Android Course Day 4

Lucio Cossio
Luis Mazoni

Course Agenda

Day 1

- **Development environment and tools**
- **Android project structure**
  - Source, tests , resources, manifest
- **Activity**
  - Lifecycle
  - Layout interaction
  - ActionBar
- **Practice**

Day 2

- **Review Day 1**
- **Resources**
  - String
  - Dimensions
  - Layout
- **Views**
  - TextView, Edit Text, Button
  - String resources
  - View listeners
- **Animation**
- **Practice**

Day 3

- **Review Day 2**
- **Intents**
  - Explicit
    - Extras
  - Implicit
    - Intent Filters
- **Android Manifest**
  - Overview
  - Add activities
- **Notifications**
- **Practice**

Day 4

- **Review Day 3**
- **Fragments**
  - Lifecycle
  - Fragment Manager
  - Arguments
- **Broadcast Receiver**
  - **Otto**
- **Async Task**
- **Practice**

Day 5

- **Review Day 4**
- **ListView**
  - Adapter
  - View Holder
- **Service**
- **Storage**
  - **SharedPreferences**
  - **SQLite**
- **Practice**

Review day 3

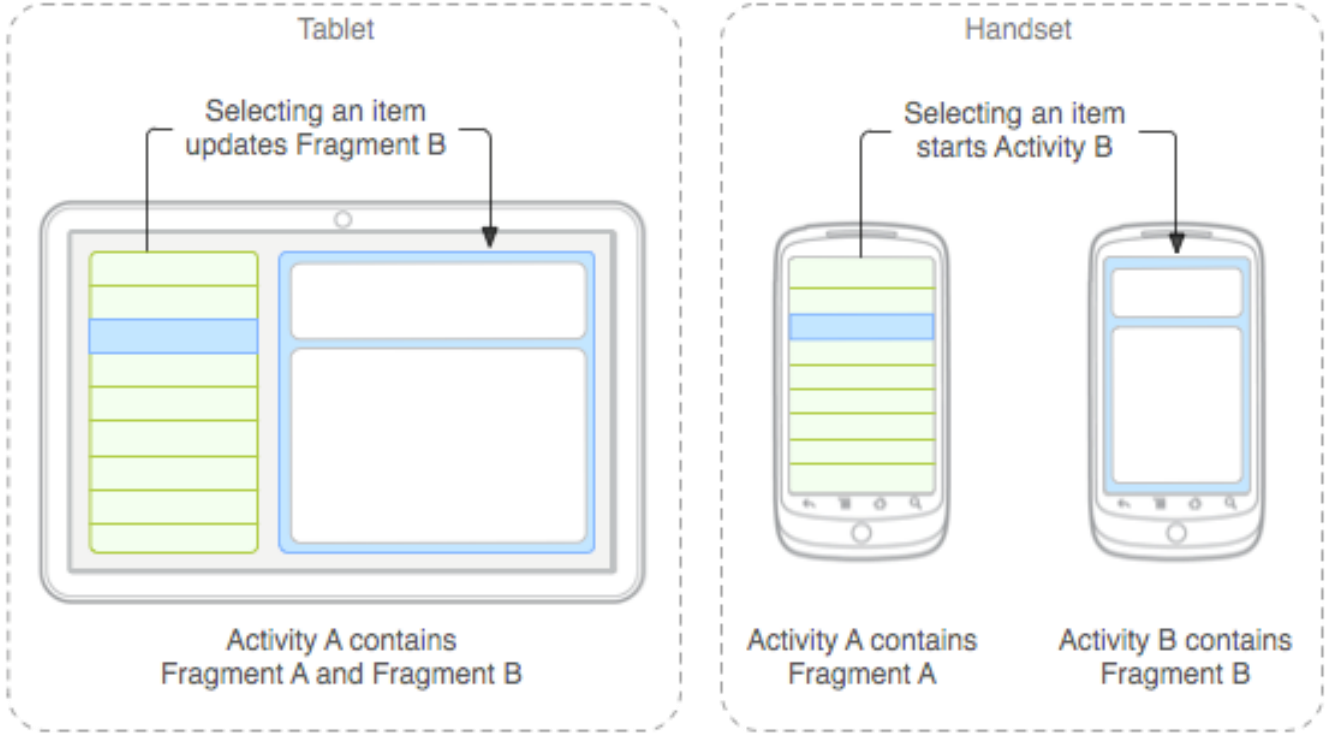
- Intents
  - Explicit
  - Implicit
- Android Manifest
- Notifications

Fragments

- A Fragment represents a **behavior or a portion of user interface** in an Activity. You can **combine multiple fragments in a single activity** to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which **has its own lifecycle, receives its own input events**, and which you **can add or remove while the activity is running**.

Fragments

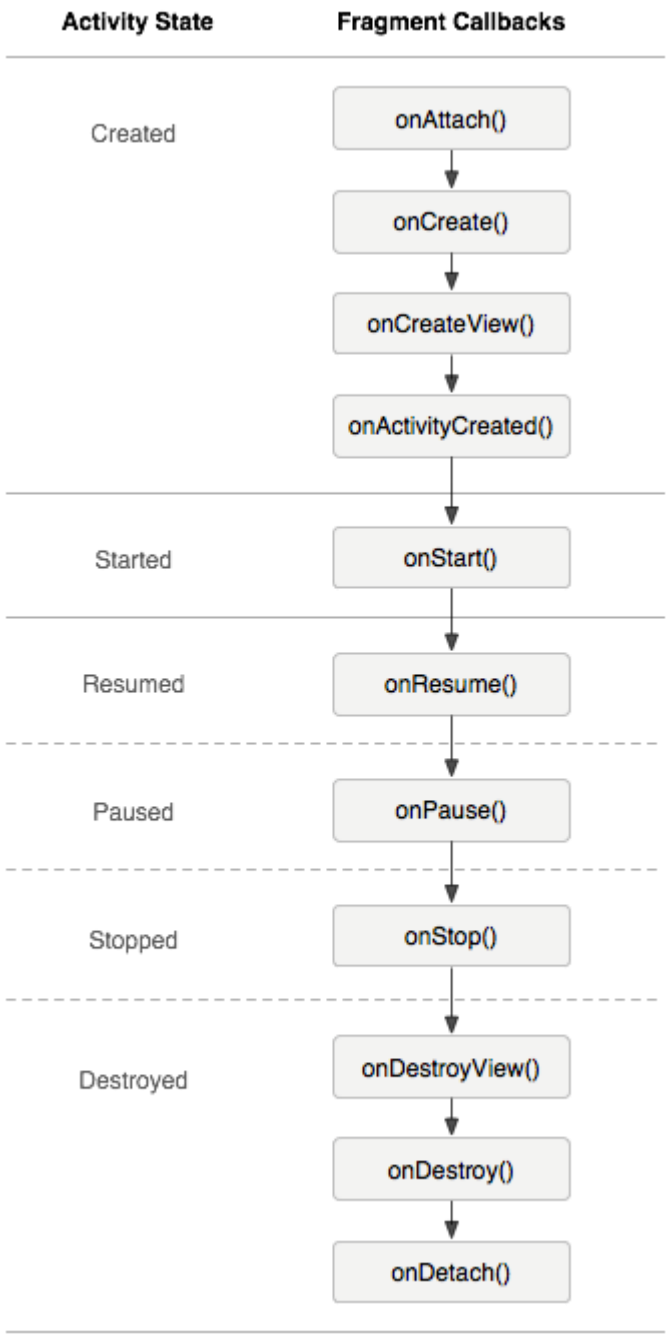
- Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets.
- Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components.

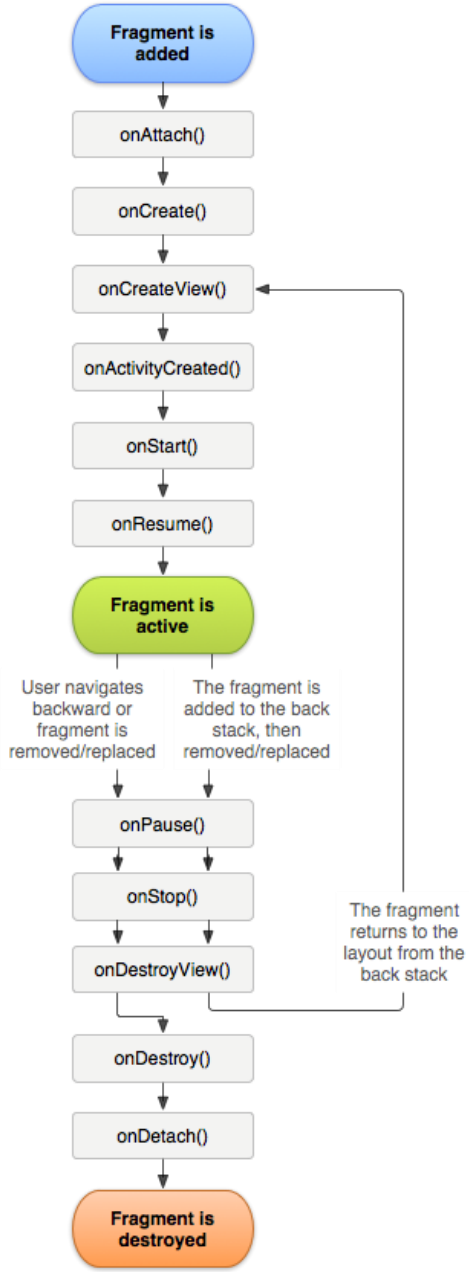


Fragments or Activities?

- Eric Burke from Square said they tried to create an application that worked with only one Activity, using fragments to change layouts. Didn't work very well.
- What he recommends is to use one activity for each "region" of the app:
  - On Boarding has several steps: name, address, bank account. One activity, several steps (fragments). :
  - Payment flow
  - Settings
- Why? Action bar stay fixed. Smooth animations. Code Organization.

Fragments - Lifecycle





Fragments - Lifecycle

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.ViewGroup;

public class ArticleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.article_view, container, false);
    }
}
```

Fragments – Add To Activity Layout

Activity

```
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.news_articles);
    }
}
```

news_articles Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <fragment android:name="com.example.android.fragments.HeadlinesFragment"
              android:id="@+id/headlines_fragment"
              android:layout_weight="1"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

    <fragment android:name="com.example.android.fragments.ArticleFragment"
              android:id="@+id/article_fragment"
              android:layout_weight="2"
              android:layout_width="0dp"
              android:layout_height="match_parent" />

</LinearLayout>
```

Fragments – Fragment Manager

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

Fragments – Arguments

```
public static DetailsFragment newInstance(int index) {
    DetailsFragment f = new DetailsFragment();

    // Supply index input as an argument.
    Bundle args = new Bundle();
    args.putInt("index", index);
    f.setArguments(args);

    return f;
}
```

```
getArguments().getInt("index", 0);
```

Practice: Drawer fragments for tablet and smartphone

Activity-Fragment communication - Listeners

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement OnArticleSelecte
        }
    }
    ...
}
```

Broadcast Receivers

- A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
- Filters for broadcast receivers can be registered dynamically by calling registerReceiver(). You can then unregister the receiver with unregisterReceiver(). Doing so allows your app to listen for specific broadcasts during only a specified period of time while your app is running.

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >

   <receiver android:name="MyReceiver">
      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED">
      </action>
      </intent-filter>
   </receiver>

</application>
```

Broadcast Receivers

- There are system broadcast intents that can be listen to:
  - ACTION_AIRPLANE_MODE_CHANGED
  - ACTION_BATTERY_LOW
  - ACTION_DEVICE_STORAGE_LOW
  - ACTION_HEADSET_PLUG
  - ACTION_POWER_CONNECTED
  - ACTION_POWER_DISCONNECTED
  - ACTION_SHUTDOWN

LocalBroadcastManager

- Helper to register for and send broadcasts of Intents to local objects within your process.

```
LocalBroadcastManager.getInstance(this).registerReceiver(
                mMessageReceiver, new IntentFilter("custom-event-name"));
```

```
LocalBroadcastManager.getInstance(this).unregisterReceiver(
                mMessageReceiver);
```

LocalBroadcastManager

```
private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
                // TODO Auto-generated method stub
                // Get extra data included in the Intent
                String message = intent.getStringExtra("message");
                Log.d("receiver", "Got message: " + message);
        }
};
```

```
Intent intent = new Intent("custom-event-name");
// You can also include some extra data.
intent.putExtra("message", "This is my message!");
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

Otto

Otto is an event bus designed to decouple different parts of your application while still allowing them to communicate efficiently.

- Use as a Singleton

```
Bus bus = new Bus();
```

- Publishing

```
bus.post(new AnswerAvailableEvent(42));
```

- Subscribing

```
bus.register(this);
```

```
@Subscribe public void answerAvailable(AnswerAvailableEvent event) {
    // TODO: React to the event somehow!
}
```

AsyncTasks

- AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int) ((i / (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) break;
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

```
new DownloadFilesTask().execute(url1, url2, url3);
```

Thank you

Lucio Cossio
Luis Mazoni