



UNIVERSITY of TARTU

INSTITUTE OF COMPUTER SCIENCE



Mobile Application Development - Android

MTAT.03.262



Mobile & Cloud Lab

Satish Srirama

satish.srirama@ut.ee



tiigriülikool

Programm IKT toetuseks kõrghariduses

Tiigriülikool on riiklik info-
ja kommunikatsioonitehnoloogia-
alane kõrghariduse toetusprogramm

Goal

- Give you an idea of how to start developing Android applications
- Introduce major Android application concepts
- Walk you through some sample applications in the development environment

References & Books

- Android developers
<http://developer.android.com/>
- Books
 - “Professional Android 4 Application Development”, By Reto Meier

Dealing with problems & Help

- Have created a mailing list
 - mclab DOT mad AT lists DOT ut DOT ee
- Post your question to mailing list
- **Practicals assistants:**
 - Mohan Liyanage (liyanage AT ut DOT ee)
 - Jakob Mass (jaks AT ut DOT ee)
- Also created a Google group
<https://groups.google.com/forum/#!forum/mob-app-dev--mtat03262-ut>
 - Post your questions there
 - Keep answering your friends

What is Android?

- Android is not a device or a product
 - It's not even limited to phones - you could build a handheld GPS, an MP3 player, TV, Watch etc.
- A free, open source mobile platform
- Open Handset Alliance
 - 100+ technology companies
 - Commitment to openness, shared vision, and concrete plans
- A Linux-based, multiprocess, multithreaded OS
 - The Android operating system is a multi-user Linux system in which each application is a different user

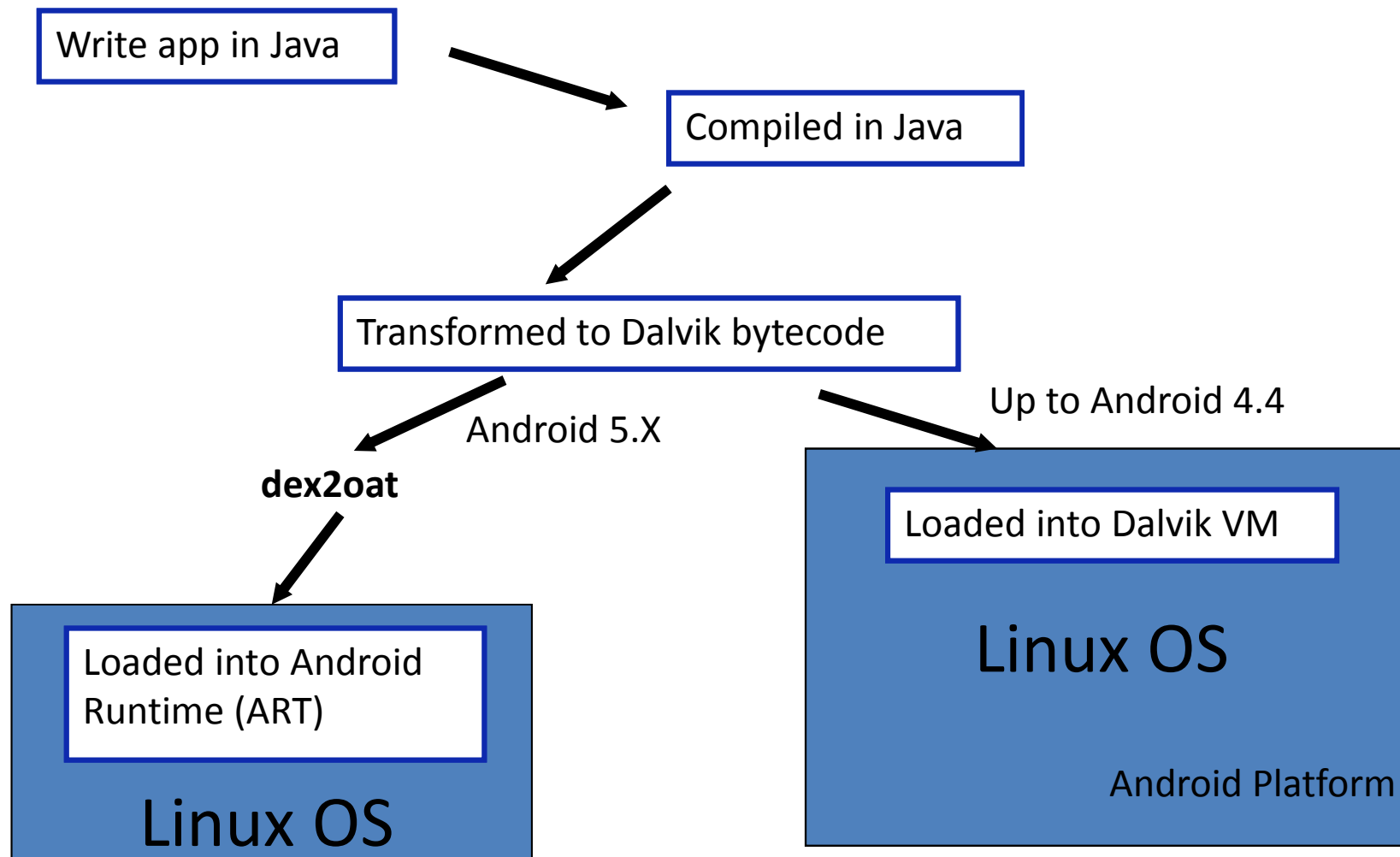
Android applications are written in Java

```
package com.google.android.helloactivity;

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

public class HelloActivity extends Activity {
    public HelloActivity() {
    }
    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.hello_activity);
    }
}
```

Android applications are compiled to Dalvik bytecode



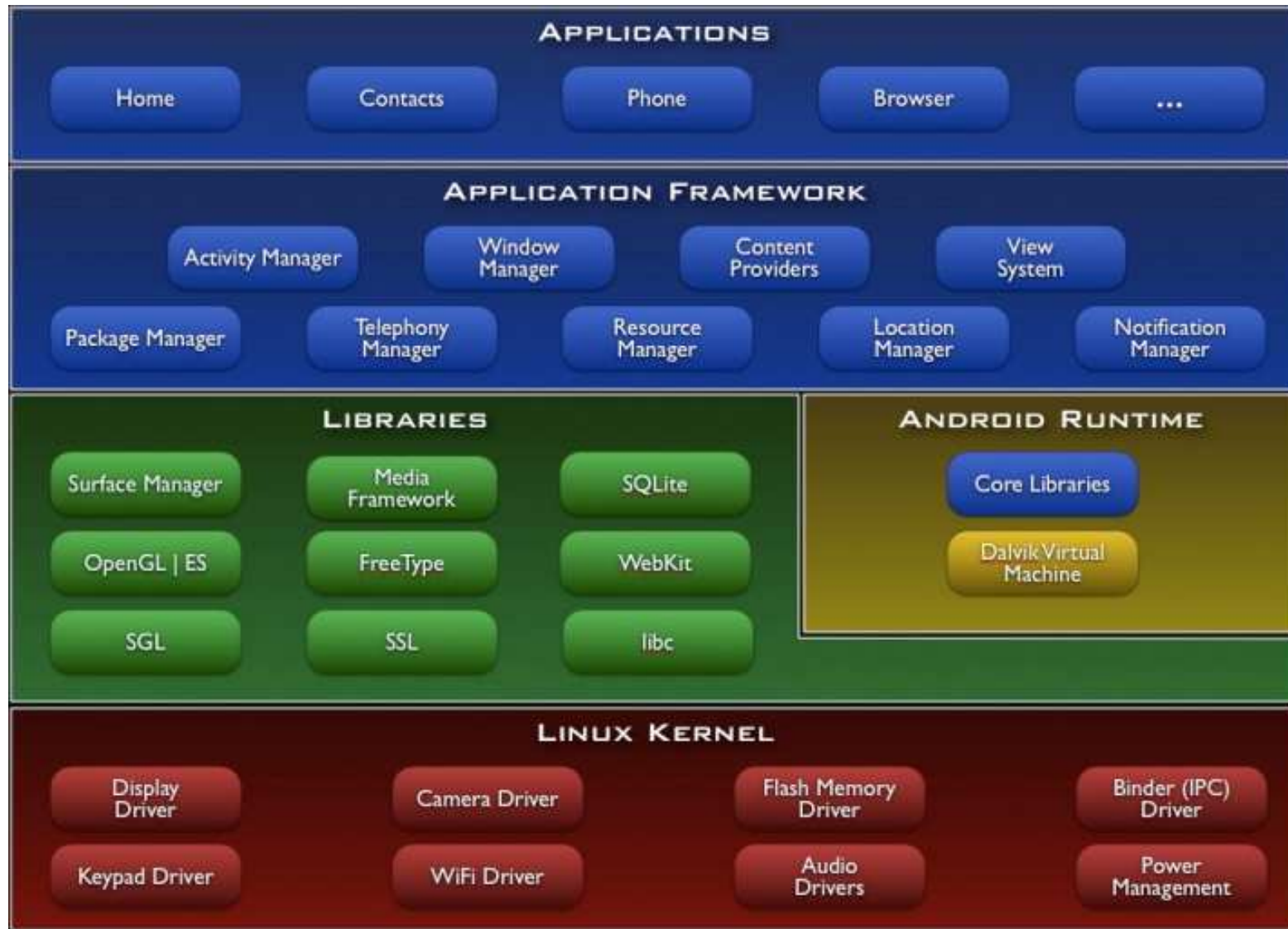
Android's ART vs Dalvik

- ART and Dalvik are compatible runtimes running Dex bytecode
- Dalvik is based on JIT (just in time) compilation
 - Each time an app is run, the part of the code required for its execution is going to be translated (compiled) to machine code
 - It has a smaller memory footprint and uses less physical space on the device
- ART compiles the intermediate language, Dalvik bytecode, into a system-dependent binary
 - Whole code of the app will be pre-compiled during install (once)
 - Ahead-of-Time compiler (AOT)
 - Installation process takes a bit longer
 - Code executes much faster
 - Less CPU usage as no more compilation required and this also results in less battery drain
- ART optimized the garbage collector (GC)

Why Android/Dalvik Runtime?

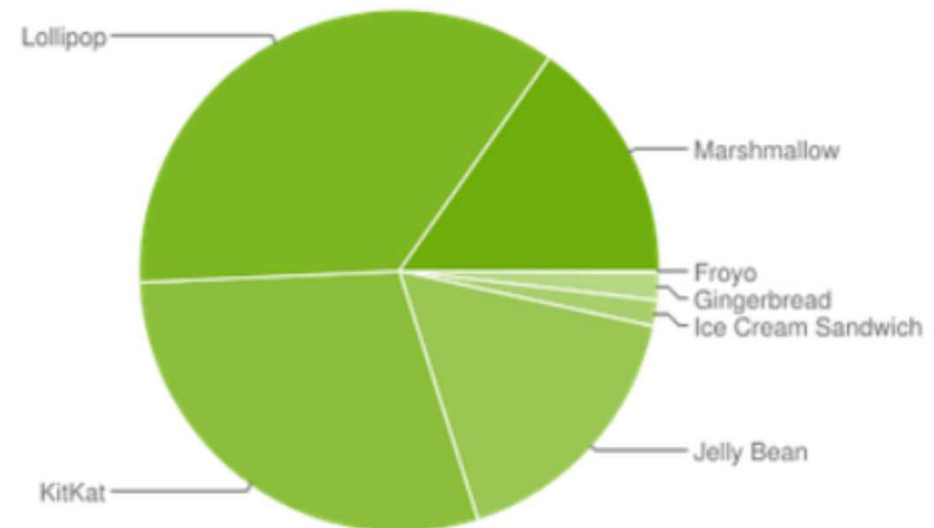
- The Android/Dalvik runtime is optimized for mobile applications
- Runs multiple VMs efficiently
- Each application has its own VM
- Minimal memory footprint
- Relies on Linux kernel for threading and low-level memory management

Android software stack



Can assume that most devices have android 4.x to 6.0

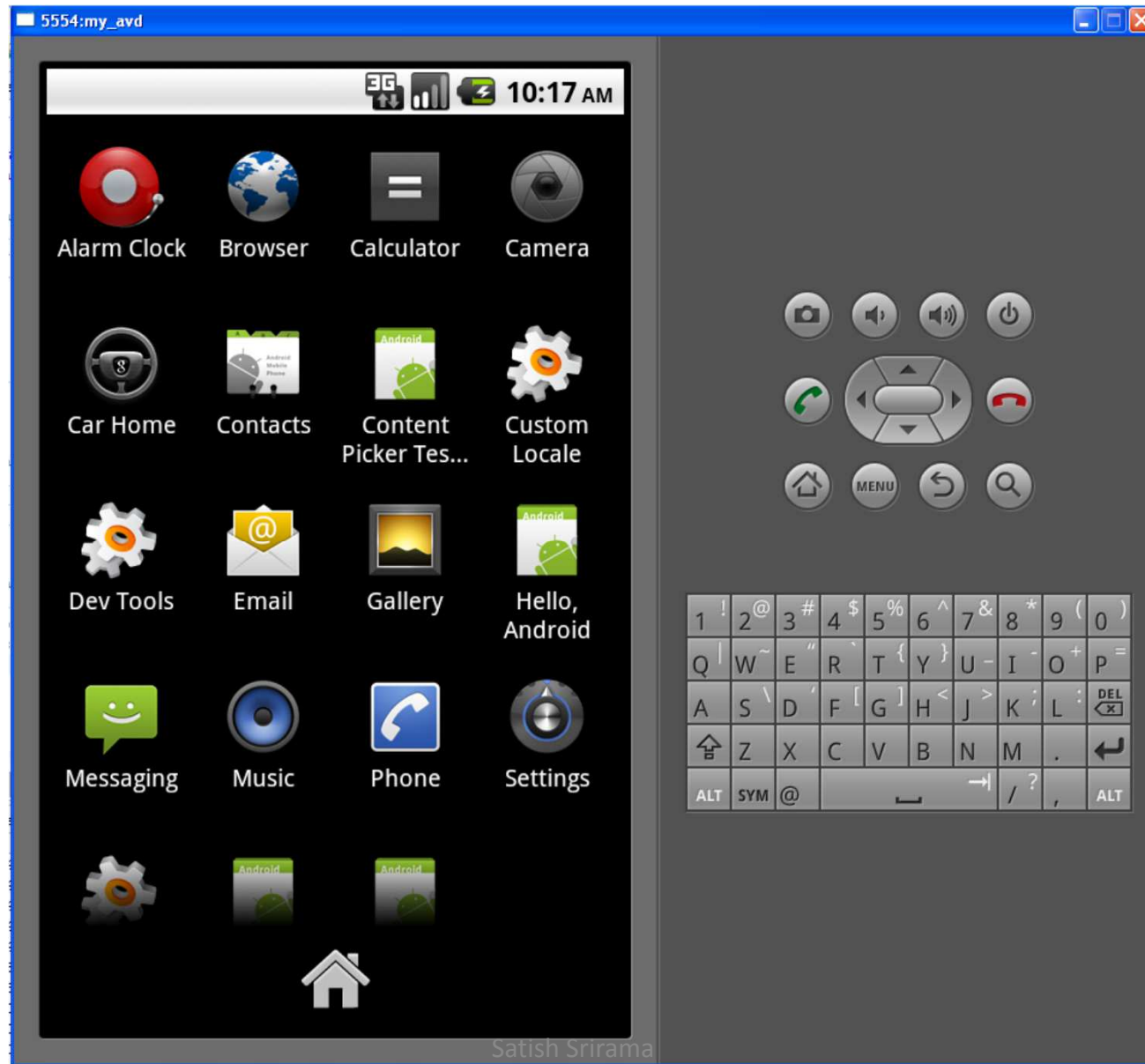
| Version | Codename | API | Distribution |
|------------------|-----------------------|-----|--------------|
| 2.2 | Froyo | 8 | 0.1% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 1.7% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 1.6% |
| 4.1.x | Jelly Bean | 16 | 6.0% |
| 4.2.x | | 17 | 8.3% |
| 4.3 | | 18 | 2.4% |
| 4.4 | KitKat | 19 | 29.2% |
| 5.0 | Lollipop | 21 | 14.1% |
| 5.1 | | 22 | 21.4% |
| 6.0 | Marshmallow | 23 | 15.2% |



<http://developer.android.com/about/dashboards/index.html>

http://en.wikipedia.org/wiki/Android_version_history

Android has a working emulator

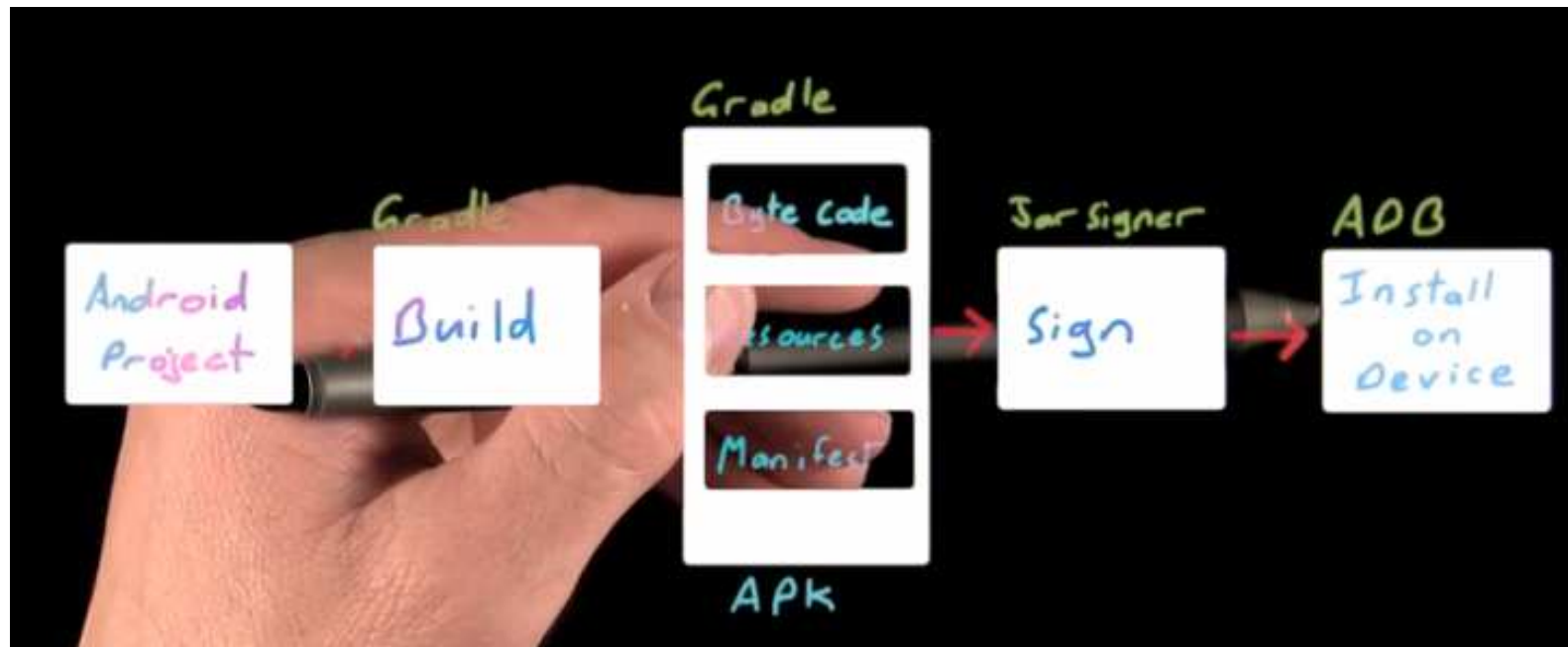


Getting started

- I hope all of you have Android Studio installed
- Downloaded the latest SDK tools and platforms (version 23) using the SDK Manager

[http://developer.android.com/training/basics/fir
stapp/index.html](http://developer.android.com/training/basics/fir
stapp/index.html)

Redraw



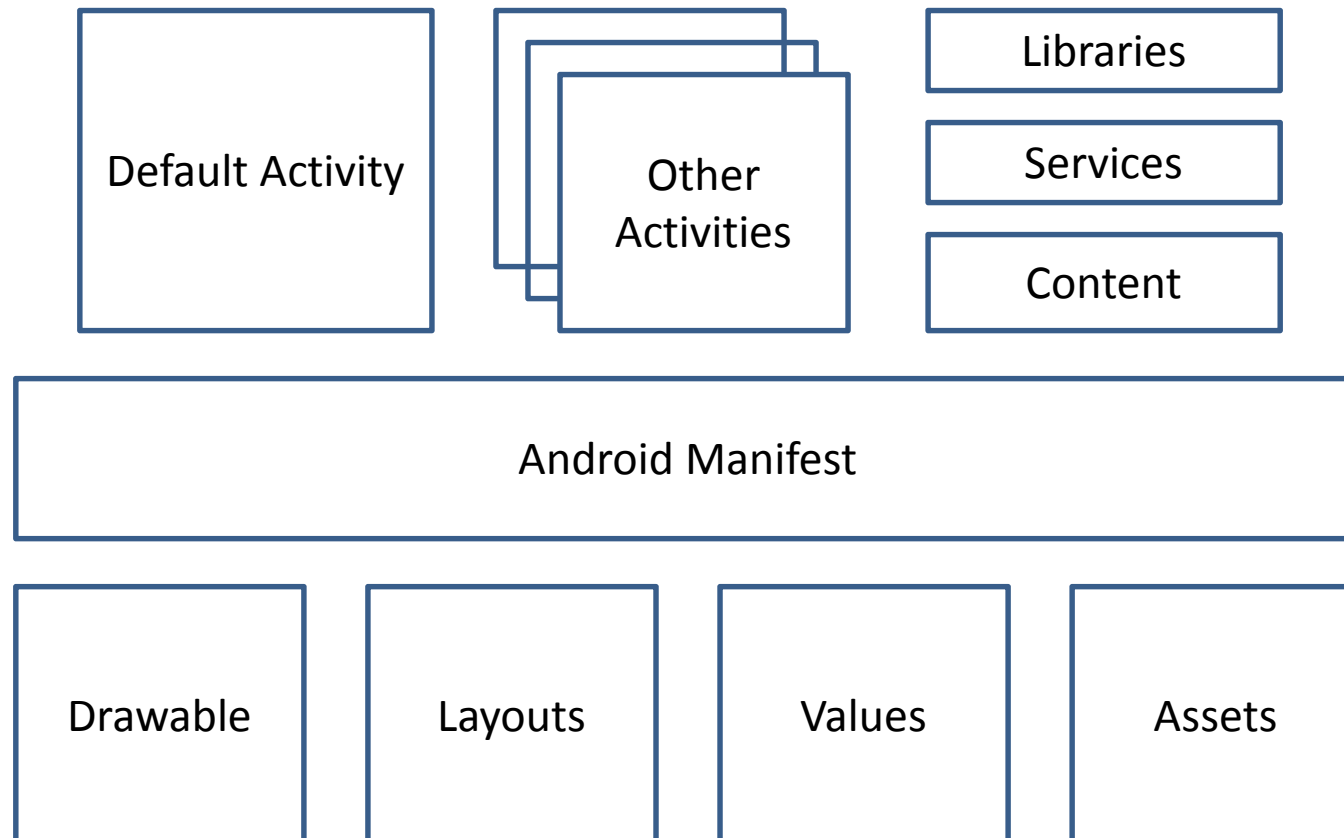
Exercise: Hello World

- Let us create an AVD (Android virtual device)
- Create a New Android Project
- Construct the UI
- Run the Application
- Upgrade the UI to an XML Layout
- Let us debug

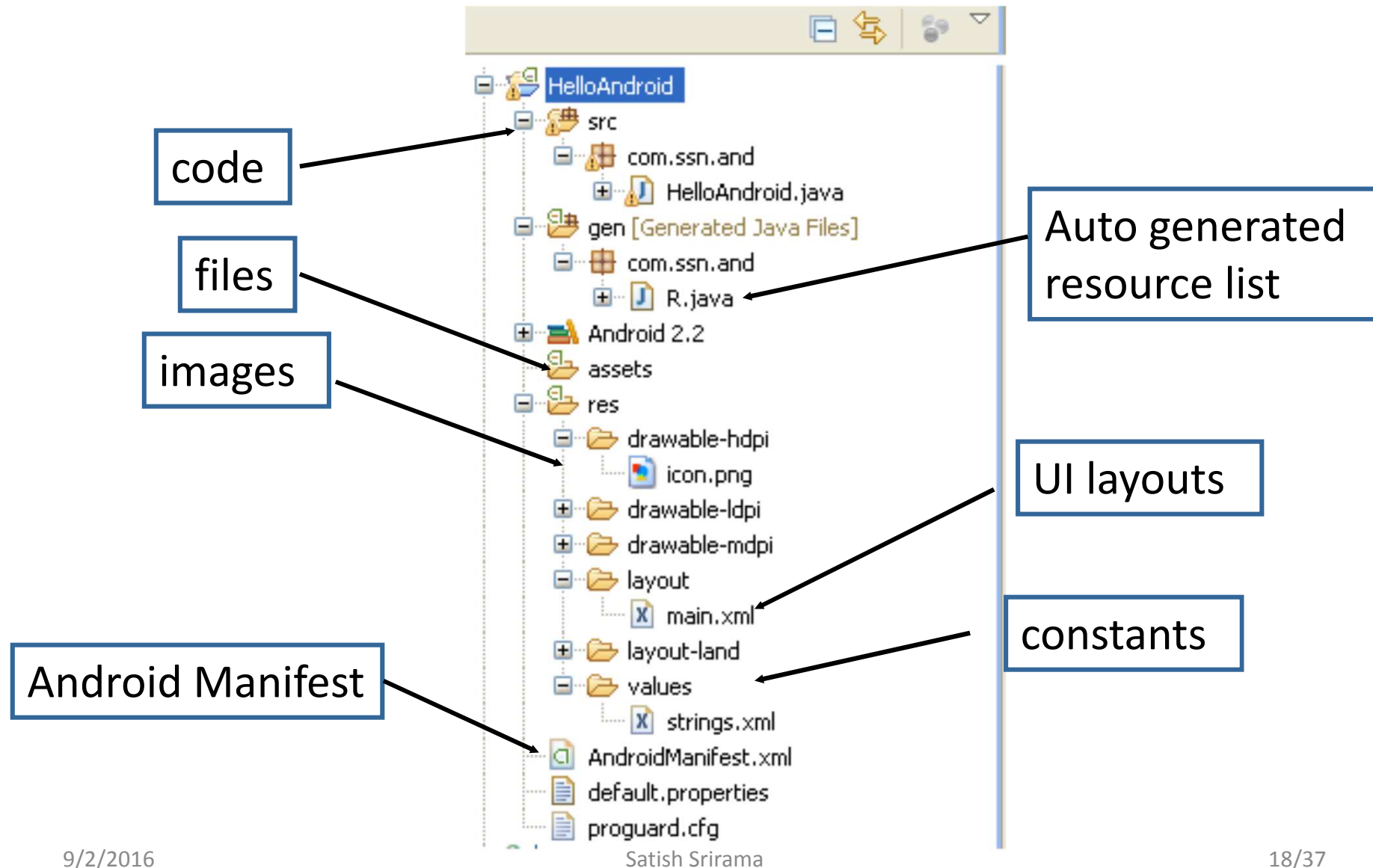
Debugging in Android

- Traditional `System.out.println` is not available in the Android system
- We can debug through the app user interface
 - Errors crash and close app
- Instead use Logging mechanisms
 - `Log.v(String tag, String message);`
 - *tag -> app name*
 - *Message -> debug message.*
- Requires importing `android.util.Log`
- Log messages appear in the LogCat component of the Android Studio interface

What's in an Application?



File structure of applications



Application Manifest

- Each Android project has a manifest file
- Defines the structure and metadata of the application
- Includes nodes for each of the application components (Activities, Services, Intents etc.)
- Also includes security permissions

Application Manifest - continued

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ssn.and.ch5" android:versionCode="1" android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".ContactPickerActivity"
            android:label="@string/app_name">
            <!--
                <intent-filter> <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            -->
            <intent-filter>
                <action android:name="android.intent.action.PICK" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:path="contacts" android:scheme="content" />
            </intent-filter>
        </activity>
        <activity android:name=".ContentPickerTesterActivity" android:label="Content Picker Test">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

Security in Android

- Follows standard Linux guidelines
- Each application runs in its own process
- Process permissions are enforced at user and group IDs assigned to processes
- Finer grained permissions are then granted (revoked) per operations
- Apps declare permissions in manifest
 - `<uses-permission id="android.permission.RECEIVE_SMS" />`

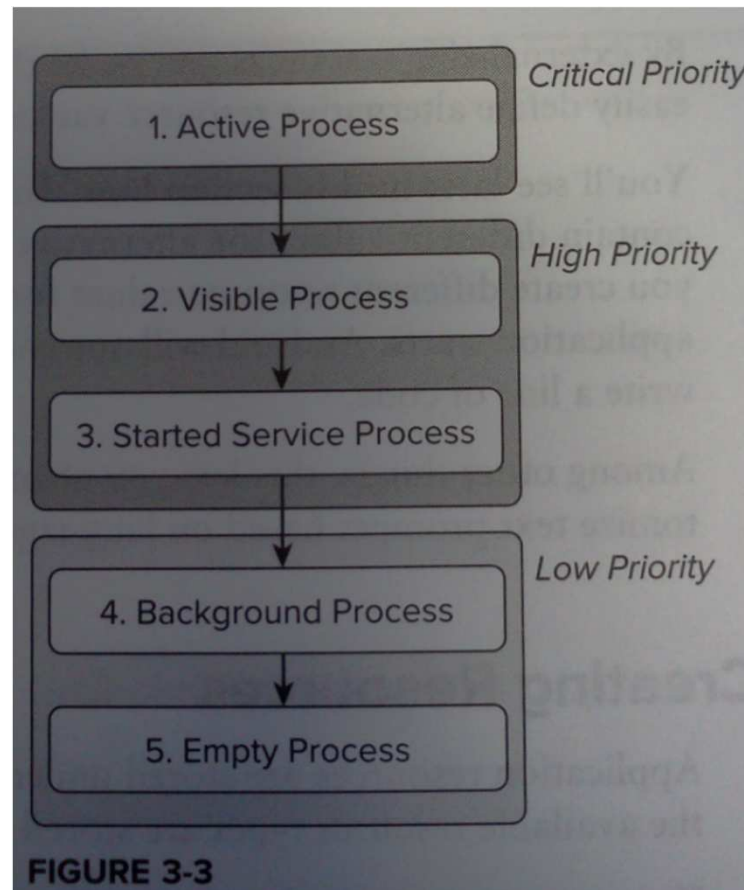
Android Design Philosophy

- Applications should be:
 - Fast
 - Resource constraints: <200MB RAM, slow processor
 - Responsive
 - Apps must respond to user actions within 5 seconds
 - Secure
 - Apps declare permissions in manifest
 - Seamless
 - Usability is key, persist data, suspend services
 - Android kills processes in background as needed

Application priority and process states

- Android applications have limited control over their life cycles
- Each application runs in its own process
 - Each process is running a separate instance of Dalvik/ART
- Memory and process management is handled by runtime
 - Runtime may kill some services in the background
 - Application priority is critical

Application priority

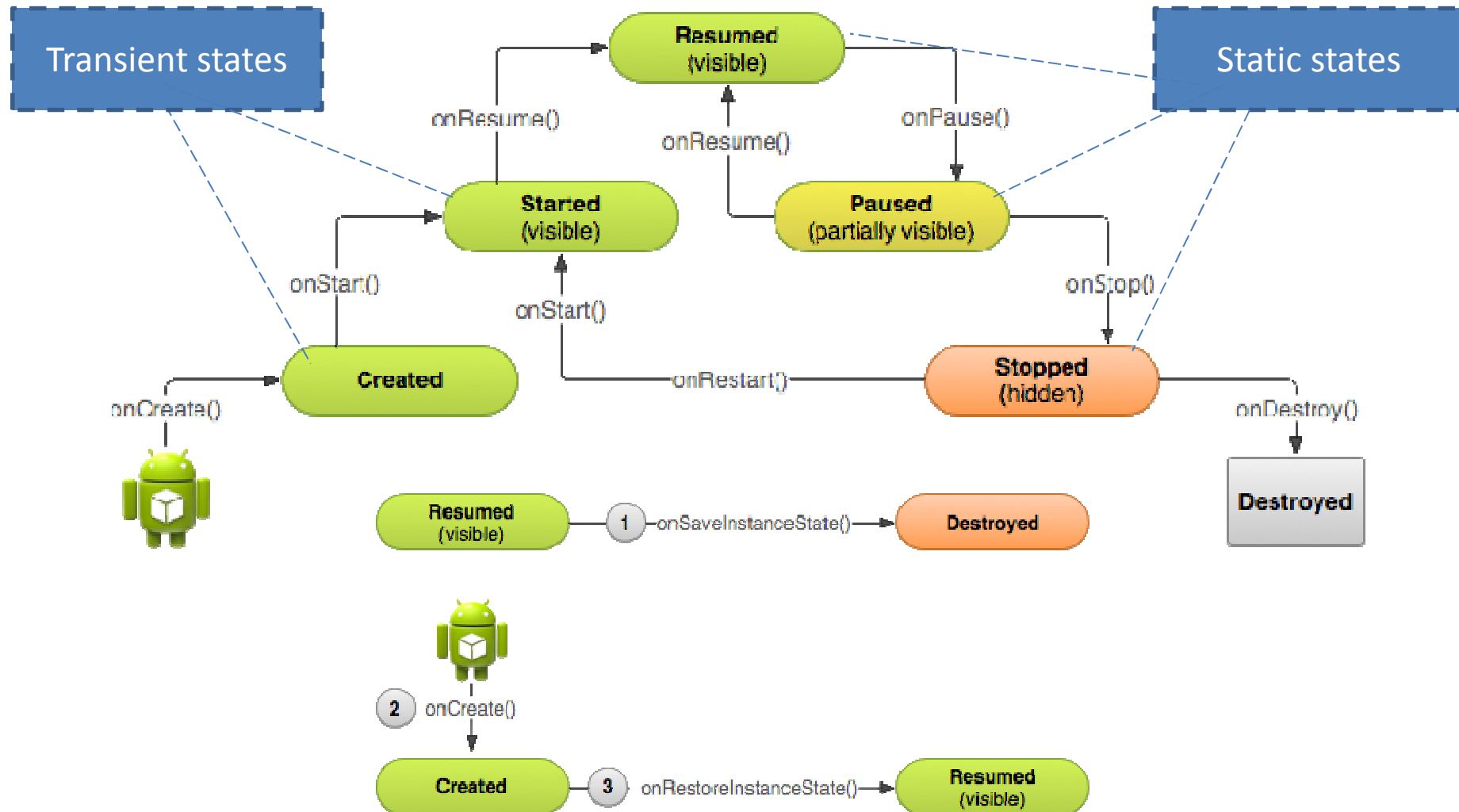


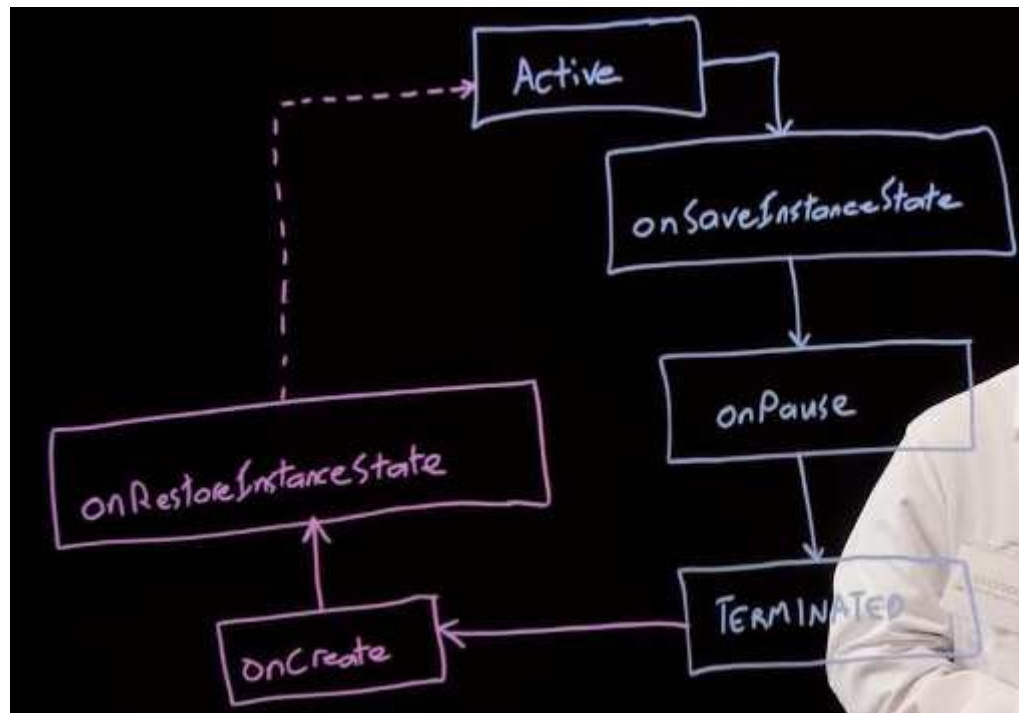
Reto Meier, Professional Android 2 Application Development, p 59

Android application lifecycle

- Apps move through states during lifecycle
- Understanding app lifecycle is necessary, so that apps:
 - Does not crash if the user receives a phone call or switches to another app while using your app
 - Does not consume valuable system resources when the user is not actively using it
 - Does not lose the user's progress if they leave your app and return to it at a later time
 - Does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation

Android application lifecycle - continued





The History of GUIs

- Hardcoded to the screen
- Hardcoded to the window
- Hardcoded within a view hierarchy
- Dynamic layout within a view hierarchy

Generating GUIs in Android

- Two ways to create GUIs: in XML or in code

```
public class LayoutExamplesActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        Button buttonOne = new Button(this);  
        Button buttonTwo = new Button(this);  
        buttonOne.setText("Press Me!");  
        buttonTwo.setText("Press Me Two!");  
  
        LinearLayout linearLayout = new LinearLayout(this);  
        linearLayout.setOrientation(LinearLayout.VERTICAL);  
        linearLayout.addView(buttonOne);  
        linearLayout.addView(buttonTwo);  
  
        setContentView(linearLayout);  
    }  
}
```

```
<?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"  
>  
    <Button  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="Press Me!"  
    >  
    </Button>  
    <Button  
        android:layout_width="fill_parent"  
        android:layout_height="wrap_content"  
        android:text="Press Me Two!"  
    >  
    </Button>  
</LinearLayout>
```

Advantages of Declarative route via XML

- Separation of appearance (presentation) from actual meaningful state-changing code
- Can change interface without having to completely recompile Java code
 - Can just recompile XML
 - View Resource is inflated at runtime

Generating GUIs in Android - continued

- A lot of your GUI-related work will take place in:
 - res/layout
 - res/values
- `@+id/name_for_component` gives you handle for referencing XML declarations in code

```
Button button = (Button) findViewById(R.id.button1);
```

```
<Button  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:layout_weight="1"  
    android:text="Record Time"  
    android:id="@+id/button1"  
></Button>
```

Working with resources

- Application resources are stored under `res/`
- There are different types of resources
- String resources

– Saved in `res/values/` and accessed from the `R.string`

XML file saved at `res/values/strings.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

This layout XML applies a string to a View:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

This application code retrieves a string:

```
String string = getString(R.string.hello);
```

```
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

This application code retrieves a string array:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

<http://developer.android.com/guide/topics/resources/available-resources.html>

Working with resources - continued

- Color resources
 - Saved in `res/color/` and accessed from the `R.color` class

XML file saved at `res/values/colors.xml`:

```
<resources>
  <color name="opaque_red">#f00</color>
  <color name="translucent_red">#80ff0000</color>
</resources>

Resources res = getResources();
int color = res.getColor(R.color.opaque_red);
<TextView
  android:textColor="@color/translucent_red"
```

- Style resources
 - Saved in `res/values/` and accessed from the `R.style` class
- Layout resources

<http://developer.android.com/guide/topics/resources/available-resources.html>

Advantages of structured resources

- One can maintain resources independently and can group based on types
- Android selects which alternative resource to use at runtime
 - Depending on the current device configuration
- Help in providing alternative resources based on device types
- Localizing the applications

Supporting different screens

- Four generalized sizes
 - small, normal, large, xlarge
- Four generalized densities
 - low (ldpi), medium (mdpi), high (hdpi), extra high (xhdpi)
 - `res/layout`
 - `res/layout-large`
 - `res/layout-land-large`
 - `res/drawable-ldpi`
 - `res/drawable-hdpi`

<http://developer.android.com/training/basics/supporting-devices/screens.html>

Localizing the applications

- Supporting different languages
 - `res/values`
 - `res/values-en`
 - `res/values-fr`
- Help in localizing the applications
 - `res/drawable-de-rDE/`
 - `res/drawable-fr-rCA/`

<http://developer.android.com/training/basics/supporting-devices/languages.html>

Homework

- Play with life cycle methods
 - Have a layout with 2 EditTexts and 2 TextViews and try to manage the input under different conditions like application paused, resumed, stopped, destroyed.
- Have the same application dealing with language, localization and landscape movement issues
- Upload the project to course tasks
- Deadline: A day before the next lecture. So 8th September 2016.