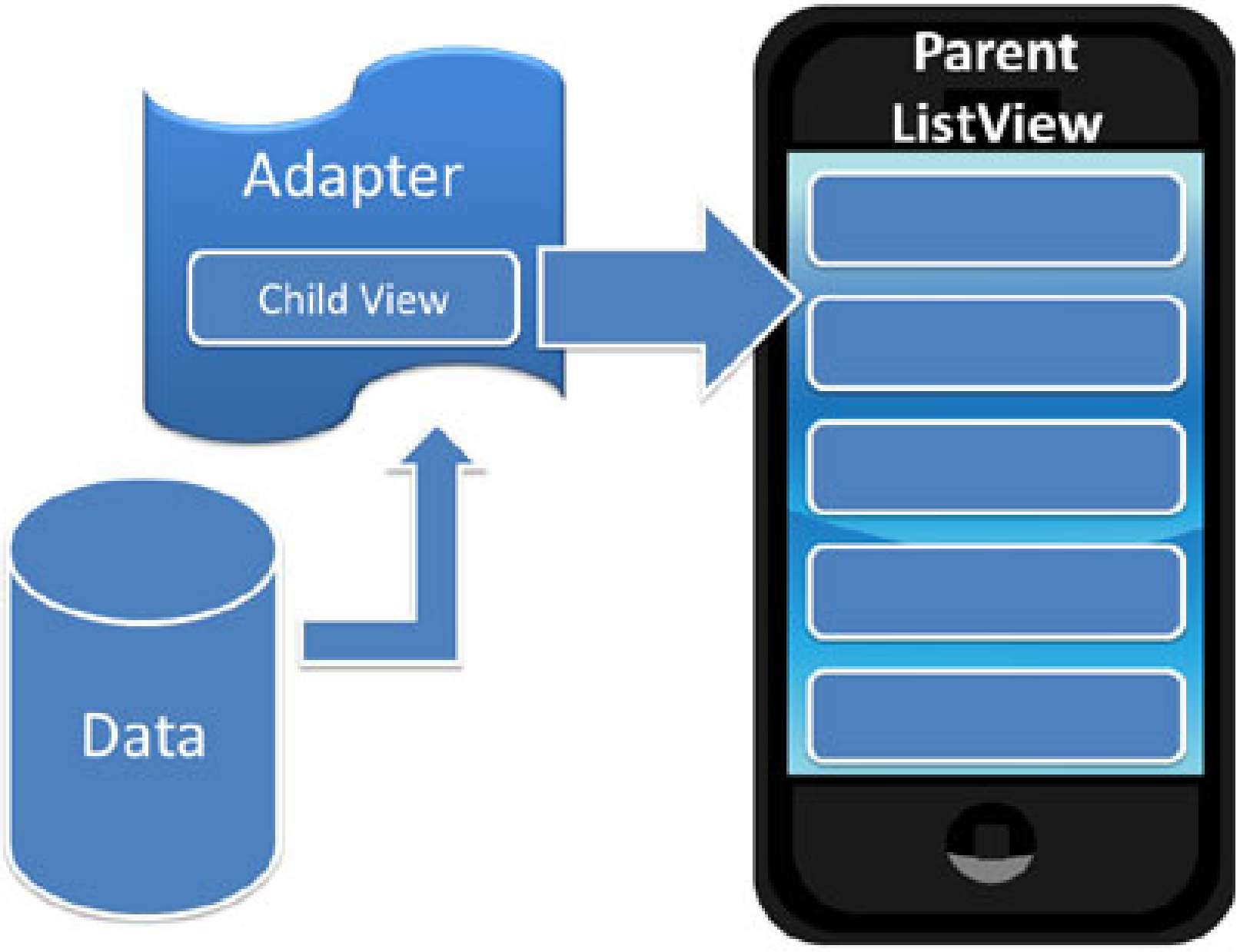
ListView? Replaced with RecyclerView

Outline I

1. Review the material design specification.
2. Apply the material theme to your app.
3. Create your layouts following material design guidelines.
4. Specify the elevation of your views to cast shadows.
5. Use system widgets for lists and cards.
6. Customize the animations in your app.

- ListView is a view group that displays a list of scrollable items
  - list items are automatically inserted to the list using an Adapter(Click!)
    - source from array or database query data set




Adapter
Child View
Data
Parent
ListView

- acts as a bridge between an AdapterView and the underlying data for that view
- provides access to the data items
- responsible for making a View for each item in the data set

abstract	int getCount()
	How many items are in the data set
abstract	Object getItem(int position)
	Get the data item associated with the specified position
abstract	View getView(int position, View convertView, ViewGroup parent)
	Get a View that displays the data at the specified position

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout>
    <!--
    Design Your Own UI
    -->

    <ListView
        android:id="@+id/listview1"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</RelativeLayout>
```

```
public class MainActivity extends AppCompatActivity {
    static final String[] LIST_MENU = {"LIST1", "LIST2", "LIST3"};
    // ...
}
```

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

    ArrayAdapter adapter = new ArrayAdapter(this,
        android.R.layout.simple_list_item_1, LIST_MENU);

    ListView listview = (ListView) findViewById(R.id.listview1);
    listview.setAdapter(adapter);
    // ...
}
```

```
@Override
public View getView(int position, View view, ViewGroup parent) {

    if(view == null) {
        LayoutInflater inflater =
            (LayoutInflater)parent.getContext().getSystemService(
            Context.LAYOUT_INFLATER_SERVICE );
        view = inflater.inflate(R.layout.item, parent, false);
    }

    TextView tName = (TextView)
         view.findViewById(R.id.item_name);
    TextView tDesc = (TextView)
         view.findViewById(R.id.item_desc);

    MyData item = getItem(position);

    tName.setText(item.getName());
    tDesc.setText(item.getDescription());

    return view;
}
```

- Question: when you scroll down, the views disappearing on the top are reused to display items on the bottom of your list? or just destroyed?

- the process of inflating views takes long time

- How to reuse the invisible views?
  - tag the views in order to avoid to inflate them again.

  1. create a class (called ViewHolder)

```
static class ViewHolder{
 TextView tName;
 TextView tDesc;
}
```

  2. change the Adapter with ViewHolder

```
@Override
public View getView(int position, View view, ViewGroup parent) {
    ViewHolder holder = null;
    if(view == null) {
        LayoutInflater inflater =
            (LayoutInflater)parent.getContext().getSystemService(
            Context.LAYOUT_INFLATER_SERVICE );
        view = inflater.inflate(R.layout.item, parent, false);
        holder = new ViewHolder();
        holder.tName = (TextView)
            view.findViewById(R.id.item_name);
        holder.tDesc = (TextView)
            view.findViewById(R.id.item_desc);
        view.setTag(holder);
    }
    else {
        holder = (ViewHolder) view.getTag();
    }
    MyData item = getItem(position);
    holder.tName.setText(item.getName());
    holder.tDesc.setText(item.getDescription());
    return view;
}
```

```
listview.setOnItemClickListener(new
     AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView parent, View v, int
         position, long id) {

        // get TextView's Text
        String strText = (String)
             parent.getItemAtPosition(position);

        Toast.makeText(getApplicationContext(), strText,
             Toast.LENGTH_SHORT);
    }
});
```

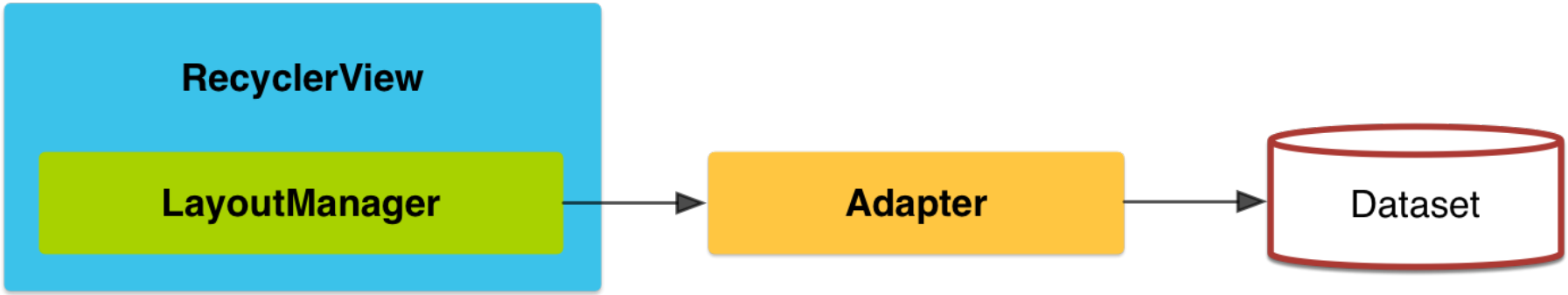
Part II

RecyclerView – More Advanced and Flexible version of ListView

Outline I

Please read **The Good, the Bad and the Ugly Things About the New RecyclerView**

Please read **RecyclerView VS ListView**



- Adapter
  - Wraps the data set and creates views for individual item
- ViewHolder
  - Holds all sub-views that depend on the current item
- LayoutManager
  - Places items within the available area
- ItemDecoration
  - Draws decorations around or on top of each item
- ItemAnimation
  - Animates items when they are added, removed or reordered

- Caches of View objects
- RecyclerView.ViewHolder subclass
  - can access the root view of your ViewHolder
  - no need to store within the ViewHolder subclass
- inner class of the Adapter

```
public static class ViewHolder extends RecyclerView.ViewHolder {
    TextView tName;
    TextView tDesc;

    public ViewHolder(View view) {
        super(view);
        tName = (TextView) itemView.findViewById(R.id.item_name);
        tDesc = (TextView) itemView.findViewById(R.id.time_desc);
    }
}
```

- Two roles of Adapters
  - provide access to the underlying data set
  - responsible for creating the correct layout for individual items
- Adapters for ListView, AutoCompleteTextView, Spinner and so on inherit from AdapterView
- RecyclerView uses a new RecyclerView.Adapter class
  - must implement three methods
    - public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)
    - public void onBindViewHolder(ViewHolder holder, int position)
    - public int getItemCount()

- Example

```
1  public class MySimpleRecyclerAdapter extends
       RecyclerView.Adapter<MySimpleRecyclerAdapter.ViewHolder>
2  {
3      private List<Item> mItems;
4      private int lastPosition = -1;
5
6      public MySimpleRecyclerAdapter(List<Item> items) {
7          mItems = items;
8      }
9
10     // must Generate : create a new View
11     @Override
12     public ViewHolder onCreateViewHolder(ViewGroup parent, int
           viewType) {
13         // create a new view
14         View v =
               LayoutInflater.from(parent.getContext()).inflate(R.layout.rec_i
               parent, false);
15         return new ViewHolder(v);
16     }
17
18     // must Generate : work like getView method of ListView
19     @Override
20     public void onBindViewHolder(ViewHolder holder, int position) {
```

```
21          holder.imageView.setImageResource(mItems.get(position).image);
22          holder.textView.setText(mItems.get(position).imagetitle);
23          setAnimation(holder.imageView, position);
24      }
25
26
27      // must Generate
28      @Override
29      public int getItemCount() {
30          return mItems.size();
31      }
32
33      private void setAnimation(View view, int position) {
34          if (position > lastPosition) {
35              Animation animation =
                        AnimationUtils.loadAnimation(view.getContext(),
                        android.R.anim.slide_in_left);
36              view.startAnimation(animation);
37
38              lastPosition = position;
39          }
40      }
41
42      public static class ViewHolder extends RecyclerView.ViewHolder {
43
44          public ImageView imageView;
```

```
45          public TextView textView;
46
47          public ViewHolder(View view) {
48              super(view);
49              imageView = (ImageView)
                    view.findViewById(R.id.recImage);
50              textView = (TextView)
                    view.findViewById(R.id.recImageTitle);
51          }
52      }
53  }
```

- responsible for the layout of all child views
- must set a LayoutManager for RecyclerView
- default: LinearyLayoutManager

```
1 LinearLayoutManager layoutManager = new
      LinearLayoutManager(context);
2 layoutManager.setOrientation(LinearLayoutManager.VERTICAL);
3 layoutManager.scrollToPosition(currPos);
4 recyclerView.setLayoutManager(layoutManager);
```

  - findFirstVisibleItemPosition()
  - findFirstCompletelyVisibleItemPosition()
  - findLastVisibleItemPosition()
  - findLastCompletelyVisibleItemPosition()

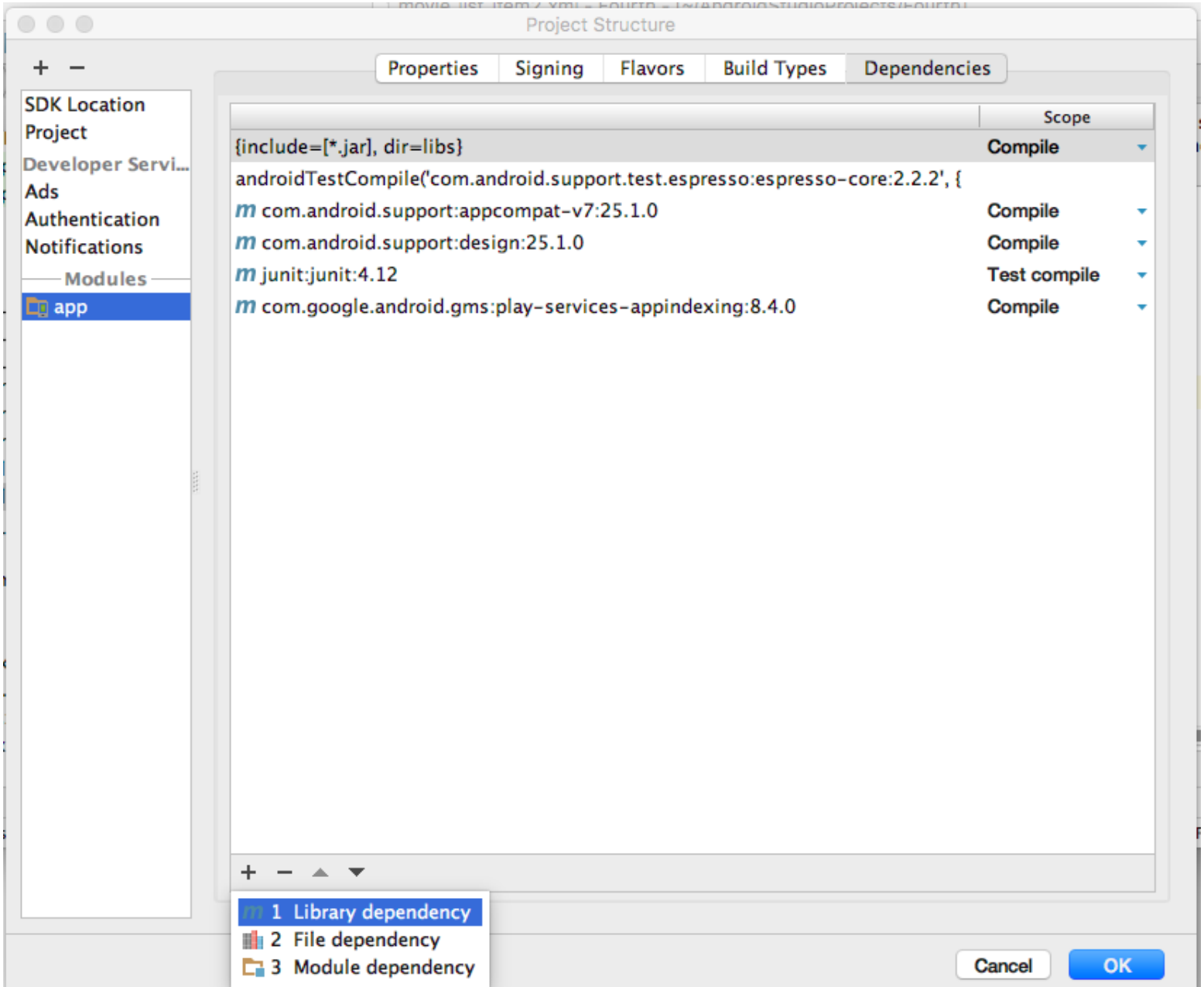
- can add an offset to each item
- can modify the item (highlighted or decorated)
- If you use a CardView for each item, no need for an ItemDecoration
- drawing methods
  - public void onDraw(Canvas c, RecyclerView parent)
    - might be hidden
  - public void onDrawOver(Canvas c, RecyclerView parent)
    - drawn on top of the items
    - if you add decorations, MUST use onDrawOver()
  - public void getItemOffsets(Rect outRect, int itemPosition, RecyclerView parent)

- animate individual items
- deal with three events
  - An item gets added to the data set
  - An item gets removed from the data set
  - An item moves as a result of one or more of the previous two operations
- DefaultItemAnimator
- Instead of using notifyDataSetChanged() in your Adapter,
  - public final void notifyItemInserted(int position)
  - public final void notifyItemRemoved(int position)

- NO OnItemClickListener or OnItemLongClickListener
- RecyclerView.OnItemTouchListener in combination with gesture detection

```
1  setContentView(R.layout.activity_recyclerview_demo);
2  recyclerView = (RecyclerView) findViewById(R.id.recyclerView);
3
4  LinearLayoutManager layoutManager = new LinearLayoutManager(this);
5  layoutManager.setOrientation(LinearLayoutManager.VERTICAL);
6  layoutManager.scrollToPosition(0);
7  recyclerView.setLayoutManager(layoutManager);
8
9  // allows for optimizations if all item views are of the same size:
10 recyclerView.setHasFixedSize(true);
11
12 List<DemoModel> items = RecyclerViewDemoApp.getDemoData();
13
14 adapter = new RecyclerViewDemoAdapter(items);
15 recyclerView.setAdapter(adapter);
16
17 RecyclerView.ItemDecoration itemDecoration =
```

```
18          new DividerItemDecoration(this,
                DividerItemDecoration.VERTICAL_LIST);
19 recyclerView.addItemDecoration(itemDecoration);
20
21 // this is the default;
22 // this call is actually only necessary with custom ItemAnimators
23 recyclerView.setItemAnimator(new DefaultItemAnimator());
24
25 // onClickDetection is done in this Activity's OnItemTouchListener
26 // with the help of a GestureDetector;
27
28 recyclerView.addOnItemTouchListener(this);
29 gesturedetector =
30          new GestureDetectorCompat(this, new
                RecyclerViewDemoOnGestureListener());
```


Project Structure
Properties
Signing
Flavors
Build Types
Dependencies
SDK Location
Project
Developer Servi...
Ads
Authentication
Notifications
Modules
app
Scope
{include=[*.jar], dir=libs}
Compile
androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
com.android.support:appcompat-v7:25.1.0
Compile
com.android.support:design:25.1.0
Compile
junit:junit:4.12
Test compile
com.google.android.gms:play-services-appindexing:8.4.0
Compile
1 Library dependency
2 File dependency
3 Module dependency
Cancel
OK