Mobile Applications

Lecture 3

Mobile Programming- Android Studio

Android Studio

- Java Eclipse was the programming language to build applications.
- In May 2013, it was announced at the Google I/O conference of 2013 that Google had bought Android Studio.
- Android Studio is the official IDE and specialized for Android application development.

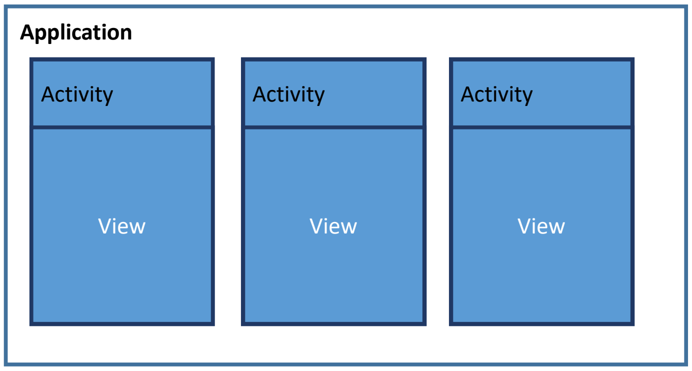
Framework Capabilities and Add-Ons

- *Tailored to mobile apps*
- *Built-in services:*
  - GUI, OS services (file I/O, threads, device management), Graphics, Device, access (GPS, camera, music and video players, sensors), Web services, Networking, XML processing, Standard language libraries.
- *Add-ons:*
  - Maps
  - Database support (SQLite)
  - WebKit

Android Studio Project Components

- **Activity:** a "single screen" that's visible to user
- **Service:** long-running background "part" of app.
- **ContentProvider:** manages app data (usually stored in database) and data access for queries.
- **BroadcastReceiver**: component that listens for particular Android system "events", e.g., "found wireless device", and responds accordingly.

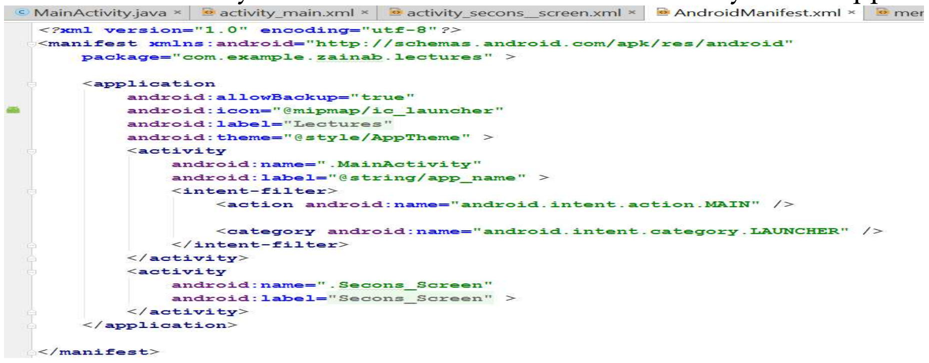
Android Activity



- Android system operation is performed by triggering an Activity.
- Single, focused thing that the developer can do to create a screen that the users can interact with.

Android Manifest

- Every app must have AndroidManifest.xml
- The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

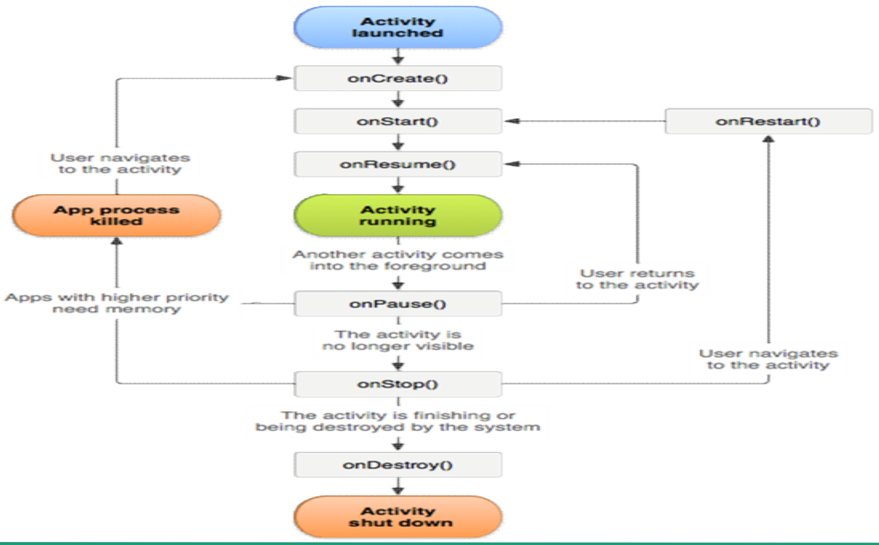


```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.zainab.lectures" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Lectures"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".Secons_Screen"
            android:label="Secons_Screen" >
        </activity>
    </application>

</manifest>
```

Android Manifest

- The manifest specifies:
- App's Activities, Services, etc.
- Permissions requested by app
- Minimum API required
- External libraries to which app is linked, e.g., Google Maps library.

Android Activity Lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity Methods

Method	Description
OnCreate()	Called when the activity is first created. Perform basic startup tasks such as creating interface.
onStart()	Make the activity visible to the user.
onResume()	Called every time activity comes into focus when the activity will start interacting with the user
onPause()	When an interruptive event occurs, the activity enters the *Paused* state. calls when the user is leaving the activity (though it does not always mean the activity is being destroyed). Stop animations or other ongoing actions that could consume CPU. Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email). Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them.

Activity Methods

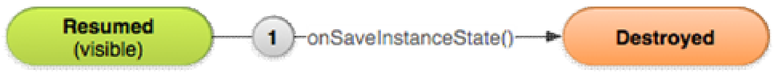
Method	Usage
onStop()	The activity is no longer visible to the user. Should release all resources not needed while user not using app.
onRestart()	Called after the activity has been stopped, prior to it being started again
onDestroy()	Called to destroy the activity either because the activity is finishing or because the system is temporarily destroying this activity to save space. Cleanup operations such as killing threads, releasing locks.

Activity Destructions

- **Normal**
- When user presses back button.
- When code calls finish().

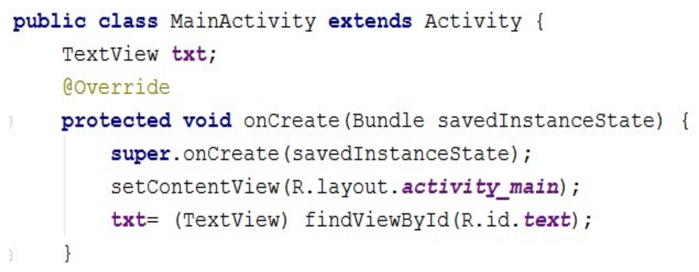
- **Abnormal**
- An activity unused for a while can be destroyed by system without warning!!.
- User rotates screen!

Recreating Activity

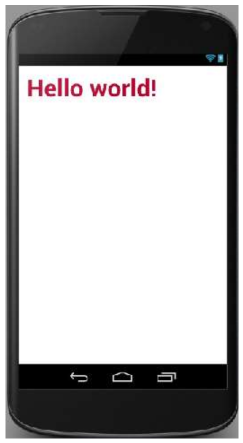


- By default, the system uses the Bundle instance state and a combined approach onSaveInstanceState() to save information about each View object in your activity layout (such as the text value entered into an EditText widget) .
- So, if your activity instance is destroyed and recreated, the state of the layout is restored to its previous state with no code required by you.

Android Activity "Hello World!" Example



```
public class MainActivity extends Activity {
    TextView txt;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txt= (TextView) findViewById(R.id.text);
    }
```

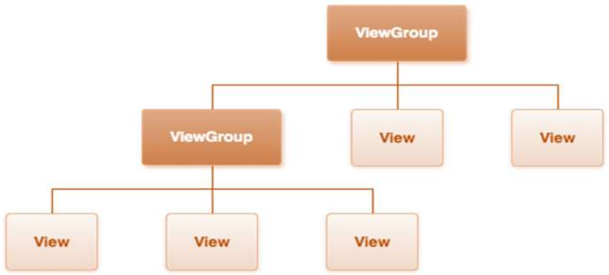


Activity UI

- If an app is to be interactive it needs a View.
- *Views* include *widgets*.
- *Views* can be grouped using *ViewGroups*.
- Derive from Android.view.view

Activity UI

- Many type of:
- Widgets (Views): Buttons, TextView, ImageView, ProgressBar…etc.
- ViewGroups: LinearLayout, ConstraintLayout, TableLayout, RelativeLayout, FrameLayout.



Views & ViewGroups Attributes

- Most are start with word `Android:`
- **Giving id**
- android:id = unique identifier.
- Must start with @ followed by type (in this case id)
- + needed when defining resource. Not needed for concrete resources such as strings.

```
<Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Next"
    android:textSize="30dp"
    android:id="@+id/button"
/>
```

Views & ViewGroups Attributes

- **LayoutParams**
- class just describes how big the view or viewgroup wants to be for both width and height. It can be specify either by numbers OR by constants

Fill_parent	uses maximum available width. `android:layout_width= "fill_parent"`
Wrap_content	height restricted by component (usually text)height. `android:layout_width= "wrap_content"`
Match_parent	Same as Fill_parent in API Level 8 and higher `android:layout_width= "match_parent"`

Views & ViewGroups Attributes

- **Measurements Units**
  - **dp** = Density-independent pixel. 160dp = 1 screen inch. (Recommended)

```
android:layout_marginBottom="427dp"
```

  - **Sp** = Scale –independent pixel (recommended for font sizes). The SP unit is sensitive to the user's display setting, so it makes text flexible for all screen sizes.

```
android:textSize="34sp"
```

Views & ViewGroups Attributes

- **Measurements Units**

mm	Millimeters - based on the physical size of the screen.
px	Pixels - corresponds to actual pixels on the screen.
pt	Points - 1/72 of an inch based on the physical size of the screen
in	Inches - based on the physical size of the screen. 1 Inch = 2.54 centimeters

Views & ViewGroups Attributes

- **Colors**
- **Programmatically (Java file)**
  - Names: Color.RED, Color.BLUE….etc.
  - OR Color.*parseColor*("#44f0a0")

```
txt.setTextColor(Color.RED);

txt.setTextColor(Color.parseColor("#44f0a0"));
```

  - **In XML file**

```
android:textColor="#ffb360"
```

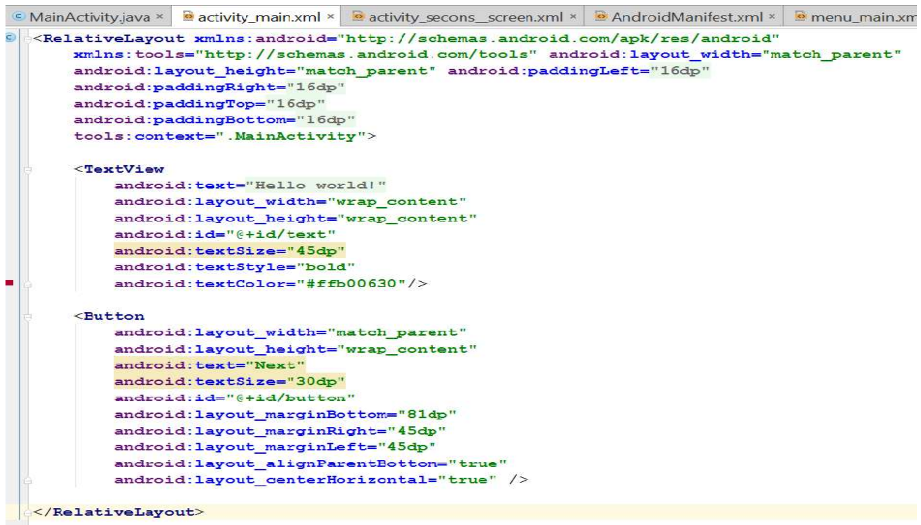
Views & ViewGroups Attributes

- Many more attributes…..

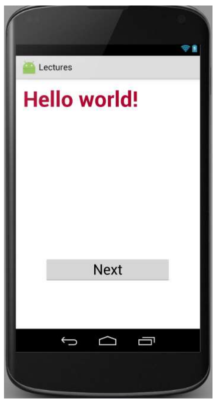
```
android:textStyle="bold"

android:gravity="center"
```

- Can we change the text size programmatically (Java file)?
- How do you deal with layouts (viewgroups) programmatically?

Implementation of Activity UI



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/text"
        android:textSize="45dp"
        android:textStyle="bold"
        android:textColor="#ffb00630"/>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Next"
        android:textSize="30dp"
        android:id="@+id/button"
        android:layout_marginBottom="81dp"
        android:layout_marginRight="45dp"
        android:layout_marginLeft="45dp"
        android:layout_alignParentBotton="true"
        android:layout_centerHorizontal="true" />

</RelativeLayout>
```



Input Events

- A way to collect data about a user's interaction with interactive components of Applications such as button presses or screen touch… etc.

- **Event listeners**
- is an interface in the View class that contains a single callback method.

Input Events

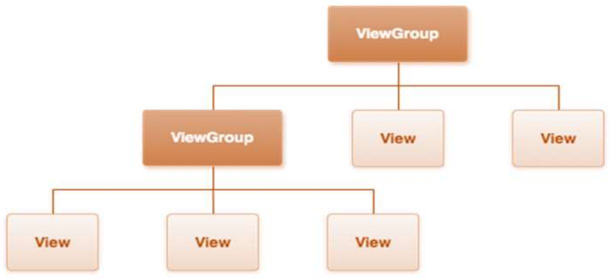
Interface	Method	Description
View.OnClickListener	onClick()	when the user either clicks or touches or focuses upon any widget like button, text, image etc
OnLongClickListener()	onLongClick()	when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds
OnTouchListener()	onTouch()	when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen.

Example

```
public class MainActivity extends Activity {
    TextView txt;
    Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txt= (TextView) findViewById(R.id.text);
        btn=(Button) findViewById(R.id.button);

        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                txt.setTextColor(Color.BLUE);
                btn.setBackgroundColor(Color.parseColor("#ff3398"));
                Toast.makeText(getBaseContext(),"Color Changed", Toast.LENGTH_SHORT).show();
            }
        });     }
```

Lab Work

- Create an Android Studio activity that simulates the diagram below. include as many view attributes as you could.
- change the text size programmatically


ViewGroup
ViewGroup
View
View
View
View
View