Activity & Layout

What's an Activity?

- Core component of Android applications
- An Activity is a means for the user to accomplish one main goal.
- An Android app is composed of one or more activities.
- Every screen you see in an app is an Activity
- Every activity has an associated layout file

For more info visit https://developer.android.com/guide/components/activities/intro-activities

Creating an Activity

```
public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        // Initialize UI components here

    }

}
```

Declaring Activity in Manifest

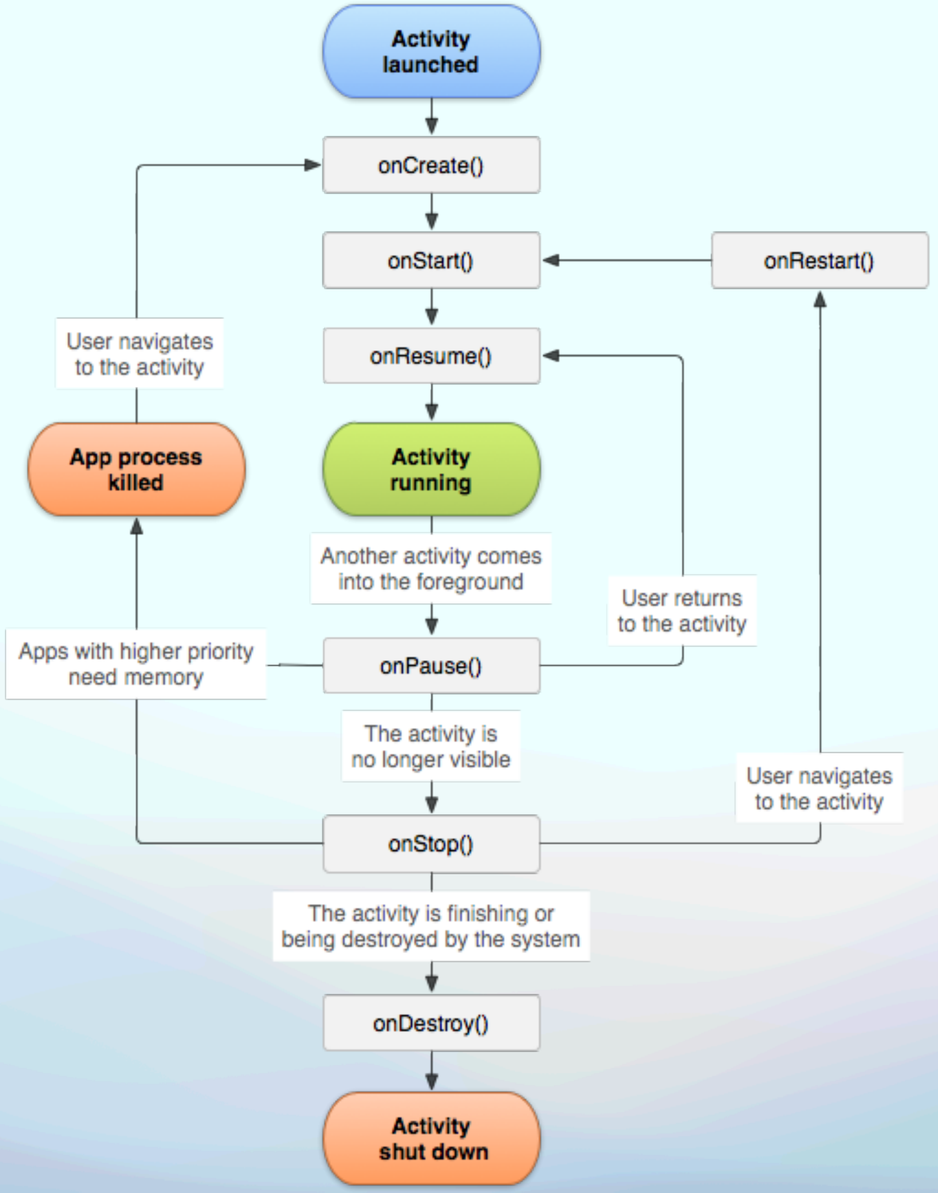
```
<manifest ...>
    <application ...>
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- Regular Activity -->
        <activity
            android:name=".SecondActivity"
            android:label="Second Screen"
            android:screenOrientation="portrait"/>
    </application>
</manifest>
```

Important Manifest Attributes

- android:name: Activity class name
- android:label: Title shown in app bar
- android:theme: Visual style of activity
- android:exported: Whether other apps can start it
- android:parentActivityName: Parent for navigation
- android:screenOrientation: Fixed/dynamic orientation
- android:launchMode: How activity should be launched

See https://developer.android.com/guide/topics/manifest/activity-element for more details

Activity Lifecycle



Activity lifecycle

- **onCreate()** Called when the activity is first created. Used for initializing UI and loading resources.
- **onStart()** The activity becomes visible but is not yet interactive
- **onResume()** The activity is in the foreground and ready for user interaction
- **onPause()** The activity is partially visible but loses focus (e.g., another activity is on top).
- **onStop()** The activity is completely hidden but still exists in memory.
- **onDestroy()** The activity is being removed from memory, and resources should be cleaned up.
- **onRestart()** – Called when an activity that was stopped is coming back to the foreground.

See https://developer.android.com/guide/components/activities/activity-lifecycle for more info

Lifecycle Method Usage

- onCreate(): Initialize components, bind data
- onStart(): Register UI-related listeners
- onResume(): Start animations, GPS updates
- onPause(): Save draft data, stop animations
- onStop(): Save persistent data, release resources
- onDestroy(): Clean up resources, unregister listeners
- onRestart(): Refresh UI data

Common Lifecycle Scenarios

1. App Launch:
   - onCreate() → onStart() → onResume()
2. Rotating Screen:
   - onPause() → onStop() → onDestroy() →
   - onCreate() → onStart() → onResume()
3. Pressing Home:
   - onPause() → onStop()
4. Returning to App:
   - onRestart() → onStart() → onResume()
5. Pressing Back:
   - onPause() → onStop() → onDestroy()

Views

- Everything visible on screen is a View
- Views are the user interface building blocks in Android
  - Bounded by a rectangular area on the screen
  - Responsible for drawing and event handling
  - Examples: TextView, ImageView, Button
- Can be grouped to form more complex user interfaces

  For more information : https://developer.android.com/reference/android/view/View

XML Layouts

You can also edit your layout in XML.

- Android uses XML to specify the layout of user interfaces (including View attributes)
- Each View in XML corresponds to a class in Kotlin that controls how that View functions

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"/>
```

Size of a View

- wrap_content

  android:layout_width="wrap_content"

- match_parent

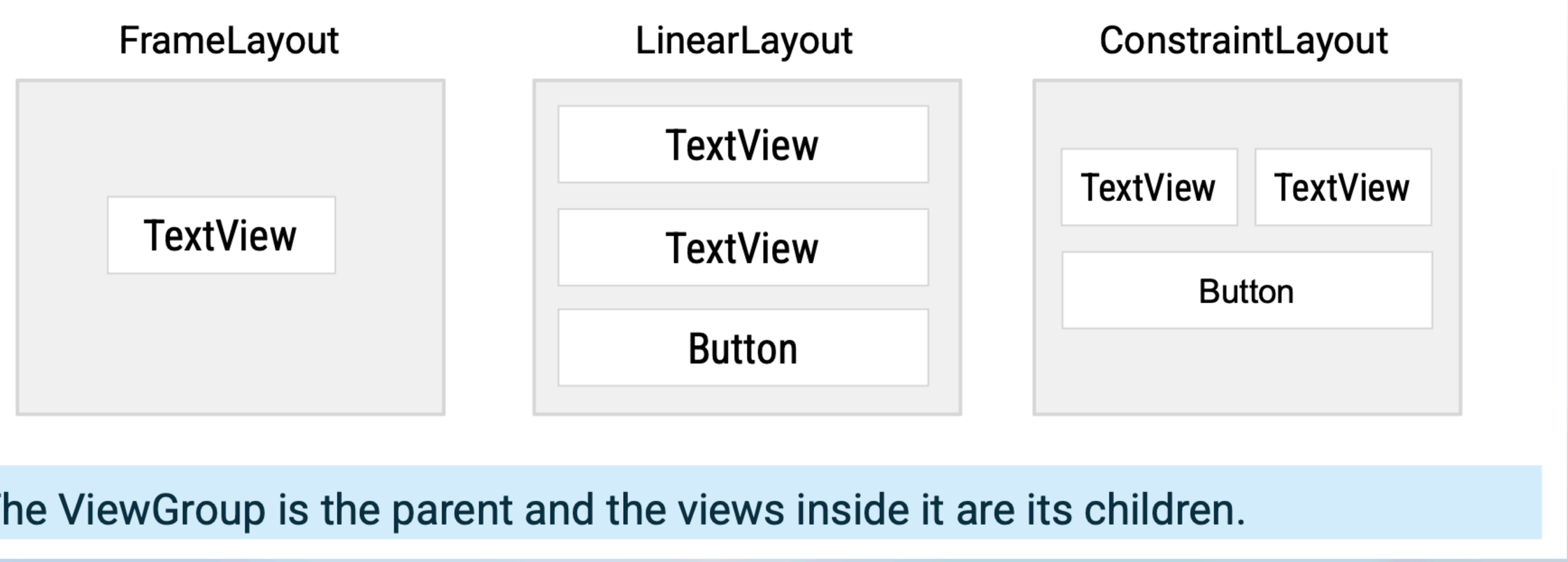
  android:layout_width="match_parent"

- Fixed value (use dp units)

  android:layout_width="48dp"

ViewGroups

A `ViewGroup` is a container that determines how views are displayed.



The ViewGroup is the parent and the views inside it are its children.

ViewGroups (Layouts)

- **LinearLayout** – Arranges views in a row/column
- **RelativeLayout** – Position elements relative to each other
- **ConstraintLayout** – More flexible positioning
- **FrameLayout** – Overlapping views
- **ScrollView** – Scrollable content

App Resources

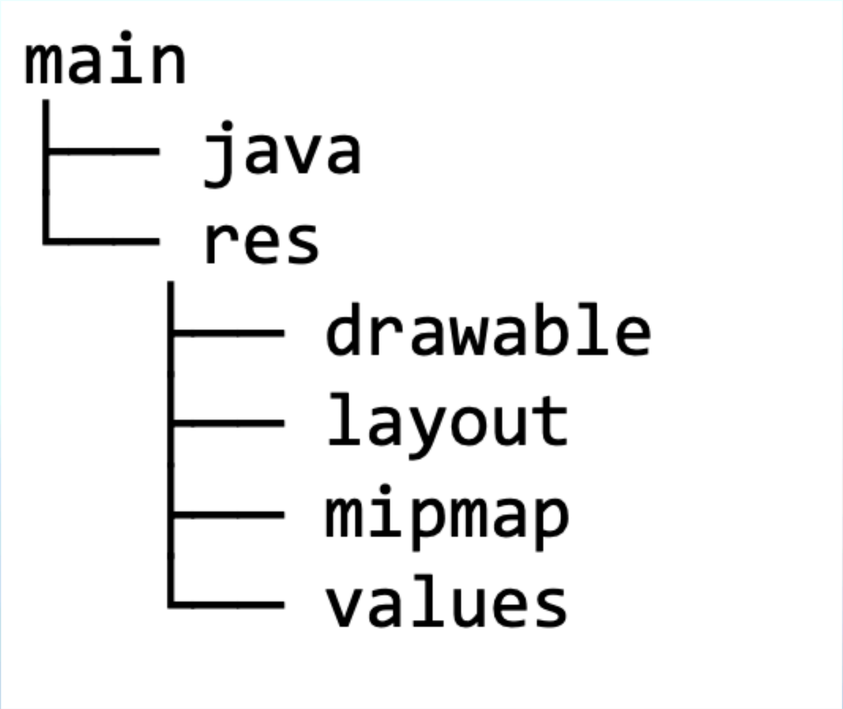
Static content or additional files that your code uses

- Layout files
- Images
- Audio files
- User interface strings
- App icon

Common resource directories

Add resources to your app by including them in the appropriate resource directory under the parent `res` folder.

.



Resource IDs

- Each resource has a resource ID to access it.
- When naming resources, the convention is to use all lowercase with underscores (for example, `activity_main.xml`).
- Android autogenerates a class file named `R.java` with references to all resources in the app.
- Individual items are referenced with: `R.<resource_type>.<resource_name>`

Examples:

R.drawable.ic_launcher (res/drawable/ic_launcher.xml)

R.layout.activity_main (res/layout/activity_main.xml)

Resource IDs for views

Individual views can also have resource IDs.

Add the `android:id` attribute to the View in XML. Use `@+id/name` syntax.

```
<TextView
    android:id="@+id/helloTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!”/>
```

Within your app, you can now refer to this specific TextView using:

R.id.helloTextView

Thank you!