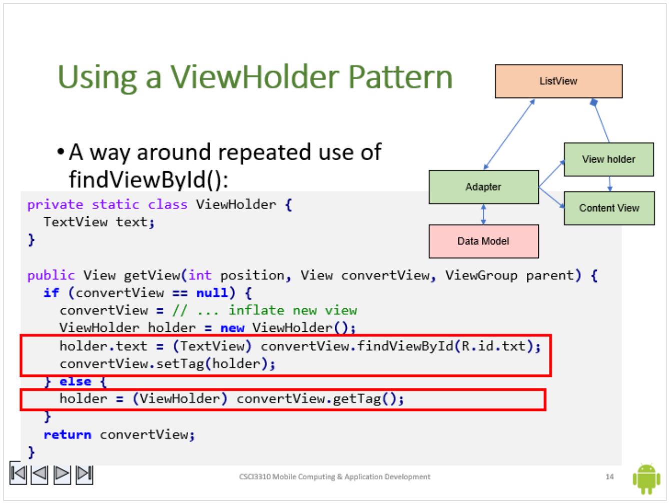
Android Dynamic UI – Adapter, View-Holder & Recycling

CSCI3310 Mobile Computing & Application Development

Overview

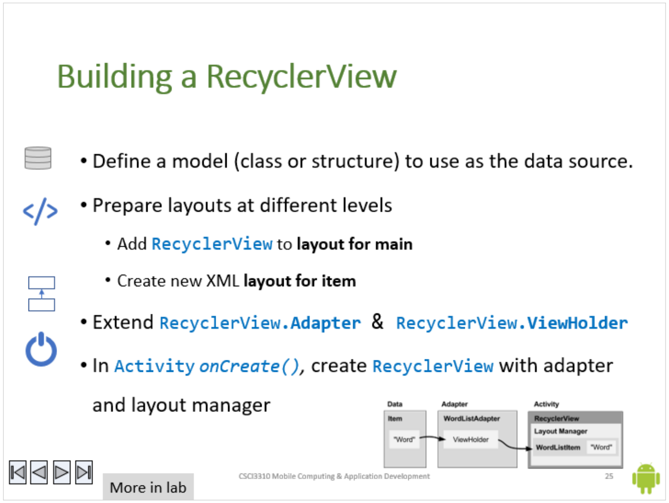
- Adapter-backed Views
  - From ListView to RecyclerView
  - View holder
- Deflating the LayoutInflater



Using a ViewHolder Pattern

- A way around repeated use of findViewById():

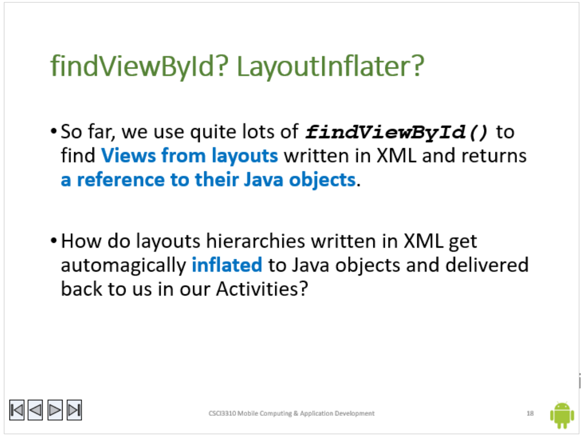
```
private static class ViewHolder {
  TextView text;
}

public View getView(int position, View convertView, ViewGroup parent) {
  if (convertView == null) {
    convertView = // ... inflate new view
    ViewHolder holder = new ViewHolder();
    holder.text = (TextView) convertView.findViewById(R.id.txt);
    convertView.setTag(holder);
  } else {
    holder = (ViewHolder) convertView.getTag();
  }
  return convertView;
}
```



Building a RecyclerView

- Define a model (class or structure) to use as the data source.
- Prepare layouts at different levels
  - Add `RecyclerView` to **layout for main**
  - Create new XML **layout for item**
- Extend `RecyclerView.Adapter` & `RecyclerView.ViewHolder`
- In `Activity onCreate()`, create `RecyclerView` with adapter and layout manager

More in lab

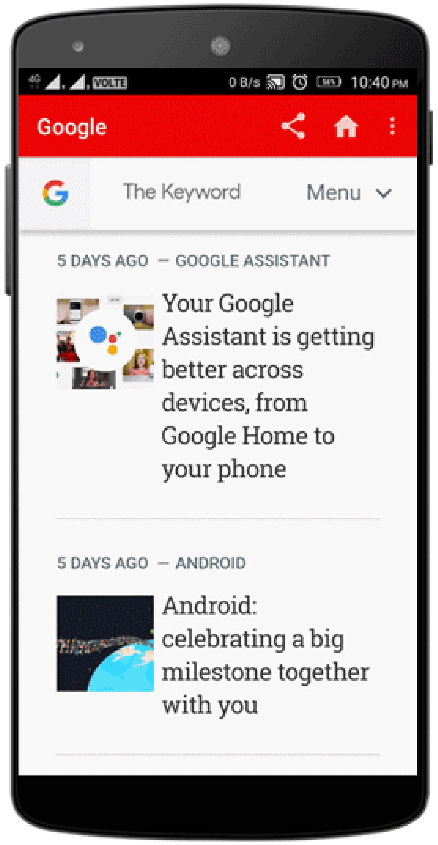


findViewById? LayoutInflater?

- So far, we use quite lots of ***findViewById()*** to find **Views from layouts** written in XML and returns **a reference to their Java objects**.
- How do layouts hierarchies written in XML get automagically **inflated** to Java objects and delivered back to us in our Activities?

Container Views

- Examples of Container Views:
  - WebView
  - SearchView
  - GridView
  - **ListView**
  - ScrollView

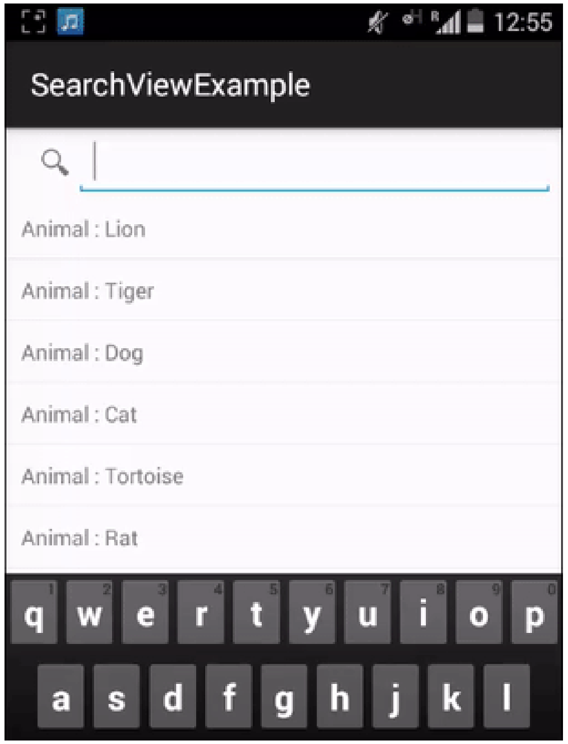


5 DAYS AGO — GOOGLE ASSISTANT

Your Google Assistant is getting better across devices, from Google Home to your phone

5 DAYS AGO — ANDROID

Android: celebrating a big milestone together with you

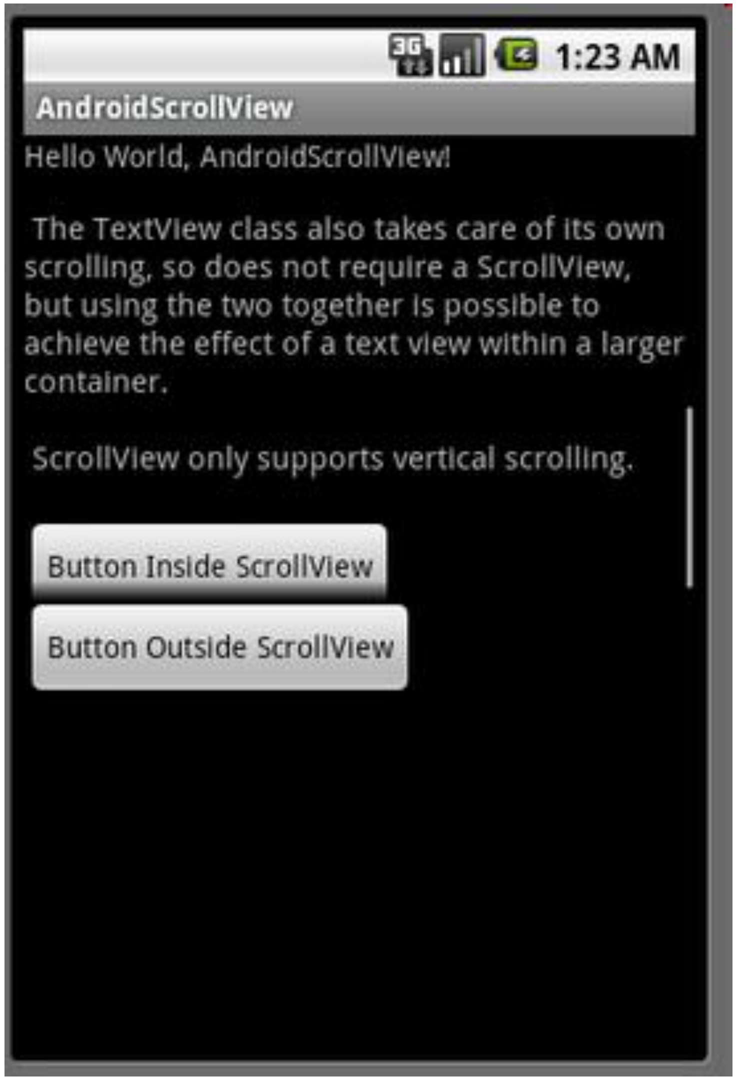


Animal : Lion
Animal : Tiger
Animal : Dog
Animal : Cat
Animal : Tortoise
Animal : Rat

ScrollView

NOT to house a **ListView** or **RecyclerView** within a ScrollView,

- because that defeats the performance optimizations of a ListView or any **Adapter-backed Views**


1:23 AM

AndroidScrollView

Hello World, AndroidScrollView!

The TextView class also takes care of its own scrolling, so does not require a ScrollView, but using the two together is possible to achieve the effect of a text view within a larger container.

ScrollView only supports vertical scrolling.

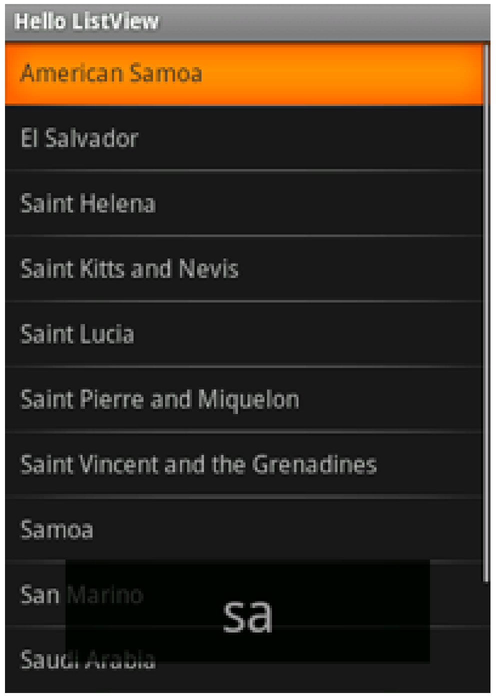
Revisited

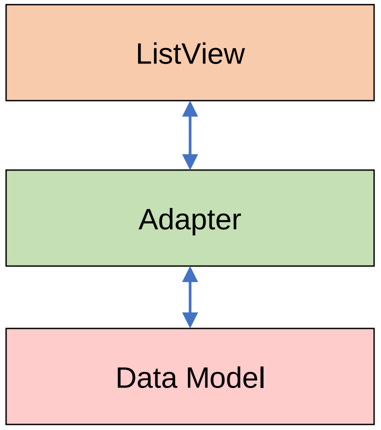
ListView

- ListView is a legacy container view supporting a vertical list (but not horizontal)

- Use AdapterView to bind the view to data source via getView,
  - Retrieving data from source based on the given position in runtime

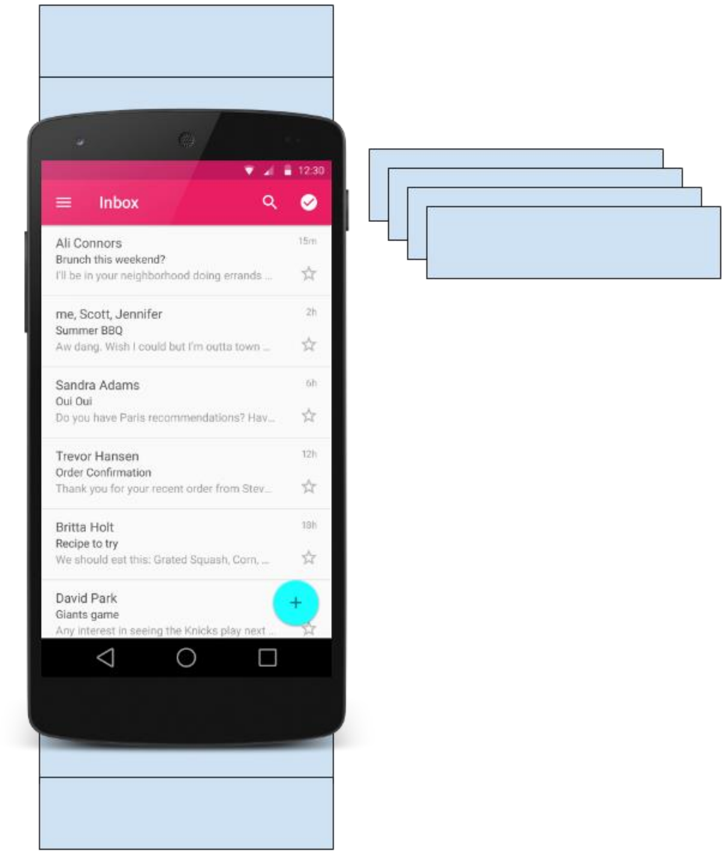
  - ViewHolder pattern are recommended but not mandatory





Revisited

RecyclerView



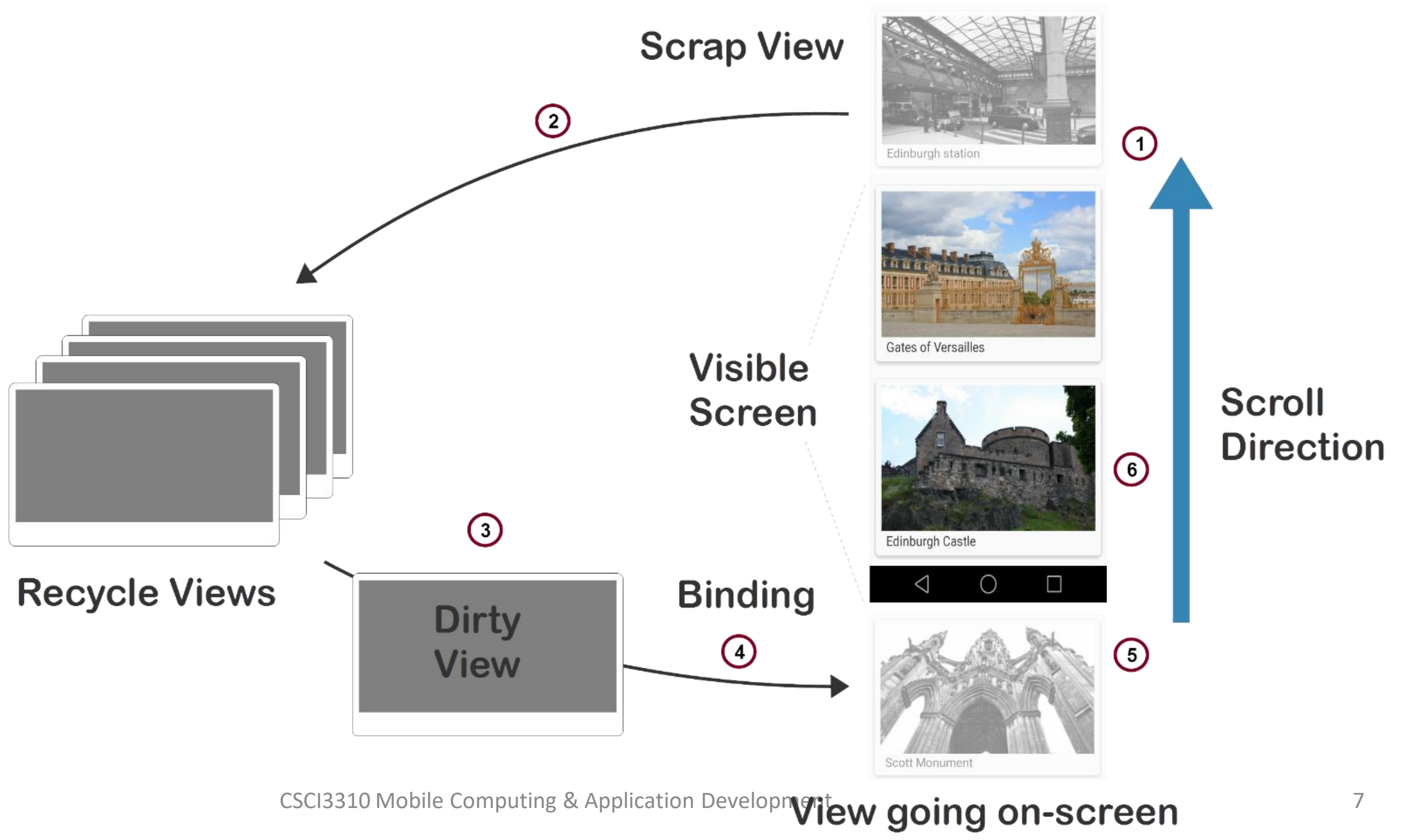
12:30
Inbox
Ali Connors
Brunch this weekend?
I'll be in your neighborhood doing errands ...
15m
me, Scott, Jennifer
Summer BBQ
Aw dang. Wish I could but I'm outta town ...
2h
Sandra Adams
Oui Oui
Do you have Paris recommendations? Hav...
6h
Trevor Hansen
Order Confirmation
Thank you for your recent order from Stev...
12h
Britta Holt
Recipe to try
We should eat this: Grated Squash, Corn, ...
18h
David Park
Giants game
Any interest in seeing the Knicks play next ...

- Scrollable container for large data sets
- Efficient
  - Uses and reuses limited number of View elements
  - Updates changing data fast

Revisited

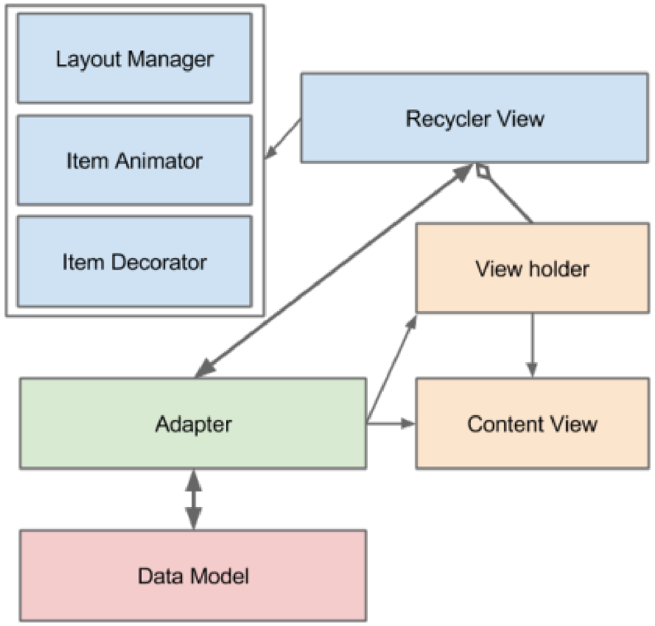
Dirty View for Recycling

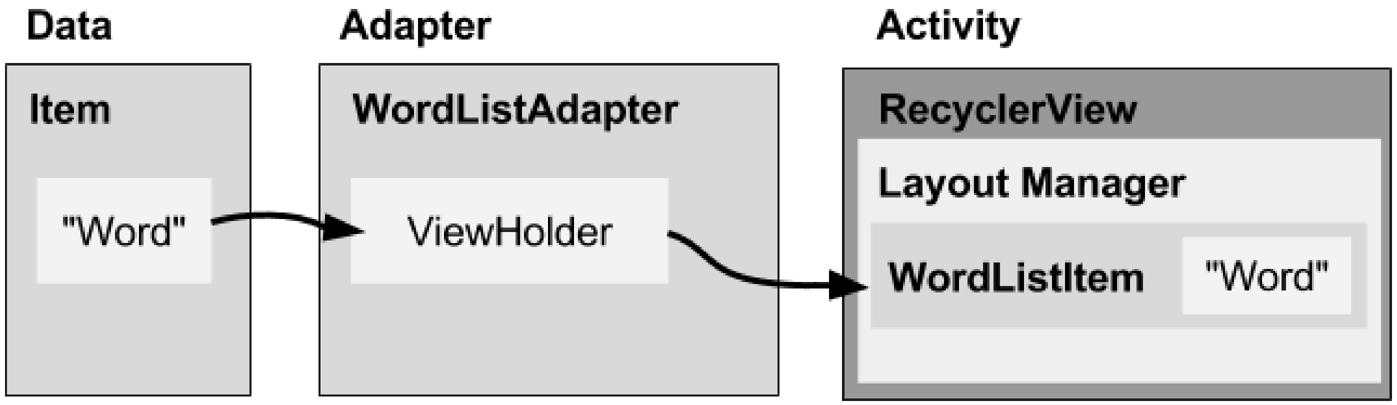
- When a view is scrolled out of screen, it can be marked **dirty** – ready for being recycled.



RecyclerView components

- **RecyclerView** scrolling list for list items
- **Adapter** connects data to the RecyclerView
- **ViewHolder** has view information for displaying one item

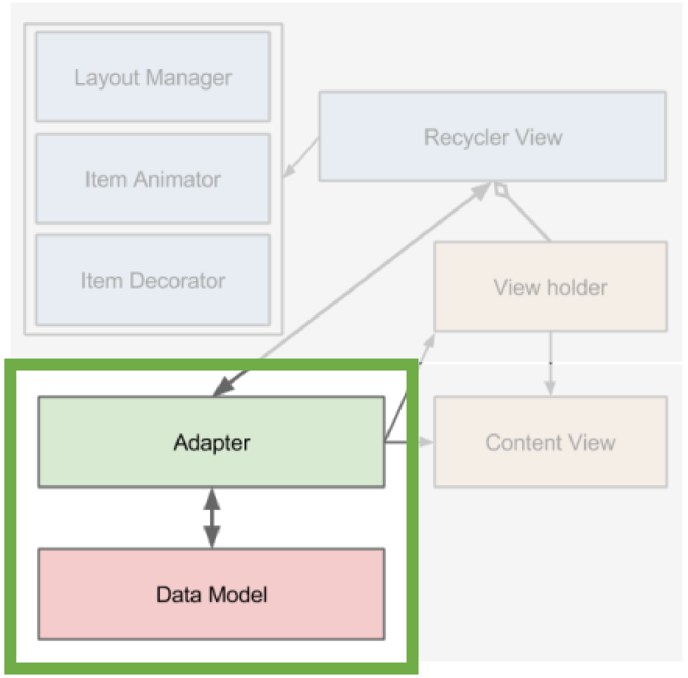


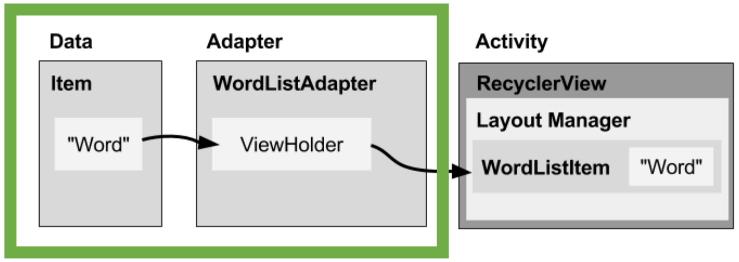


Revisited

RecyclerView.Adapter

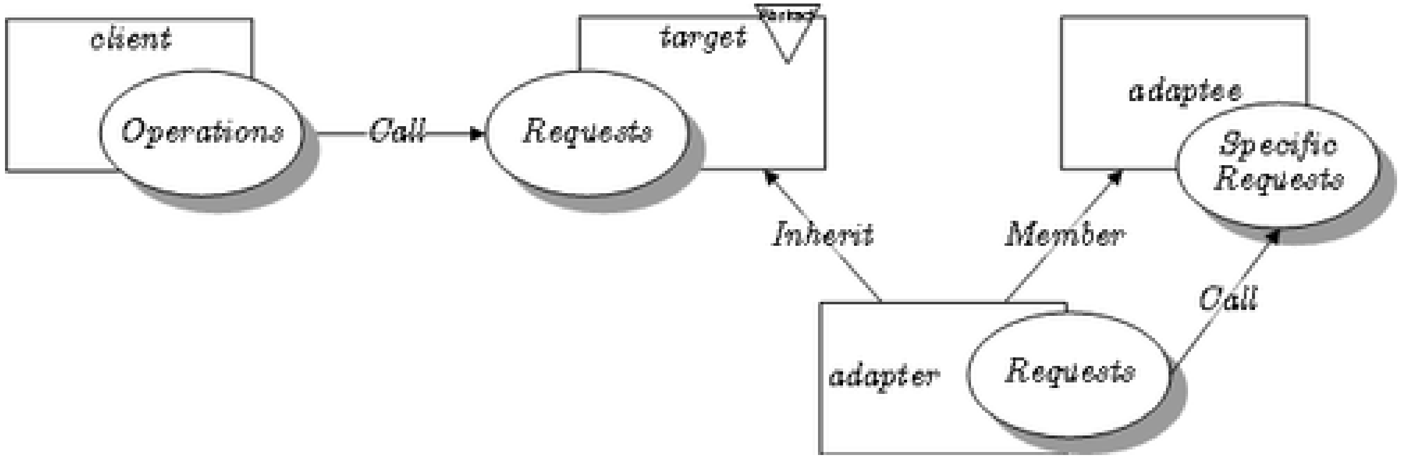
- Helps incompatible interfaces work together
  E.g.: Takes data from database and prepares strings to put into a `View`
- **Intermediary** between **data** and **`View`**
- Manages creating, updating, adding, deleting `View` items as underlying **data changes**





Gang-of-Four Adaptor Pattern?

- "*Convert the **interface** of a **class** into another interface clients expect, also known as wrapper*" – Gang of Four

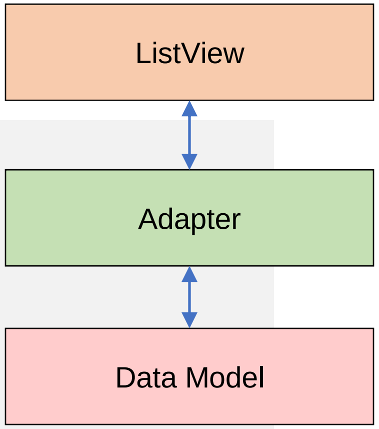


- But in Android, Adapter refers to an **encapsulation** process between the *View* and *Model, could* be regarded as part of the MVP pattern

ViewHolder?

- Let’s re-visit ListView again, one can instantiate new View object via Adapter’s getView:

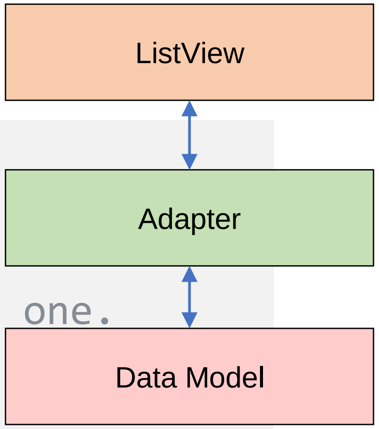
```
// override the getView for an adapter to be used in ListVIew
@Override public View getView(int pos, View convertView,
      ViewGroup container) {
  /* We create a new convertView no matter what,
     by inflating an xml layout */

    convertView = getLayoutInflater().inflate(
                            R.layout.list_item, container, false);

  (TextView) myText = new TextView(this);
  myText.setText(getItem(pos));
  // ... some extra lines to add views into the layout
  return convertView;
}
```



Without a ViewHolder

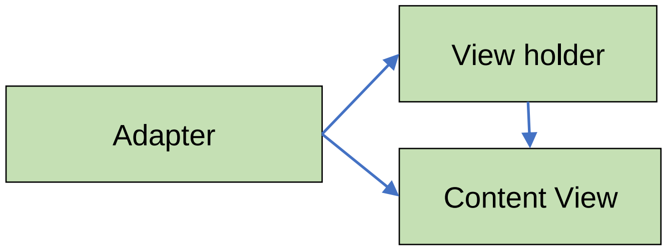
- This is better but for every new item, an extra lookup via findViewById :

```
// override the getView for an adapter to be used in ListVIew
@Override public View getView(int pos, View convertView,
      ViewGroup container) {
  /* Only if there's no view at this position, we create a new one.
     by inflating an xml layout */
  if (convertView == null) {
    convertView = getLayoutInflater().inflate(
                           R.layout.list_item, container, false);
  }

  ((TextView) convertView.findViewById(R.id.txt)).setText(getItem(pos));

  return convertView;
}
```

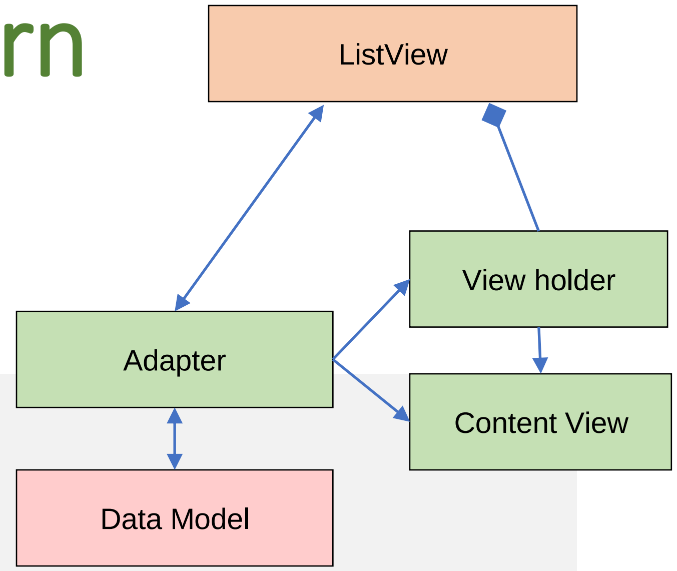


What is a ViewHolder Pattern?

- A **View Holder** pattern is for holding references to the sub-views after you "find" them.
- The **View Holder** stores each of the component views inside the **tag** field of the Layout, so you can immediately access them without the need to look them up repeatedly.
- You store it as a **tag** in the row's view after inflating it.



Using a ViewHolder Pattern

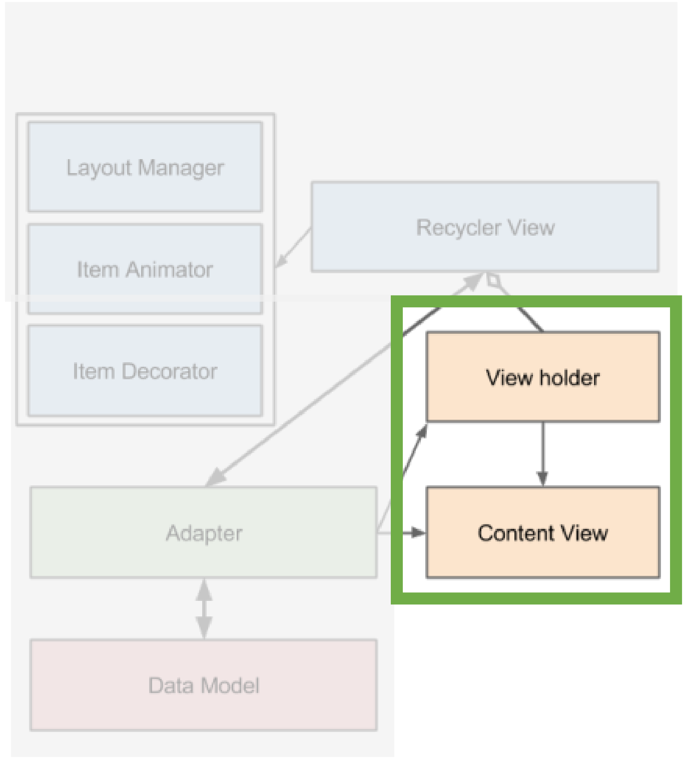


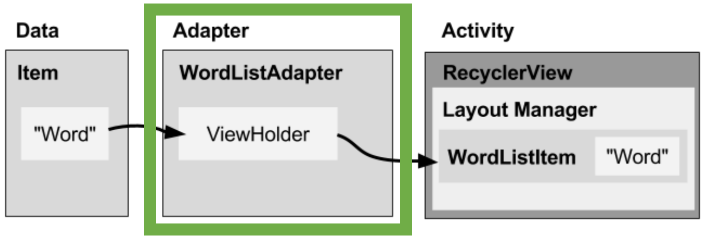
- A way around repeated use of findViewById():

```
private static class ViewHolder {
  TextView text;
}

public View getView(int position, View convertView, ViewGroup parent) {
  if (convertView == null) {
    convertView = // ... inflate new view
    ViewHolder holder = new ViewHolder();
    holder.text = (TextView) convertView.findViewById(R.id.txt);
    convertView.setTag(holder);
  } else {
    holder = (ViewHolder) convertView.getTag();
  }
  return convertView;
}
```

RecyclerView.ViewHolder

- Used by the adapter to **prepare one `View`** with data for one list item
- Layout specified in an XML resource file
- Can have clickable elements
- Is placed by the layout manager





Layout Manager

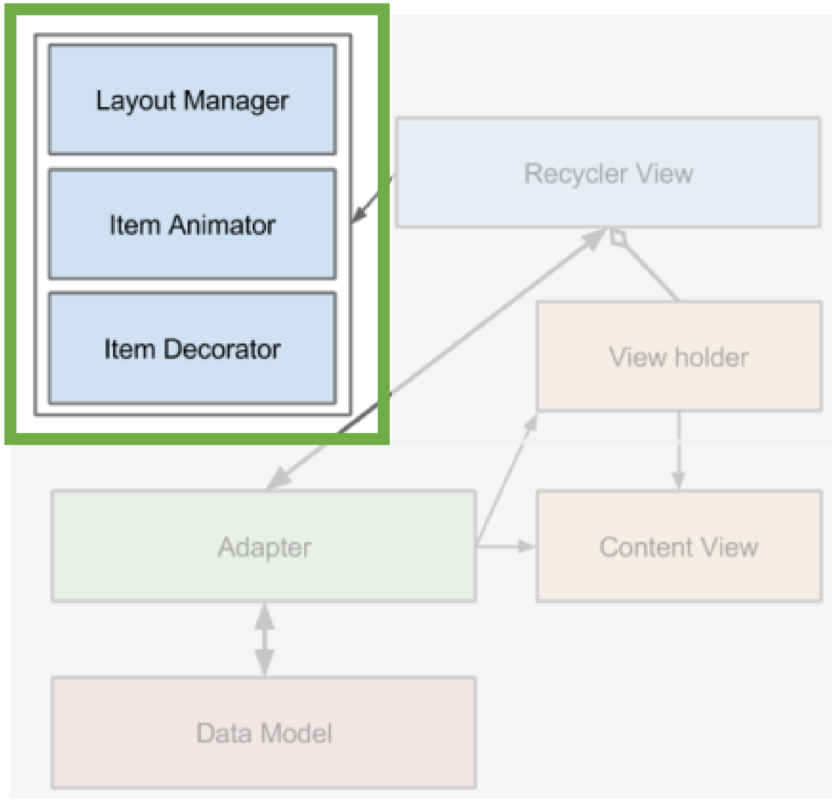
Each ViewGroup has a layout manager

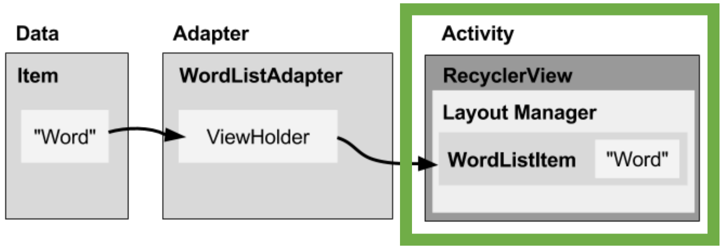
- Use to position View items inside a RecyclerView

Reuses View items that are no longer visible to the user

Built-in layout managers
- LinearLayoutManager
- GridLayoutManager
- StaggeredGridLayoutManager

Extend RecyclerView.LayoutManager

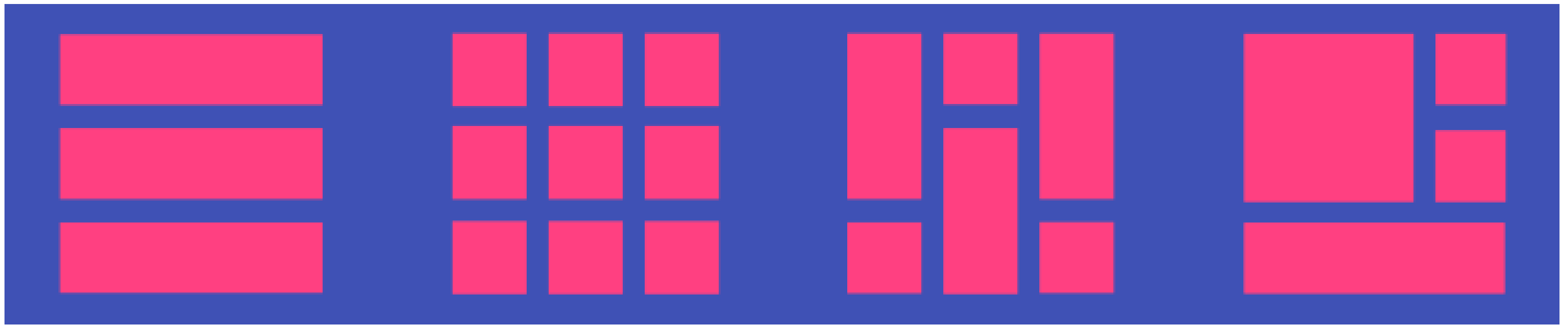




More in tutorial

Layout Manager

- With ListView, the onlyoption is vertical lists.
- LayoutManager for every layout **horizontal**, **vertical**, **grid** or even **mixed**
- May build up customized layoutManger through inherenting the RecyclerView.LayoutManager class



Vertical List, Grid View, Staggered View, Mixed View (from left to right)

More in tutorial

findViewById? LayoutInflater?

- So far, we use quite lots of ***`findViewById()`*** to find **Views from layouts** written in XML and returns **a reference to their Java objects**.

- How do layouts hierarchies written in XML get automagically **inflated** to Java objects and delivered back to us in our Activities?

LayoutInflater

- we need to look at Android as a **pure Java framework**. For anything written in **XML**, the framework spends extra effort **converting** that **into Java**.

- Layouts are the best example of this and the class responsible for "**inflating**" them is *`LayoutInflater`*

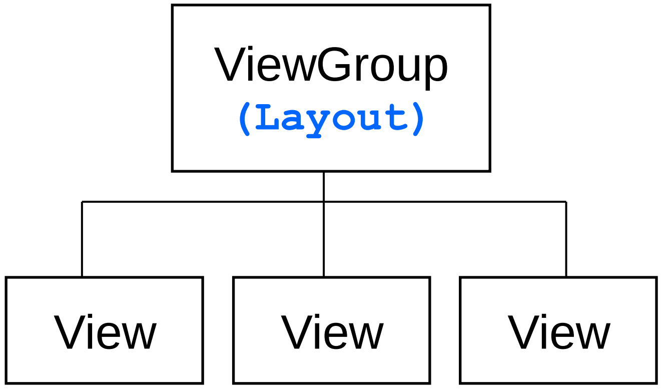
Deflating the LayoutInflater

The entry point to *LayoutInflater* is the ***`inflate()`*** method and we use it in a lot of places:

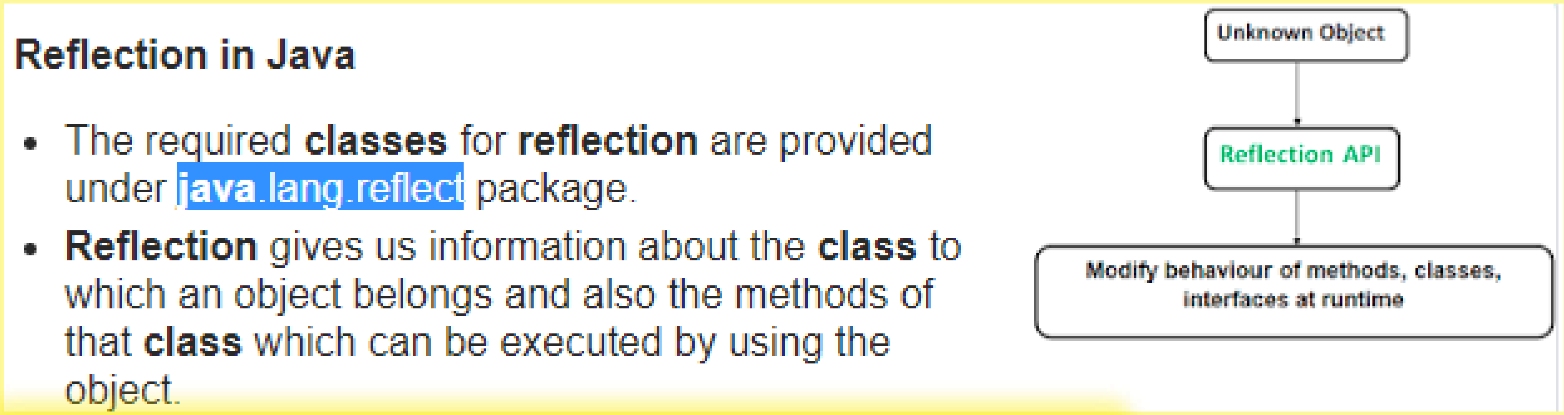
- **Adapter backed Views**, for inflating the layout of each item in a RecyclerView, Spinner, etc
- **Activities**: This is not obvious, but every call to ***`setContentView()`*** internally gets routed to *LayoutInflater*
- **Fragments**, for inflating their layouts (to be discussed in other chapter)

Working of LayoutInflater

1. Parsing XML using XmlPullParser
   - parsing the View name (e.g., TextView) and its attributes from the XML layout file
2. Constructing attributes using AttributeSet
3. Instantiating View object using **java.lang.reflect**

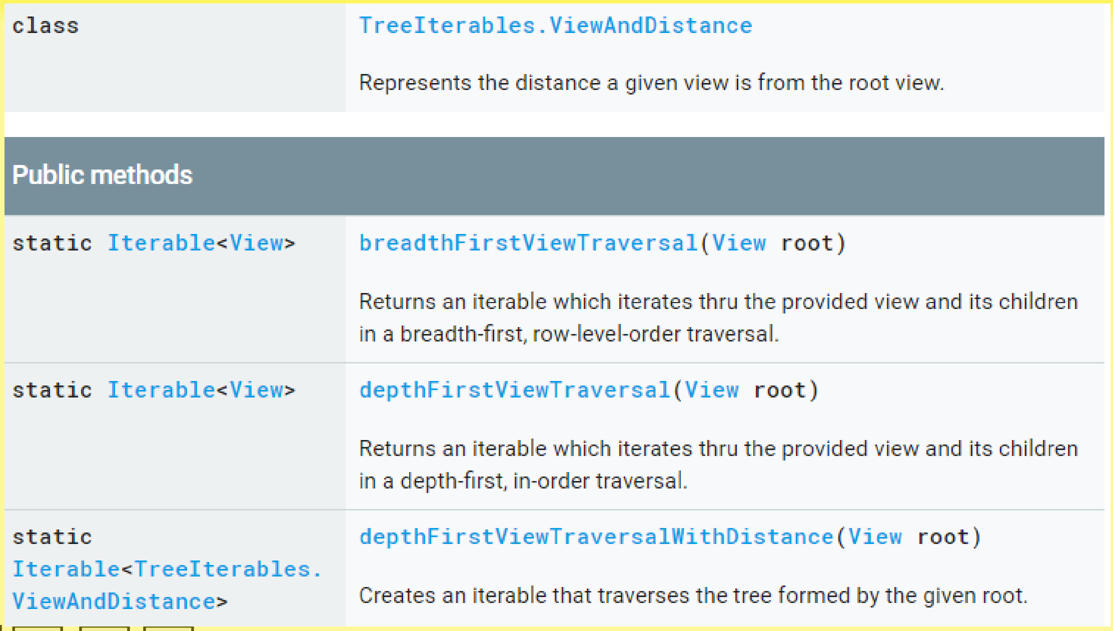


Working of LayoutInflater



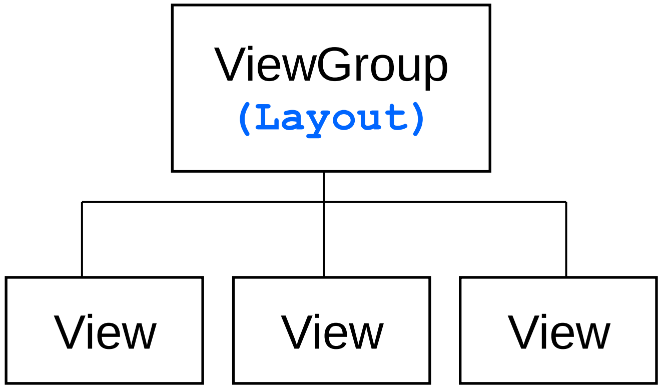
Reflection in Java

- The required **classes** for **reflection** are provided under java.lang.reflect package.
- **Reflection** gives us information about the **class** to which an object belongs and also the methods of that **class** which can be executed by using the object.



class	TreeIterables.ViewAndDistance Represents the distance a given view is from the root view.

Public methods	
static Iterable<View>	breadthFirstViewTraversal(View root) Returns an iterable which iterates thru the provided view and its children in a breadth-first, row-level-order traversal.
static Iterable<View>	depthFirstViewTraversal(View root) Returns an iterable which iterates thru the provided view and its children in a depth-first, in-order traversal.
static Iterable<TreeIterables.ViewAndDistance>	depthFirstViewTraversalWithDistance(View root) Creates an iterable that traverses the tree formed by the given root.



Working of findViewById

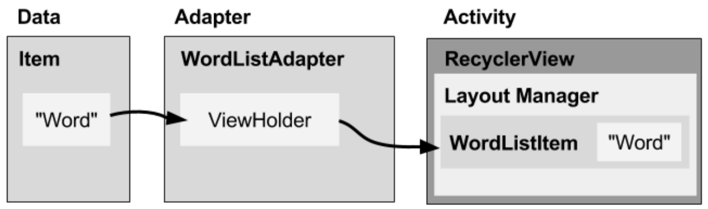
- With each inflation, LayoutInflater links the instantiated View to its parent ViewGroup and its children Views, essentially creating a tree of our View hierarchy.
- The View then simply **traverses these links every time** findViewById() gets called.

Implications

- Avoid frequent View finding, say try to cache views
- The use of reflection by LayoutInflater makes it relatively expensive, which is one of the reasons we should
  - avoid complex View hierarchies.
  - Not doing so directly affects the startup time for Activities, Fragments or any adapter backed ViewGroup.

Building a RecyclerView

- Define a model (class or structure) to use as the data source.
- Prepare layouts at different levels
  - Add `RecyclerView` to **layout for main**
  - Create new XML **layout for item**
- Extend `RecyclerView.`**`Adapter`** & `RecyclerView.`**`ViewHolder`**
- In `Activity` *`onCreate()`*, create `RecyclerView` with adapter and layout manager



More in lab

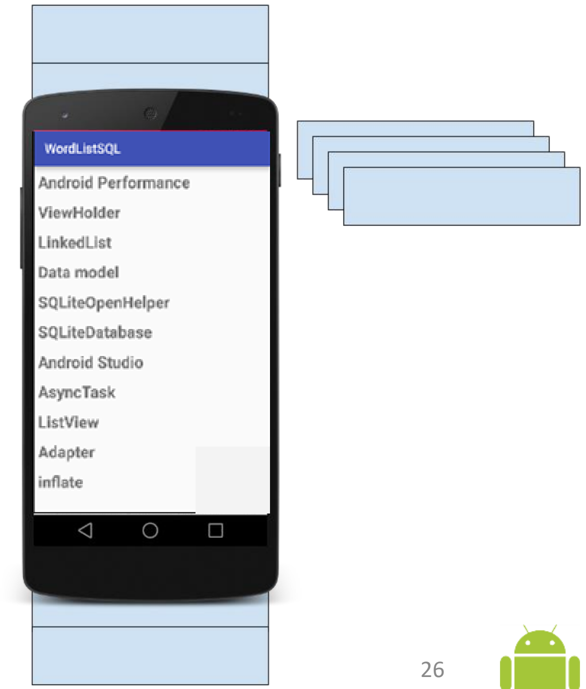
XML Layouts

Activity layout - Add RecyclerView to XML Layout

```
<androidx.recyclerview.widget.RecyclerView
  android:id="@+id/recyclerview"
  android:layout_width="match_parent"
  android:layout_height="match_parent”/>
```

Item layout - Create layout for 1 list item

```
<LinearLayout …>
  <TextView
   android:id="@+id/word"
   style="@style/word_title" />
</LinearLayout>
```



More in lab

Implement the adapter

Adapter: Create

- **LayoutInflater** instantiates a layout XML file into its corresponding View objects.

```
public class WordListAdapter
   extends RecyclerView.Adapter<WordListAdapter.WordViewHolder>
{
      public WordListAdapter(Context context,
                   LinkedList<String> wordList) {
          Inflater = LayoutInflater.from(context);
          this.mWordList = wordList;
   }
}
```

More in lab

Adapter has 3 required methods

`getItemCount()`

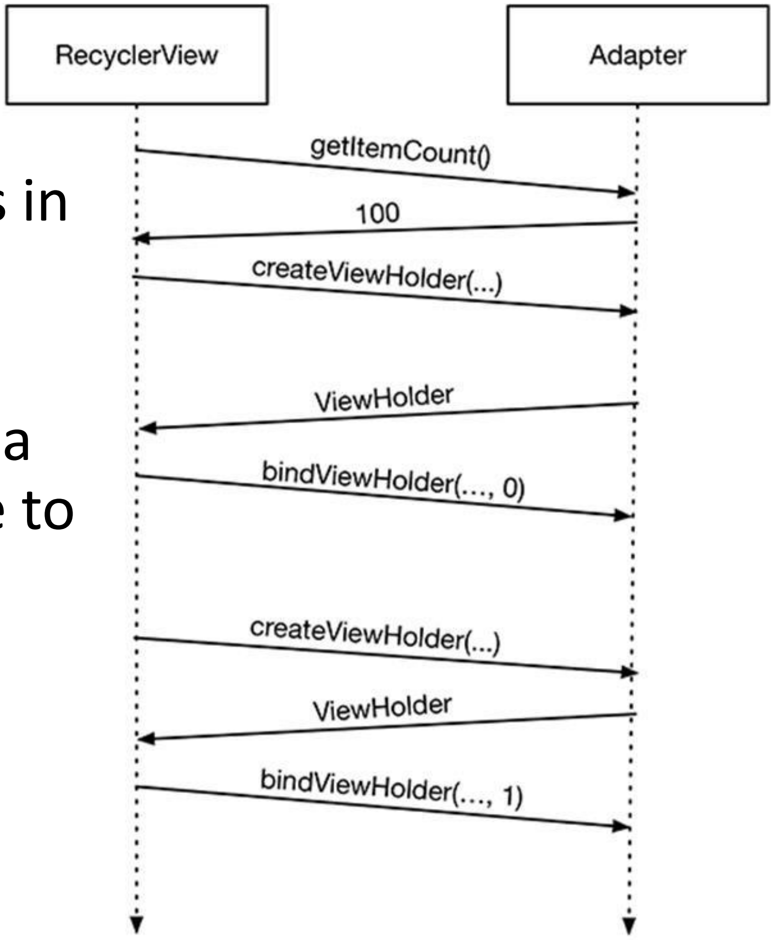
- Returns the total number of items in the data set held by the adapter.

`onCreateViewHolder()`

- Called when RecyclerView needs a new ViewHolder of the given type to represent an item

`onBindViewHolder()`

- Called by RecyclerView to display the data at the specified position.



More in lab

Adapter: onCreateViewHolder()

- The View object (of individual item) is instantiated from the item layout XML file using the **LayoutInflater** obtained in Adapter creation.
- We get **new unused view holders initially** and have to fill them with data you want to display.
- We scroll to get view holders that were used for rows that went **off screen** and **replace old data** with new one.

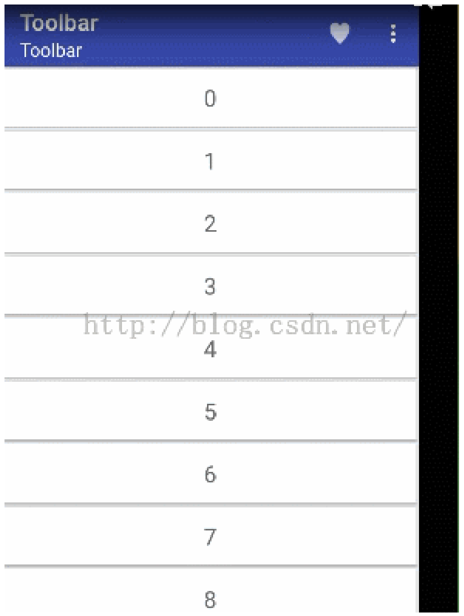
```
@Override
public WordViewHolder onCreateViewHolder(ViewGroup
parent, int viewType) {
        // Create view from layout
        View mItemView = mInflater.inflate(
             R.layout.wordlist_item, parent, false);
        return new WordViewHolder(mItemView, this);
}
```

More in lab

Adapter: onBindViewHolder()

- Called by RecyclerView to display the data at the specified **position**.

```
@Override
public void onBindViewHolder(WordViewHolder holder,
int position) {
        // Retrieve the data for that position
        String mCurrent = mWordList.get(position);
        // Add the data to the view
        holder.wordItemView.setText(mCurrent);
}
```



More in lab

Adapter: getItemCount()

- Returns the **total number** of items in the data set held by the adapter.
- Make sure the returned value is **updated after any add/delete**

```
  @Override
  public int getItemCount() {
     // Return the number of data items to 
display
     return mWordList.size();
  }
```

More in lab

Adapter: ViewHolder Class

Create the view holder in adapter class

```
class WordViewHolder extends RecyclerView.ViewHolder { //.. }
```

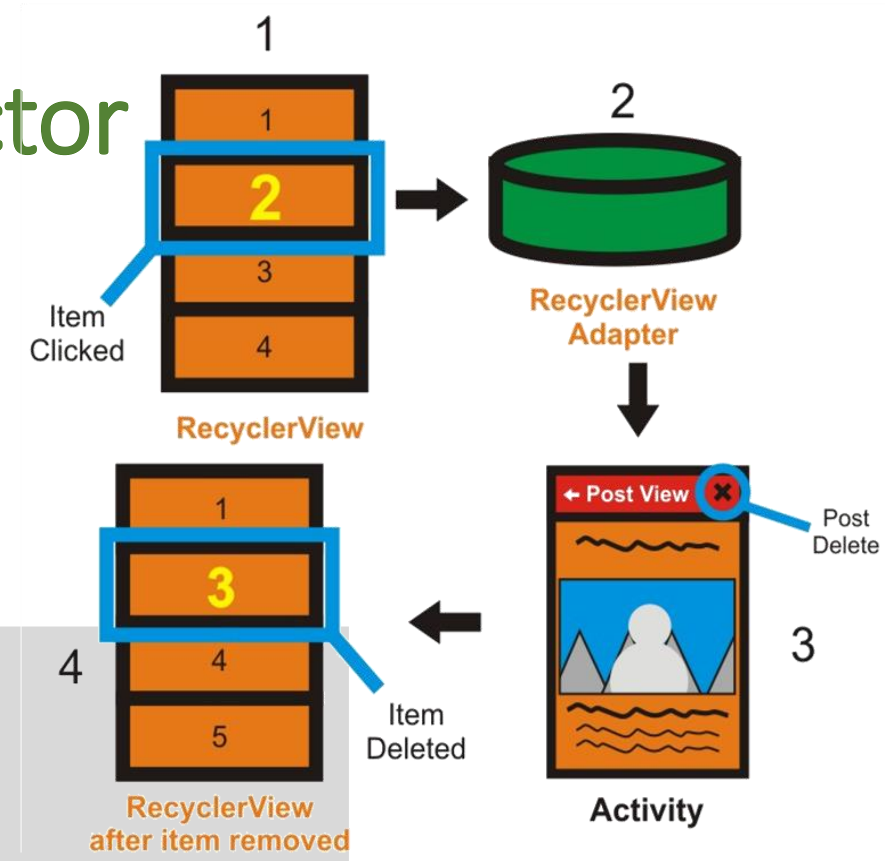
If you want to handle mouse clicks:

```
class WordViewHolder extends RecyclerView.ViewHolder
                     implements View.OnClickListener { //.. }
```

More in lab

View holder constructor

- Get the layout (view) of item
- Associate with current adapter
- Add event handler (Optionally)



```
public WordViewHolder(View itemView,
WordListAdapter adapter) {
   super(itemView);
   // Get the layout
   wordItemView = itemView.findViewById(R.id.word);
   // Associate with this adapter
   this.mAdapter = adapter;
   // Add click listener, if desired
   itemView.setOnClickListener(this);
}
// Implement onClick() as required
```

More in lab

In Activity onCreate()

- Create the RecyclerView in onCreate() of Activity

```
mRecyclerView = findViewById(R.id.recyclerview);
mAdapter = new WordListAdapter(this, mWordList);
mRecyclerView.setAdapter(mAdapter);
mRecyclerView.setLayoutManager(new
LinearLayoutManager(this));
```

More in lab

Reference

1. Android RecyclerView
   http://developer.android.com/reference/android/support/v7/widget/RecyclerView.html

2. Inflate View from XML
   https://stackoverflow.com/questions/4576330/what-does-it-mean-to-inflate-a-view-from-an-xml-file