What is Android ?

Android is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets.

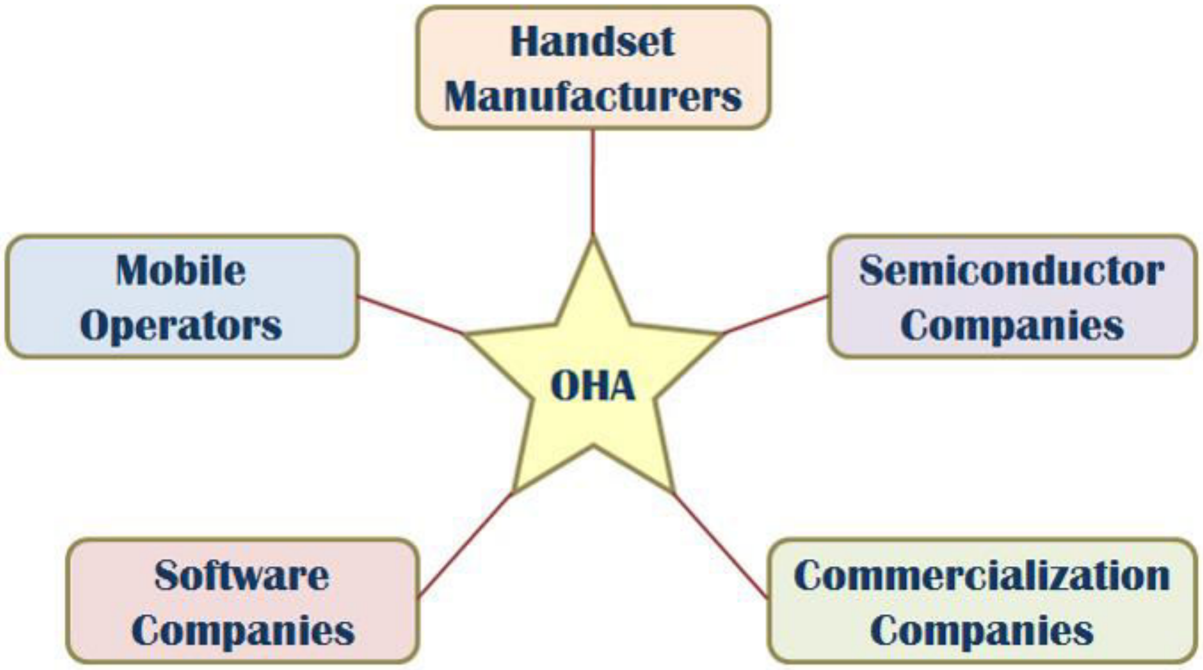
History

Android, Inc. was founded in Palo Alto, California in October 2003 by Andy Rubin (co-founder of Danger), Rich Miner (co-founder of Wildfire Communications, Inc.), Nick Sears (once VP at T-Mobile), and Chris White (headed design and interface development at WebTV)

In July 2005, Google acquired Android Inc. for at least $50 million

OHA (Open Handset Alliance)

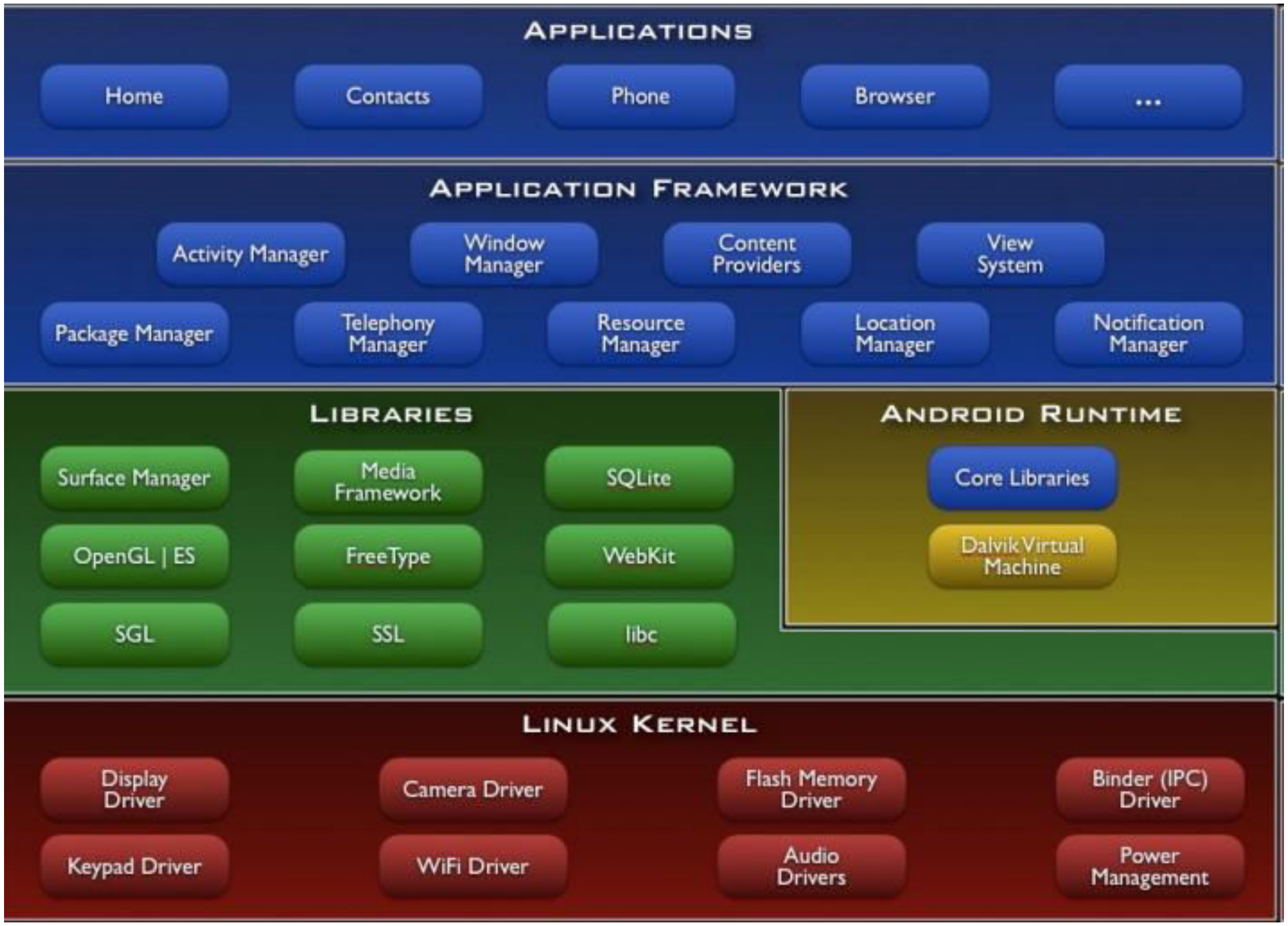
- A business alliance consisting of 47 companies to develop open standards for mobile devices



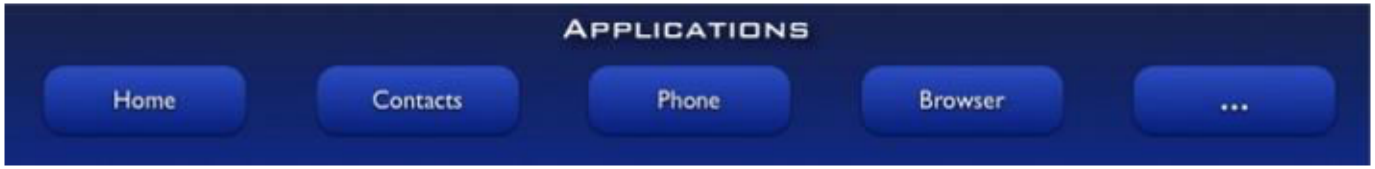
Why Android ?

- The world’s most powerful mobile platform?
- Take Google with you
- Navigate your world
- Connect and share
- Be Entertained
- Create & Collaborate
- Powerful, Simple and Beautiful.

Architecture



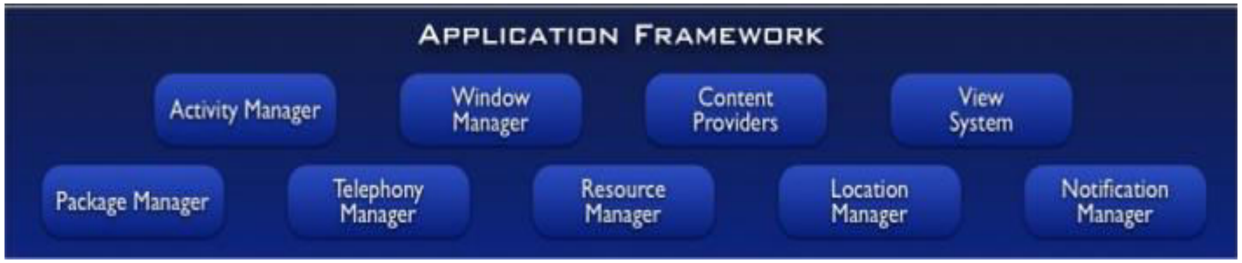
Android S/W Stack - Application



- Android provides a set of core applications:
  - ✓ Email Client
  - ✓ SMS Program
  - ✓ Calendar
  - ✓ Maps
  - ✓ Browser
  - ✓ Contacts
  - ✓ Etc

- All applications are written using the Java language.

Android S/W Stack – App Framework

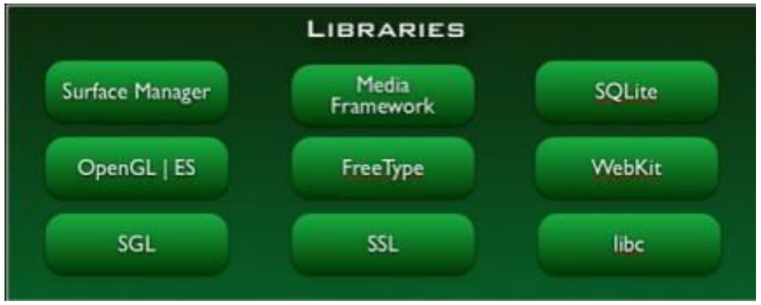


- Enabling and simplifying the reuse of components
  - ✓ Developers have full access to the same framework APIs used by the core applications.
  - ✓ Users are allowed to replace components.

Android S/W Stack – App Framework (Cont)

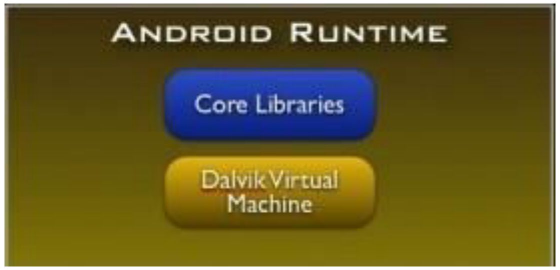
Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized strings, graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation backstack

Android S/W Stack - Libraries



- Including a set of C/C++ libraries used by components of the Android system
- Exposed to developers through the Android application framework

Android S/W Stack - Runtime



- Core Libraries
  - ✓ Providing most of the functionality available in the core libraries of the Java language
  - ✓ APIs
    - Data Structures
    - Utilities
    - File Access
    - Network Access
    - Graphics
    - Etc

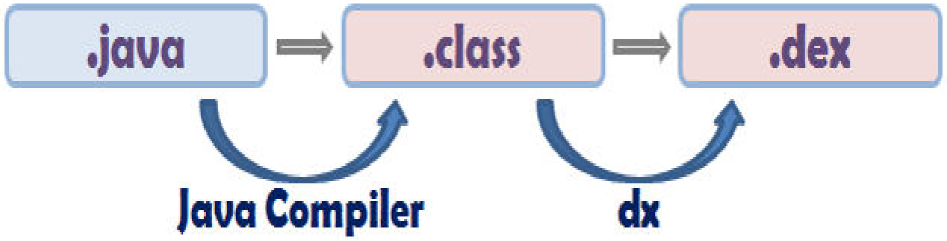
Android S/W Stack - Runtime

ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine

- Dalvik Virtual Machine
  - ✓ Providing environment on which every Android application runs
    - Each Android application runs in its own process, with its own instance of the Dalvik VM.
    - Dalvik has been written such that a device can run multiple VMs efficiently.
  - ✓ Register-based virtual machine

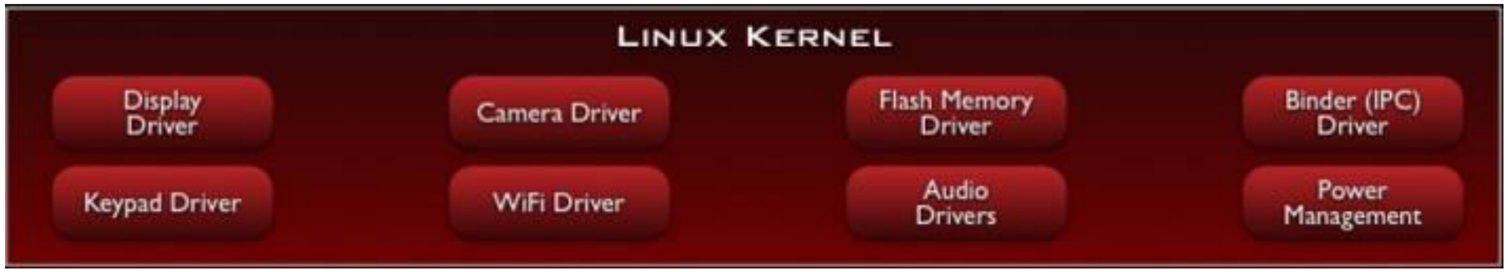
Android S/W Stack – Runtime (Cont)

- Dalvik Virtual Machine (Cont)
  - ✓ Executing the Dalvik Executable (.dex) format
    - .dex format is optimized for minimal memory footprint.
    - Compilation



  - ✓ Relying on the Linux Kernel for:
    - Threading
    - Low-level memory management

Android S/W Stack – Linux Kernel



- Relying on Linux Kernel 2.6 for core system services
  - ✓ Memory and Process Management
  - ✓ Network Stack
  - ✓ Driver Model
  - ✓ Security
- Providing an abstraction layer between the H/W and the rest of the S /W stack

Application Building Blocks

- Activity
- IntentReceiver
- Service
- ContentProvider

Activities

- Typically correspond to one UI screen
- But, they can:
  - Be faceless
  - Be in a floating window
  - Return a value

Intents

- Think of Intents as a verb and object; a description of what you want done
  - E.g. VIEW, CALL, PLAY etc..
- System matches Intent with Activity that can best provide the service
- Activities and IntentReceivers describe what Intents they can service

Services

- Faceless components that run in the background
  - E.g. music player, network download etc…

ContentProviders

- Enables sharing of data across applications
  - E.g. address book, photo gallery
- Provides uniform APIs for:
  - querying
  - delete, update and insert.
- Content is represented by URI and MIME type

Installation

- Android Studio
- Stand-Alone SDK Tools

Installation(Cont)

- Android Studio - Android Studio provides everything you need to start developing apps for Android, including the Android Studio IDE and the Android SDK tools.

Reference :

http://developer.android.com/sdk/index.html

Installation(Cont)

- Stand-Alone SDK Tools- The stand-alone SDK Tools package does not include a complete Android development environment. It includes only the core SDK tools, which you can access from a command line or with a plugin for your favorite IDE (if available)..

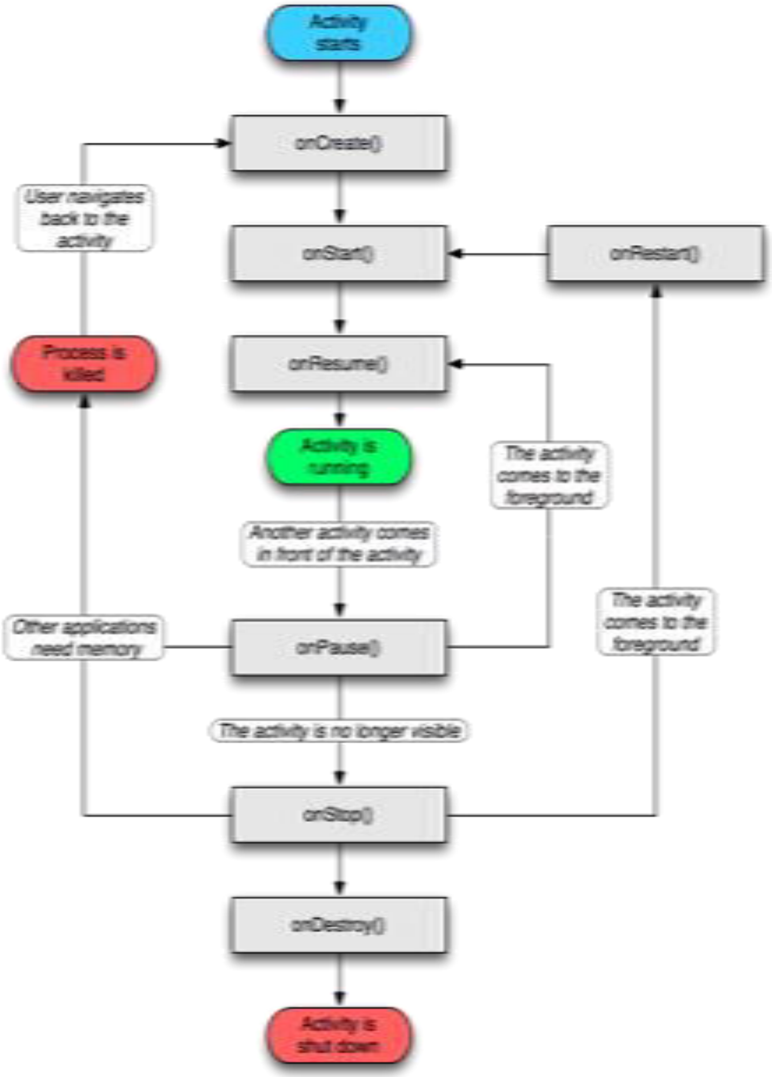
Reference :

http://developer.android.com/sdk/index.html#Other

Application Lifecycle

- Application run in their own processes (VM, PID)
- Processes are started and stopped as needed to run an application's components
- Processes may be killed to reclaim resources

Application Lifecycle (Cont)


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

UI : Two Alternatives Code or XML

- You have two ways you can create the interface(s) of your Application.

1. Code = write code using SDK with classes like LinearLayout, TextView, ……

2. XML = create XML files in res/Layout (i.e. main.xml) that contain Android XML view tags like <LinearLayout> <TextView>, etc.

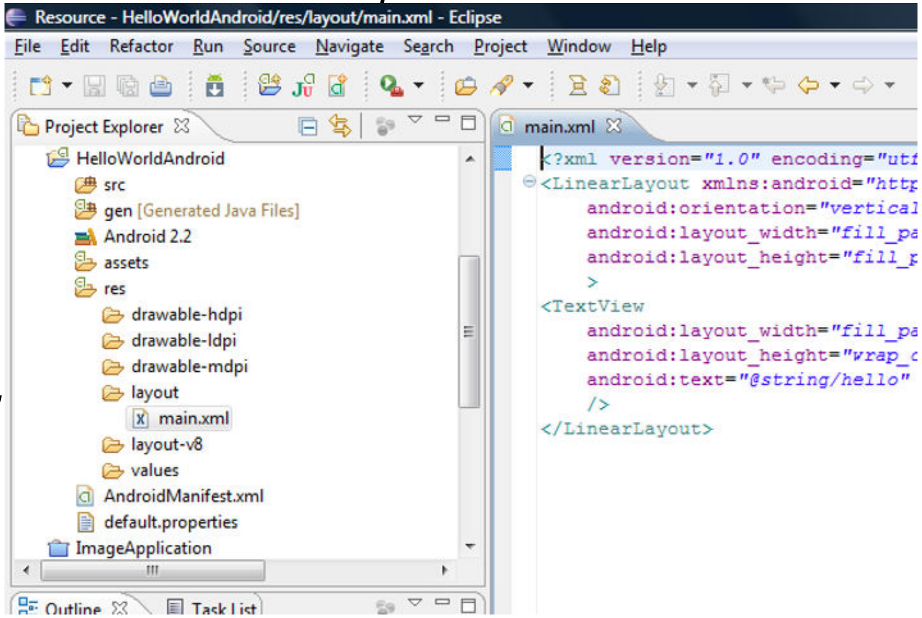
XML Interface Creation

- Generally, I would say if it is possible, doing XML would be better as it means a decoupling of design from Java code.
- You can have both in your system….
- Lets discuss this first.

The Layout --- the interface

- res/layout/main.xml = contains layout for interface

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



```
<?xml version="1.0" encoding="utf
<LinearLayout xmlns:android="http
    android:orientation="vertical
    android:layout_width="fill_pa
    android:layout_height="fill_p
    >
<TextView
    android:layout_width="fill_pa
    android:layout_height="wrap_c
    android:text="@string/hello"
    />
</LinearLayout>
```

The above will create an interface in vertical (versus portrait) mode that fills the parent

Both in width and write and wraps and content as necessary.

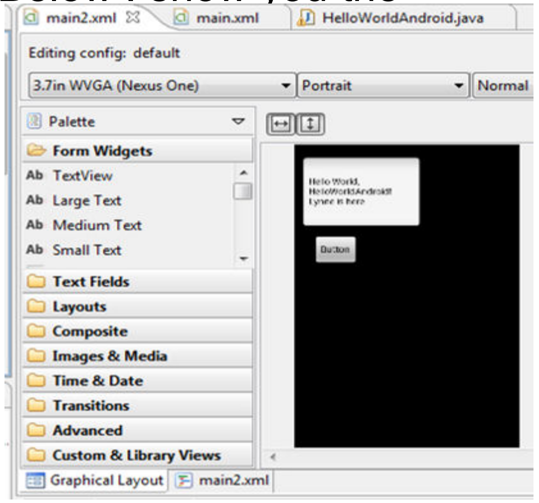
- .

XML interface

-
  ```
  <TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:text="@string/hello"/>
  ```
  - xmlns:android XML namespace declaration that tells the Android tools that you are going to refer to common attributes defined in the Android namespace. The outermost tag in every Android layout file must have this attribute.
  - android:layout_width This attribute defines how much of the available width on the screen this View should consume. As it's the only View so you want it to take up the entire screen, which is what a value of "fill_parent" means. android:layout_height This is just like android:layout_width, except that it refers to available screen height.
  - android:text This sets the text that the TextView should display. In this example, you use a string resource instead of a hard-coded string value. The *hello* string is defined in the *res/values/strings.xml* file.
- .

Visually Creating XML interface

- I dragged and dropped an EditText view and a Button. Below I show you the corresponding code.



res/layout/main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
    <EditText android:text="@string/hello" android:id="@+id/editText1" android:inputType="textMultiLine"
      android:layout_width="169dp" android:layout_height="115dp" android:layout_x="11dp"
      android:layout_y="20dp"></EditText>
    <Button android:id="@+id/button1" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Button" android:layout_x="27dp"
      android:layout_y="146dp"></Button>

</AbsoluteLayout>
```

- .

Each View or ViewGroup can have its own set of attributes…but, some are very common

Attribute	Description
layout_width	specifies width of View or ViewGroup
layout_height	specifies height
layout_marginTop	extra space on top
layout_marginBottom	extra space on bottom side
layout_marginLeft	extra space on left side
layout_marginRight	extra space on right side
layout_gravity	how child views are positioned
layout_weight	how much extra space in layout should be allocated to View (only when in LinearLayout or TableView)
layout_x	x-coordinate
layout_y	y-coordinate

SQLITE

- SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

- SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

SQLITE(Cont)

- Database – Package - The main package is android.database.sqlite that contains the classes to manage your own databases
- Database – Creation - In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own

```
SQLiteDatabse mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

SQLITE(Cont)

- Database – Insertion

we can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS COMPANY(Username VARCHAR,Password
VARCHAR);");


mydatabase.execSQL("INSERT INTO COMPANY VALUES('admin','admin');");
```

SQLITE(Cont)

- Database - Fetching
- We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from COMAPANY",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

Creating a Menu

Two methods (again):

- XML

  Place a file inside res/menu/

  Inflate the menu inside the Activity

  Useful if you want to create the same menu inside different activities

- Java

  Create the menu directly inside the activity

Menu(Cont)

- Create res/menu/menu.xml
- We need:
  - IDs of menu’s elements
  - Title of each element
  - Icon of each element
- Inside the Activity, create onCreateOptionsMenu()
  - Inflate the menu
  - Add functionality to the buttons

Menu(Cont)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="First Option"></item>
    <item android:id="@+id/item2" android:title="Second Option">
        <menu>
            <item android:id="@+id/item3" android:title="Third Option"/>
            <item android:id="@+id/item4" android:title="Fourth Option"/>
        </menu>
    </item>
</menu>
```

Menu Inflation(Cont)

```
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);

    getMenuInflater().inflate(R.menu.myMenu, menu);

    menu.findItem(R.id.menu_first).setIntent(new Intent(this, First.class));

    return true;
}
```

Toast

Tiny messages over the Activity

Used to signal to the user confirmation, little errors

Can control the duration of the Toast

As simple as:

```
Toast msg = Toast.makeText(this, “Toast!”,
Toast.LENGTH_SHORT).show();
```

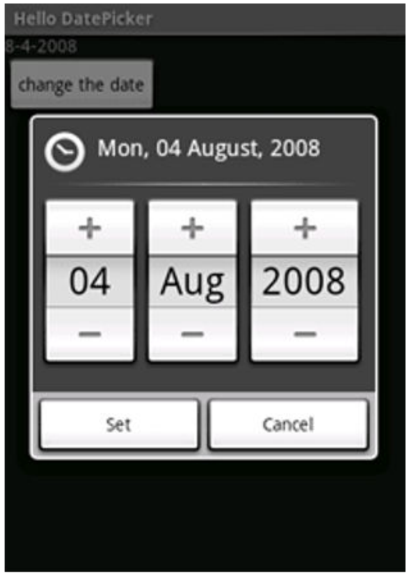
Dialog

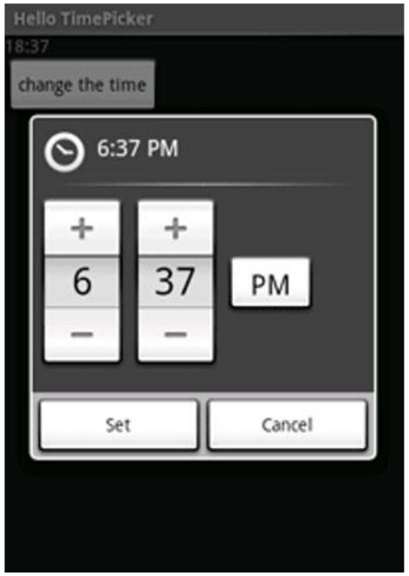
Used to interact with the user

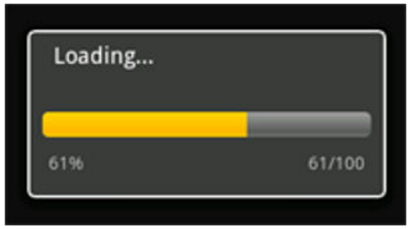
Little messages, easy answers

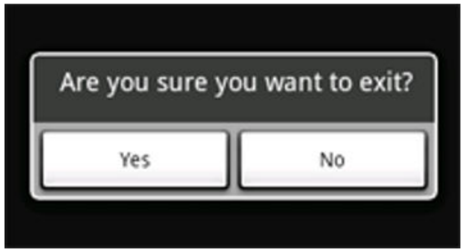
Different kinds:

- AlertDialog
- ProgressDialog
- DatePickerDialog
- TimePickerDialog





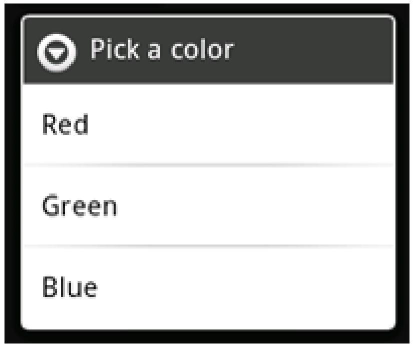


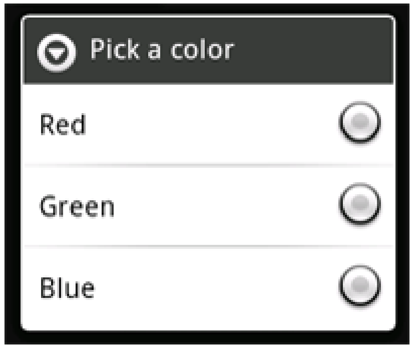


Alert Dialog with a list

```
final CharSequence[] items = {"Red", "Green", "Blue"};
   AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int item) {
       Toast.makeText(getApplicationContext(), items[item],
        Toast.LENGTH_SHORT).show();
     }
   });// OR
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
     public void onClick(DialogInterface dialog, int item) {
       Toast.makeText(getApplicationContext(), items[item],

        Toast.LENGTH_SHORT).show();
     }
   });
   AlertDialog alert = builder.create();
```



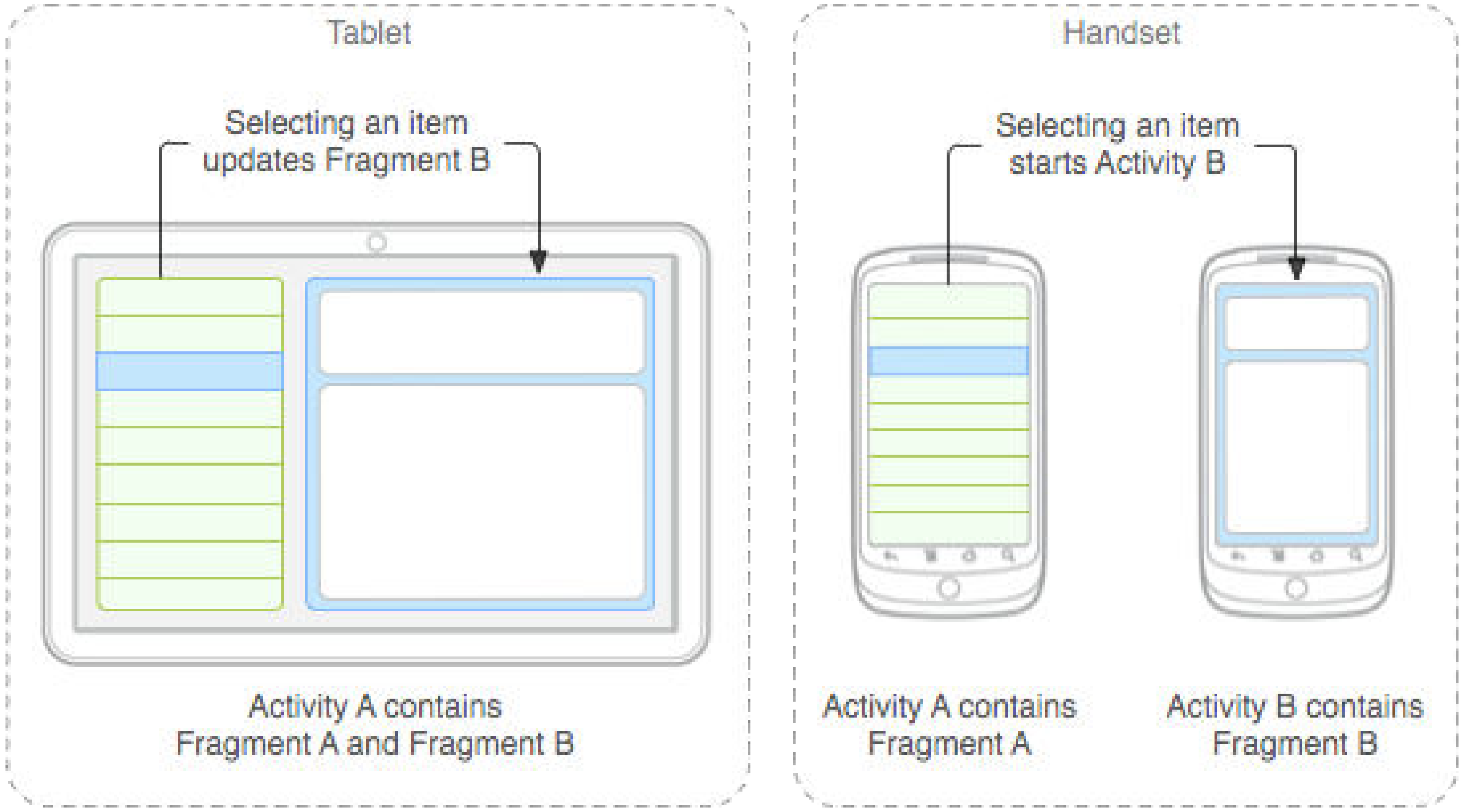


Fragments- what are they?

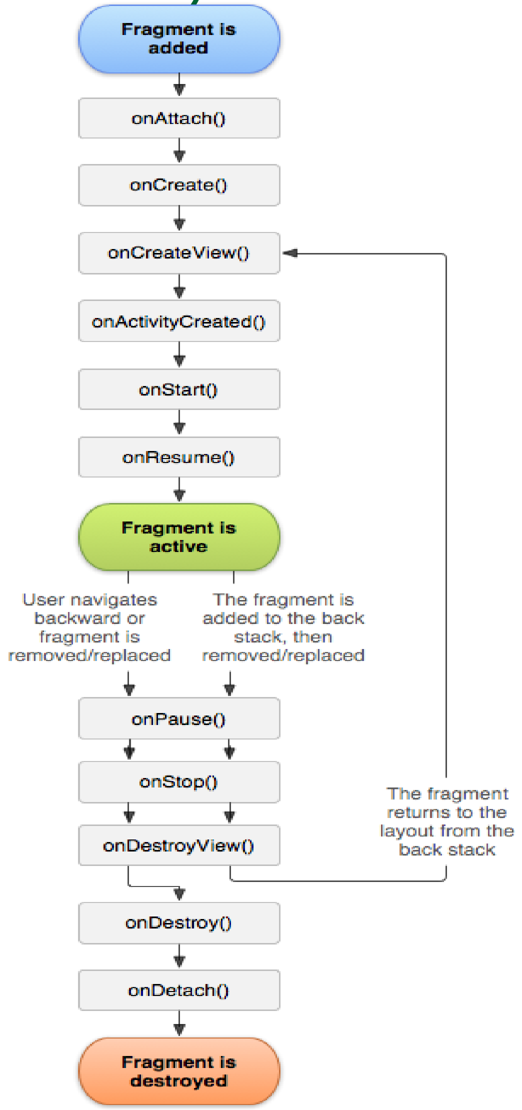
- Mini-Activities
- Own life-cycle

  separate from activity
- Live inside an activity

Fragments(Cont)


Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Fragments(Cont)



Location API

- **Getting the Last Known Location** - Learn how to retrieve the last known location of an Android device, which is usually equivalent to the user's current location.

- **Receiving Location Updates** - Learn how to request and receive periodic location updates

Location API(Cont)

- **Displaying a Location Address**- Learn how to convert a location's latitude and longitude into an address (reverse geocoding).
- **Creating and Monitoring Geofences**- Learn how to define one or more geographic areas as locations of interest, called geofences, and detect when the user is close to or inside a geofence.

Shared Preference

- Good for basic data storage, simple examples:
  - Has the user completed the application settings? (boolean)
  - What is the user's username? (string)

- Shared Preferences accessed by *string* key, value can be:
  - boolean, float, int, long, string

- Arbitrary objects *cannot* be stored in Shared Preferences, best two options for arbitrary objects:
  - Marshal to/from a private database *(best)*
  - Marshal to/from binary files in private storage

Switching Between Activities

- Specify class name of new Activity
- New Activity must be in same project as original Activity

Syntax

– Java (original Activity)

```
Intent activityIntent = new Intent(this, NewActivity.class);
startActivity(activityIntent);
```

– XML (AndroidManifest.xml)

```
<activity android:name=".NewActivity"
android:label="@string/some_app_name">
<intent-filter>
<action android:name="android.intent.action.VIEW" />
<category android:name="android.intent.category.DEFAULT"/>
</intent-filter>
</activity>
```

Future of Android

Android payments and security

- The Google Authenticator app could work with your device's NFC chip to automatically log you into Gmail when you sit down at your laptop, for example, or pay for your flight when you step on a plane.

Android Maps

- Better imagery of most public buildings, as well as tappable info as you move around.

Android messaging

- Don't be surprised to see free 5G video calling and texting between Android devices by 2020, with all of your conversations grouped by person rather than platform, and archived and searchable in Gmail.

Thank You