# Creating Applications and Activities

Ilir Dema

September 10, 2017

## Table of contents

## What makes an android application?

- **Activities** - your application's presentation layer.
- **Services** - the invisible workers of your application.
- **Content Providers** - manage and persis application data and interact with SQL databases.
- **Intents** - a powerful interapplciation message-passing framework.
- **Broadcast Receivers** - they enable your app to listen for intents that match your criteria.
- **Widgets** - visual components added to your device home screen.
- **Notifications** - they allow you to alert users to various events without interrupting current activity.

## Manifest File

### AndroidManifest.xml

- It is an XML file
- Contains name of the application and a default package,
- Sets up the various permissions required by this application,
- Defines the application-wide parameters,
- Defines the startup Activity for the application.

## AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ca.georgebrown.comp3074.programadviser">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ProgramAdviser">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## Resources

### /res/values/strings.xml

You can use file `string.xml` to keep global constants such as name of the app, communicates, headers etc.

```
<resources>
    <string name="app_name">Beer Adviser</string>
    <string name="action_settings">Settings</string>
    <string name="find_beer">Find Beer!</string>
    <string name="brands"></string>
    <string-array name="beer_colors">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
    </string-array>
</resources>
```

## /res/values/colors.xml

You can use `colors.xml` to define new color resources.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

## How?

### Declarative

Statically (offline) declare the interface using XML-based language. It is similar to building XHTML web page.

### Procedural

Simply means in code. You write Java code to create and manipulate all the user interface objects called Views.

### Mixed

Android allows you also to mix both methods. You can build a skeleton declaratively and fill in required elements during the execution. This approach allows for flexibility minimizing coding.

## Advice from Google

**Google's advice is to use declarative XML as much as possible.**

The XML code is often shorter and easier to understand than the corresponding Java code, and its less likely to change in future versions.

**Why XML?**

Although you see XML when writing your program, the Android resource compiler, aapt, does preprocess the XML into a compressed binary format. It is this format, not the original XML text, that is stored on the device.

## Basic Elements

**View**

An object that knows how to draw itself to the screen.

**ViewGroup**

Views that can contain or group other Views.

**Layout**

Gives Android hints about how youd like to see the Views arranged.

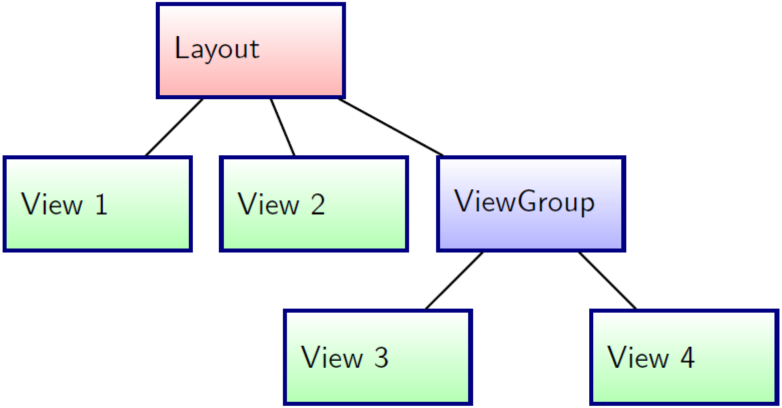


Figure: A concept of layouts, groups and views

## Layout

**Layout is:**

- a container for one or more child objects
- a description how to position them on the screen within the rectangle of the parent object

## Layout

### Commonly used layouts

- FrameLayout-all children start at the top left of the screen
- LinearLayout-children in a single column or row
- RelativeLayout-children in relation to each other or to the parent
- TableLayout-children in rows and columns, similar to an HTML table

## Vertical vs. Horizontal

### Horizontal

res/layout/main.xml
Can contain e.g. ListLayout

### Vertical

res/layout-land/main.xml
Can contain TableLayout nested in ListLayout to make better use of the space

## Enchant me

**Delight me in surprising ways**

A beautiful surface, a carefully-placed animation, or a well-timed sound effect is a joy to experience.

**Real objects are more fun than buttons and menus**

Allow people to directly touch and manipulate objects in your app. It reduces the cognitive friction.

**Let me make it mine**

People love to add personal touches because it helps them feel at home and in control.

**Get to know me**

Learn peoples' preferences over time. Place previous choices within

## Simplify my life

- Keep it brief

  Use short phrases with simple words. People are likely to skip sentences if they're long.
- Pictures are faster than words.

  Consider using pictures to explain ideas. They get people's attention and can be much more efficient than words.
- Decide for me but let me have the final say

  Take your best guess and act rather than asking first. Too many choices and decisions make people unhappy. Just in case you get it wrong, allow for 'undo'.
- Only show me when I need it

  People get overwhelmed when they see too much at once. Break tasks and information into small, digestible chunks. Hide options that aren't essential at the moment, and teach

## Simplify my life

- I should always know where I am

  Give people confidence that they know their way around. Make places in your app look distinct and use transitions to show relationships among screens. Provide feedback on tasks in progress.

- Never lose my stuff

  Save what people took time to create and let them access it from anywhere.

- If it looks the same, it should act the same.

  Help people discern functional differences by making them visually distinct rather than subtle. Avoid modes, which are places that look similar but act differently on the same input.

- Only interrupt me if it is important.

## Make me amazing

- Give me tricks that work everywhere

  People feel great when they figure things out for themselves.
- It is not my fault

  Be gentle in how you prompt people to make corrections. They want to feel smart when they use your app.
- Sprinkle encouragement

  Break complex tasks into smaller steps that can be easily accomplished. Give feedback on actions, even if it's just a subtle glow.
- Do the heavy lifting for me

  Make novices feel like experts by enabling them to do things they never thought they could.

Source:
`https://developer.android.com/design/get-started/principles`

## Activity

**What is it?**

An activity is a user interface screen. Applications can define one or more activities to handle different phases of the program.

**Declaration**

In AndroidManifest.xml.

**Definition**

In Java as an extension of Activity class.

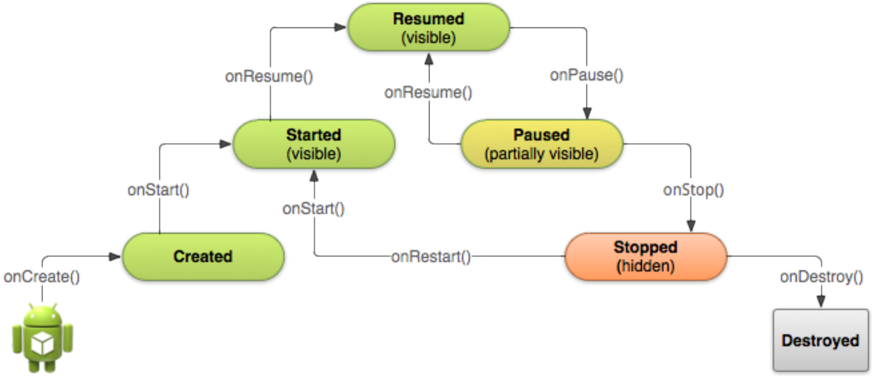
## Definition

```
// various package imports here

public class FindBeerActivity extends Activity {
    // Instance variables
    private BeerExpert expert = new BeerExpert();
   // Called when activity is first created
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Initialize, build UI elements
    }

    public void onClickFindBeer(View view) {
        // event handling
    }
}
```

## Starting an activity

Unlike other programming paradigms in which apps are launched with a `main()` method, the Android system initiates code in an `Activity` instance by invoking specific callback methods that correspond to specific stages of its lifecycle. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

## Android Application Lifecycle



## Activity Lifecycle Methods

Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods. Implementing your activity lifecycle methods properly ensures your app behaves well in several ways, including that it:

- Does not crash if the user receives a phone call or switches to another app while using your app.
- Does not consume valuable system resources when the user is not actively using it.
- Does not lose the user's progress if they leave your app and return to it at a later time.
- Does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation.

## Activity States

The activity can exist in one of only three states for an extended period of time:

- Resumed: the activity is in the foreground and the user can interact with it. ("running" state.)
- Paused: the activity is partially obscured by another activity. The paused activity does not receive user input and cannot execute any code.
- Stopped: the activity is completely hidden and not visible to the user; the activity instance and all its state, but it cannot execute any code.

The other states (Created and Started) are transient and the system quickly moves from them to the next state by calling the next lifecycle method: after the system calls onCreate(), it quickly

## Intent

**Whatis it?**

An intent is a mechanism for describing a specific action, such as pick a photo, phone home, or show map

**How is it done?**

An instance of Intent class is created and parameters such as action, category and data are set.

**Fundamental uses**

- Tostart an activity
- To start a service
- To deliver a broadcast

## Calling specific activity

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as
// "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this,
                             DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

## Component based architecture

### Ask others to do it for you

- Android allows you to call activities and services from other apps!
- You can send a request e.g. to open a photo or show a map and (if there is an app accepting such call) it will be served.
- If multiple apps can handle it then user can choose who (which app) will be used.

## Calling activity based on action/category

```
Intent myIntent = new Intent(
        android.content.intent.ACTION_VIEW,
        // While house location
        Uri.parse("geo:38.899533,-77.036476"));
startActivity(myIntent);

// or startActivityForResult(myIntent resCode);
// if you want to get a result
```

**Fragment**

Encapsulated, reusable component with own lifecycle and UI that can be used to build activity.
It is tightly bound to the Activity into which it is placed.
They can be reused in various activities.

## Creating a fragment

To create a fragment, you must create a subclass of `Fragment` (or an existing subclass of it). The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as `onCreate()`, `onStart()`, `onPause()`, and `onStop()`.

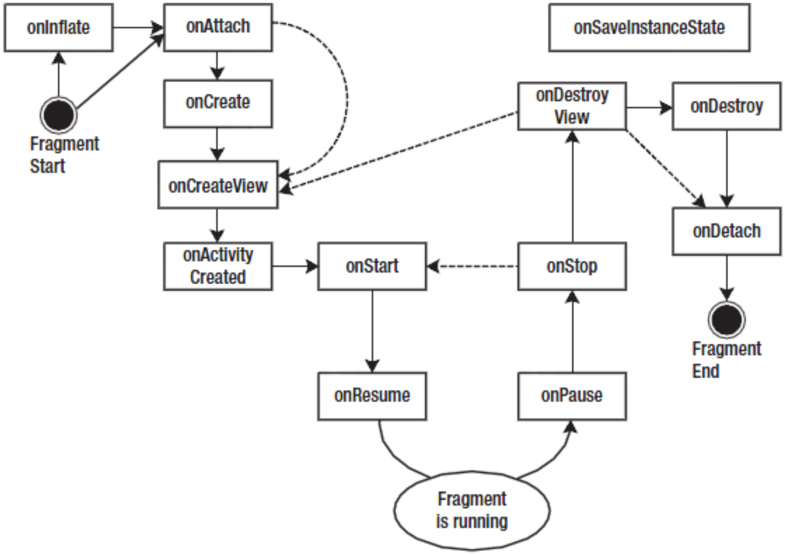
## Create a Fragment Class

### Extend Fragment Class

- Create layout file with view hierarchy (optionally) If your fragment requires UI override onCreateView handler to inflate and return the view hierarchy (from layout file or automatically).

## Adding a user interface to a fragment

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


onInflate
onAttach
onSaveInstanceState
onCreate
onDestroy View
onDestroy
Fragment Start
onCreateView
onDetach
onActivity Created
onStart
onStop
Fragment End
onResume
onPause
Fragment is running

## Fragment-specific Lifecycle Events

**Attach/Detach**

Start and End of the life-cycle

**Create/Destroy**

Start and End of the created lifetime

**Create/Destory UI**

`onCreateView` should be used to initialize fragment's UI

## Fragment States

**State**

A state of the fragment is closely related to the corresponding Activity state and it's transitions are related to Activity's transitions.

## Fragment Manager

FragmentManager is used to manage fragments that the Activity contains. It provides the methods to perform fragment transaction to add, remove and replace fragments.