ANDROID

Android Basics

- Bhaumik Shukla

Android Application Developer

@ STEALTH FLASH

Introduction to Android

- Android is a software stack for mobile devices that includes an operating system, middleware and key applications.
- The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

Introduction to Android

- Android is an open-source platform developed under the Open Handset Alliance to enable faster development of mobile applications and provision of services to the user.
- Google is the leading company to develop and promote Android, however there are other companies as well who are involved in the development of Android. Some of these include T-Mobile, HTC, Qualcomm and others.

Features

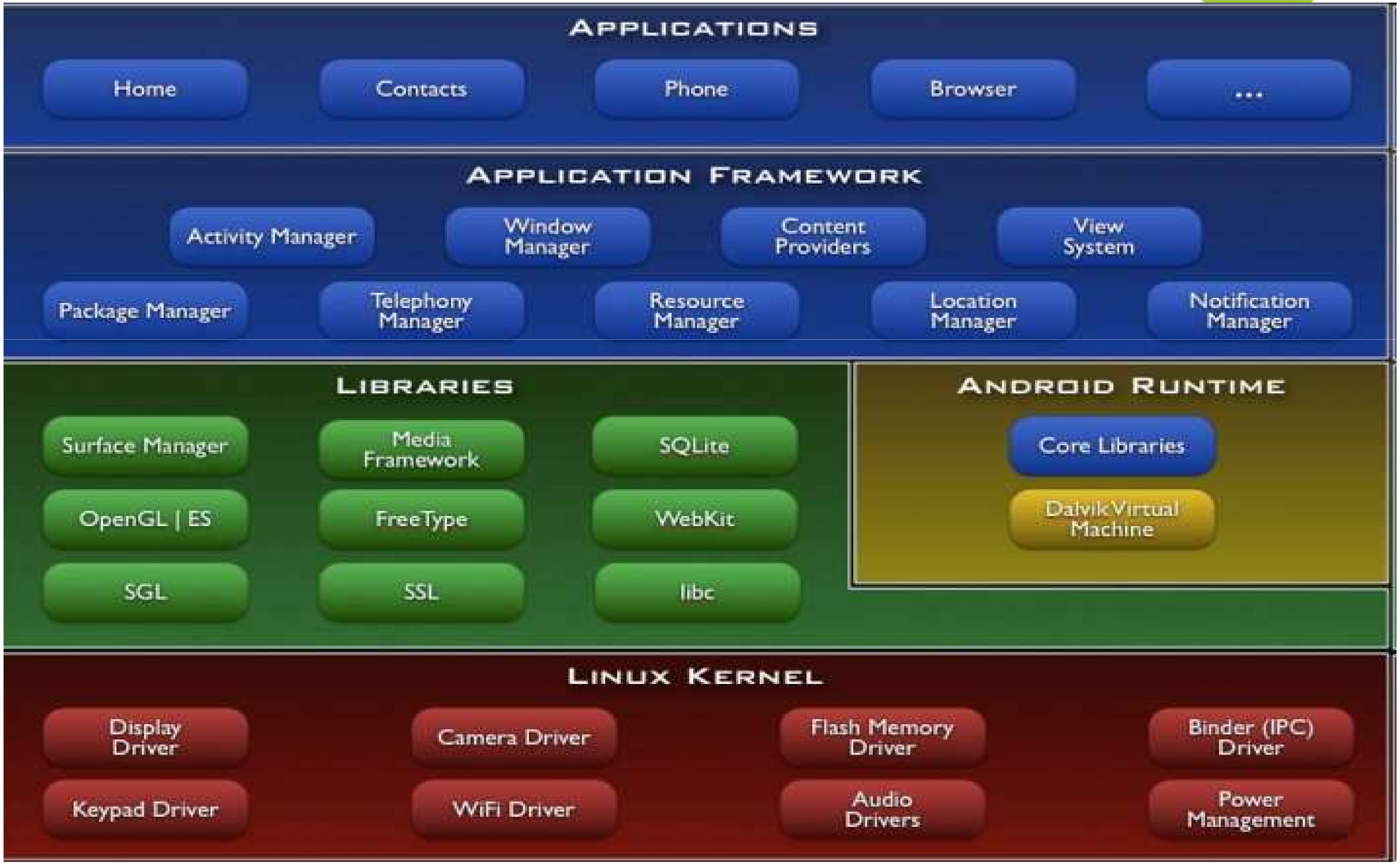
- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine
- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

Tools

- Emulator
  - Android applications may be run on a real device or on the Android Emulator, which ships with the Android SDK.

- ADB (Android Debug Bridge)
  - The ADB utility lets you connect to the phone itself and issue rudimentary shell commands, such as copying files to and from the device.

Architecture



Applications

- Android will ship with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all applications to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of applications and provides a common navigation backstack

Libraries

- **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- **FreeType** - bitmap and vector font rendering
- **SQLite** - a powerful and lightweight relational database engine available to all applications

Android Runtime

- Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.
- Every Android application runs in its **own process**, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.
- The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

Linux Kernel

- Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Application Fundamentals

- Apps are written in Java
- Bundled by Android Asset Packaging Tool
- Every App runs its own Linux process
- Each process has it’s own Java Virtual Machine
- Each App is assigned a unique Linux user ID
- Apps can share the same user ID to see each other’s files

Application Fundamentals

- Android applications are written in the Java programming language. The Android SDK tools compile the code—along with any data and resource files—into an *Android package*, an archive file with an <u>**.apk**</u> suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

Application Components

- Application components are the essential building blocks of an Android application. Each component is a different point through which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your application's overall behavior.

Application Components

- **Activities**
  - An *activity represents a single screen with a user interface.*
  - *Present a visual user interface for one focused endeavor the user can undertake*
  - *Example: a list of menu items users can choose from, or* ***anything.***

Application Components

- **Services**
  - A *service* is a component that runs in the background to perform long-running operations or to perform work for remote processes.
  - A service does not provide a user interface. For example, a service might **play music** in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

Application Components

- **Content providers**
  - A *content provider* manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any application with the proper permissions can query part of the content provider (such as ContactsContract.Data) to read and write information about a particular person.

Application Components

- **Broadcast receivers**
  - A *broadcast receiver* is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
  - Receive and react to broadcast announcements
  - Example: announcements that the time zone has changed

Application Components

- Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an *intent*.
- Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.
- An intent is created with an Intent object, which defines a message to activate either a specific component or a specific *type* of component—an intent can be either explicit or implicit, respectively.

Application Components

- There are separate methods for activiting each type of component:
- You can start an activity (or give it something new to do) by passing an Intent to startActivity() or startActivityForResult() (when you want the activity to return a result).
- You can start a service (or give new instructions to an ongoing service) by passing an Intent to startService(). Or you can bind to the service by passing an Intent to bindService().
- You can initiate a broadcast by passing an Intent to methods like sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().
- You can perform a query to a content provider by calling query() on a ContentResolver.

The Manifest File

- Before the Android system can start an application component, the system must know that the **component** exists by reading the application's AndroidManifest.xml file (the "manifest" file). Your application must declare all its components in this file, which must be at the root of the application project directory.
- The manifest does a number of things in addition to **declaring the application's components**, such as:
- Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.
- Declare the minimum API Level required by the application, based on which APIs the application uses.
- Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.
- API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.
- And more

The Manifest File

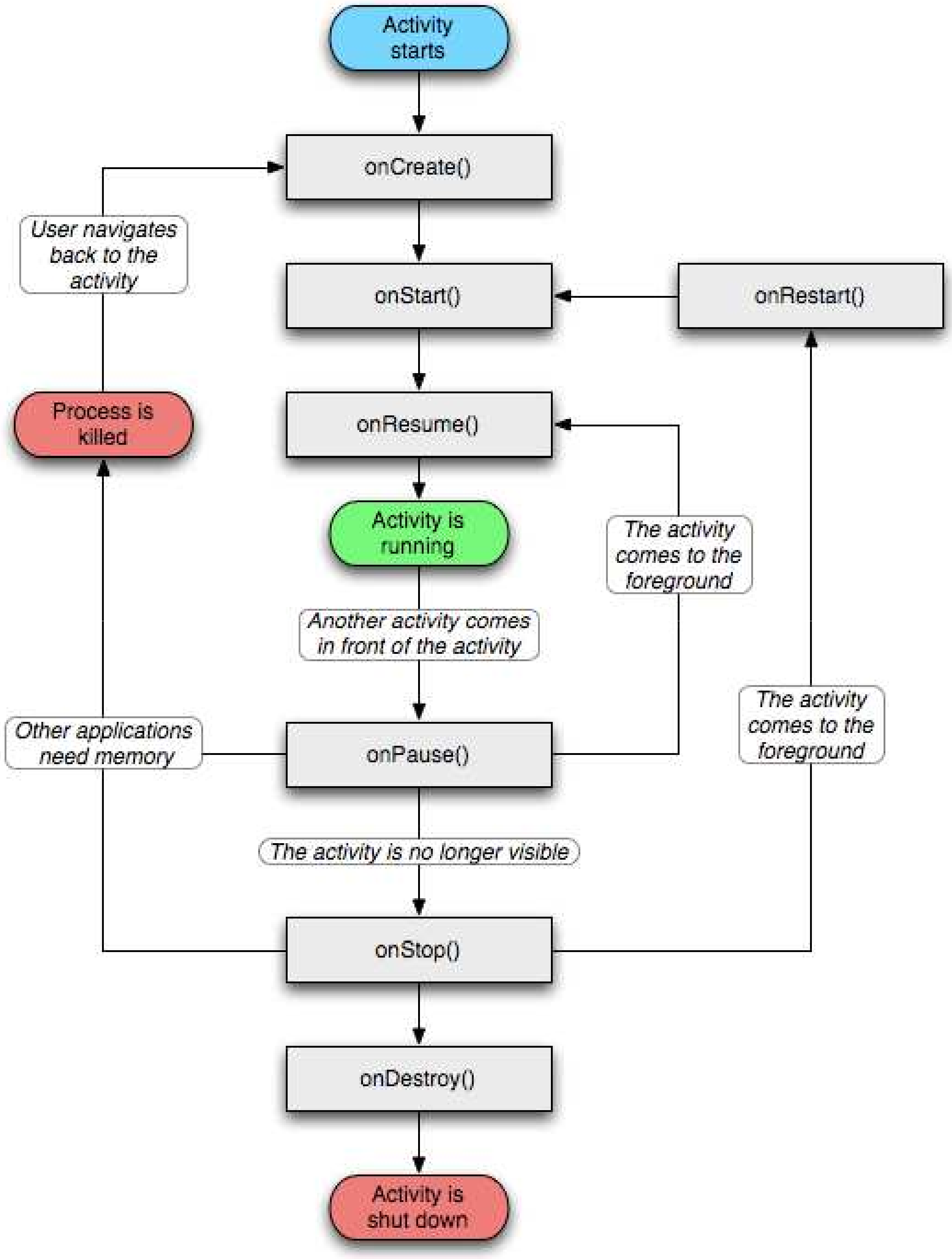
The primary task of the manifest is to inform the system about the application's components. For example, a manifest file can declare an activity as follows:

```
<?xml version="1.0" encoding="utf-8"?>
    <manifest ... >
        <application android:icon="@drawable/app_icon.png" ... >
            <activity android:name=".ExampleActivity"
                      android:label="@string/example_label" ... >
            </activity>
            ...
        </application>
    </manifest>
```

Application Resources

- An Android application is composed of more than just code—it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application.
- For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes).

Activity life cycle


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

Activity callbacks

- onCreate() Called when the activity is first created. This is where you should do all of your normal static set up — create views, bind data to lists, and so on. This method is passed a Bundle object containing the activity's previous state, if that state was captured (see Saving Activity State, later). Always followed by onStart().
- onRestart() Called after the activity has been stopped, just prior to it being started again. Always followed by onStart()
- onStart() Called just before the activity becomes visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.
- onResume() Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it. Always followed by onPause().

Activity callbacks

- onPause() Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. Followed either by onResume() if the activity returns back to the front, or by onStop() if it becomes invisible to the user.
- onStop() Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it. Followed either by onRestart() if the activity is coming back to interact with the user, or by onDestroy() if this activity is going away.

Activity callbacks

- onDestroy() Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing (someone called finish() on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.

Declaring Activity

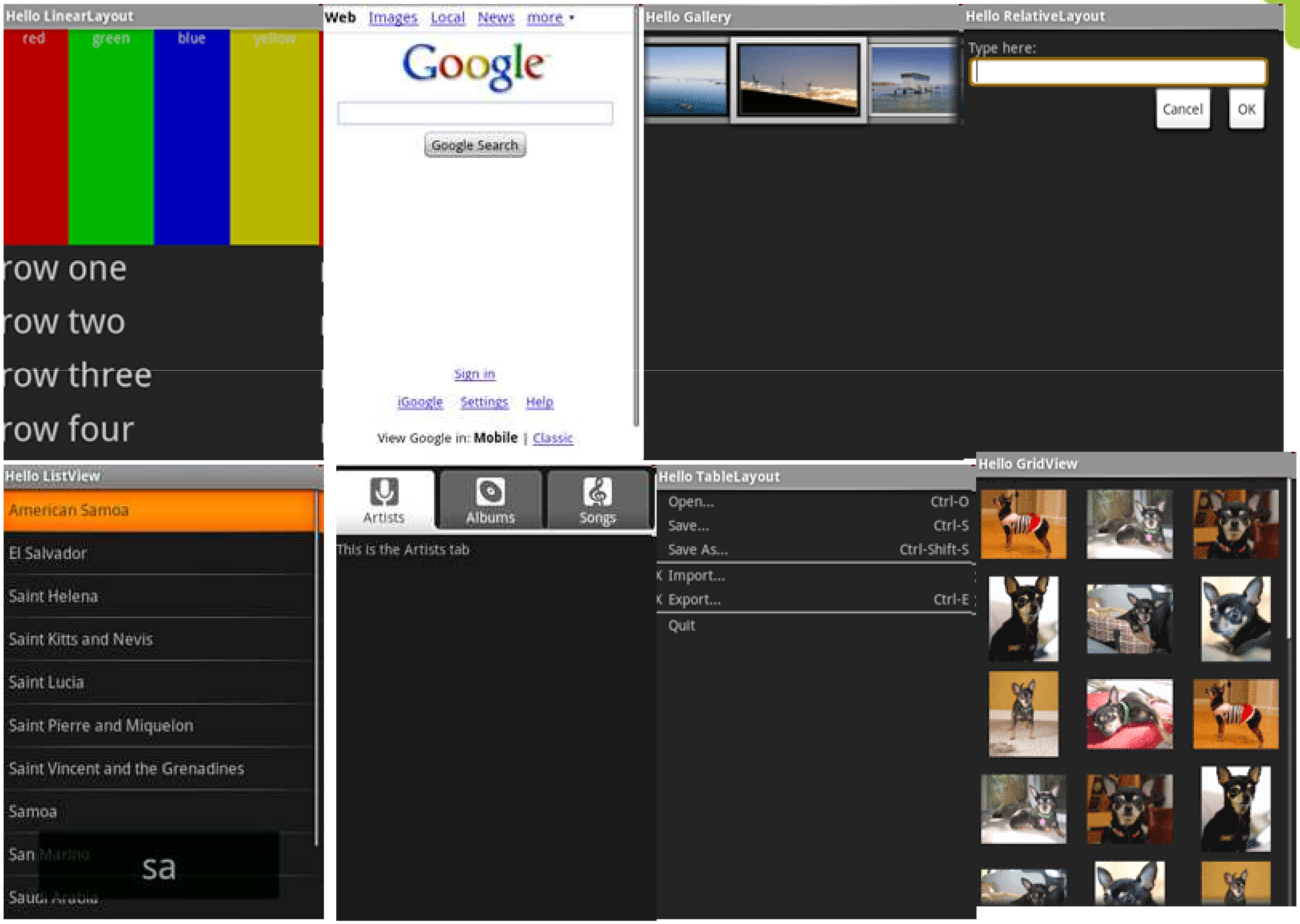
- **Declaring the activity in the manifest**
- You must declare your activity in the manifest file in order for it to be accessible to the system. To declare your activity, open your manifest file and add an <activity> element as a child of the <application> element. For example:

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

View Groups

- FrameLayout Layout that acts as a view frame to display a single object.
- Gallery A horizontal scrolling display of images, from a bound list.
- GridView Displays a scrolling grid of m columns and n rows.
- LinearLayout 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.
- ListView Displays a scrolling single column list.
- RelativeLayout 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).
- ScrollView A vertically scrolling column of elements.
- etc..

View Groups


Hello LinearLayout
red
green
blue
yellow
row one
row two
row three
row four
Web
Images
Local
News
more
Google
Google Search
Sign in
iGoogle
Settings
Help
View Google in: Mobile | Classic
Hello Gallery
Hello RelativeLayout
Type here:
Cancel
OK
Hello ListView
American Samoa
El Salvador
Saint Helena
Saint Kitts and Nevis
Saint Lucia
Saint Pierre and Miquelon
Saint Vincent and the Grenadines
Samoa
San Marino
Saudi Arabia
sa
Artists
Albums
Songs
This is the Artists tab
Hello TableLayout
Open... Ctrl-O
Save... Ctrl-S
Save As... Ctrl-Shift-S
X Import...
X Export... Ctrl-E
Quit
Hello GridView

Write XML for UI design

- Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

sample **main.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/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>
```

Installation

- How to install & configure Android SDK and Eclipse

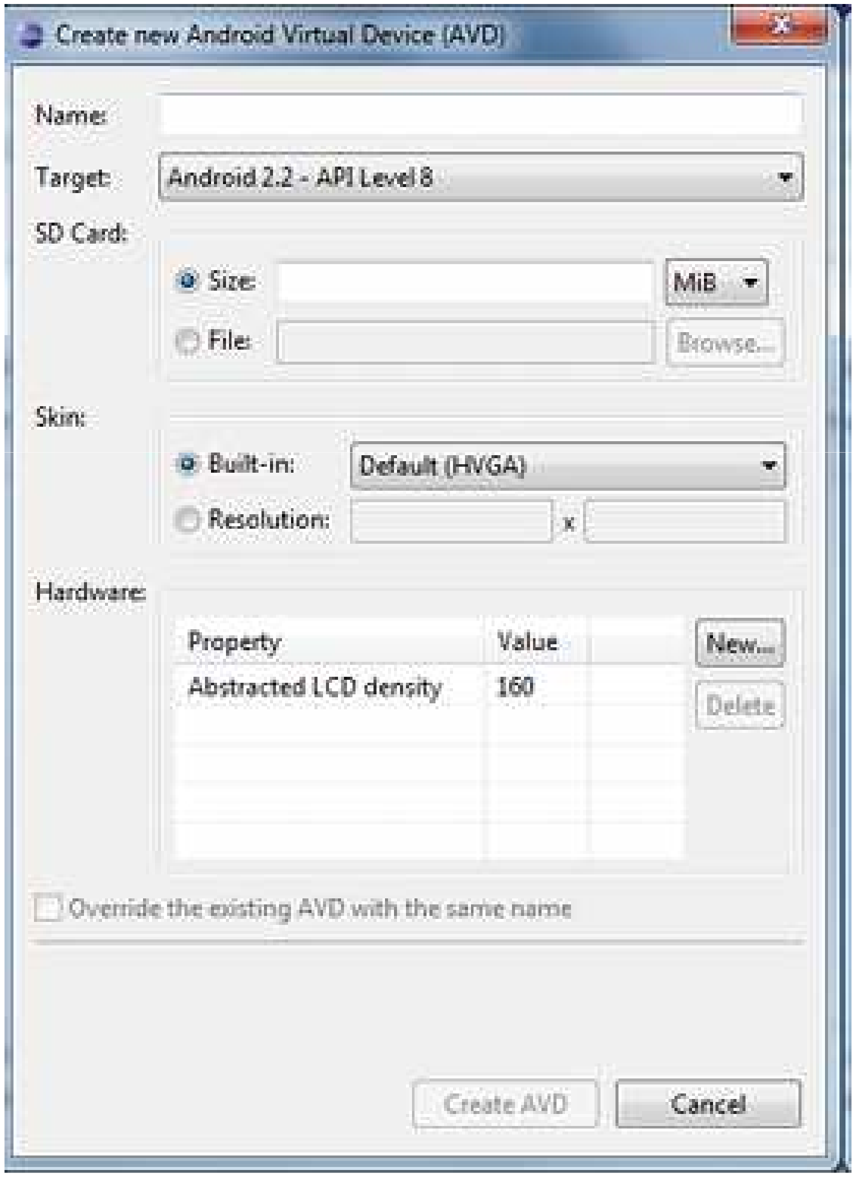
  http://developer.android.com/sdk/installing.html

Configure AVD

http://developer.android.com/guide/developing/devices/managing-avds.html

- **Creating an AVD**
- You can create as many AVDs as you would like to test on. It is recommended that you test your applications on all API levels higher than the target API level for your application.
- To create an AVD:
- Start the AVD Manager:
  - In Eclipse: select **Window > Android SDK and AVD Manager**, or click the Android SDK and AVD Manager icon in the Eclipse toolbar.
  - In other IDEs: Navigate to your SDK's tools/ directory and execute the android tool with no arguments.
- In the *Virtual Devices* panel, you'll see a list of existing AVDs. Click **New** to create a new AVD. The **Create New AVD** dialog appears.

Configure AVD



Create new Android Virtual Device (AVD)

Name:

Target: Android 2.2 - API Level 8

SD Card:

[✓ Size:] MiB

File: Browse...

Skin:

[✓ Built-in:] Default (HVGA)

Resolution: x

Hardware:

Property	Value
Abstracted LCD density	160

Start Developing on Android

First Android Project

- Select **File** > **New** > **Project**.
- Select **Android** > **Android Project**, and click **Next**.
- Select the contents for the project:
  - Enter a *Project Name*.
  - Under Target, select an Android target to be used as the project's Build Target
  - Under Properties, fill in all necessary fields.
    - Enter an *Application name*.
    - Enter a *Package name*.
    - Select *Create Activity* (optional, of course, but common) and enter a name for your main Activity class.
    - Enter a *Min SDK Version*
- Click **Finish**.

- Your Android project is now ready. It should be visible in the Package Explorer on the left. Open the HelloAndroid.java file, located inside *HelloAndroid* > *src* > *com.example.helloandroid*). It should look like this:
-

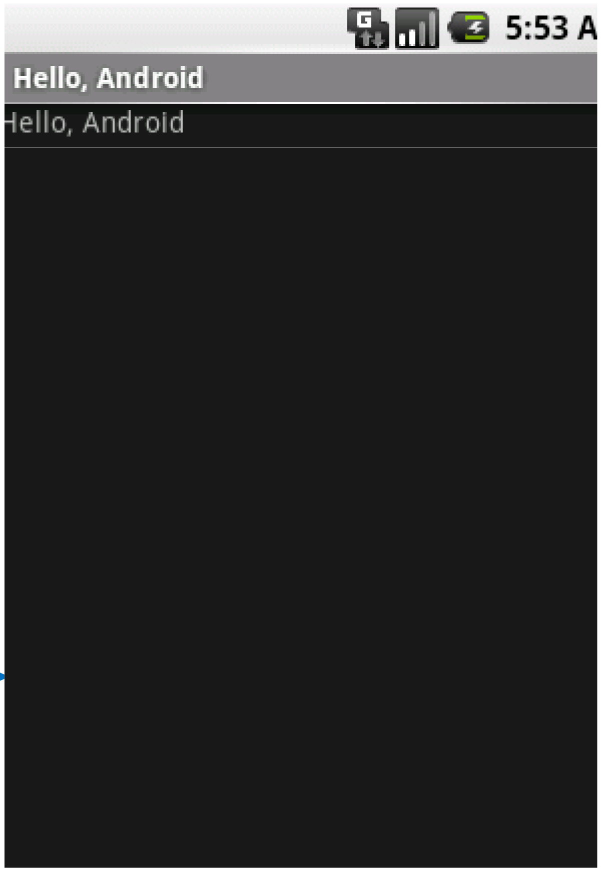
```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class HelloAndroid extends Activity {
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
   }
}
```

res/layout/main.xml

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

res/layout/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello, Android! I am a string
resource!</string>
    <string name="app_name">Hello, Android</string>
</resources>
```



5:53 A

Hello, Android

Hello, Android

Handling UI Events

- On Android, there's more than one way to intercept the events from a user's interaction with your application. When considering events within your user interface, the approach is to capture the events from the specific View object that the user interacts with. The View class provides the means to do so.

- Event Listeners An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Handling UI Events

- onClick() From View.OnClickListener.
- onLongClick() From View.OnLongClickListener.
- onFocusChange() From View.OnFocusChangeListener.
- onKey() From View.OnKeyListener.
- onTouch() From View.OnTouchListener.
- onCreateContextMenu() From View.OnCreateContextMenuListener.

How to write Click event

- The example below shows how to register an on-click listener for a Button.

// Create an anonymous implementation of OnClickListener

```
private OnClickListener mCorkyListener = new OnClickListener() {
        public void onClick(View v) {
          // do something when the button is clicked
        }
    };

    protected void onCreate(Bundle savedValues) {
        ...
        // Capture our button from layout
        Button button = (Button)findViewById(R.id.btn);
        // Register the onClick listener with the implementation above
        button.setOnClickListener(mCorkyListener);
        ...
    }
```

How to write Click event

- You may also find it more convenient to implement OnClickListener as a part of your Activity. This will avoid the extra class load and object allocation. For example:

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

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

Example

Button Click example counts the clicks.

- We need a to declare activity in **AndroidManifest.xml** file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.test"
      android:versionCode="1"
      android:versionName="1.0“>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".test"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Example

Button Click example counts the clicks.

- We need a button in **main.xml**

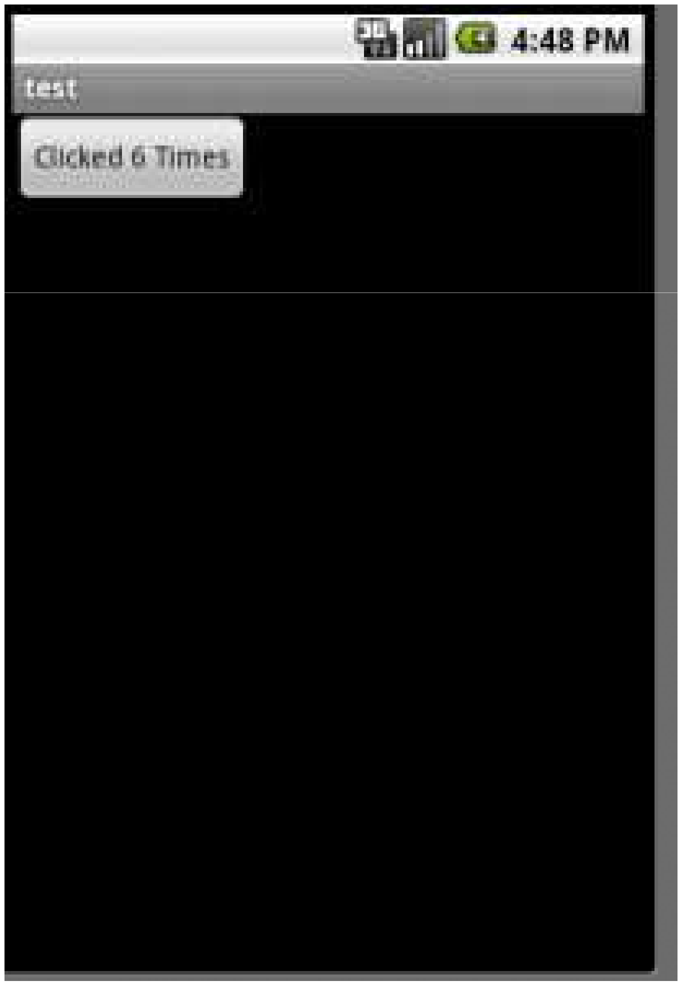
```
<?xml version=”1.0″ encoding=”utf-8″?>
    <LinearLayout android:id=”@+id/linear1″
    android:layout_width=”fill_parent” android:layout_height=”fill_parent”
    xmlns:android=”http://schemas.android.com/apk/res/android”>
    <Button android:text=”ClickMe” android:id=”@+id/button1″
    android:layout_width=”wrap_content”
    android:layout_height=”wrap_content”></Button>
</LinearLayout>
```

Example

Button Click example counts the clicks.

Activity.Java

```
package com.example.test;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class test extends Activity implements OnClickListener{
Button btn;
int count=0;
        public void onCreate(Bundle icicle)
        {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        btn= (Button)findViewById(R.id.button1);
        btn.setOnClickListener(this);
        }
        public void onClick(View v) {
        // TODO Auto-generated method stub
        btn.setText("Clicked "+ ++count + " Times");
        }
}
```



Questions ?

My Android Blog:

1. http://www.boomtech.in
2. http://androidrox.wordpress.com