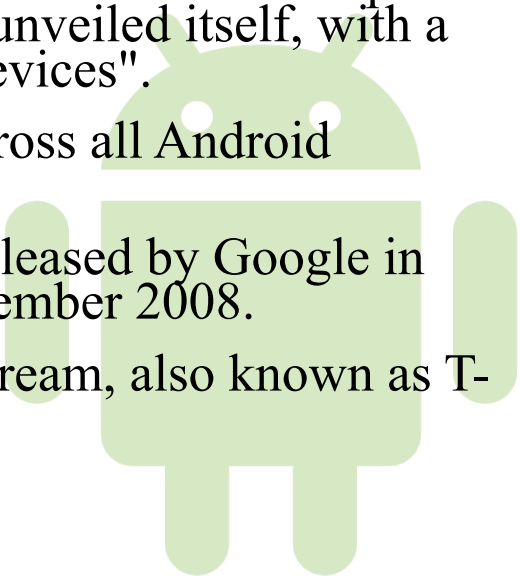
Mobile Design - Android

Outline

- Android – background, architecture and features
- Android Studio basics
- Basic concepts – Resources, Activity, Services, etc.
- Event handling
- HTTP networking and JSON parser
- Running application on an Android device/Emulator
- References

Background

- Android is an open source and Linux-based Operating System for smartphones.
- Android Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White.
- Android was pitched as a handset operating system that would rival Symbian and Microsoft Windows Mobile.
- In July 2005,Google acquired Android Inc. for at least $50 million. Its key employees, including Rubin, Miner and White, joined Google as part of the acquisition
- On November 5, 2007, the *Open Handset Alliance*, a consortium of technology companies including Google, device manufacturers such as HTC, Motorola and Samsung, wireless carriers such as Sprint and T-Mobile, and chipset makers such as Qualcomm and Texas Instruments, unveiled itself, with a goal to develop "the first truly open and comprehensive platform for mobile devices".
- Developers need to develop only once for Android and the applications run across all Android devices.
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- The first commercially available smartphone running Android was the HTC Dream, also known as T-Mobile G1, announced on September 23, 2008.



Features

- Open Source
- Larger Developer Community
- High Marketing
- Rich Development Environment
- Inter App Integration Feature
- High Security
- Reduced Cost of Development
- Higher Success Ratio

Categories of Applications

Music

News

Multimedia

Sports

Lifestyle

Food & Drinks

Books

Weather

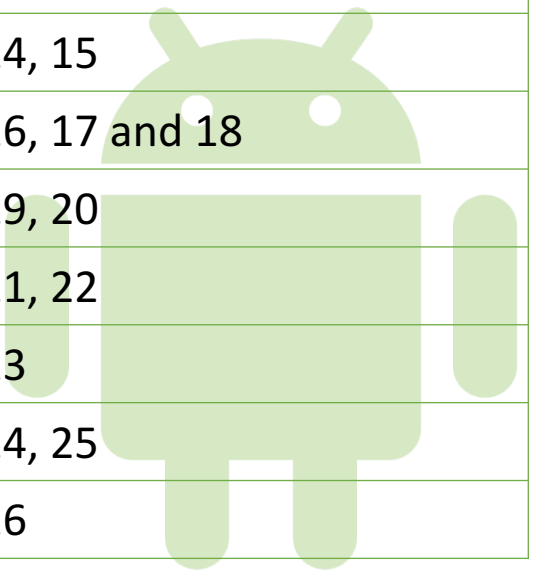
Reference

Navigation

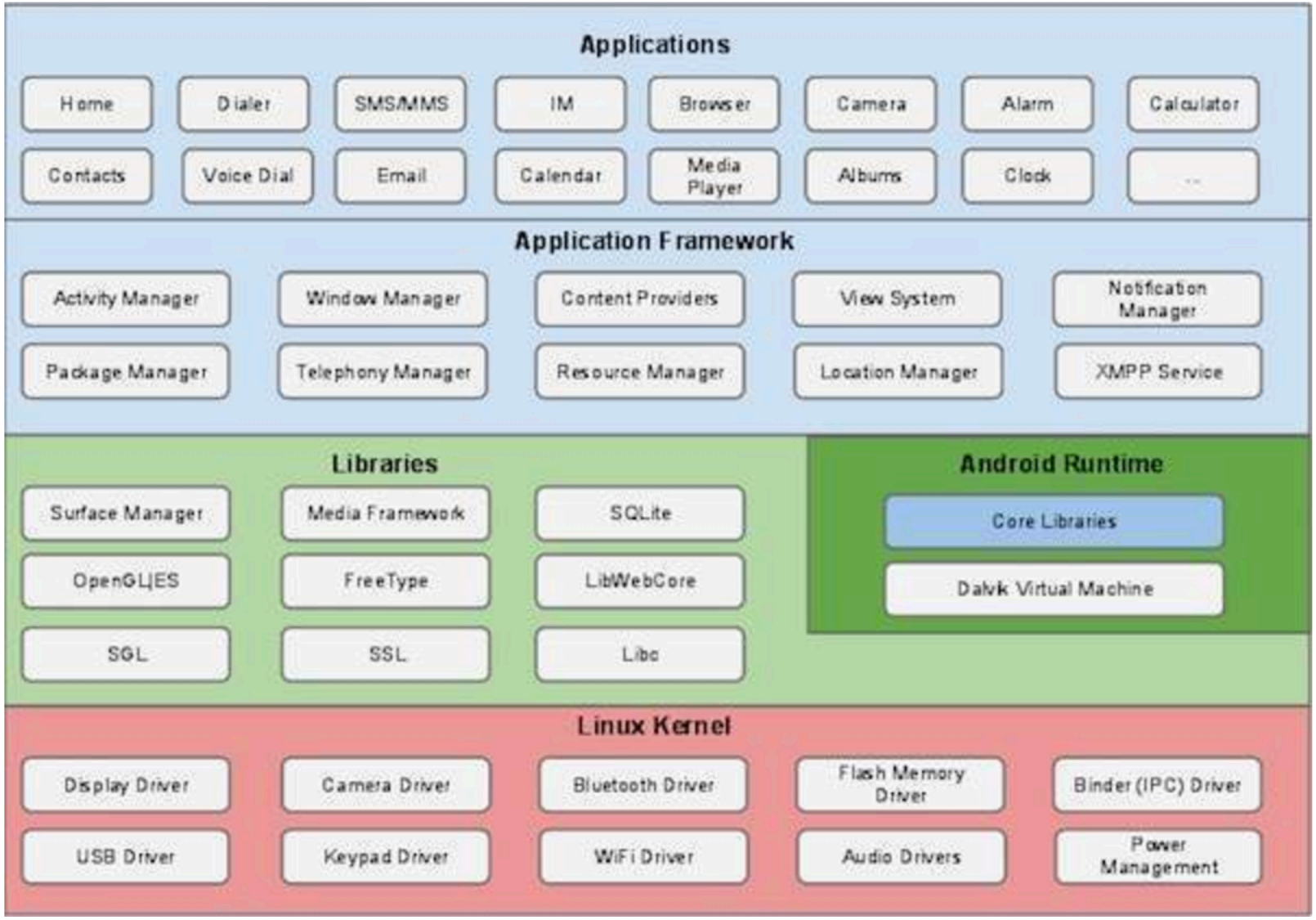
Social Media

History

Android Version	Code name	API level
1.0	APPLE PIE	1
1.1	BANANA BREAD	2
1.5	CUPCAKE	3
1.6	DONUT	4
2.0, 2.1	ECLAIR	5, 6, 7
2.2	FROYO	8
2.3	GINGERBREAD	9 and 10
3.0, 3.1, 3.2	HONEYCOMB	12 and 13
4.0	ICE CREAM SANDWICH	14, 15
4.1, 4.2, 4.3	JELLY BEAN	16, 17 and 18
4.4	KITKAT	19, 20
5.0, 5.1	LOLLIPOP	21, 22
6.0	MARSHMALLOW	23
7.0, 7.1	NOUGAT	24, 25
8.0	OREO	26



Architecture



Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

Architecture

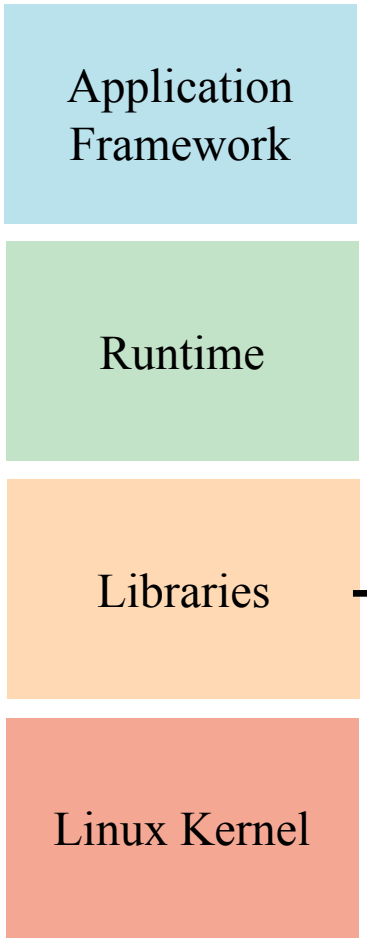
Linux Kernel

Linux Kernel

Hardware drivers

Hardware/software interface

Architecture



Libraries

android.app

android.content

android.database

android.opengl

android.os

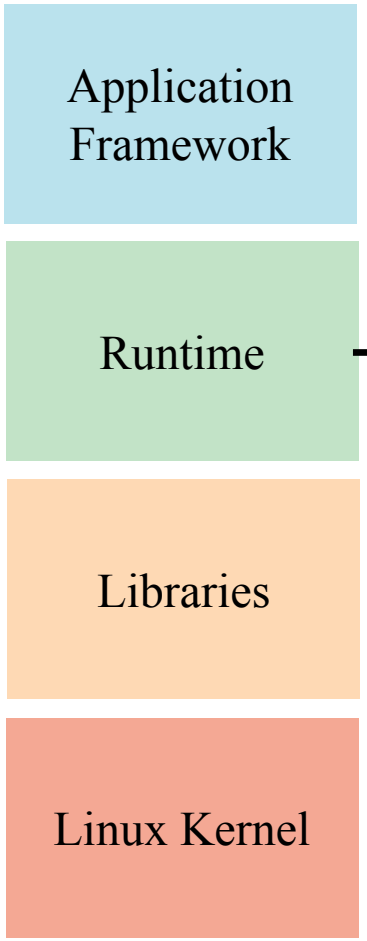
android.text

android.view

android.widget

android.webkit

Architecture



Runtime

Dalvik Virtual Machine

Core libraries

Architecture

Application Framework

Activity Manager

Content Providers

Resource Manager

Notifications Manager

View System

Android Studio Basics

- Setup
- Create a new project
- Get familiar with Android Studio
- Design UI in XML using drag and drop in xml resource layout editor
- Control UI using Activity
- Change Activity properties in Manifest file
- Run your app in the emulator

Setup

You can start Android application development on either of the following operating systems −

- Microsoft Windows
- Mac OS X
- Linux

Following is the list of software's you will need before you start your Android application programming:

- Java JDK5 or later version
- Android Studio

Setup

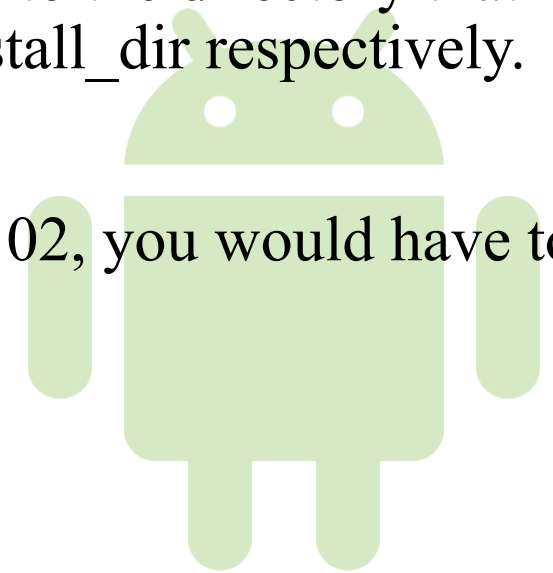
- You can download the latest version of Java JDK from Oracle's Java site − Java SE Downloads

- Install and configure the setup.

- Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java_install_dir/bin and java_install_dir respectively.

- If you are running Windows and installed the JDK in C:\jdk1.8.0_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```



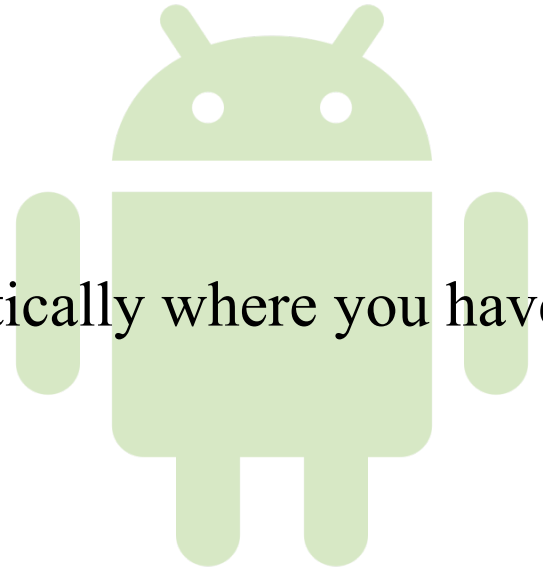
Setup

- Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.

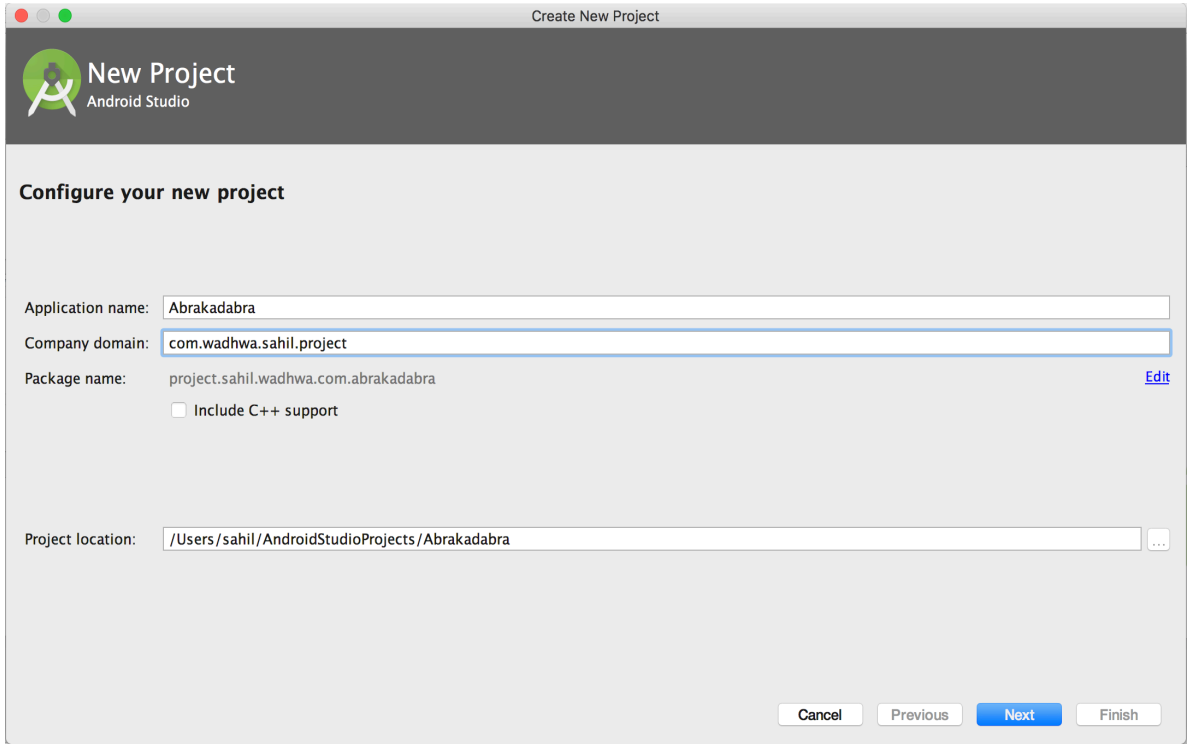
- On Linux, if the SDK is installed in /usr/local/jdk1.8.0_102 and you use the C shell, you would put the following code into your **.cshrc** file.

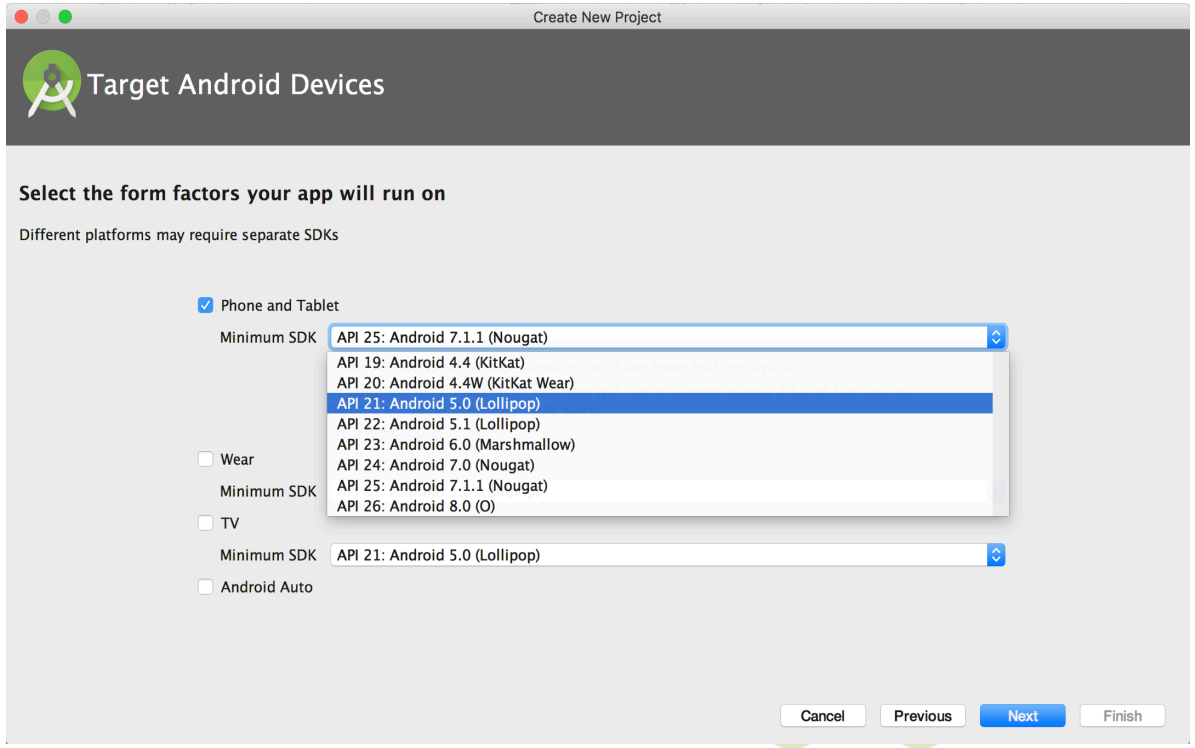
```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

- Alternatively, if you use Android Studio, then it will know automatically where you have installed your Java.

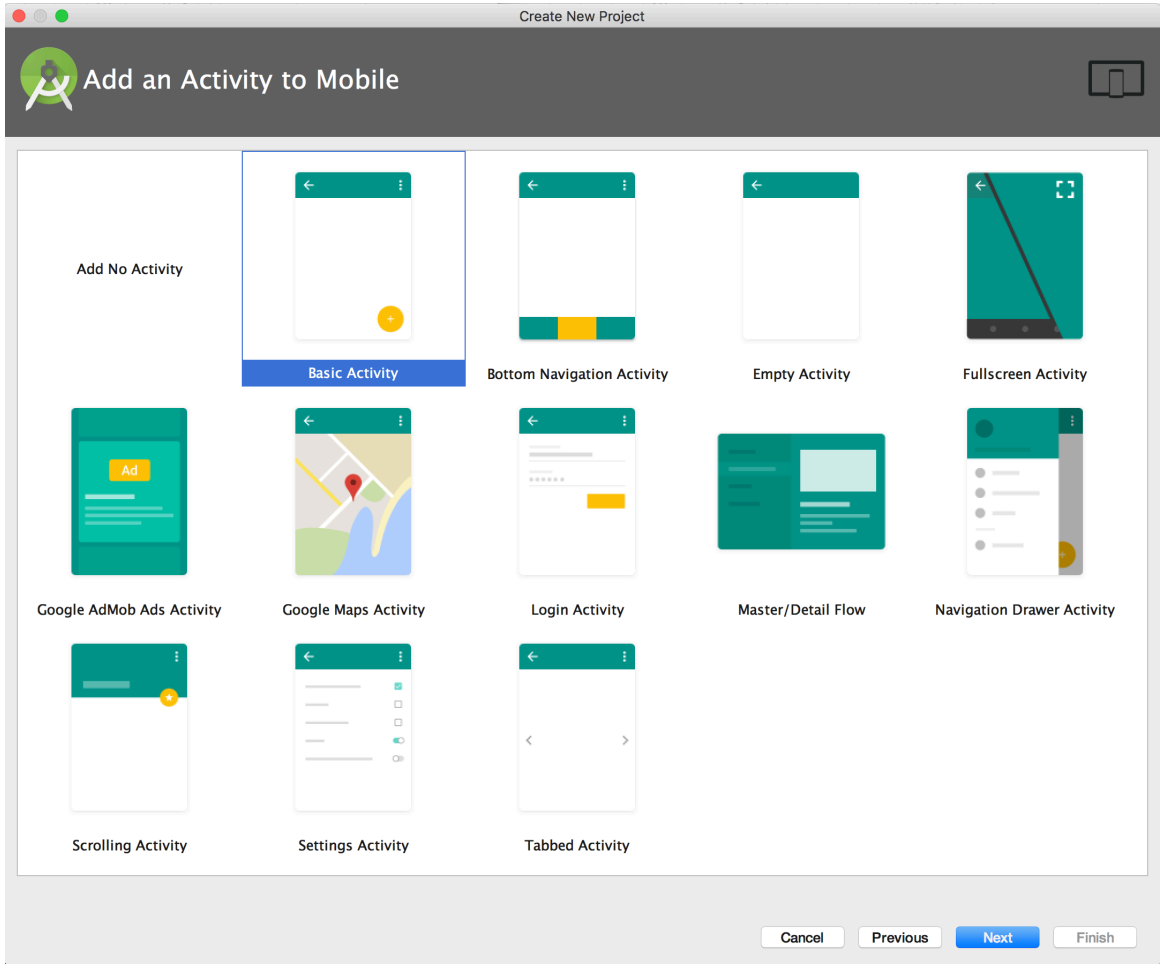


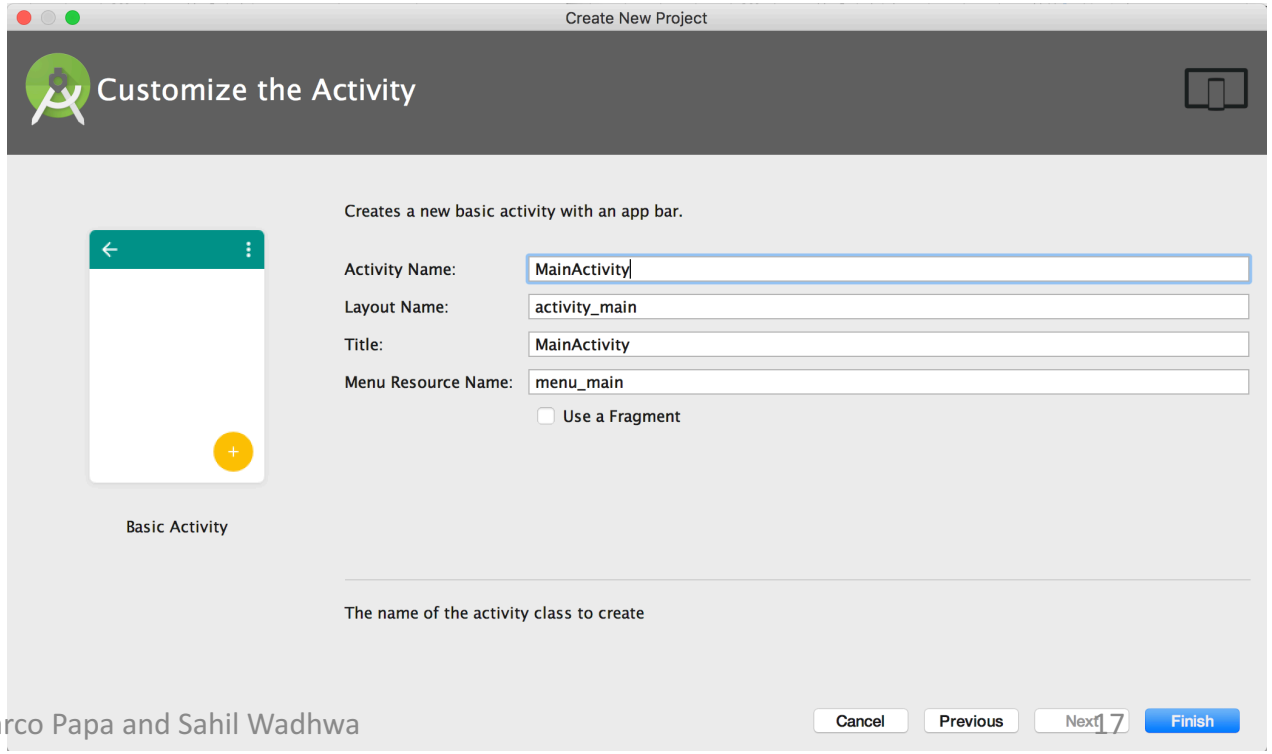
Android Studio - Create a new project



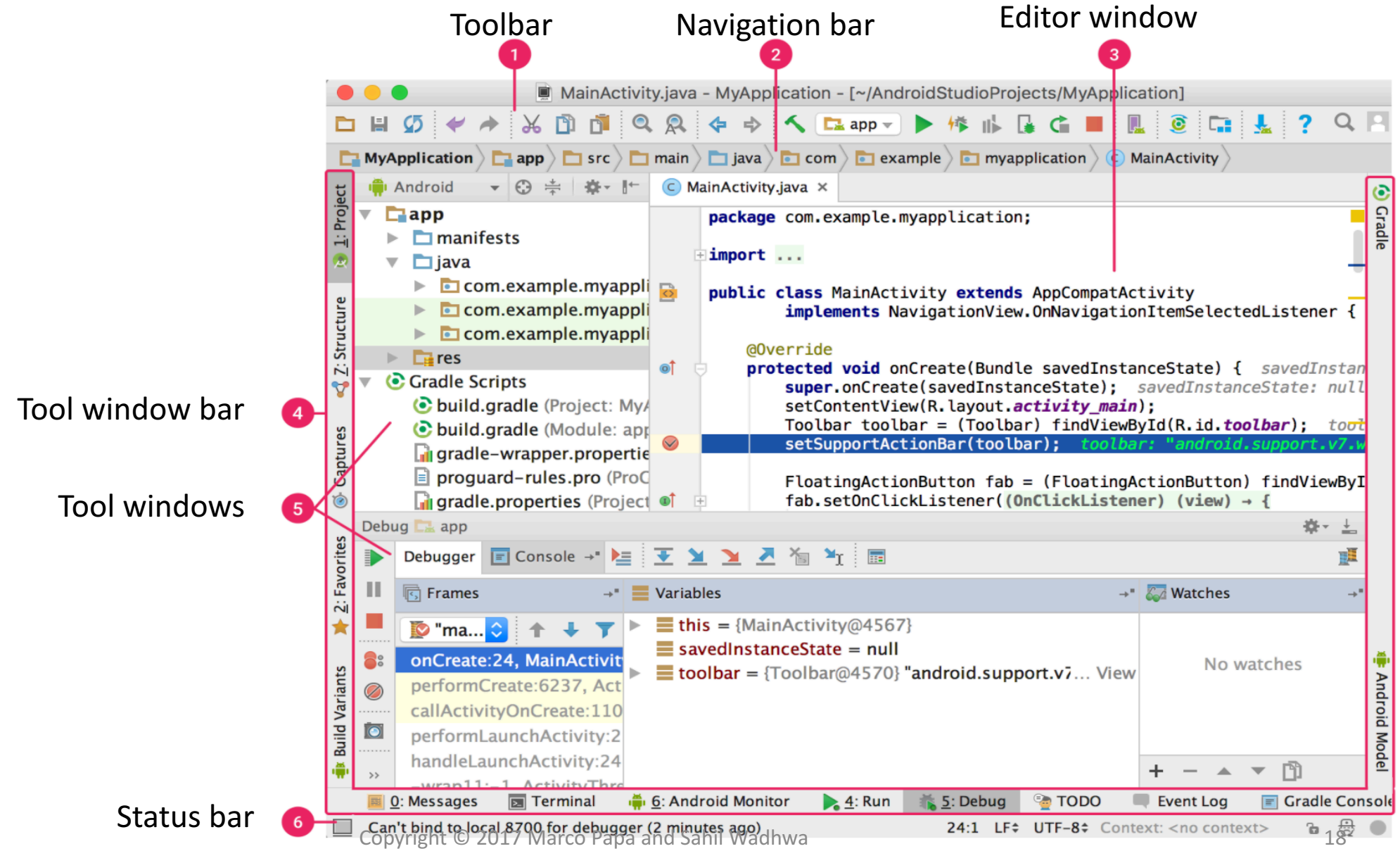


Android Studio - Create a new project (cont'd)





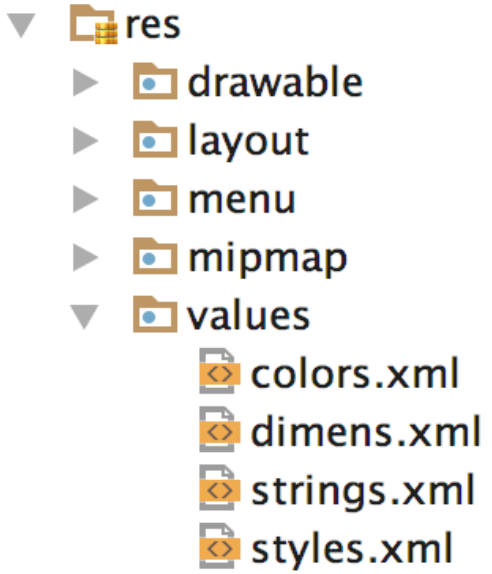
Get familiar with Android Studio



Resources

- Types of resources: anim, color, drawable, layout, menu, raw, values, xml.
- Alternately, you can create a new directory in res/ named in the form **<resources_name>-<config_qualifier>**
- To access *res/drawable/myimage.png* and set an ImageView you will use following code −

```
ImageView imageView = (ImageView)findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```



Activity

- An activity represents a single screen with a user interface.
- Android system initiates its program within an Activity starting with a call to *onCreate()* callback method.

Life-cycle of an Activity:

1. **onCreate():** This is the first callback and called when the activity is first created.
2. **onStart():** This callback is called when the activity becomes visible to the user.
3. **onResume():** This is called when the user starts interacting with the app.

Activity (cont'd)

4. **onPause():** This callback is called when the current activity is being paused and the previous activity is being resumed. During this time, it does not receive user input or execute any code.
5. **onStop():** This callback is called when the activity is no longer visible.
6. **onDestroy():** This callback is called before the activity is destroyed by the system.
7. **onRestart():** This callback is called when the activity restarts after getting stopped.

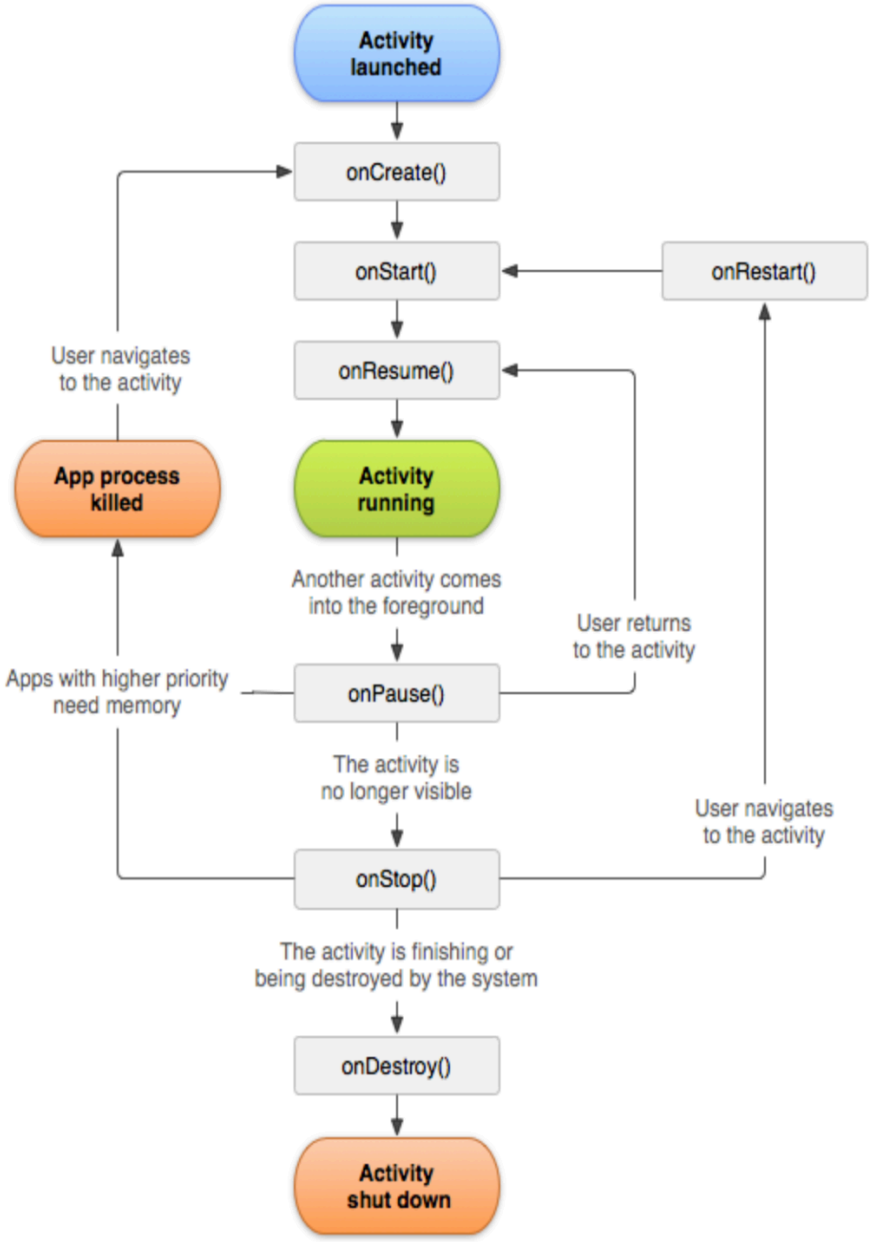


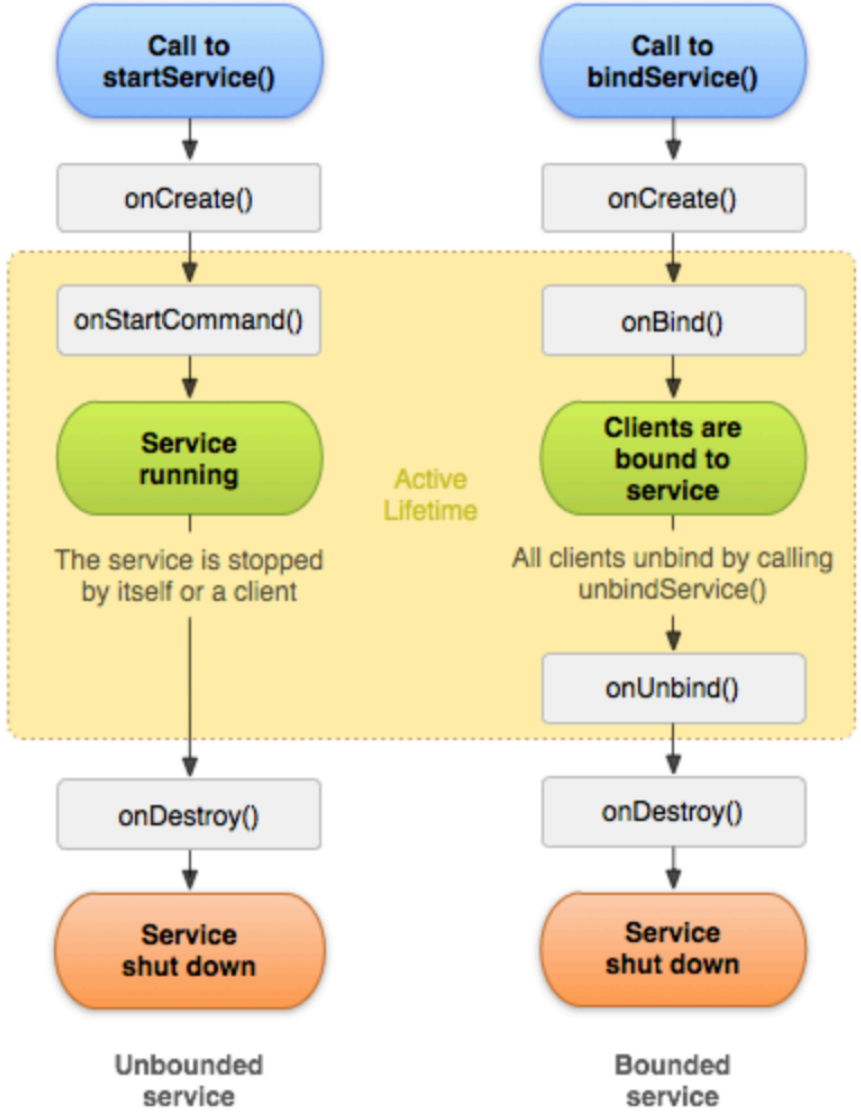
Figure 1. A simplified illustration of the activity lifecycle.

Services

A service runs in the background to perform a task without user interaction and keeps working even if application is destroyed.

A service can essentially take two states −

1. **Started:** A service is started when an activity starts it by calling *startService()*.
2. **Bound:** A service is bound when an application component binds to it by calling *bindService()*.



Broadcast Receivers

- A Broadcast Receiver reacts to the system generated messages or the messages generated from other applications. These messages are called events or intents.
- The following steps are needed to use and respond to events using Broadcast Receiver:
  1. **Creating the Broadcast Receiver**: A receiver can be created in an Activity by extending the abstract class BroadcastReceiver.

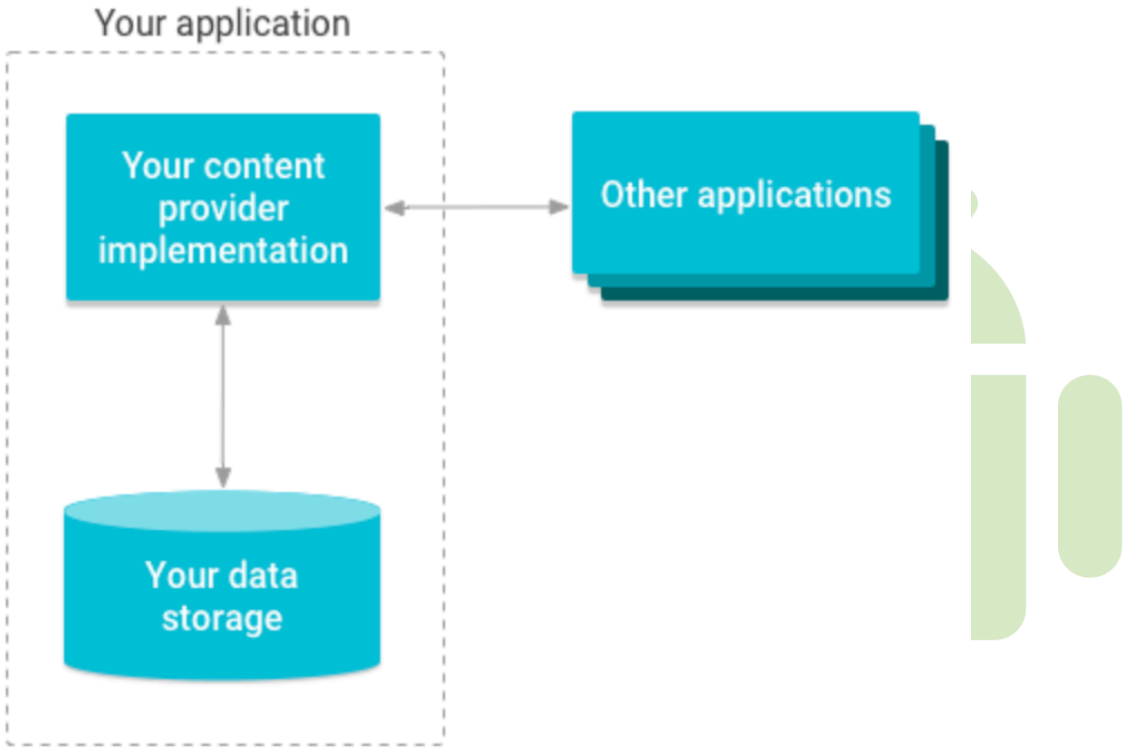
```
public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected (Power Connected)", Toast.LENGTH_LONG).show();
    }
```

  2. **Registering Broadcast Receiver**: A receiver can be registered in Android Manifest file.

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

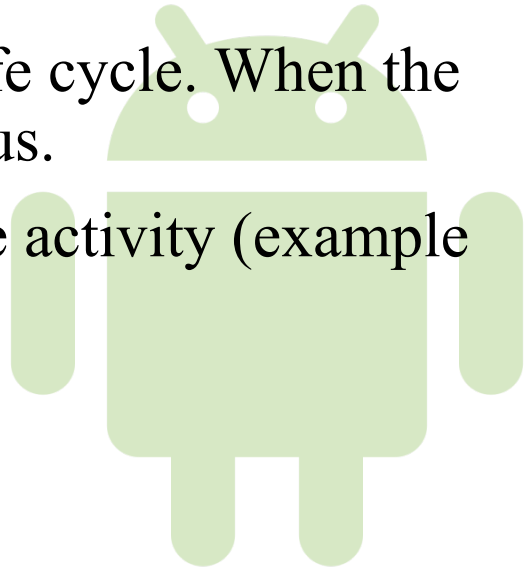
Content Providers

- Often data from one application is needed in another application. A content provider supplies data on demand.
- Using content providers, you can have centralized data and other applications can access it whenever needed. It behaves more like a database which can be queried, updated, inserted into like any other DB.

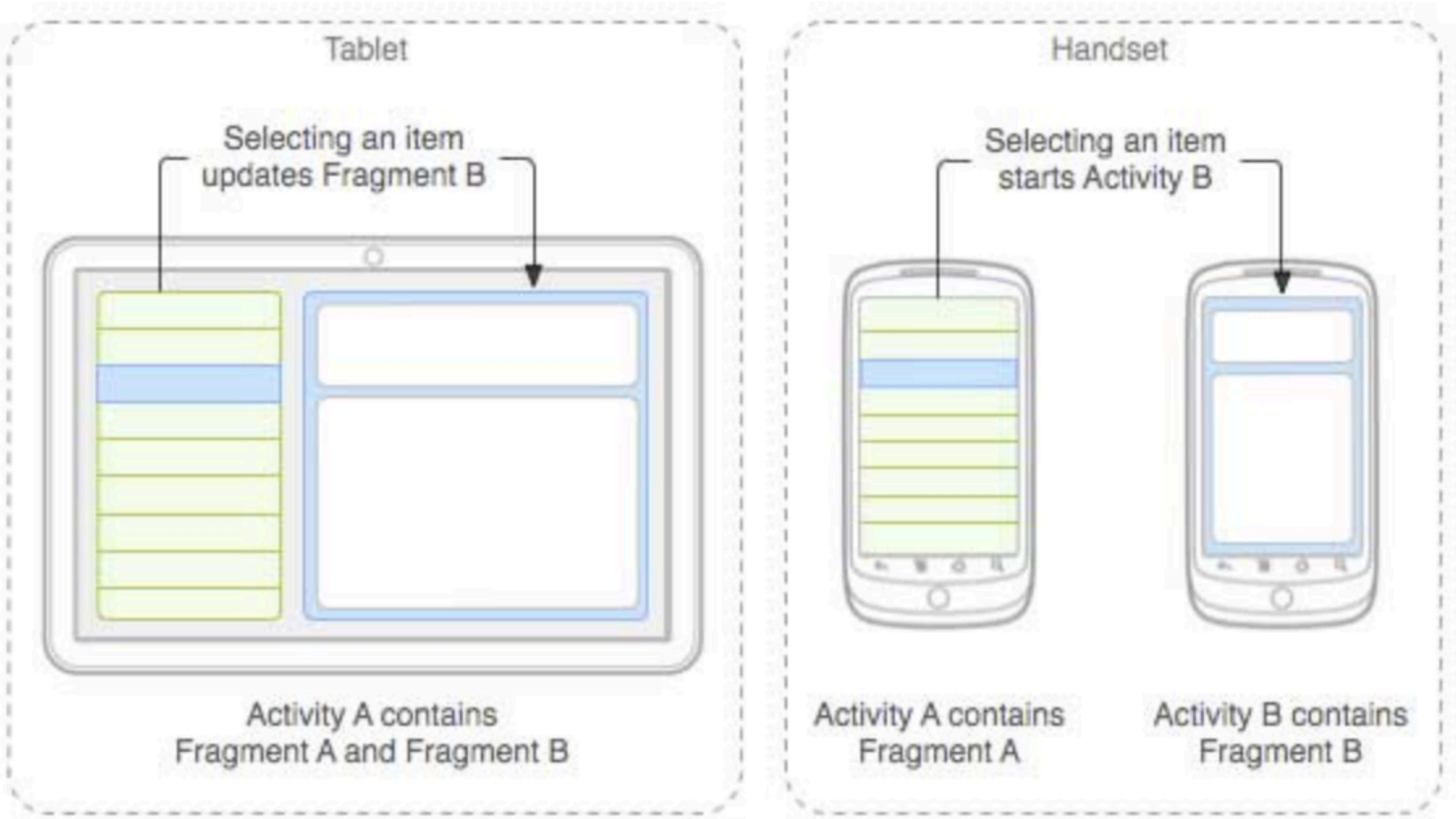


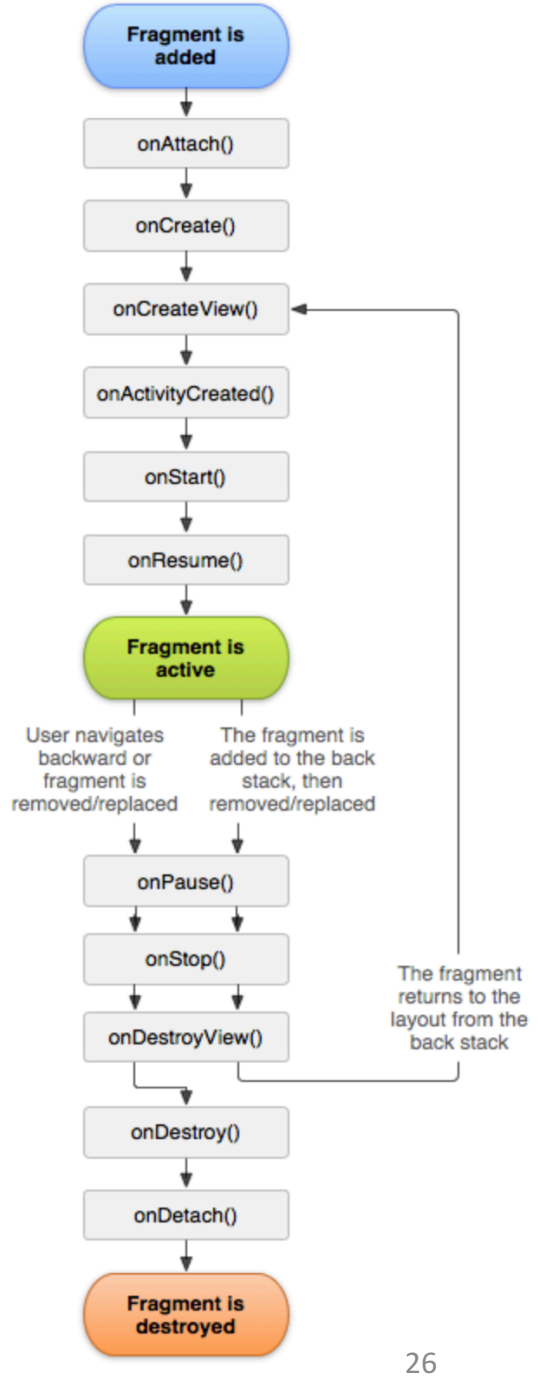
Fragments

- A Fragment is often used for a more modular design- it may or may not be a part of an Activity. It can also be considered as a sub-activity.
- A fragment can run without a user interface as well.
- A fragment has its own layout and its own behavior with its own life cycle callbacks.
- It is possible to dynamically add fragments to a running activity.
- A fragment once created, can be used in multiple activities.
- A fragment's life cycle is closely knitted with its parent activity's life cycle. When the activity is paused or stopped, fragments also replicate the same status.
- Usually tablets have multiple fragments running side by side, in one activity (example on next slide)



Life cycle of a Fragment



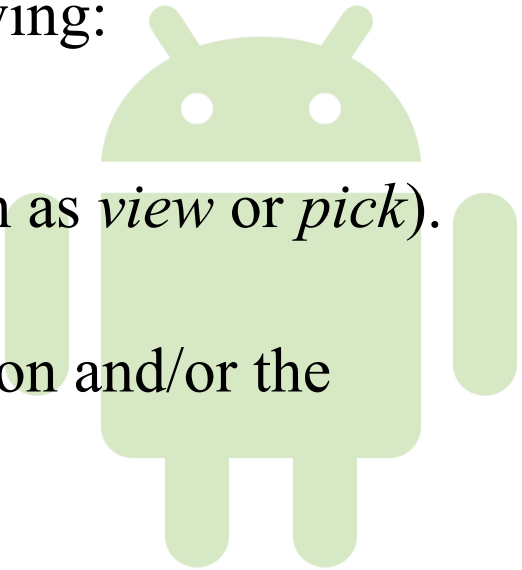


Intents & Intent Filters

- An *Android Intent* is a messaging object you can use to request an action from another app component. The 3 main use cases are:
  - Starting an Activity
  - Starting a Service
  - Delivering a broadcast

The primary information contained in an Intent Object is the following:

- **Component name:** The name of the component to start.
- **Action:** A string that specifies the generic action to perform (such as *view* or *pick*). This is mandatory part of the Intent object.
- **Data**: The URI (a Uri object) that references the data to be acted on and/or the MIME type of that data.

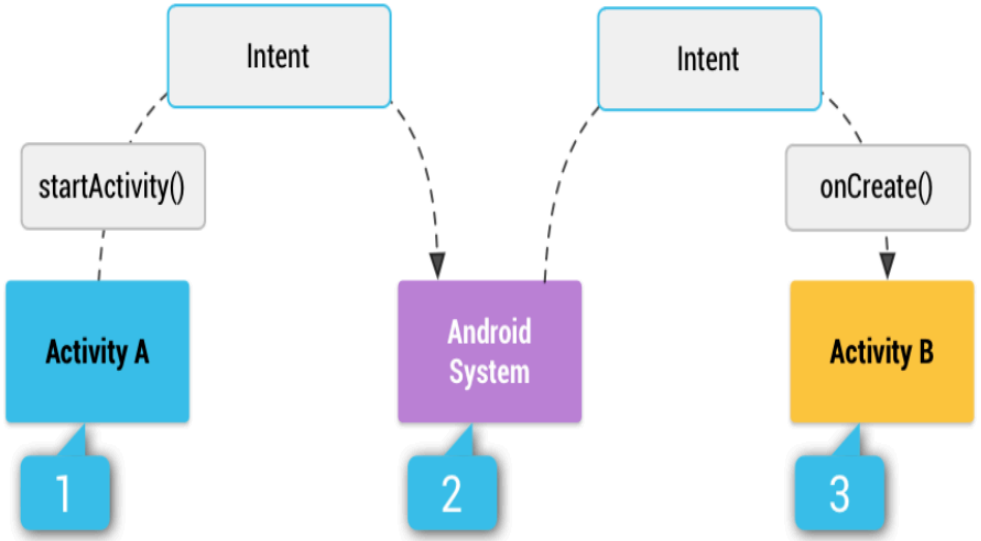


Intents & Intent Filters (cont'd)

- **Category**: A string containing additional information about the kind of component that should handle the intent. The category is an optional part of Intent object.
- **Extras**: This will be in key-value pairs for additional information that should be delivered to the component handling the intent.
- **Flags**: Flags function as metadata for the intent. The flags may instruct the Android system how to launch an activity and how to treat it after it's launched.

Example:

```
Intent i = new Intent(FirstActivity.this,
SecondActivity.class);
startActivity(i);   // Starts SecondActivity
```



UI Layouts

A layout, usually defined in XML, describes the structure and placement of UI components for an Activity. You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Next slide: Example code for defining a TextView and a Button in Linear Layout.

UI Layouts (cont'd)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Loading the XML resource in Java:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

UI Layouts (cont'd)

Common Layouts

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

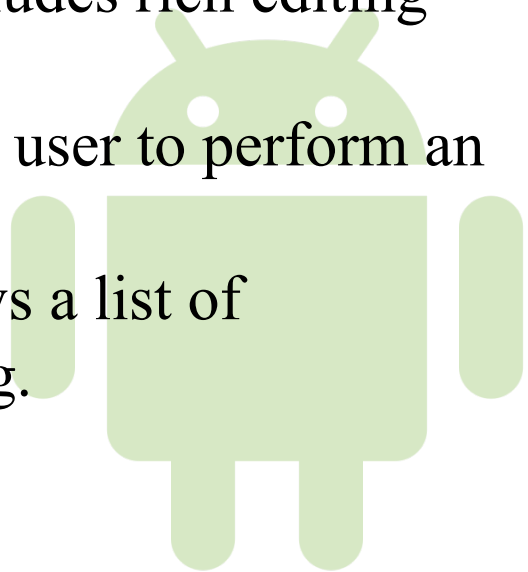
Displays web pages.

Input Controls

Input Controls controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

Common controls include:

1. **TextView:** This control is used to display text to the user.
2. **EditText:** This is a predefined subclass of TextView that includes rich editing capabilities.
3. **Button**: A push-button that can be pressed, or clicked, by the user to perform an action.
4. **AutoCompleteView**: Similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.



Input Controls (cont'd)

5. **ImageButton:** This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6. **CheckBox:** An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7. **RadioButton:** The RadioButton has two states: either checked or unchecked.
8. **ProgressBar:** The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
9. **Spinner:** A drop-down list that allows users to select one value from a set.
10. **TimePicker:** The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
11. **DatePicker:** The DatePicker view enables users to select a date of the day.

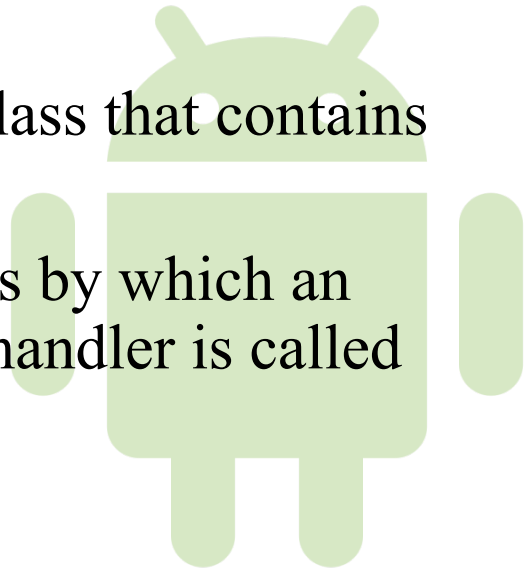
Event Handling

Events are the way a User interacts with the Android system like button presses or screen touch etc.

The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are three concepts related to Android Event Management.

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method.
- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.



Event Handling (cont'd)

- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Two ways of Handling events:

1. Using Event Listeners

```
public class ExampleActivity extends Activity implements OnClickListener {
    protected void onCreate(Bundle savedValues) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
      // do something when the button is clicked
    }
    ...
}
```

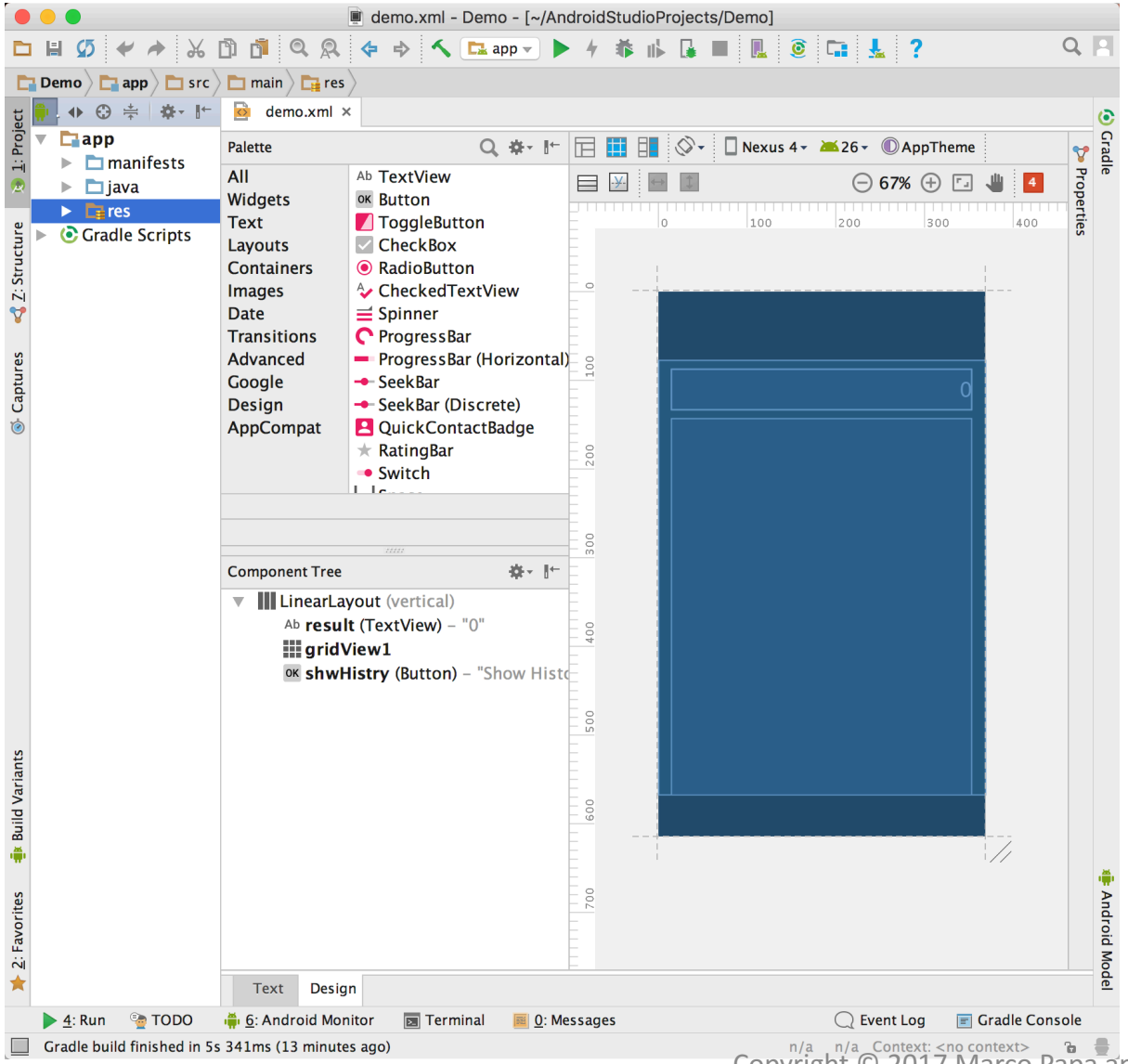
Event Handling (cont'd)

2. Using Event Handler (after declaring method name in the XML)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="android.demo.marco.papa.com.demo.ServiceActivity">

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:layout_marginBottom="20dp"
        android:layout_marginTop="30dp"
        android:background="@android:color/holo_purple"
        android:onClick="startService"
        android:text="Start Services"
        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="0dp" />
```

```
public class ServiceActivity extends AppCompatActivity {

    String msg = "Android : ";
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_service);
        setTitle("Service");
        Log.d(msg, "The onCreate() event");
    }
    public void startService(View view) {
        startService(new Intent(getBaseContext(), MyService.class));
    }
```

Design UI in XML



demo.xml

LinearLayout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/result"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="10dp"
        android:background="@color/peacockBlue"
        android:paddingBottom="6dp"
        android:paddingTop="6dp"
        android:paddingRight="2dp"
        android:text="0"
        android:textAlignment="viewEnd"
        android:textColor="@color/black"
        android:textSize="30sp" />

    <GridView
        android:id="@+id/gridView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numColumns="3"
        android:verticalSpacing="10dp"
        android:horizontalSpacing="10dp"
        android:stretchMode="columnWidth"
        android:layout_marginTop="0dp"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:gravity="center" />

    <Button
        android:id="@+id/shwHistry"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Show History"
        android:textColor="@color/peacockBlue"
        android:background="@color/gray"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="15dp"
        android:layout_marginRight="15dp"
        android:stateListAnimator="@null"
        android:textSize="15sp"
        android:padding="10dp"
        android:onClick="showHistory"/>

</LinearLayout>
```

Control UI & Logic using Activity

setContentView() sets the first UI - activity_main.xml

Steps to produce a Grid UI:

1. Define an ArrayAdapter which would contain a list of items.
2. findViewById(R.id.*gridView1*) is used to connect UI elements with ID – gridView1 with the Java code.
3. Set the adapter in the Grid view using the code gridView.setAdapter(ad);
4. gridView.setOnItemClickListener() is the click listener for the grid items of the calculator.

MainActivity.java

MainActivity | onCreate()

```
27
28  public class MainActivity extends AppCompatActivity
29          implements NavigationView.OnNavigationItemSelectedListener {
30      public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";
31      private GridView gridView;
32      private NavigationView nvDrawer;
33      private Toolbar toolbar;
34      private DrawerLayout mDrawer;
35      private TextView result;
36      boolean hitPlus = false;
37      private int count = 0;
38      int newNum = 0;
39      LinkedHashMap<String, ArrayList<String>> lhm = new LinkedHashMap<>();
40
41      static final String[] numbers = new String[] {
42              "7", "8", "9",
43              "4", "5", "6",
44              "1", "2", "3",
45              "+", "0", "="
46      };
47
48      @Override
49      protected void onCreate(Bundle savedInstanceState) {
50          super.onCreate(savedInstanceState);
51          setContentView(R.layout.activity_main);
52          toolbar = (Toolbar) findViewById(R.id.toolbar);
53          setSupportActionBar(toolbar);
54
55          ArrayAdapter<String> ad = new ArrayAdapter<>(getApplicationContext(),
56                  android.R.layout.simple_list_item_1, numbers) {
57              @Override
58              public View getView(int position, View convertView, ViewGroup parent) {
59                  View view = super.getView(position, convertView, parent);
60                  view.setBackgroundColor(ContextCompat.getColor(getContext(), R.color.gray));
61                  view.setPadding(0, 80, 0, 80);
62                  TextView textView= (TextView) view;
63                  textView.setTextColor(Color.parseColor("#1C99FC"));
64                  textView.setTextSize(25);
65                  textView.setTextAlignment(View.TEXT_ALIGNMENT_CENTER);
66                  return view;
67              }
68          };
69
70          final LinkedHashMap<Integer, Integer> numbers = new LinkedHashMap<Integer, Integer>();
71          numbers.put(count, 0);
72          gridView = (GridView) findViewById(R.id.gridView1);
73          gridView.setAdapter(ad);
74          result = (TextView) findViewById(R.id.result);
75
76          gridView.setOnItemClickListener((parent, v, position, id) → {
```

Control UI & Logic using Activity

Logic for Calculator:

- While consecutive numbers are pressed, treat them as part of a single number by multiplying the term by 10 every time.
- While "+" is clicked, keep updating the UI with the current number.
- When "=" is clicked, it marks the end of an addition. Add up all the numbers and display in the result bar.
- Add all the queries to a HashMap to display it in the History section.
- History is temporary.

MainActivity.java

MainActivity > onCreate()

```
76        gridView.setOnItemClickListener((parent, v, position, id) → {
79                switch(((TextView) v).getText().toString()){
80                    case "=":
81                        int sum = 0;
82                        Set set = numbers.entrySet();
83                        Iterator i = set.iterator();
84                        ArrayList<String> lhmList = new ArrayList<String>();
85                        while(i.hasNext()) {
86                            Map.Entry me = (Map.Entry) i.next();
87                            int value = (int)me.getValue();
88                            sum += value;
89                            lhmList.add(Integer.toString(value));
90                        }
91                        lhmList.add(Integer.toString(sum));
92                        lhm.put(Integer.toString(count), lhmList);
93                        numbers.clear();
94                        result.setText(Integer.toString(sum));
95                        hitPlus = false;
96                        break;
97                    case "0":
98                    case "1":
99                    case "2":
100                   case "3":
101                   case "4":
102                   case "5":
103                   case "6":
104                   case "7":
105                   case "8":
106                   case "9":
107                       if(hitPlus == false && numbers.size() != 0){
108                           int oldValue = numbers.get(count);
109                           newNum = (oldValue * 10) + Integer.parseInt(((TextView) v).getText()
110                                   .toString());
111                           numbers.put(count, newNum);
112                       } else {
113                           newNum = Integer.parseInt(((TextView) v).getText().toString());
114                           numbers.put(++count, newNum);
115                       }
116                       result.setText(Integer.toString(newNum));
117                       hitPlus = false;
118                       break;
119                   case "+":
120                       hitPlus = true;
121                       result.setText("");
122                       Toast.makeText(getApplicationContext(),
123                               "+", Toast.LENGTH_SHORT).show();
124                       break;
125               }
126       });
128
129       mDrawer = (DrawerLayout) findViewById(R.id.drawer_layout);
130       ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
131               this, mDrawer, toolbar, "Open navigation drawer", "Close navigation drawer");
132       mDrawer.setDrawerListener(toggle);
133       toggle.syncState();
134
```

Change Activity properties in Manifest file

1. Every Activity that is created must be defined in the Manifest file under application tag.
2. The **action** for the intent filter indicates that this activity serves as the entry point for the application.
3. The **category** for the intent-filter indicates that the application can be launched from the device's launcher icon.
4. Parent activity can be shown for the up arrow navigation.

AndroidManifest.xml

manifest | application

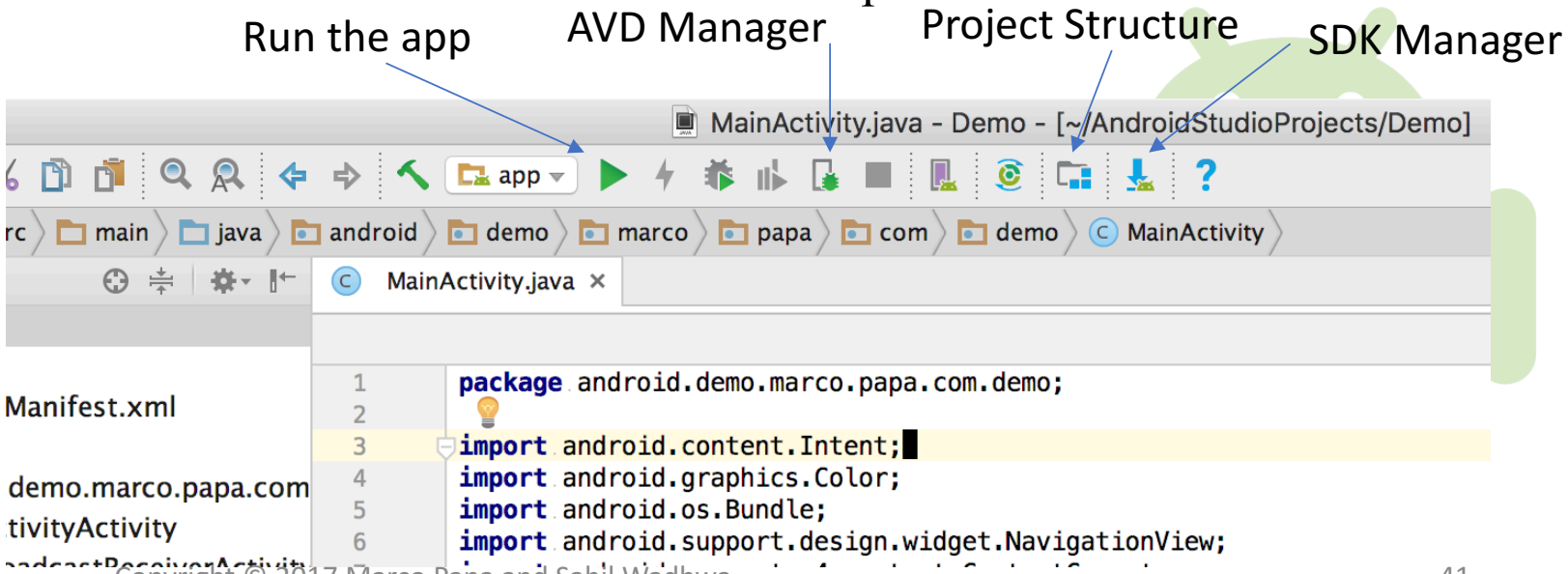
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.demo.marco.papa.com.demo">

    <uses-permission android:name="android.permission.INTERNET">
    </uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Calculator Demo"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="Calculator Demo"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".DisplayHistoryActivity"
            android:parentActivityName=".MainActivity">

            <!-- The meta-data tag is required if you support API level 15 and lower -->
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity" />
        </activity>
```

Run your app in the Emulator

To run your app in your own Android device, tap on Build Number from Settings several times. This will enable developer options in your phone. Then enable USB debugging.

To run your app in the Emulator, make sure you have correct SDK & tools installed. Verify using the following steps:

1. Click on the Project Structure icon and then click on the “app” section under modules.
    1. Under properties, check the Compile SDK Version and Build Tools Version. These should be same.
    2. Under Flavors, Min Sdk Version should be lower than the Compile SDK Version.



Run your app in the Emulator (cont'd)

2. Click on SDK Manager icon and then Appearance & Behavior->Android SDK from the left menu.
    1. Under SDK Platforms, make sure the Compile Android version you selected in the last step is installed.
    2. Under SDK tools, make sure Android Emulator, Android SDK Platform Tools, Android SDK Tools & Intel x86 Emulator Accelerator are all installed.
3. Click on AVD Manager and then click on Create Virtual Device to fire up a new Emulator.
    1. Under Phone category, click on the profile (mobile type) that you wish to start. Click Next.
    2. Under Recommended section, make sure the Release is already downloaded. If not, download it. Click Next.
    3. Check the AVD Name. That will be the name of the device shown every time you start Emulator. Click Finish.
    4. Now you will see the same device under Your Virtual Devices. Click on the green play button on the right to start the Emulator.

Volley- HTTP Networking and JSON Parsing

- Volley is an HTTP library that makes networking for Android apps easier and most importantly, faster. Volley is available on GitHub.

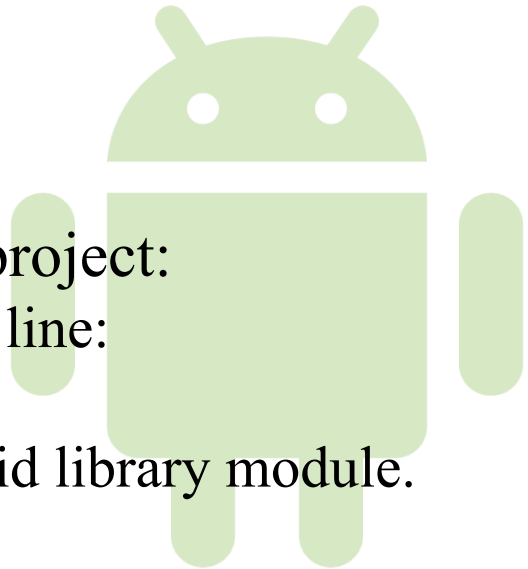
  https://github.com/google/volley

Two ways to include Volley in your project:

1. The easiest way to add Volley to your project is to add the following dependency to your app's build.gradle file:

```
dependencies {
    ...
    compile 'com.android.volley:volley:1.0.0'
}
```

2. You can also clone the Volley repository and set it as a library project:
   1. Git clone the repository by typing the following at the command line:
      git clone https://github.com/google/volley
   2. Import the downloaded source into your app project as an Android library module.



Volley- HTTP Networking and JSON Parsing (cont'd)

To use Volley, you must add the android.permission.INTERNET permission to your app's manifest. Without this, your app won't be able to connect to the network.

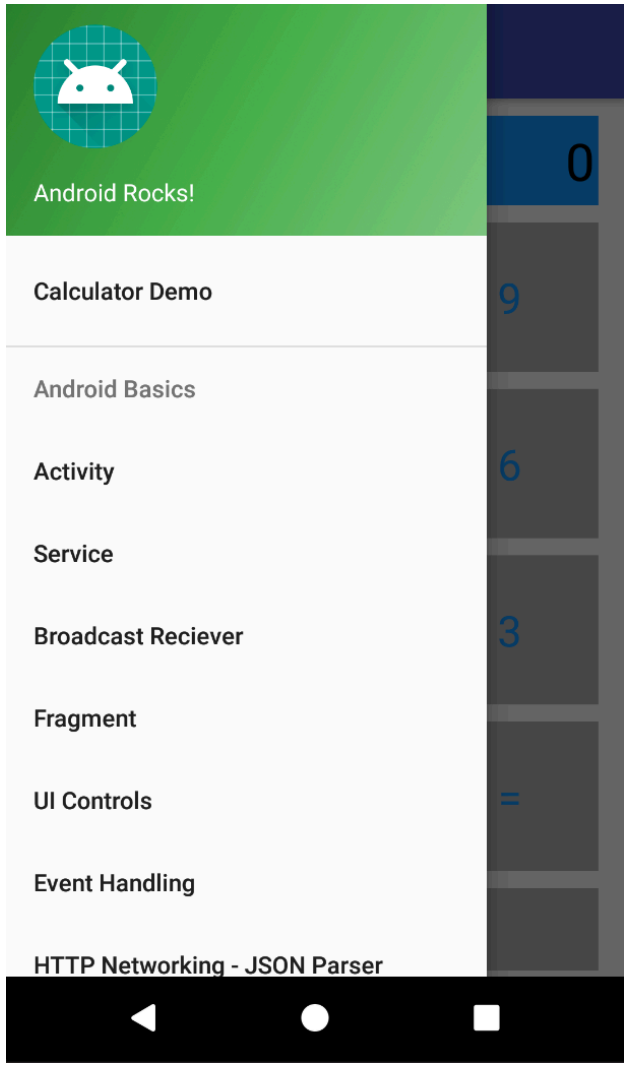
manifest

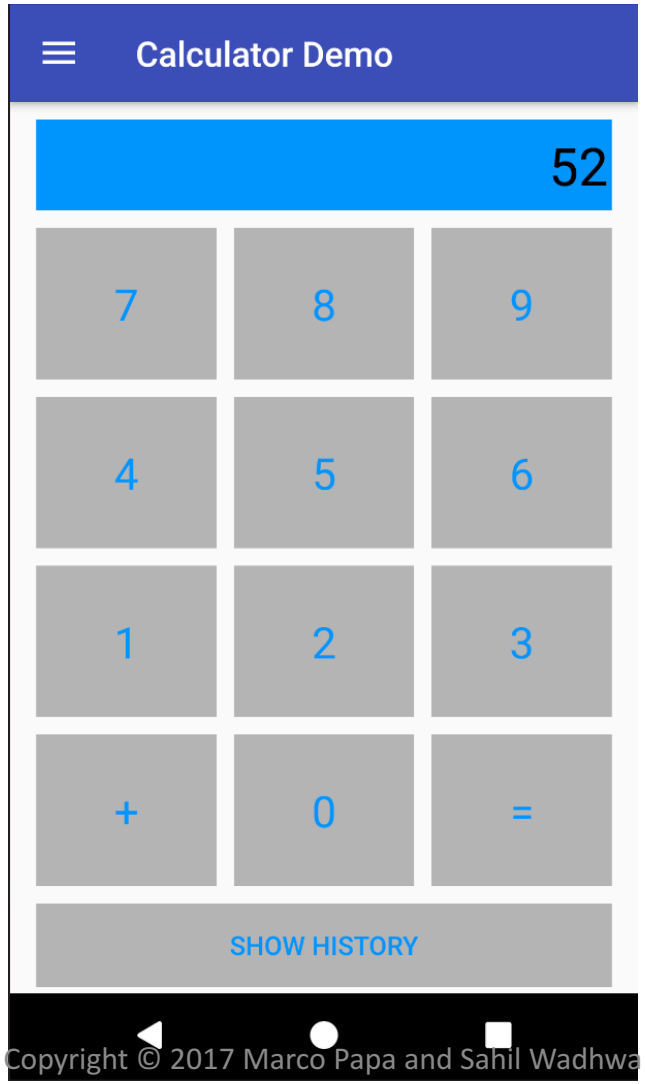
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.demo.marco.papa.com.demo">

    <uses-permission android:name="android.permission.INTERNET">
    </uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

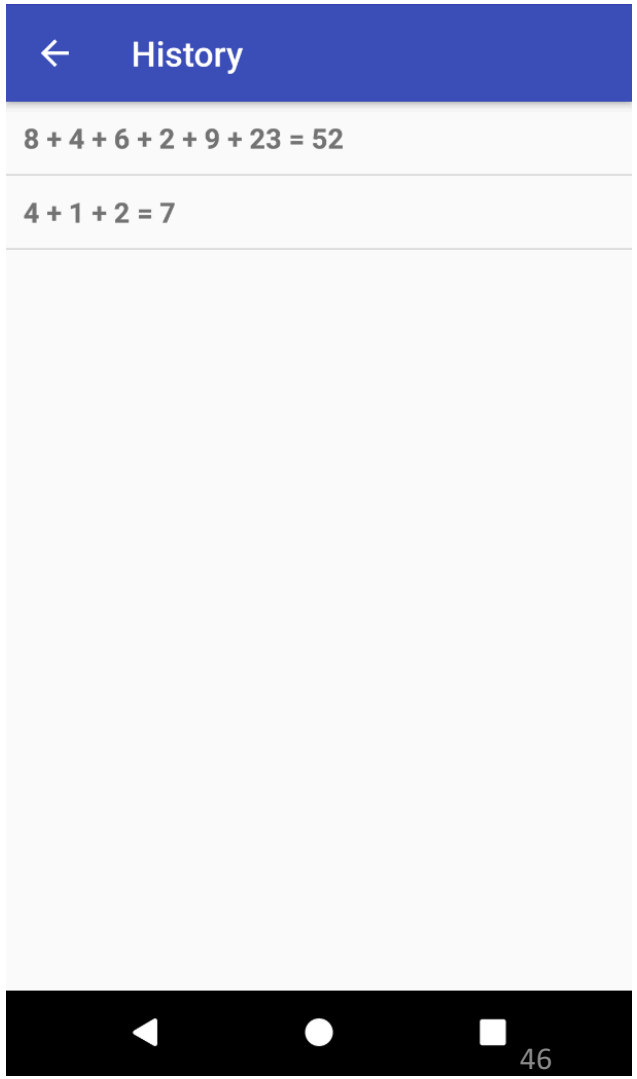
Volley at work

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_http);
        setTitle("HTTP Networking & JSON Parser");

        textView = (TextView) findViewById(R.id.volleyText);
        responseName = (TextView) findViewById(R.id.responseName);
        requestQueue = Volley.newRequestQueue(this); // 'this' is the Context
        String url = "https://jsonplaceholder.typicode.com/users";

        JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET, url, null,
                new Response.Listener<JSONArray>() {
                    @Override
                    public void onResponse(JSONArray response) {
                        textView.setText("Trimmed response: " + response.toString());
                        Toast.makeText(getApplicationContext(), response.toString(), Toast.LENGTH_SHORT);
                        StringBuilder names = new StringBuilder();
                        names.append("Parsed names from the response: ");
                        try {
                            for(int i = 0; i < response.length(); i++){
                                JSONObject jresponse = response.getJSONObject(i);
                                String name = jresponse.getString("name");
                                names.append(name).append(", ");
                                Log.d("Name", name);
                            }
                            names.deleteCharAt(names.length() -2);
                            responseName.setText(names.toString());
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    }
                },
                new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                        Toast.makeText(getApplicationContext(), "Nothing found!", Toast.LENGTH_SHORT);
                    }
                });
        //add request to queue
        requestQueue.add(jsonArrayRequest);
    }
}
```

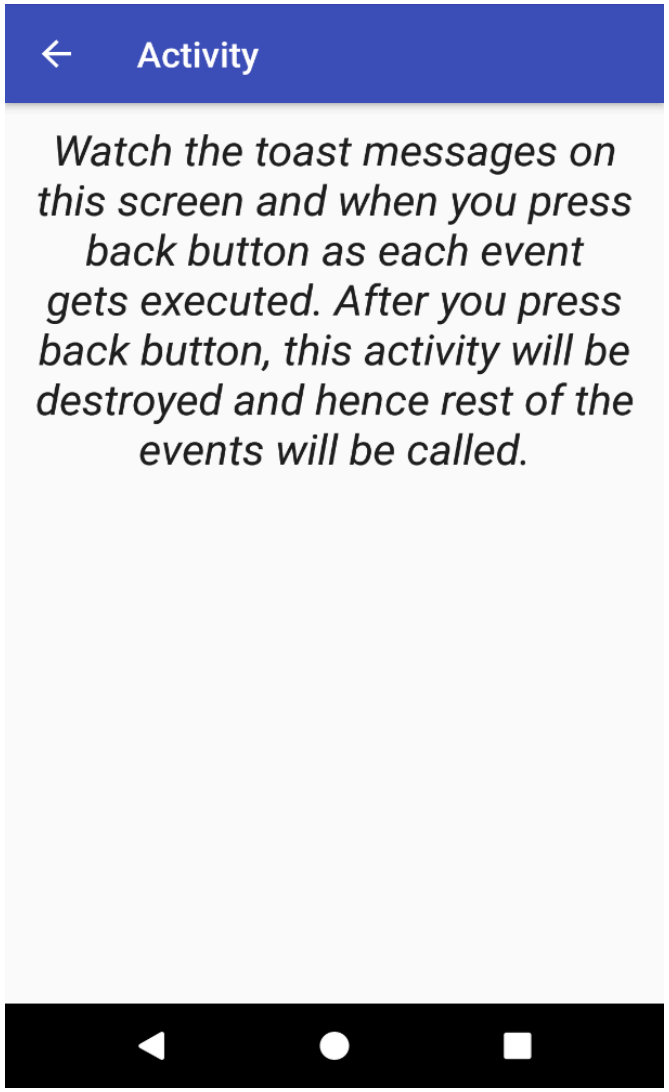
Calculator Demo

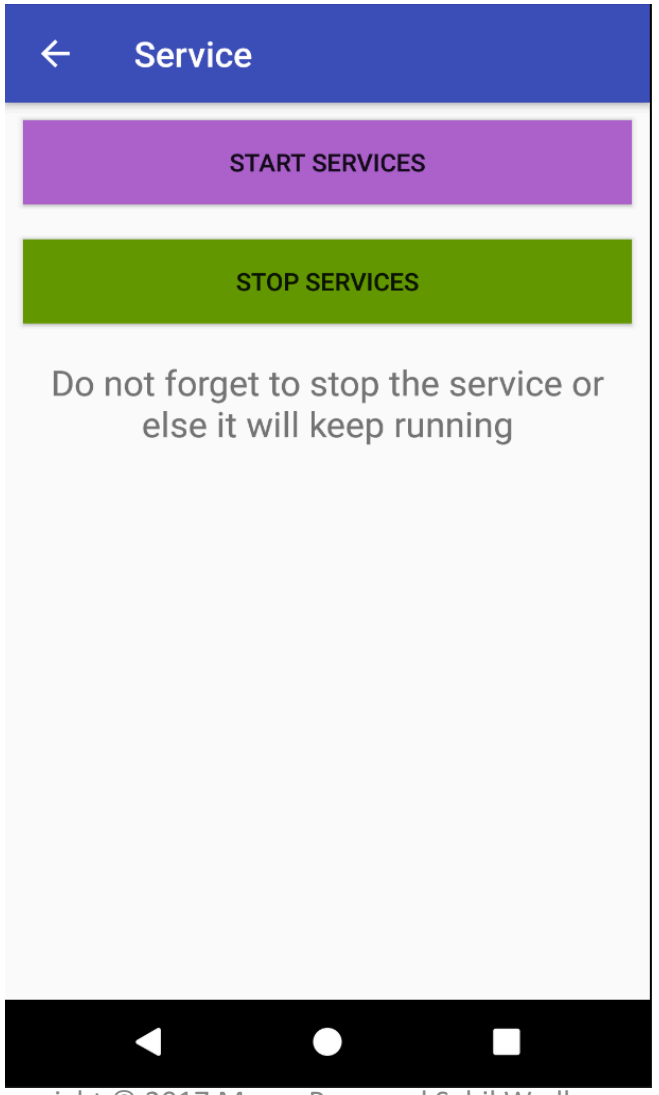




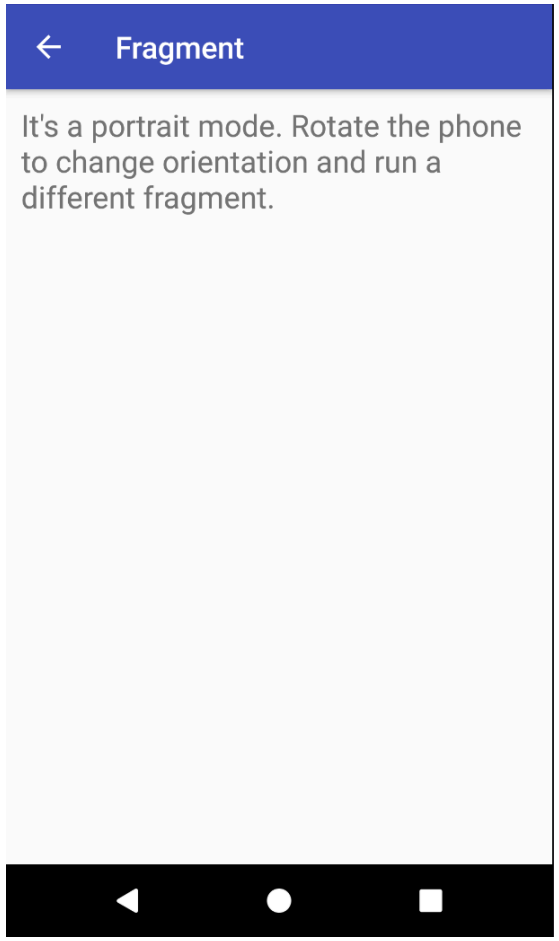


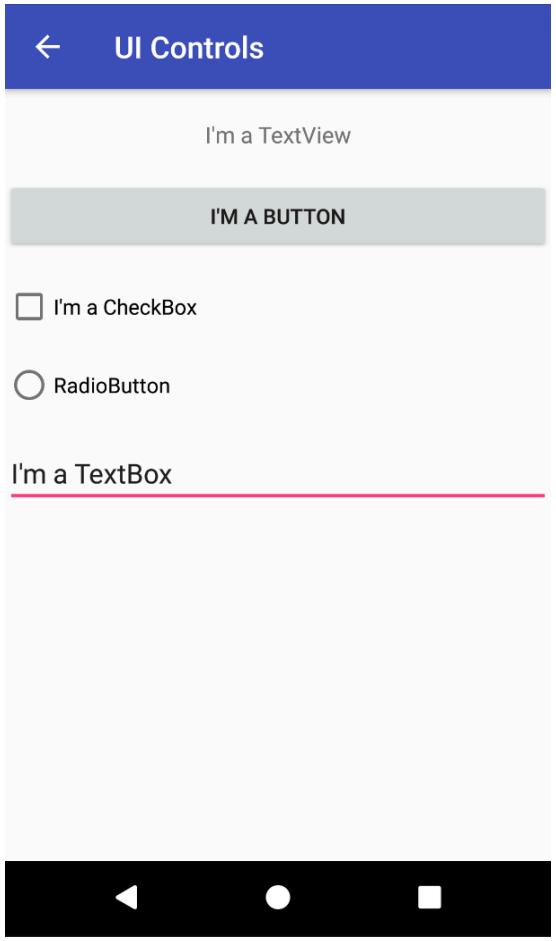
Demo: Activity, Service, Broadcast Receiver

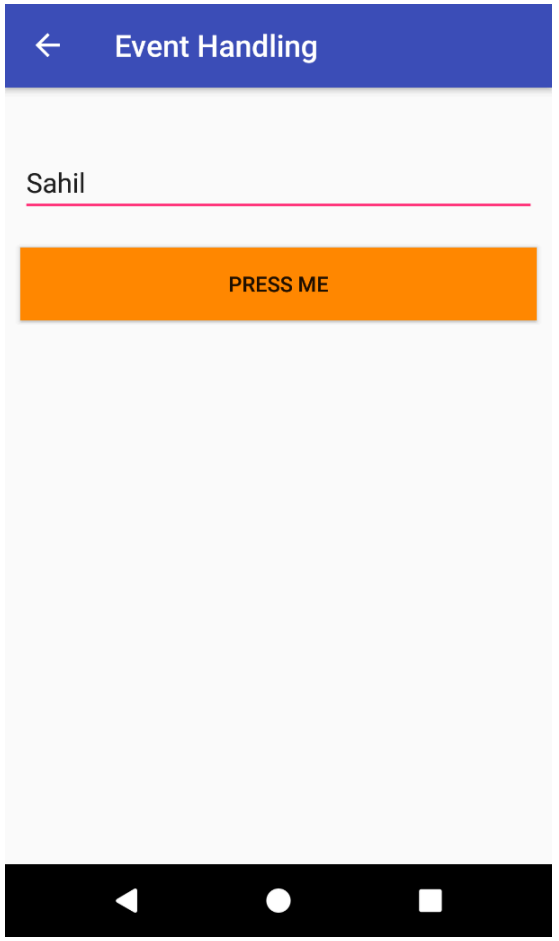


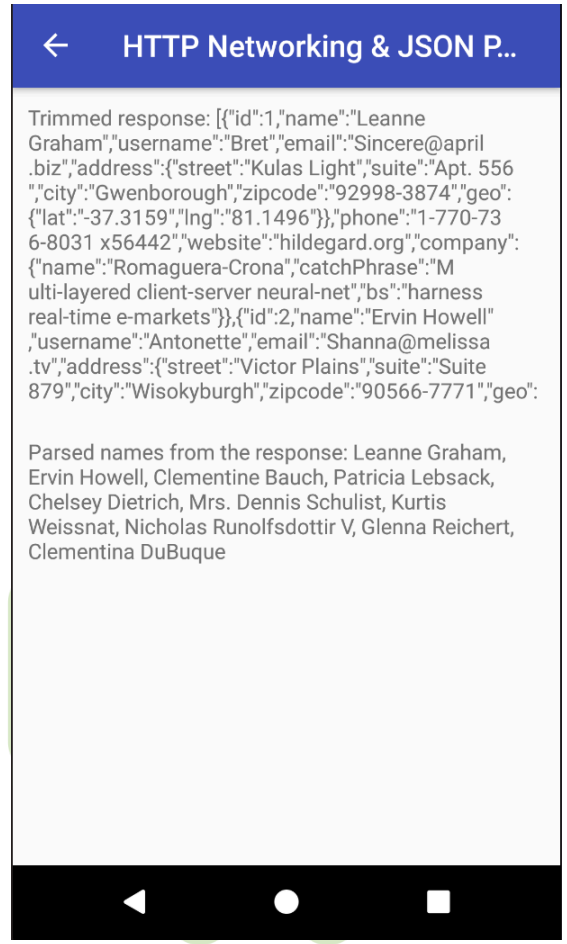


Demo: Fragment, UI controls, Event Handling, HTTP Networking & JSON Parser









References

- https://developer.android.com/studio/intro/index.html
- https://www.tutorialspoint.com/android
- https://www.javatpoint.com/android-tutorial
- http://socialcompare.com/en/comparison/android-versions-comparison