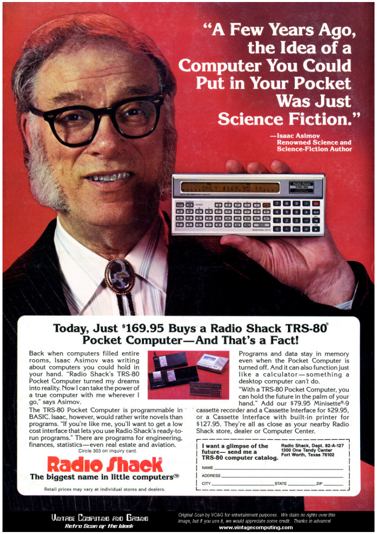
Mobile Application Development (COMP2008)

Lecture 3: UI Structure

Updated: 19th September, 2018

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)


"A Few Years Ago, the Idea of a Computer You Could Put in Your Pocket Was Just Science Fiction."
—Isaac Asimov Renowned Science and Science-Fiction Author
....ONLY $169.95 !!....
Radio Shack
The biggest name in little computers®
Vintage Computing and Gaming
Retro Scan of the Week

Today, Just $169.95 Buys a Radio Shack TRS-80® Pocket Computer—And That's a Fact!

Back when computers filled entire rooms, Isaac Asimov was writing about computers you could hold in your hand. "Radio Shack's TRS-80 Pocket Computer turned my dreams into reality. Now I can take the power of a true computer with me wherever I go," says Asimov.

The TRS-80 Pocket Computer is programmable in BASIC. Isaac, however, would rather write novels than programs. "If you're like me, you'll want to get a low cost interface that lets you use Radio Shack's ready-to-run programs." There are programs for engineering, finances, statistics—even real estate and aviation.

Programs and data stay in memory even when the Pocket Computer is turned off. And it can also function just like a calculator—something a desktop computer *can't* do.

"With a TRS-80 Pocket Computer, you can hold the future in the palm of your hand." Add our $79.95 Minisette®-9 cassette recorder and a Cassette Interface for $29.95, or a Cassette Interface with built-in printer for $127.95. They're all as close as your nearby Radio Shack store, dealer or Computer Center.

Circle 303 on inquiry card.

I want a glimpse of the future— send me a TRS-80 computer catalog.

Radio Shack, Dept. 82-A-127
1300 One Tandy Center
Fort Worth, Texas 76102

NAME ______
ADDRESS ______
CITY ______ STATE ______ ZIP ______

Retail prices may vary at individual stores and dealers.

Original Scan by VC&G for entertainment purposes. We claim no rights over this image, but if you use it, we would appreciate some credit. Thanks in advance!
www.vintagecomputing.com

(http://www.vintagecomputing.com/index.php/archives/806/retro-scan-of-the-week-asimovs-pocket-computer.)

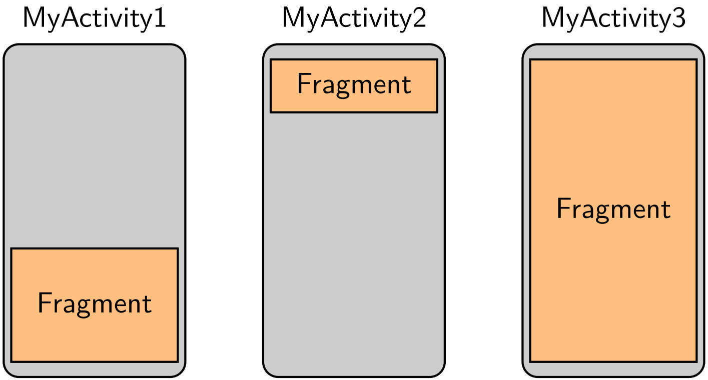
Outline

Fragments

- An activity is a controller (the brains) for a given screen.
- But it can delegate.
- A *fragment* is a “sub”-controller for *part* of the UI.
  - The activity decides which fragments (if any) it will have.
  - The activity’s layout reserves some space for the fragment’s mini-UI.
  - (Sometimes an activity may have a fragment that occupies the entire screen.)
- Improves flexibility and code re-use, because...
  1. You can re-use a fragment across different activities.
  2. You can switch between fragments within a single activity.
  3. Less need to rip everything down and rebuild it whenever the UI changes.

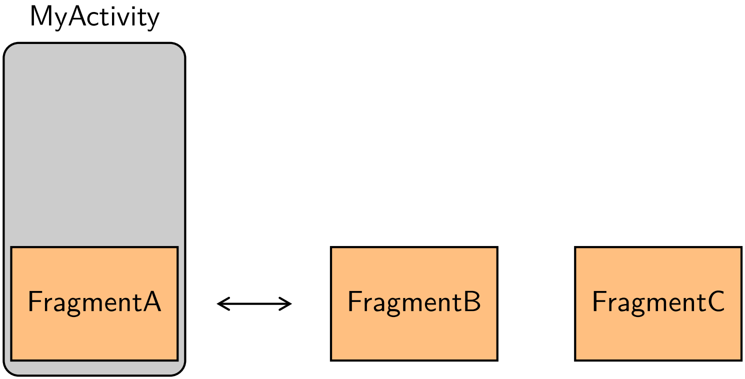
[1] https://developer.android.com/guide/components/fragments

Re-Using Fragments



- You can re-use the same fragment in multiple different activities.
- Each activity gets to say where to put it.

Swapping Fragments



- An activity can put different fragments in the same space.
- One can be replaced by another, when desired.
- Adding/replacing fragments is called a *fragment transaction*.

UI Layout Files

- Recall `ViewGroups` – UI elements that contain other UI elements.
  - We've discussed `LinearLayout` and `ConstraintLayout`.
- `FrameLayout` is another one.
  - Defines where to put a fragment.
  - In the XML, it's actually empty. It gets "filled up" at runtime when the fragment is attached.
  - Needs an ID (so our code can find it at runtime):

```
<FrameLayout
    android:id="@+id/f_container"
    ... />
```

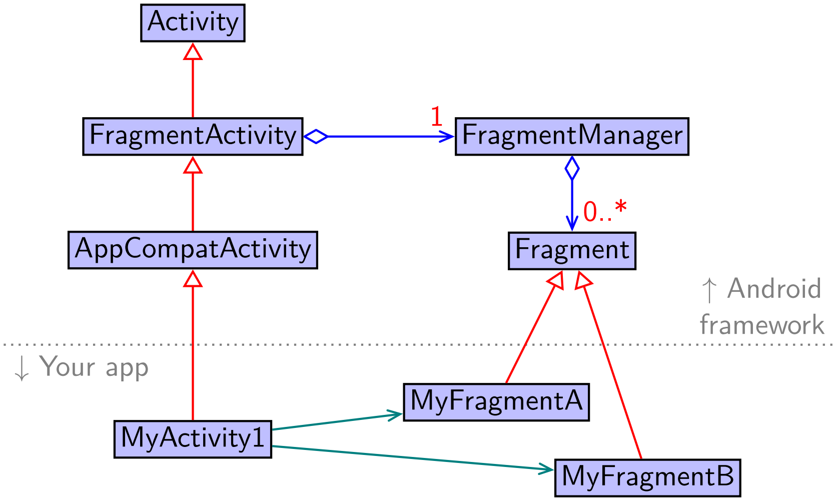
- Each fragment gets its own separate XML layout file.
  - Creating the fragment's UI is just creating an activity's UI.

Code and XML

With fragments, we end up with more .java and .xml files:

- Java code, in app/src/main/**java/**:
  - `Activity1.java`
  - `Activity2.java`
  - `FragmentA.java`
  - `FragmentB.java`
  - ...
- XML layout files, in app/src/main/**res/layout/**:
  - `activity_1.xml`
  - `activity_2.xml`
  - `fragment_A.xml`
  - `fragment_B.xml`
  - ...
- Note: NOT always a one-to-one mapping.
  - You can also re-use layout files in different activities/fragments.

Activities and Fragments



Adding Fragments to Activities

```
public class MyActivity extends AppCompatActivity
{
    @Override protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        setContentView(R.layout.activity_ui); // As before

        FragmentManager fm = getSupportFragmentManager();
        MyFragmentA frag = (MyFragmentA) fm.findFragmentById(
            R.id.f_container);
        if(frag == null) // It might already be there!
        {
            frag = new MyFragmentA();
            fm.beginTransaction()
                .add(R.id.f_container, frag).commit();
        }
    }
}
```

Finding Attached Fragments

```
FragmentManager fm = getSupportFragmentManager();
MyFragmentA frag = (MyFragmentA) fm.findFragmentById(
    R.id.f_container);
```

- The FragmentManager keeps track of the fragments.
- Here we ask it to find a fragment, based on where it's attached.
- Recall that "`f_container`" was the ID of the FrameLayout:

```
<FrameLayout
    android:id="@+id/f_container"
    ... />
```

- `findFragmentById()` returns null if there's no fragment there.
    - *Then* we create one...

Creating and Attaching Fragments

```
frag = new MyFragmentA();
fm.beginTransaction()
    .add(R.id.f_container, frag)
    .commit();
```

- First, *you* create the fragment object (unlike for activities).
- To manipulate fragments, you need a *fragment transaction*:
  - add() queues up an operation to attach a new fragment.
    - You tell it *where* to add, and *what* to add.
  - You could also detach, show, hide, replace.
  - `commit()` actually makes it happen.
- Why the complication? Flexibility. Perhaps you want to:
  - Simultaneously replace 3 fragments with 3 others.
  - Do this through animations.
  - Sometime later, reverse the process.

FragmentManager and the Fragment Lifecyle

- If the activity is destroyed and re-created, the FragmentManager can re-create the fragments too.
- Fragments have overridable methods, some very similar to activities:
  - Some very similar to activities: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onDestroyView()`.
  - Some extra: `onCreateView()`, `onActivityCreated()`, `onAttach()`.
- FragmentManager keeps fragments in-sync with the activity's lifecycle.
  - The OS only sees activities. It can pause them, stop them, etc.
  - Fragments are internal.

Defining Fragments

```java
public class MyFragmentA extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater,
                             ViewGroup ui, Bundle bundle)
    {
        View view = inflater.inflate(
            R.layout.fragment_a, ui, false);

        Button myButton = (Button) view.findViewById(
            R.id.my_button);
        // Set up event handlers

        return view;
    }
    ...
}
```

Defining Fragments: UI Inflation

- What is `LayoutInflater`? Basically:
  - It takes a layout reference (e.g. `R.layout.xyz`);
  - It instantiates all the `View` objects.
  - i.e. it reads the XML and creates the UI based on it.
  - It returns the root `View` object.
- You've already used it, indirectly.
  - This is what happens behind the scenes in activities:

```
// Uses LayoutInflater internally:
setContentView(R.layout.activity_fragment);
```

- The root `View` object lets you find specific UI elements: `view.findViewById(...)`.
  - FYI, you've already used Activity's own `findViewById()`.
  - That just calls `View`'s one internally.

UIs for Large Amounts of Data

- Real-world apps often have lots of data in a scrolling list.
- First, managing this *is not* like getting/setting `EditText`.
- **But why not?** In another reality, things could be simpler:
  - A "`BigListView`" UI element (not a real thing) could display a scrolling list.
  - It could have assorted `get`, `set`, add and `remove` methods.
  - It would display strings (or Objects, by calling `toString()`).
- Nice and easy, *except*. . .
  1. UI lists are not just rows of strings.
     - Each row may contain several text fields, buttons, etc.
  2. You don't need more UI objects than you can actually display.
     - You may have a million data elements.
     - But, at one time, maybe only 12 of them fit on the screen.
     - Having a million UI list rows (and all the UI elements inside them) just wastes memory.

Actual UI Lists

Android UI lists involve a number of parts, most notably:

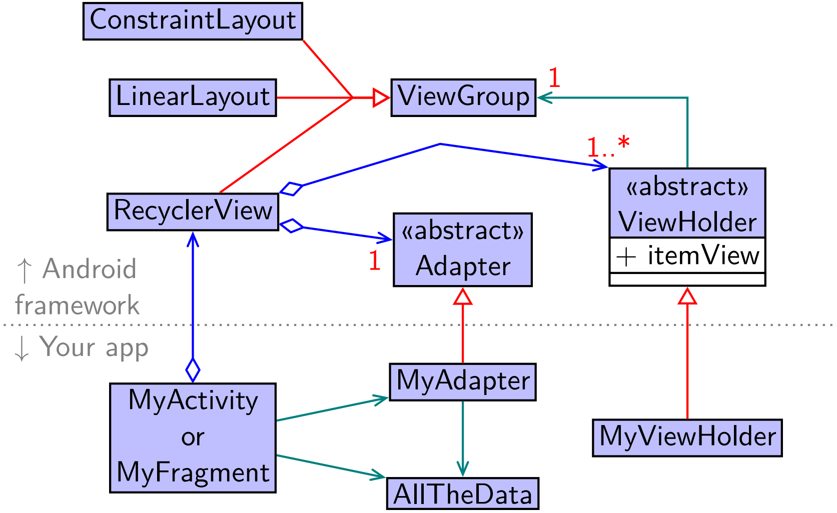
RecyclerView – the UI element that contains the list.
- Only has enough rows to fit on the screen.

ViewHolder – a tiny controller object.
- One per *visible* list row.
- ViewHolders must update their row with new data when the list scrolls.
- ViewHolders also respond to events from (say) any buttons in their row.

Adapter – creates ViewHolders and assigns data to them as needed.
- There is one adapter (for a given RecyclerView).
- It gets its data from your own model classes.

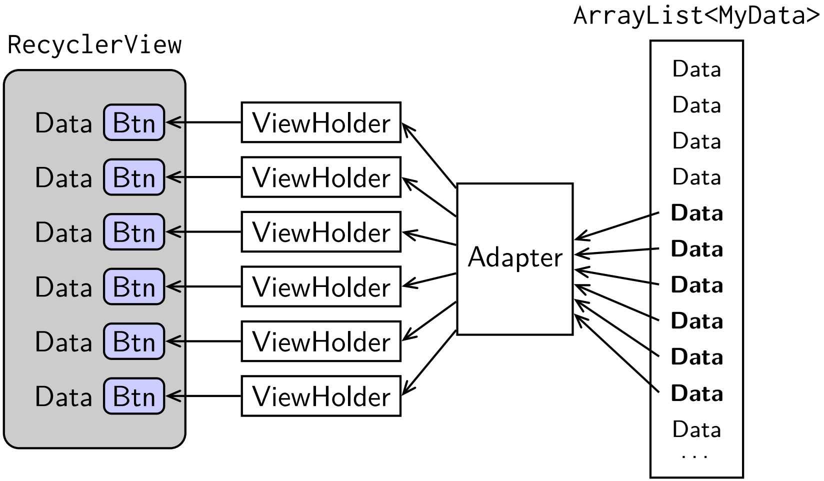
- `ViewHolder` and `Adapter` are *abstract* – you must create subclasses.

RecyclerView Class Relationships



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

Adapters and ViewHolders: How They Interact



- Note: "Data Btn" is just an example of what could be in each list row (a `TextView` and a `Button`).

Setting Up RecyclerView

```
// Obtain the RecyclerView UI element
RecyclerView rv =
    (RecyclerView) view.findViewById(R.id.my_list);

// Specify how it should be laid out
rv.setLayoutManager(new LinearLayoutManager(getActivity()));

// Have your data ready
List<MyData> data = ...;

// Create your adapter (see next slides)
MyAdapter adapter = new MyAdapter(data);

// Hook it up
rv.setAdapter(adapter);
```

- Goes inside your fragment's `onCreateView()` method.
- (Can also go inside an activity, of course, with minor tweaks.)

Adapters

Adapter subclasses must override these methods:

- **`getItemCount()`** – must return the total number of data elements.
- **`onCreateViewHolder(...)`**
  - RecyclerView calls this when it needs a new ViewHolder.
  - The adapter must create and return one.
- **`onBindViewHolder(ViewHolder,int)`** – called when RecyclerView needs to rewrite a particular list row.
  - RecyclerView calls this when it needs to (re-)assign the data in a particular list row.
  - The adapter updates the supplied ViewHolder.
  - The int parameter identifies the data element that the ViewHolder should now display.

Adapter Definition

- It can make things easy to *nest* your adapter inside your activity/fragment:

```
public class MyFragment extends Fragment
{
    ...
    private class MyAdapter
        extends RecyclerView.Adapter<MyDataVHolder>
    {
        private List<MyData> data;
        public MyAdapter(List<MyData> data)
        {
            this.data = data;
        }
        ...
    }
    ...
}
```

Adapter.getItemCount()

```
private class MyAdapter
    extends RecyclerView.Adapter<MyDataVHolder>
{
    private List<MyData> data;
    public MyAdapter(List<MyData> data)
    {
        this.data = data;
    }

    @Override
    public int getItemCount()
    {
        return data.size();
    }
    ...
}
```

- The RecyclerView needs to know the total data size.

Adaptor.onCreateViewHolder()

```
private class MyAdapter
    extends RecyclerView.Adapter<MyDataVHolder> {
    ...
    @Override
    public MyDataVHolder onCreateViewHolder(ViewGroup parent,
                                            int viewType)
    {
        LayoutInflater li = LayoutInflater.from(
            getActivity()); // <-- Fragment method
        return new MyDataVHolder(li, parent);
    }
}
```

- Called when the RecyclerView needs a new ViewHolder.
  - MyDataVHolder will be our ViewHolder subclass.
- We need a LayoutInflater to create a ViewHolder.
  - LayoutInflater belongs to the activity; hence the call to Fragment's getActivity() method.

Adaptor.onCreateViewHolder() (What If. . .)

- We were able to call `getActivity()` because we're nested inside a `Fragment` class.
- *What if* `MyAdapter` was nested inside `MyActivity`?

```
LayoutInflater li = LayoutInflater.from(MyActivity.this);
```

- *What if* `MyAdapter` was a top-level class (i.e. not nested in anything)?
  - We'd need a field to refer to the activity instead:

```
private Activity activity;
private List<MyData> data;

public MyAdapter(Activity activity, List<MyData> data)
{
    this.activity = activity;
    this.data = data;
}
```

Adapter.onBindViewHolder()

```
private class MyAdapter
    extends RecyclerView.Adapter<MyDataVHolder> {
    ...
    @Override
    public void onBindViewHolder(MyDataVHolder vh, int index)
    {
        vh.bind(data.get(index));
    }
}
```

- Called when RecyclerView needs to use a ViewHolder to display a different data element.
- `index` identifies which element to display.
- We just pass the data to `bind()`, our own method which we haven't yet defined. . .

ViewHolder Definition

- Probably easiest to nest your view holder too, in the same place as your adapter.

```
public class MyFragment extends Fragment
{
    ...
    private class MyDataVHolder
        extends RecyclerView.ViewHolder
    {
        ...
    }

    private class MyAdapter extends ... {...}
}
```

ViewHolder UI Inflation

```
private class MyDataVHolder extends RecyclerView.ViewHolder
{
    public MyDataVHolder(LayoutInflater li,
                         ViewGroup parent)
    {
        super( li.inflate(R.layout.list_mydata,
                          parent, false) );
        ...
    }
}
```

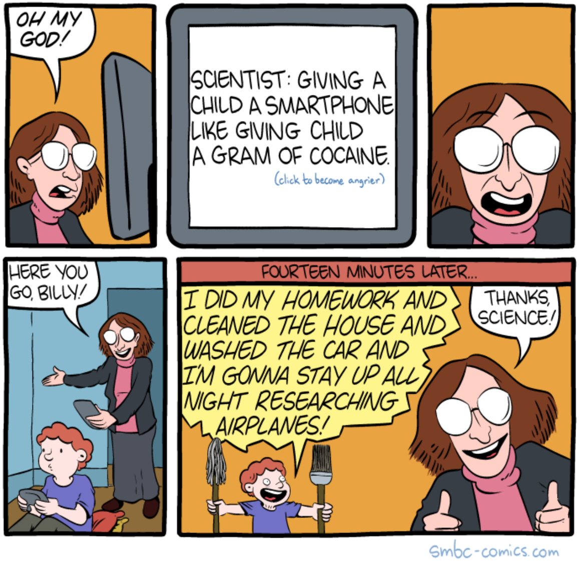
- There is a separate XML layout file for list rows.
  - In this case, `list_mydata.xml`.
  - Can contain any UI elements.
  - Applies to all list rows at the same time.
- Use the `LayoutInflater` to convert the XML to a `View` object tree.
  - The superclass then makes it available as “`itemView`”.

ViewHolder Updating Data

```
private class MyDataVHolder extends RecyclerView.ViewHolder
{
    private TextView textView; // Reference to UI element(s)

    public MyDataVHolder(LayoutInflater li,
                         ViewGroup parent)
    {
        super(...);
        textView =            // Grab UI element reference(s)
            (TextView) itemView.findViewById(R.id.list_data);
    }

    public void bind(MyData data) // Called by your adapter
    {
        textView.setText(data.getSomeStringData());
    }
}   // Note: 'itemView' is a superclass field.
```


OH MY GOD!
SCIENTIST: GIVING A CHILD A SMARTPHONE LIKE GIVING CHILD A GRAM OF COCAINE.
(click to become angrier)
HERE YOU GO, BILLY!
FOURTEEN MINUTES LATER...
I DID MY HOMEWORK AND CLEANED THE HOUSE AND WASHED THE CAR AND I'M GONNA STAY UP ALL NIGHT RESEARCHING AIRPLANES!
THANKS, SCIENCE!
smbc-comics.com

(https://www.smbc-comics.com/comic/smartphones)