MOBILE INTELLIGENCE 01

Android Basics - Quick Tour

Minsoo Kim (Pukyong Nat'l Univ.)

Major In Industrial Data System & Engineering

부산대학교 PUSAN NATIONAL UNIVERSITY
Department of
Industrial Engineering

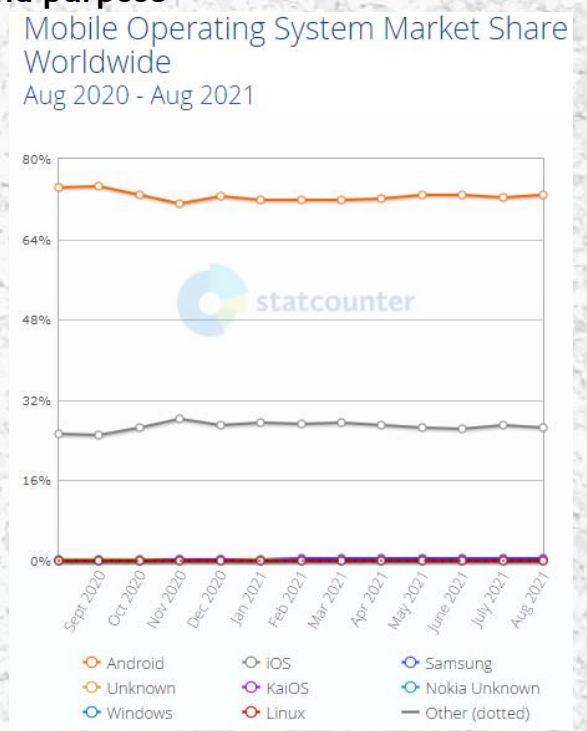
국립부경대학교 PUKYONG NATIONAL UNIVERSITY
Department of
Industrial and Data Engineering

Logistics of this Slide

- Smartphone, Mobile OS, App Market, Android Overivew
- Application Development
- Sample Project
- Modifying Sample Project

Smartphone

- Smart + Phone = Computer(Multimedia) + Network + Peripherals
  - Close Environment (Feature Phones) → Open Environment
    - Phone Vendor: provide additional peripherals with user environment
    - User: install Apps for user's own usage and purpose
    - Developer: easy distribution of Apps
  - Smartphone as an integrated device
    - All-in-one device for easy usage
    - Computation + Networking



- Mobile OS
  - Google Android vs. Apple iOS
  - Etc.

- Mobile Development Env.
  - 전용 IDE: Android Studio, Xcode IDE
  - Cross-Platform Mobile Dev.
    - Xamarin, PhoneGap, Appcelerator
    - NativeScript, Monocross, Cordova, …

Mobile OS Comparison

- Android (v10) vs. iOS (v13.3)
  - Android: Linux Kernel based OS
  - iOS: MacOS X 10.5 based OS (BSD + Mach Kernel)

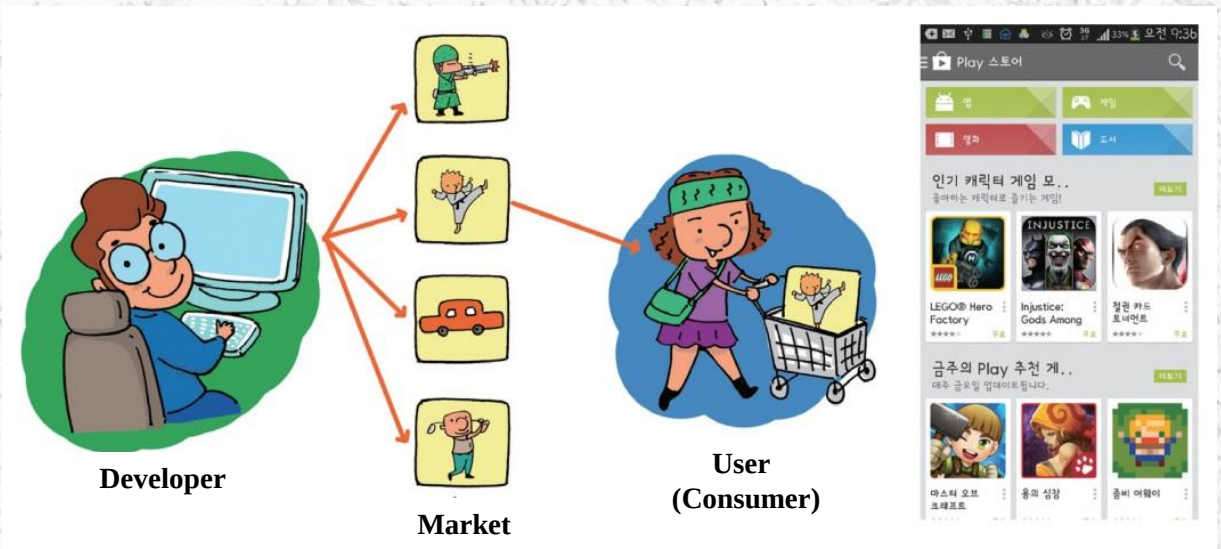
	Android	iOS
Provider	Google Open Handset Alliance	Apple Inc.
Dev. Language	Java(UI), Kotlin(UI), C(core), C++	C, C++, Objective-C, Swift
Recent Version	Android 11	iOS 15
Distn. Date	2020.09	2021. Fall (Scheduled)
Platform	32-bit 64-bit ARM, x86, x86-64	ARM
Kernel	Linux	Hybrid(XNU)
License	Apache 2.0	Proprietary
Web Site	android.com	apple.com/ios
User Interface	Graphical(multi-touch)	Cocoa Touch(Multi-touch, GUI)

(a) 안드로이드


(b) 아이폰

Application Market Concept

- Space for distribution and purchase of applications
  - Open trading space where developers and consumers meet
    - Android: Google Play Store
    - iOS: App Store
  - Sales revenue of the App + Ad. Revenue within the App
  - Network Effect → Customer(or developer) lock-in via platformization



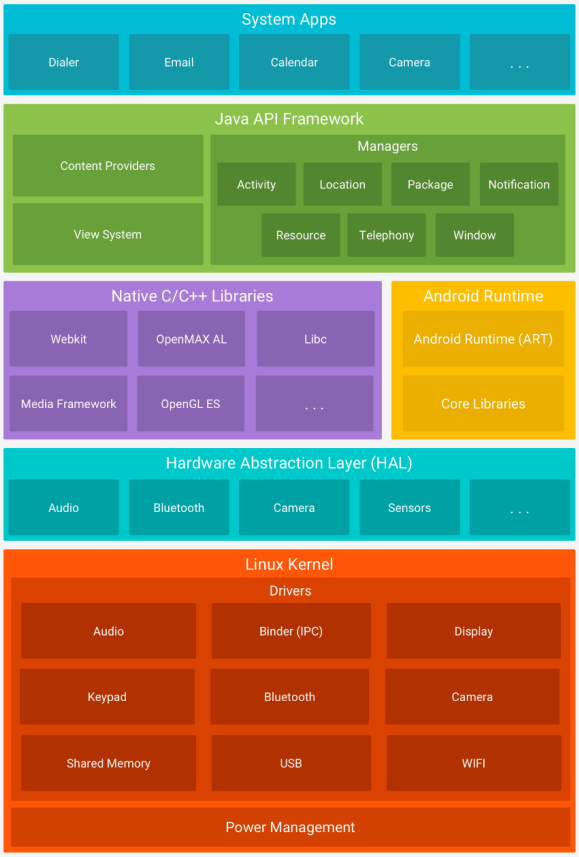
Android Version History - API Level

- https://en.wikipedia.org/wiki/Android_version_history

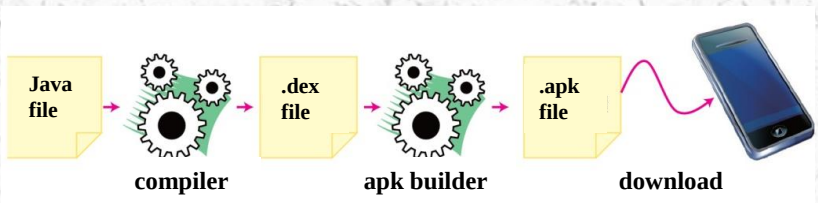
Code name	Version number(s)	Initial release date	API level	References
No codename	1.0	September 23, 2008	1	[9]
	1.1	February 9, 2009	2	[9][11]
Cupcake	1.5	April 27, 2009	3	
Donut	1.6	September 15, 2009	4	[12]
Eclair	2.0 – 2.1	October 26, 2009	5 – 7	[13]
Froyo	2.2 – 2.2.3	May 20, 2010	8	[14]
Gingerbread	2.3 – 2.3.7	December 6, 2010	9 – 10	[15]
Honeycomb	3.0 – 3.2.6	February 22, 2011	11 – 13	[16]
Ice Cream Sandwich	4.0 – 4.0.4	October 18, 2011	14 – 15	[17]
Jelly Bean	4.1 – 4.3.1	July 9, 2012	16 – 18	[18]
KitKat	4.4 – 4.4.4	October 31, 2013	19 – 20	[19]
Lollipop	5.0 – 5.1.1	November 12, 2014	21 – 22	[20]
Marshmallow	6.0 – 6.0.1	October 5, 2015	23	[21]
Nougat	7.0 – 7.1.2	August 22, 2016	24 – 25	[22][23][24][25]
Oreo	8.0 – 8.1	August 21, 2017	26 – 27	[26]
Pie	9.0	August 6, 2018	28	[27]
Android 10	10.0	September 3, 2019	29	[28]

Android Architecutre & Application Execution

- Android Software Stack: 'developer.android.com/guide/platform/'
  - System Apps
  - Java API Framework
  - Native C/C++ Libraries
  - ART (Android Runtime)
  - HAL (Hardware Abstraction Layer)
  - Linux Kernel

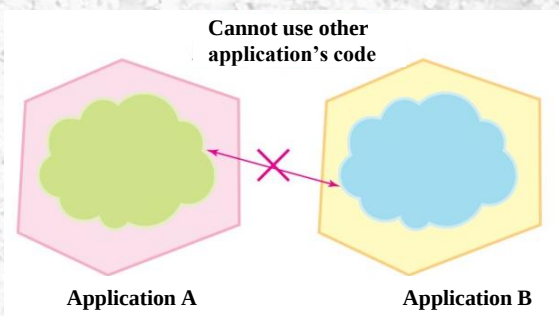


- Application Execution
  - Java: use VM to support multiple H/W
  - Google's new VM development
    - Dalvik VM → migrate to ART(v4.4)
    - Dex (Dalvik Executable) file

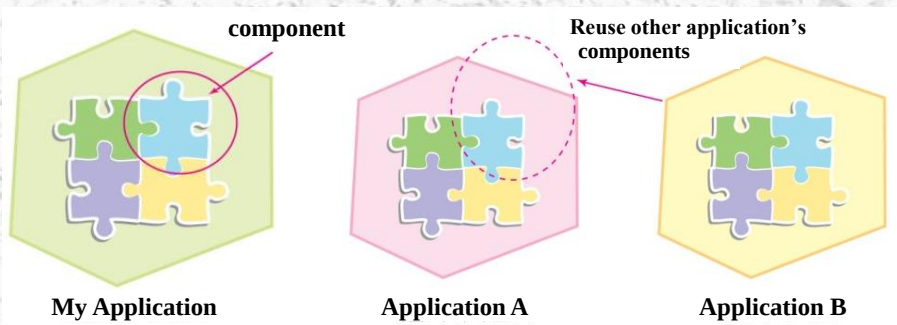


Basic Concept of Android Application

- Usual PC Applications
  - Strict separation between applications
    - install & execution is basically separated.
  - Self-sufficient Application codes
    - Except external shard libraries
  - Build separate library(DLL) for reuse

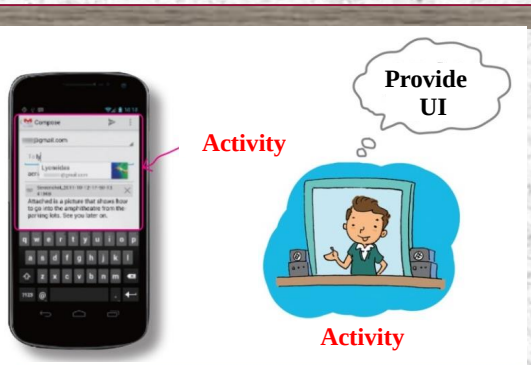


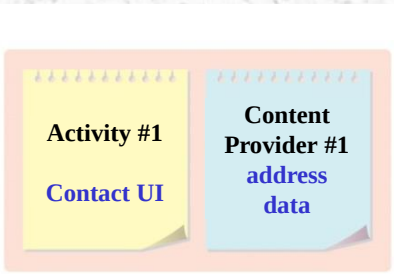
- Android Application = coalition of (reusable) component
  - Designed to reuse other application's compoents
    - Save memory & storage space, facilitate App development via reusability
  - 4 types of component
    - Activity
    - Service
    - Broadcast Receiver
    - Content Provider



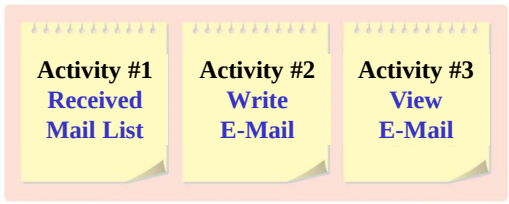
Android Application Components

- Activity: unit task having UI screen
- Service: background task/service
- Broadcast Receiver: react to notification
- Content Provider: give data to others

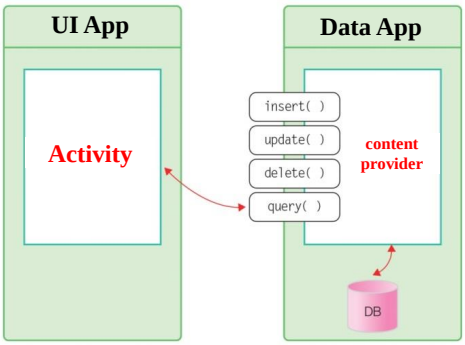


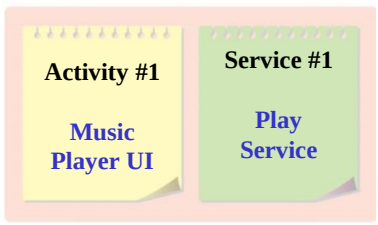


Contact Application



E-Mail Application

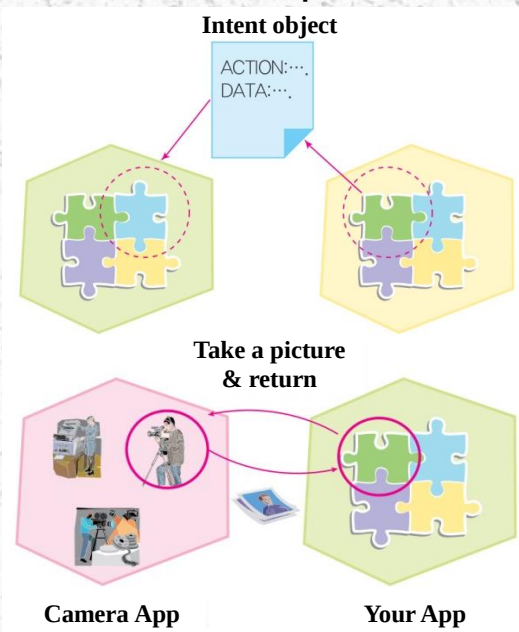


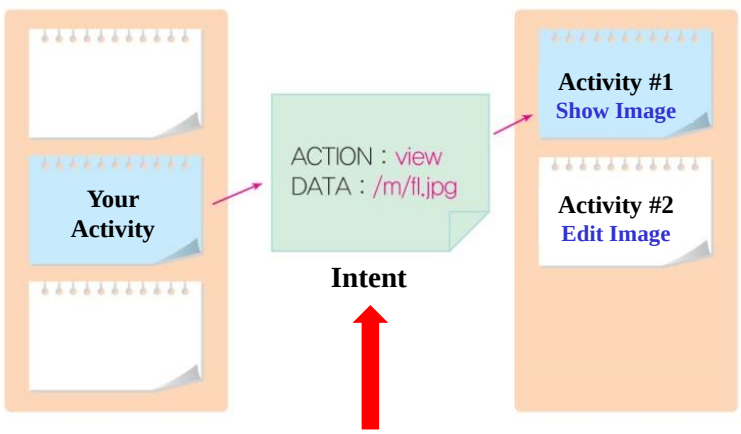


Media Player Application

Inter-operation between Android Components

- Brokerage between Components: Android System
  - Async-message using Intent: Android finds and invokes component that best fits to requested intention and returns its result to the caller.
  - eg: when your app needs to take a 'camera snapshot'
    - Access camera from your app and implement all necessary functions
    - Reuse provided Camera App's snapshot function





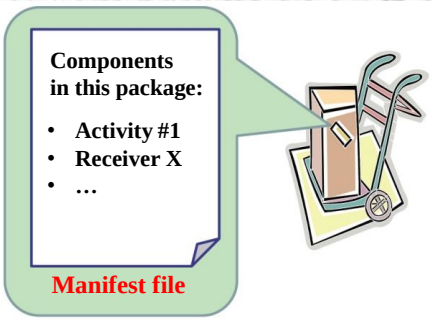
- Intents are reusable whether they are system-defined or application defined.
- Invocation result is also delivered via Intent.

Use of XML

- What is an XML(eXtensible Markup Language)?
  - Mark-up language for definition and processing of structured text documents over the Web
    - Document, Element, Attribute
  - Android actively use XML internally in various ways.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
  <application android:icon="@drawable/ic_launcher" . . . >
    <activity android:name="kr.co.company.MainActivity"
        android:label="@string/app_name"
        . . . >
    </activity>
    . . .
  </application>
</manifest>
```

Describe Activity

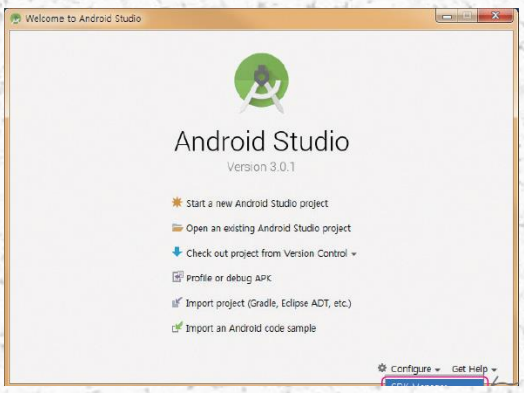


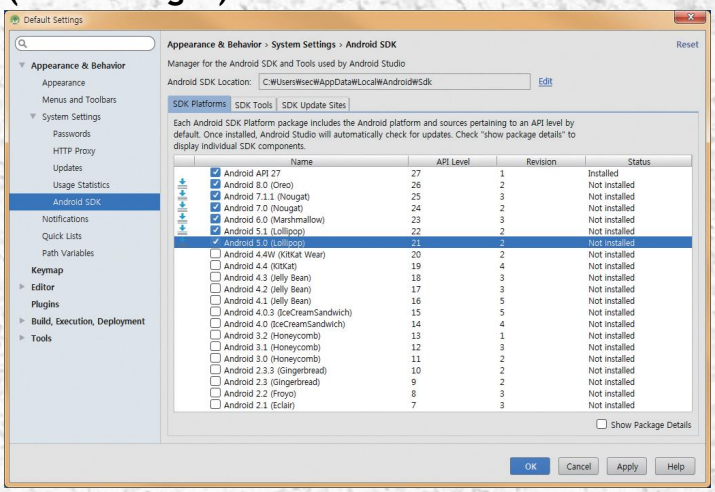
  - Android Application's AndroidManifest.xml file
    - Content list of Package: App's components, permission, settings …
  - Declarative UI: XML-defined GUI
    - Separate program codes and UI composition by using XML
    - res/layout folder
  - Resource separation
    - string, color, …

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hello, world!" />
```


Android Development Process & Tools

- Install Dev. Env. → Code → Debug(& Test) → Distribute!
  - Install Android Development Tool
    - JDK(SE) → Android Studio → Android SDK/Tools → prepare Device(or AVD)
    - http://java.oracle.com, http://developer.android.com
    - JDK Install: JDK v8 or above
    - SDK Install: Android Studio (SDK Manager)
      - Choose target version

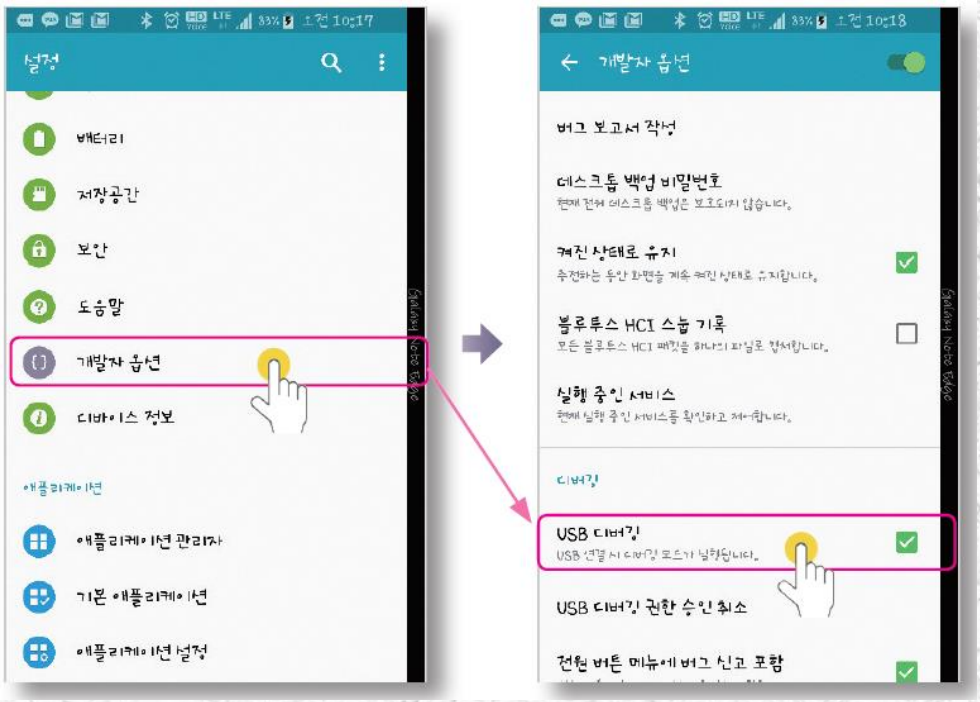




    - Create AVDs
      - Check your phone's specification → Android Studio (AVD Manager)
    - Turn-on developer mode from your phone: vendor's model dependent
      - (google it!) Settings → Developer

Connecting Physical Device

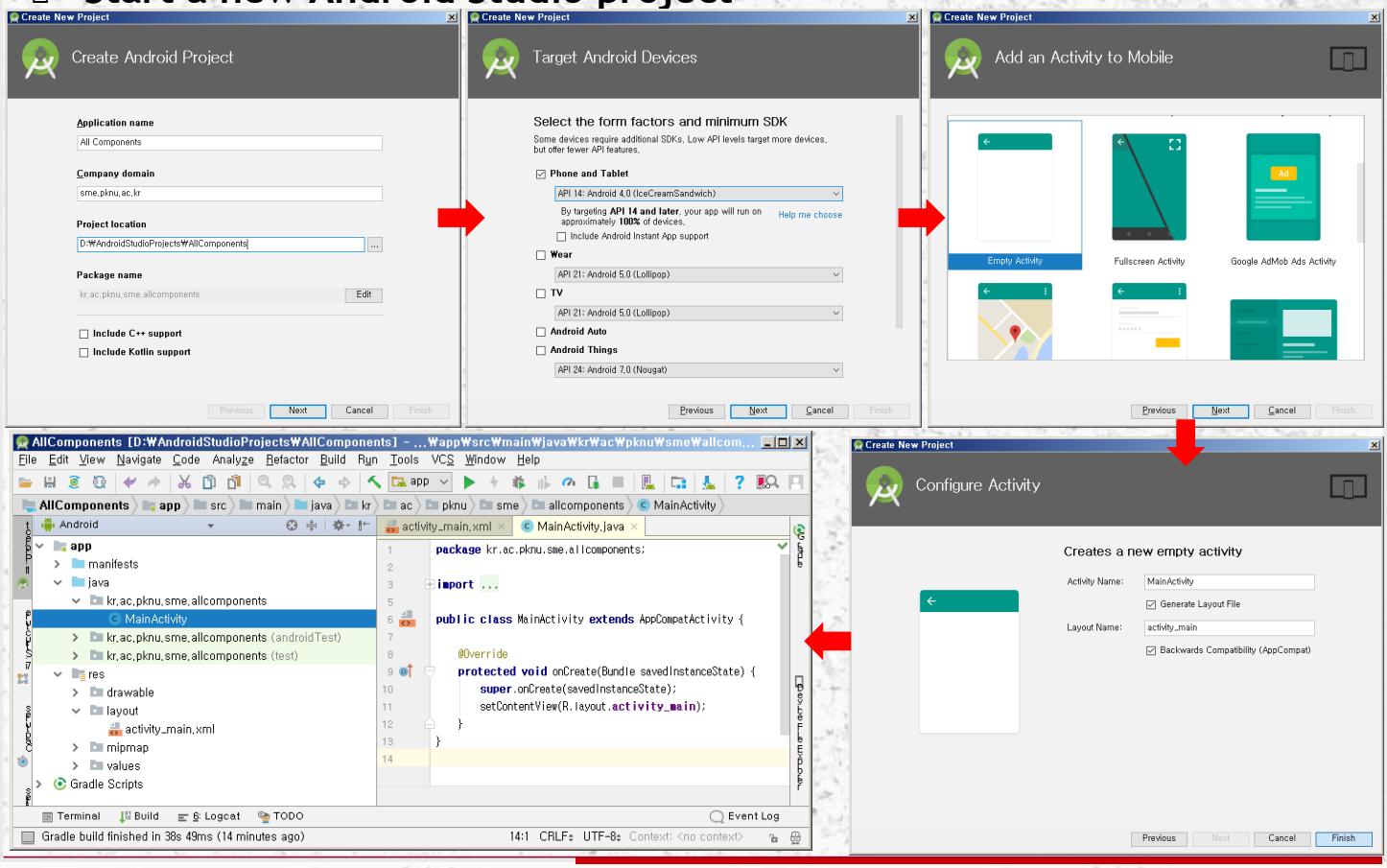
1. Install USB Driver (auto-detect or google it if necessary)
2. Turn-of "USB debugging" option at the phone.



3. Choose connected device at the Android Studio when runs an App

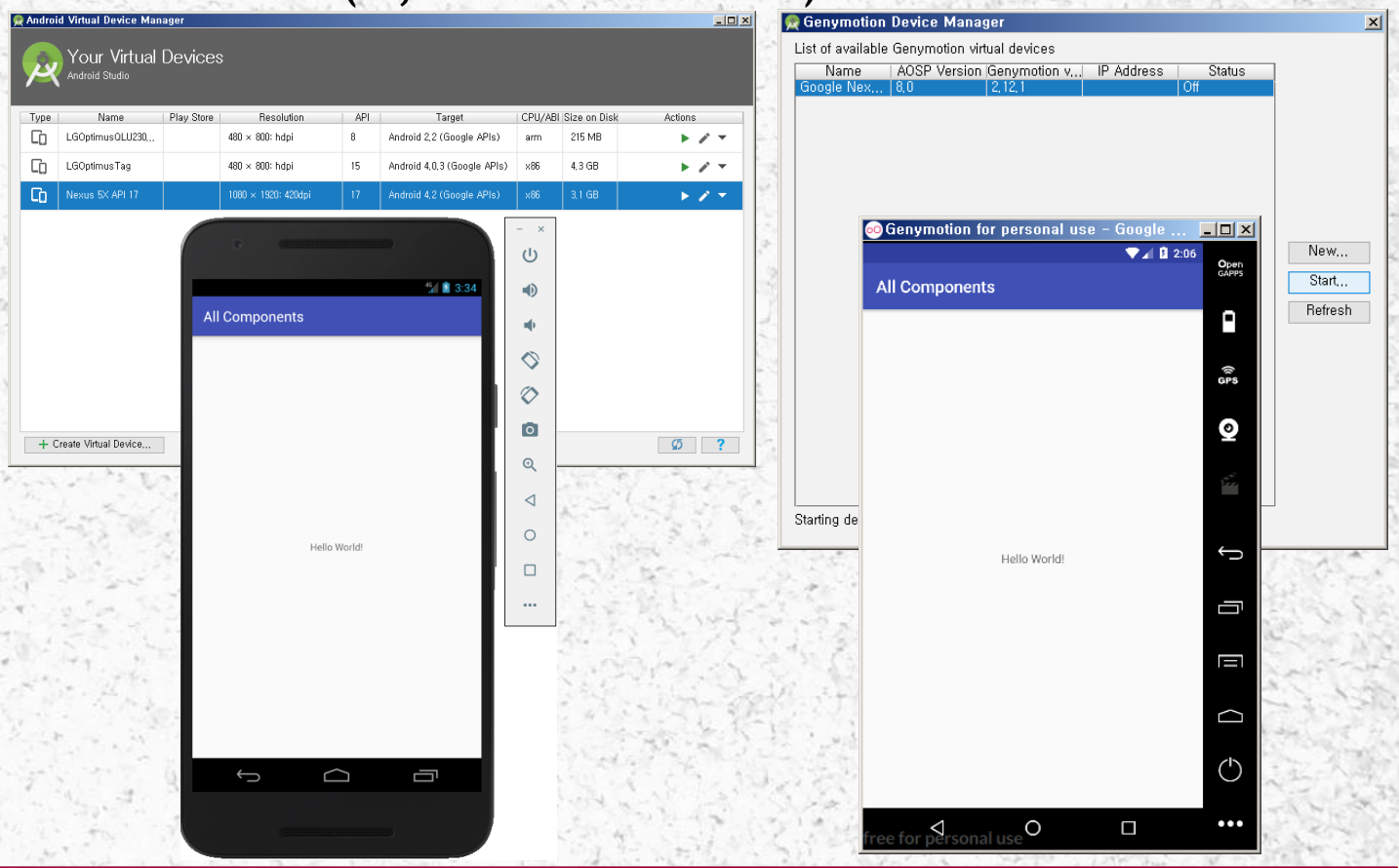
Create Your First Android Application!

- Start a new Android Studio project



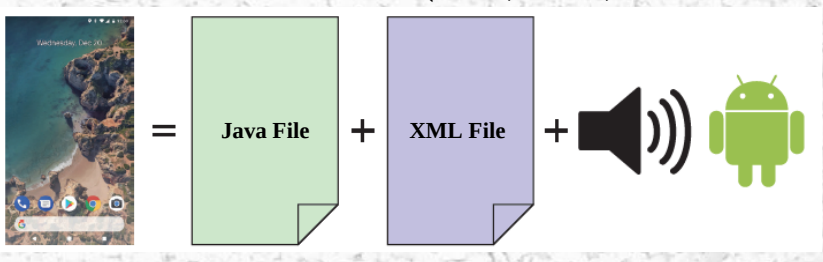
First Run

- Use emulator (or, at the actual device)



Application Development: Overview

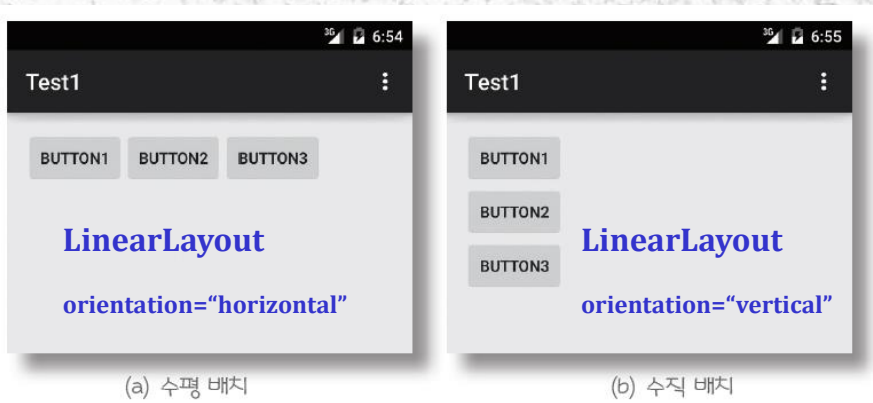
- Building Android Application = Combining Components
  - Activity(GUI composition) + Service(background task support) + Content Provider(data processing) + Broadcast Receiver(situation awarness)
- Application's Compositon
  - Java Source Code + Resouce Files(XML, Icon, multimedia, …)



- Steps: GUI design → Coding → Manifest edit (→ Test & Distribute)

Application Development: 1. UI Design

- Start with Android Studio's auto-generated template layouts.
  - basic layout file: res/layout/activity_main.xml
    - Add additional layout files if you need more Activities (or UIs)!
  - Use Layout (ViewGroup) class and View to compose GUI (in XML)
    - LinearLayout*
    - TableLayout
    - GridLayout
    - RelativeLayout
    - TabLayout
    - AbsoluteLayout
    - FrameLayout
    - ConstraintLayout*

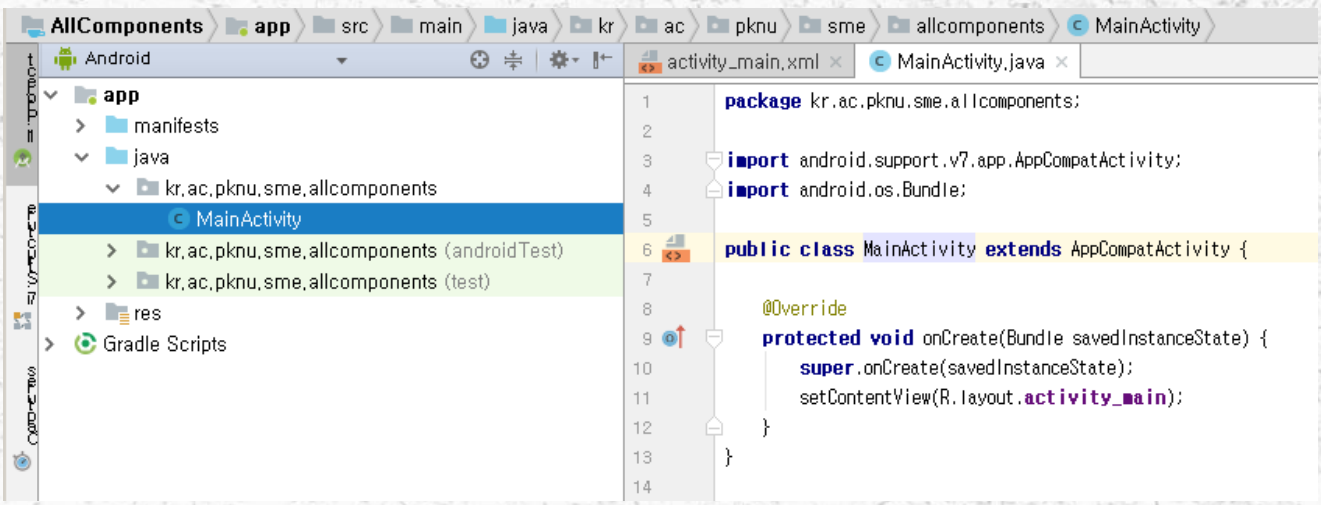


(a) 수평 배치 (b) 수직 배치

  - Layout and View's can be manipulated later by program code.
    - Relocation or dynamic handling Layout
    - Animation
    - Use of Custom View class, …

Application Development: 2. JAVA coding

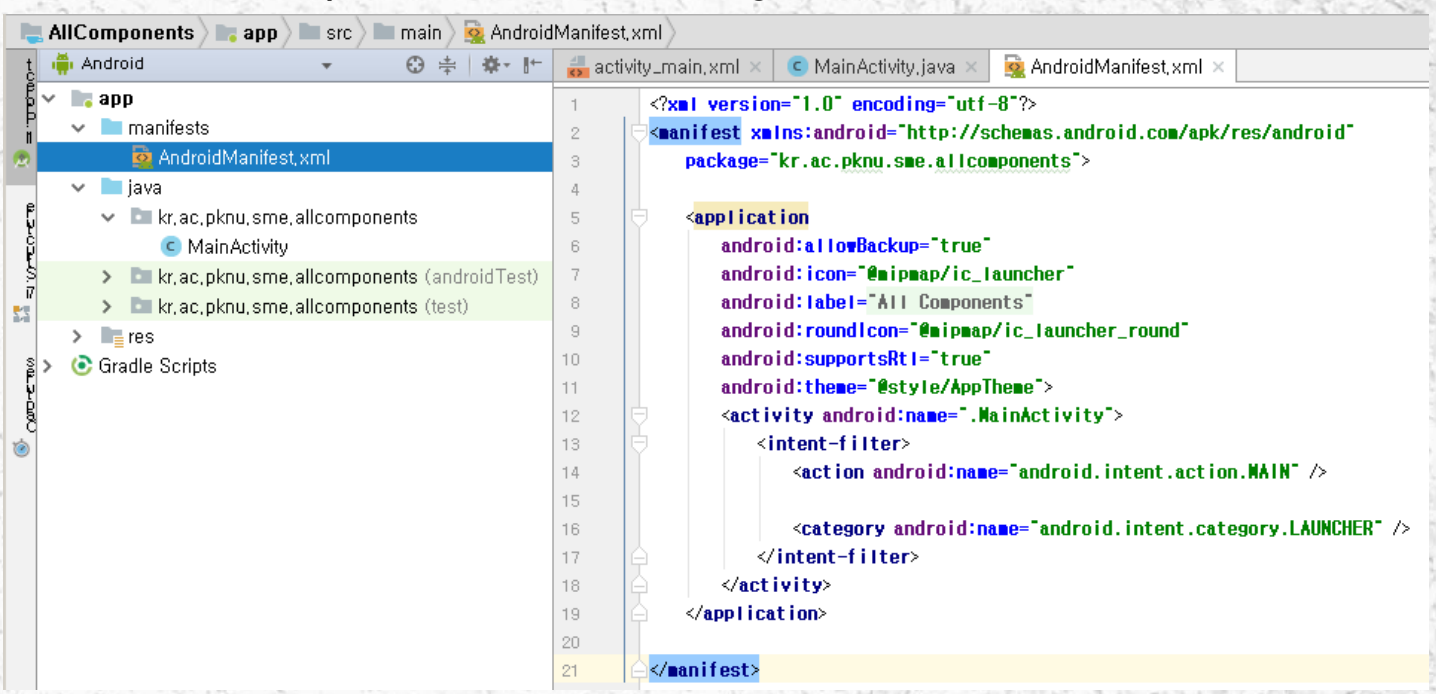
- Start with Android Studio's auto-generated template codes.
  - basic Activity file: app/java/<Your_Package>.MainActivity.java
    - Inherit AppCompatActivity (… → Activity) class



  - Change JAVA code, add components or classes.
    - Application's 4 component types and corresponding classes.
    - User defined Custom View class
  - Refer resources.
    - R.layout.activity_main, @string/hello

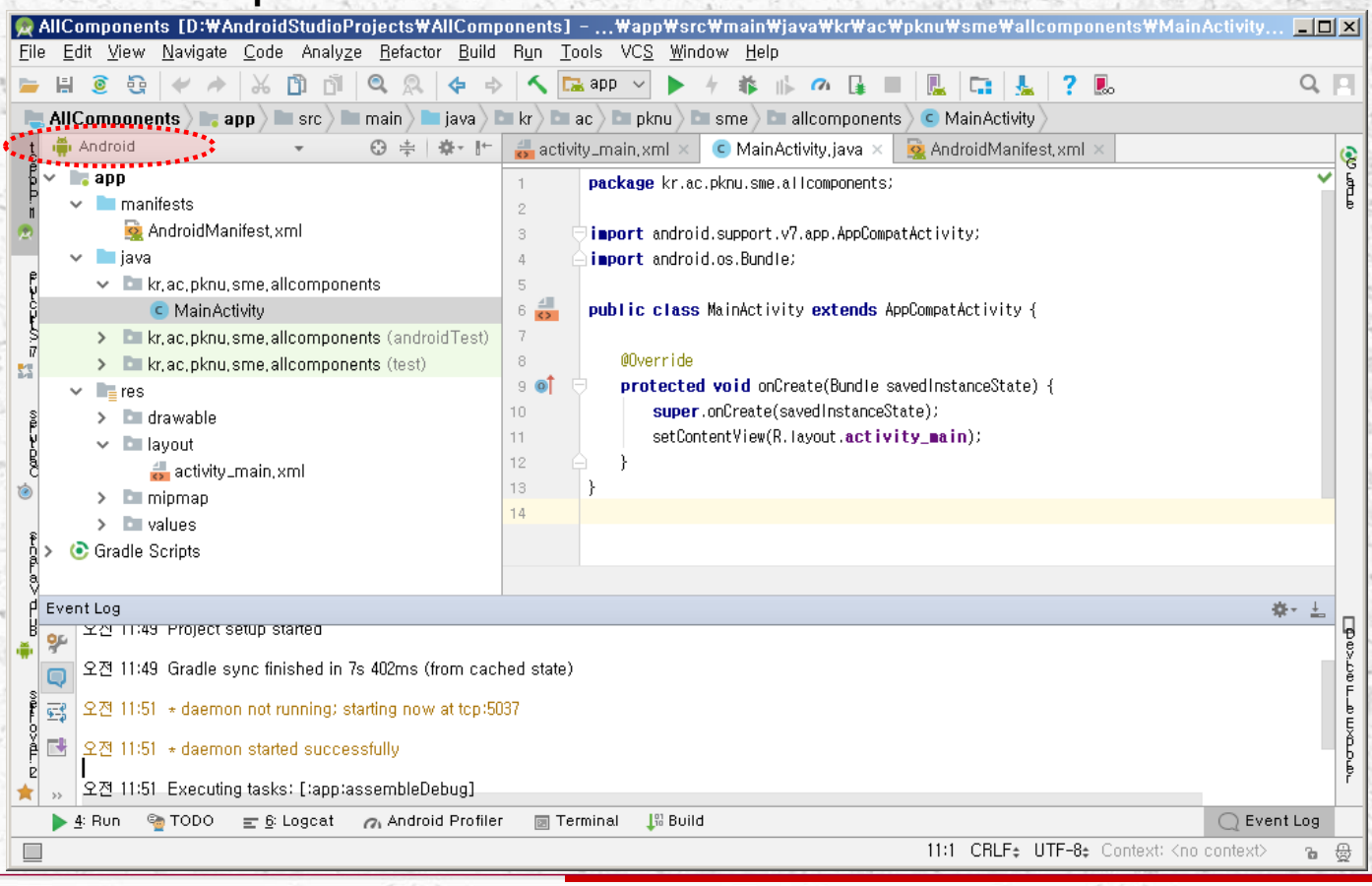
Application Development: 3. Edit Manifest

- Start with Android Studio's auto-generated manifest file.
  - basic Manifest.xml: app/manifests/AndroidManifest.xml
    - Define all components that constitute Application and their properties.
    - Access permission and meta-setting for Device control.



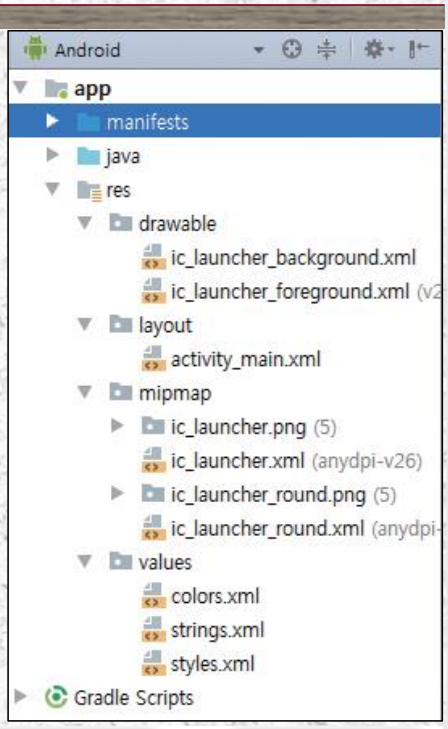
Sample Project: IDE & Project Hierarchy

- Basic Perspective: Android Studio's 'Android View'



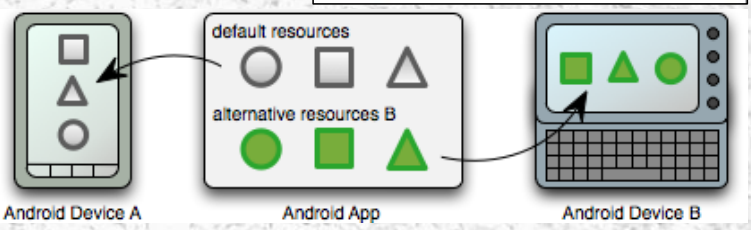
Sample Project: IDE & Project Hierarchy

- Android View: app, Gradle Scripts (folder)
  - manifest: AndroidManifest.xml file
  - java: source files
  - res: resource files used for app
  - Gradle Scripts: Gradle build scripts



- app\res: layout, image, string …
  - Mostly XML files
  - Overlapped files for each device configurations

- Separation of codes and resources!
  - Very important for configuration management.
  - With the introduction of many Android devices, supporting different display size, density, locale, etc. should be handled without changing the codes



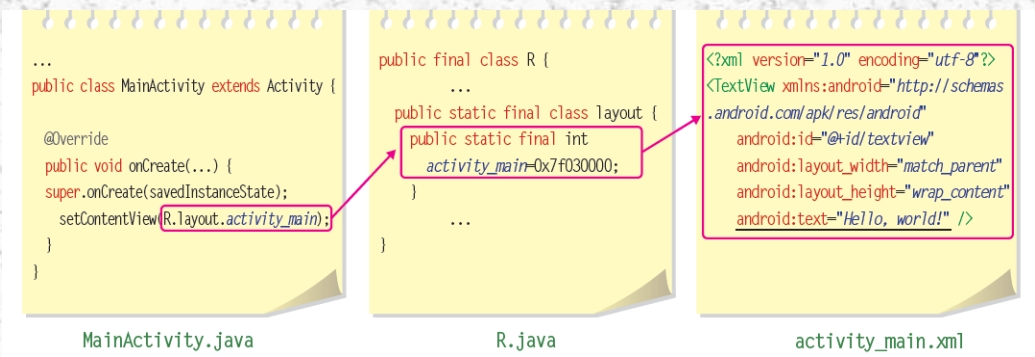
Sample Project: IDE & Project Hierarchy

- R.class: linking codes & resource (app/generatedJava/<package>/R)

MainActivity.java

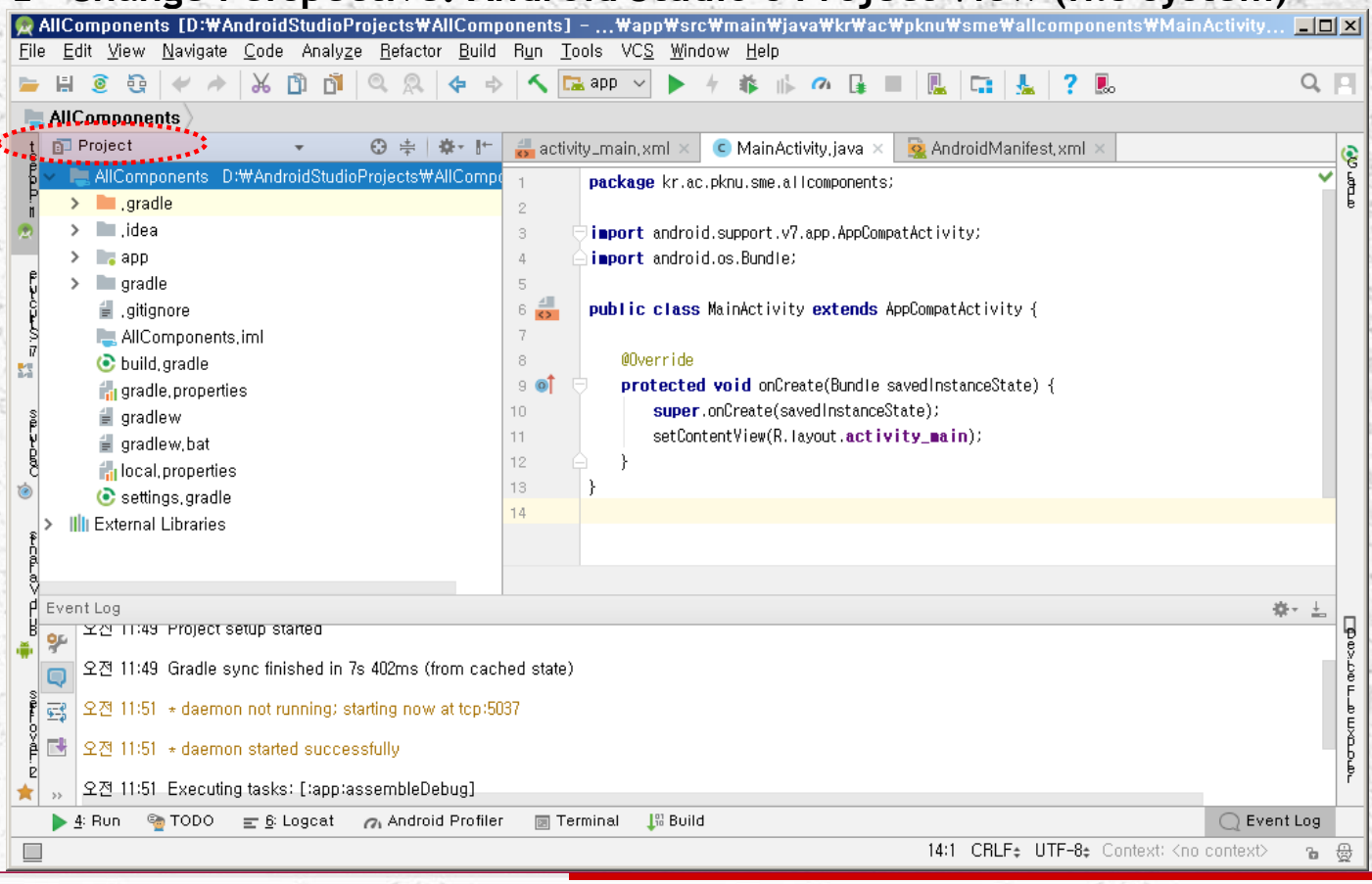
```
package kr.co.company.hello3;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

액티비티의 화면을 activity_main.xml로 설정



Sample Project: IDE & Project Hierarchy

- Change Perspective: Android Studio's Project View (file system)



Sample Project: JAVA code

- MainActivity.java
  - package declaration: folder-wise JAVA class structuring
    - Reverse your Internet domain name(sme.pknu.ac.kr)
  - import statements: use of external packages & classes
    - Use of JDK(except AWT & Swing packages), Android & external packages
    - Auto-Import in Android Studio: 'ALT + ENTER'
  - public class MainActivity extends AppCompatActivity { ... }
    - Inherit AppCompatActivity class: Activity component responsible for UI
    - There's no 'public static void main(String[] args) { ... }'
    - @Override: Annotation(compiler hint, ok to remove!)
    - onCreate(Bundle ...) { ... }
      - Starting point of Activity & set-up
      - Called just once for UI build-up

Activity interacts with user via screen.

```
package kr.ac.pknu.sme.allcomponents;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

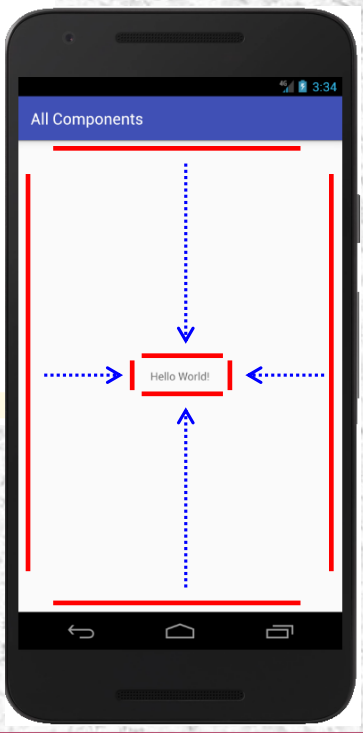
Sample Project: Layout

- Basic layout file: res/layout/activity_main.xml
  - Use of android.support.constraint.ConstraintLayout

activity_main.xml | MainActivity.java

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </android.support.constraint.ConstraintLayout>
```

layout constraints



  - Use of layout in the JAVA code
    - setContentView(R.layout.activity_main);
      - Setup the UI for this Activity
      - R.layout.activity_main: activity_main.xml file
  - find R.class (Android Studio's Project View)
    - app/build/generated/source/r/debug/<package>/R

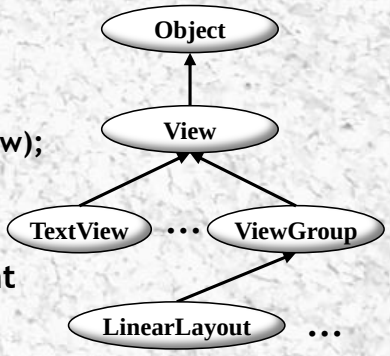
Sample Project: Layout

- XML Quick Overview
  - Document Tag + Root Element
    - Document Tag: <?xml version="1.0" encoding="utf-8"?>
    - Element: encompass all the logical components between start-tag & end-tag
      - <Greeting>Hello, world.</Greeting>, <MyElement />
    - Attribute is for additional properties for Element: "name=value" in start-tag
      -
  - NameSpace:NamePrefix
    - Introduced to separate URI spaces by prefixing Element & Attribute.
      - xmlns:android="http://schemas.android.com/apk/res/android"
      - xmlns:app="http://schemas.android.com/apk/res-auto"

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="horizontal"
    tools:context=".MainActivity">
</LinearLayout>
```

Sample Project: Layout

- GUI description using XML
  - View(UI component, Widget) and ViewGroup(Layout)
    - Each UI component(View) corresponds to one Element in layout XML
      - TextView component: corresponds to <TextView … /> element in layout XML file.
      - public class View extends Object implements Drawable.Callback, …
    - A layout element is for holding multiple UI components.
      - <LinearLayout><Button … />…<TextView … /></LinearLayout>
      - public abstract class ViewGroup extends View implements ViewParent, …
  - TextView attributes
    - xmlns:android ← XML namespace declaration for Android documents
      - <TextView android:text="Hello World" … />
    - android:id ← assign unique ID to the element
      - android:id="@+id/myTextView"
      - TextView tv = (TextView) findViewById(R.id.myTextView);
    - android:layout_width ← define the width of element
      - "match_parent" or "wrap_content"
    - android:layout_height ← define the height of element
      - "match_parent" or "wrap_content"
    - android:text ← text value to be shown in the screen



Sample Project: Layout

- Designing UI
  1. Using codes(add UI components programmatically on runtime)

```
TextView tv = new TextView(this);      <…… Construction of Java object
tv.setText("Hello, world!");           <…… Set the property of Java object
```

  2. Using XML separately from the code(recommended way)
     - Separate UI looks from the application logics.
     - Divide roles between developer & designer → fast UI design, high maintainablity
  - Usually: define UI in XML, manipulate later in Java code

activity_main.xml

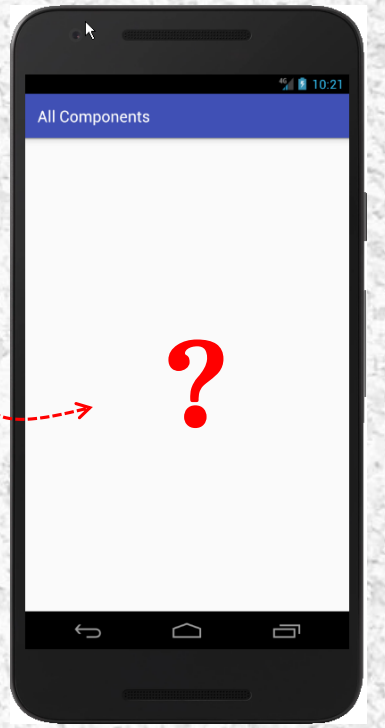
```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textview"                    <…… Assigning ID enables the reference of this object is enlisted in the R.class file.
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Hello, world!" />
```

```
TextView tv = (TextView) findViewById(R.id.textview);   <…… enlisted in R.class
tv.setText("Hello, world!");                            <…… Set Java object property
```

Modify Sample Project: JAVA code

- MainActivity.java
  - super.onCreate(…)
    - Set-up for Activity using parent method(AppCompatActivity → … Activity)
    - This should be called first of all!
  - setContentView(…) ← comment out, run and see!
    - Comment out: declaring not to use any layout!

```
1   package kr.ac.pknu.sme.allcomponents;
2
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11  //          setContentView(R.layout.activity_main);
12      }
13  }
14
```



    - Designate other layout xml for different UIs
      - setContentView(*R.layout.my_new_layout*)
  - Uncommont setContentView(…) and save.

Modify Sample Project: JAVA code

- Modify 'res/layout/activity_main.xml' & Run
  - change: ConstraintLayout → LinearLayout (horizontal)

activity_main.xml | MainActivity.java

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       tools:context=".MainActivity">
8
9       <TextView
10          android:layout_width="wrap_content"
11          android:layout_height="wrap_content"
12          android:text="Hello World!"
13          app:layout_constraintBottom_toBottomOf="parent"
14          app:layout_constraintLeft_toLeftOf="parent"
15          app:layout_constraintRight_toRightOf="parent"
16          app:layout_constraintTop_toTopOf="parent" />
17
18  </android.support.constraint.ConstraintLayout>
```

delete

activity_main.xml | MainActivity.java | AndroidManifest.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto"
4       xmlns:tools="http://schemas.android.com/tools"
5       android:layout_width="match_parent"
6       android:layout_height="match_parent"
7       android:orientation="horizontal"
8       tools:context=".MainActivity">
9
10      <TextView
11          android:layout_width="wrap_content"
12          android:layout_height="wrap_content"
13          android:text="안녕하세요!" />
14
15  </LinearLayout>
```

add

Change text

  - Add Button to Layout
    - Use Design Tab
    - Verify at the Text Tab
    - Run and check the UI

Modify Sample Project: Layout

- (res/layout/activity_main.xml)

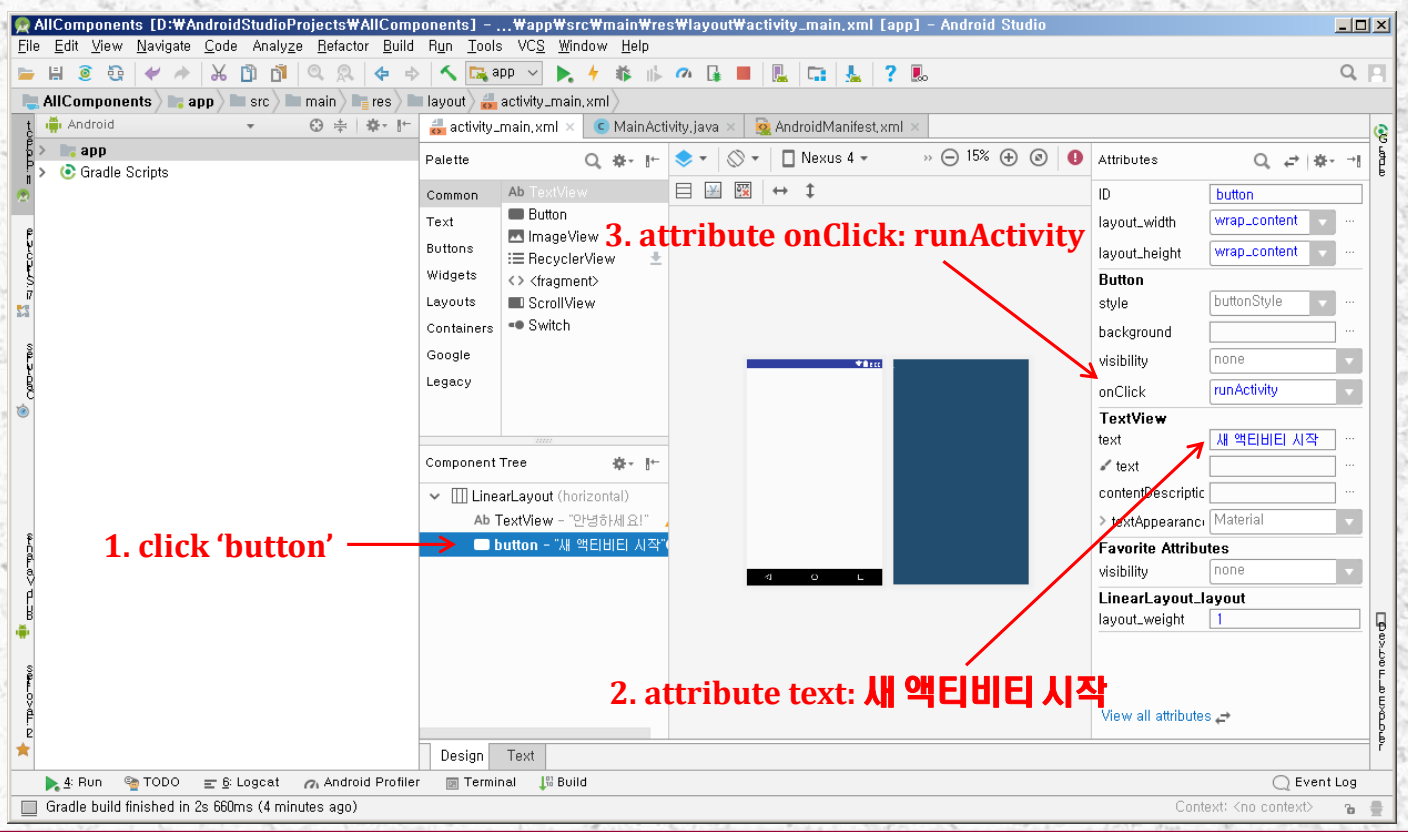
activity_main.xml | MainActivity.java | AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout 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"
    android:orientation="horizontal"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="안녕하세요!" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button" />

</LinearLayout>
```

- Change Button's text value
- Change Button's layout_width, layout_height to 'match_parent'
- Change Button's layout_weight value to '0'(or delete)
- Change LinearLayout's orientation to 'vertical'

Modify Sample Project : Event Handling

- Use Visual Tool
  - Set Button's text * onClick attributes.



Modify Sample Project : Event Handling

- Add event handler to MainActivity.java file.
  - add 'public void runActivity(View v) { … }' method
    - 'Alt + Enter' → auto-import!
    - Specify task for button click.
  - Show Toast message
    - Plain notification w/o interaction.

```
1  package kr.ac.pknu.sme.allcomponents;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5  import android.view.View;
6  import android.widget.Toast;
7
8  public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14     }
15
16     public void runActivity(View v) {
17         Toast message = Toast.makeText(this, "You clicked button!", Toast.LENGTH_LONG);
18         message.show();
19     }
20 }
21
```

auto-import
Alt + Enter

```
1  package kr.ac.pknu.sme.allcomponents;
2
3  import android.support.v7.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_main);
12     }
13
14     public void runActivity(View v) {
15
16     }
17 }
```

? android.view.View? Alt+Enter

Add event handler

Modify Sample Project : Event Handling

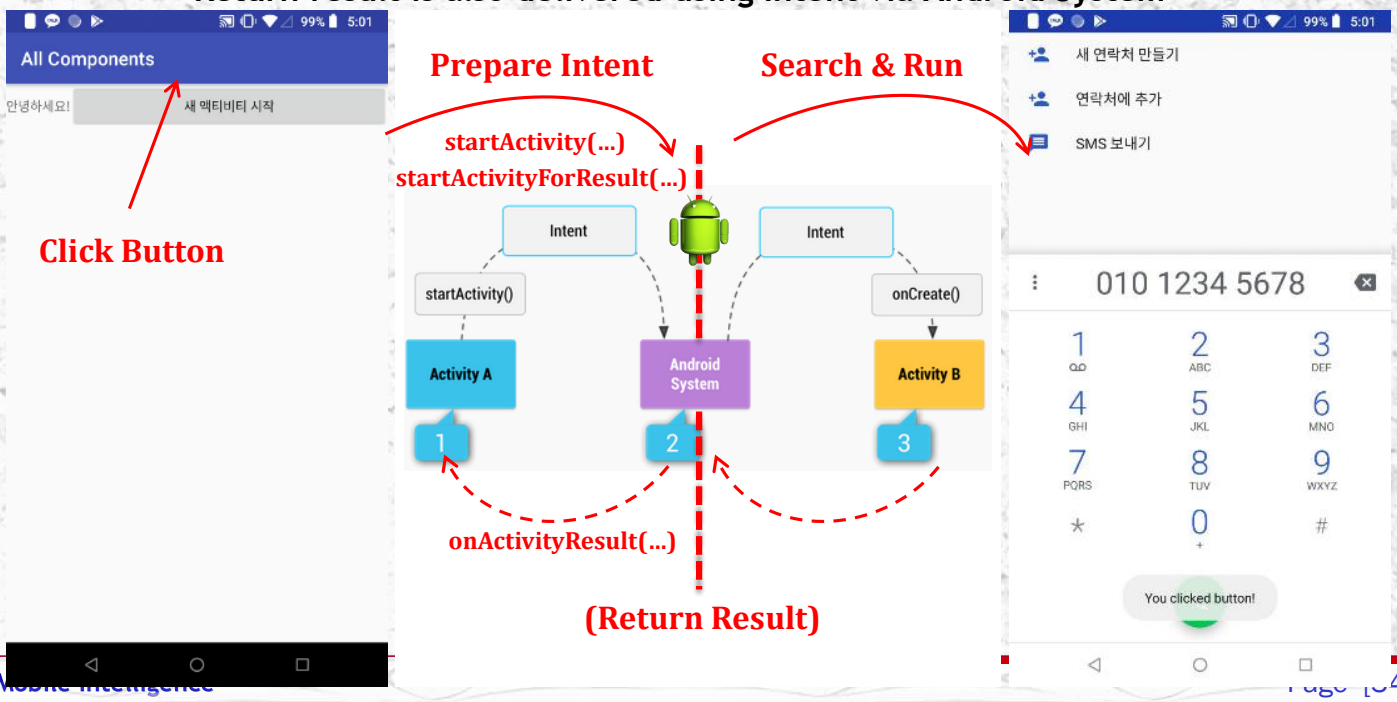
- Using Button's onClick attribute
  - See the result and change the message & location of Toast
    - https://developer.android.com/guide/topics/ui/notifiers/toasts#java
  - Toast.makeText(context, text, duration)
    - duration: set LENGTH_SHORT
    - Check pre-set object using getView() method
    - Customize with setView(…) method
      - Layout inflate → Toast creationg → setView(…)
- Edit MainActivity.runActivity(View v) { … }
  - Start Activity using Intent & startActivity(…)
    - Start 'Dialer Activity'
    - Call other app's Activity via Intent
  - Import Intent & Uri class ← 'Alt + Enter'
  - Save & Run Project

```
public void runActivity(View v) {
    Toast message = Toast.makeText(this, "You clicked button!", Toast.LENGTH_LONG);
    message.show();

    Intent call = new Intent(Intent.ACTION_VIEW, Uri.parse("tel:010-1234-5678"));
    startActivity(call);
}
```

add

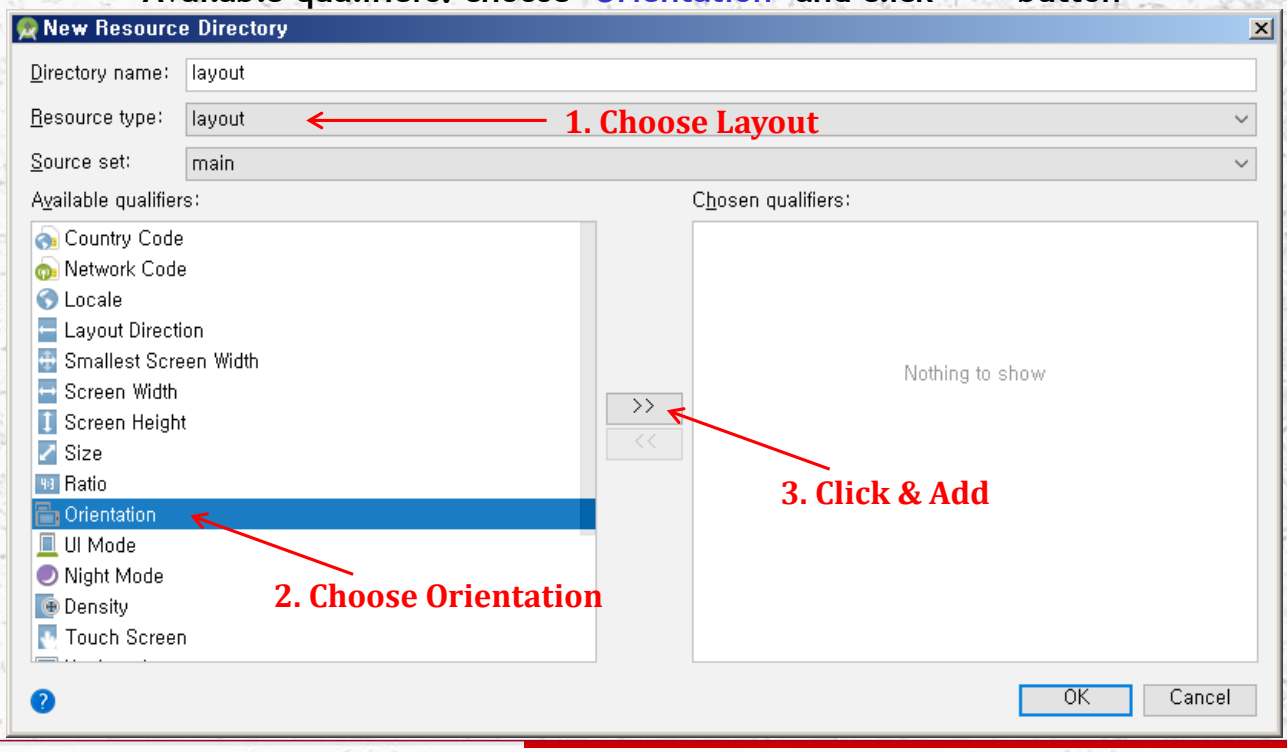
Modify Sample Project: Event Handling

- Start an Activity using Intent
  - Activity: UI component that represents individual screen
    - Running an Activity: asking Android system by sending an Intent object
      - Instancing an Intent object → set desired task → call startActivity(…)
      - Always need Android System's brokerage
    - Return result is also delivered using Intent via Android System



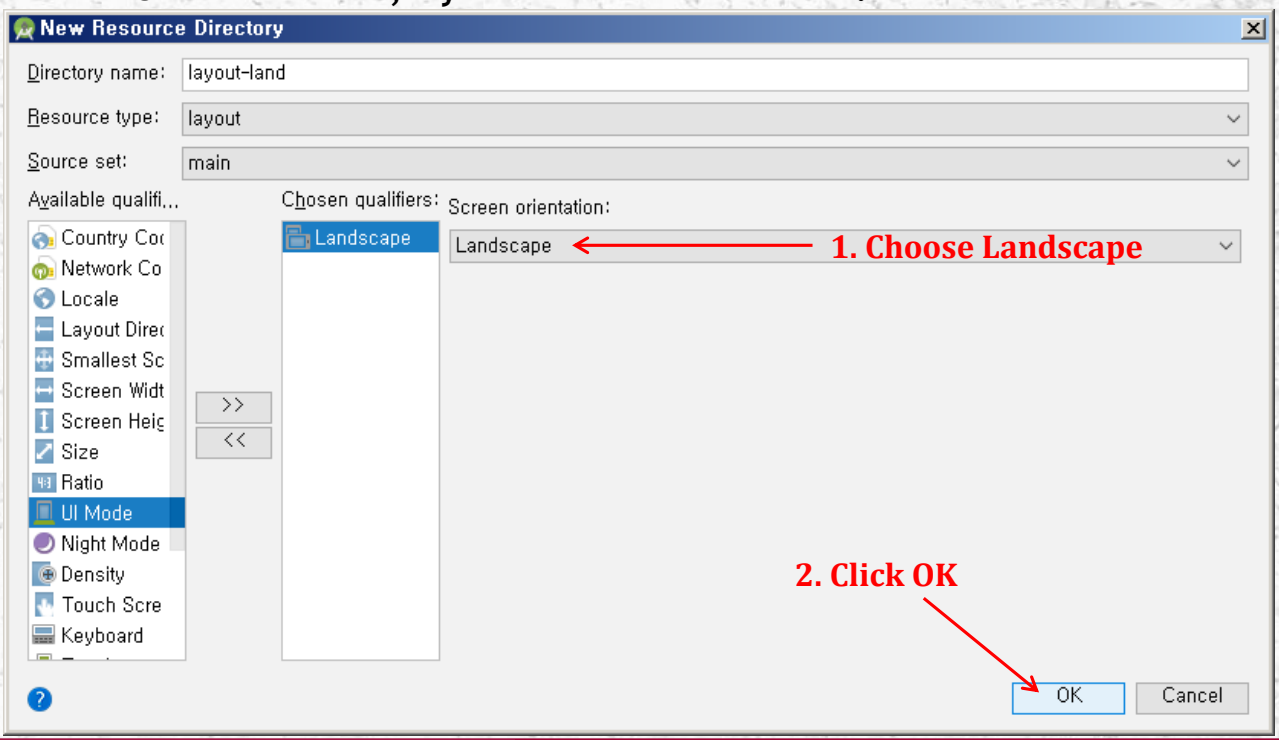
Modify Sample Project: Using Resource

- Adding layout for landscape mode
  - Right-click 'res' folder → New → Android Resource Directory
    - Resource Type: choose 'layout'
    - Available qualifiers: choose 'Orientation' and click '>>' button



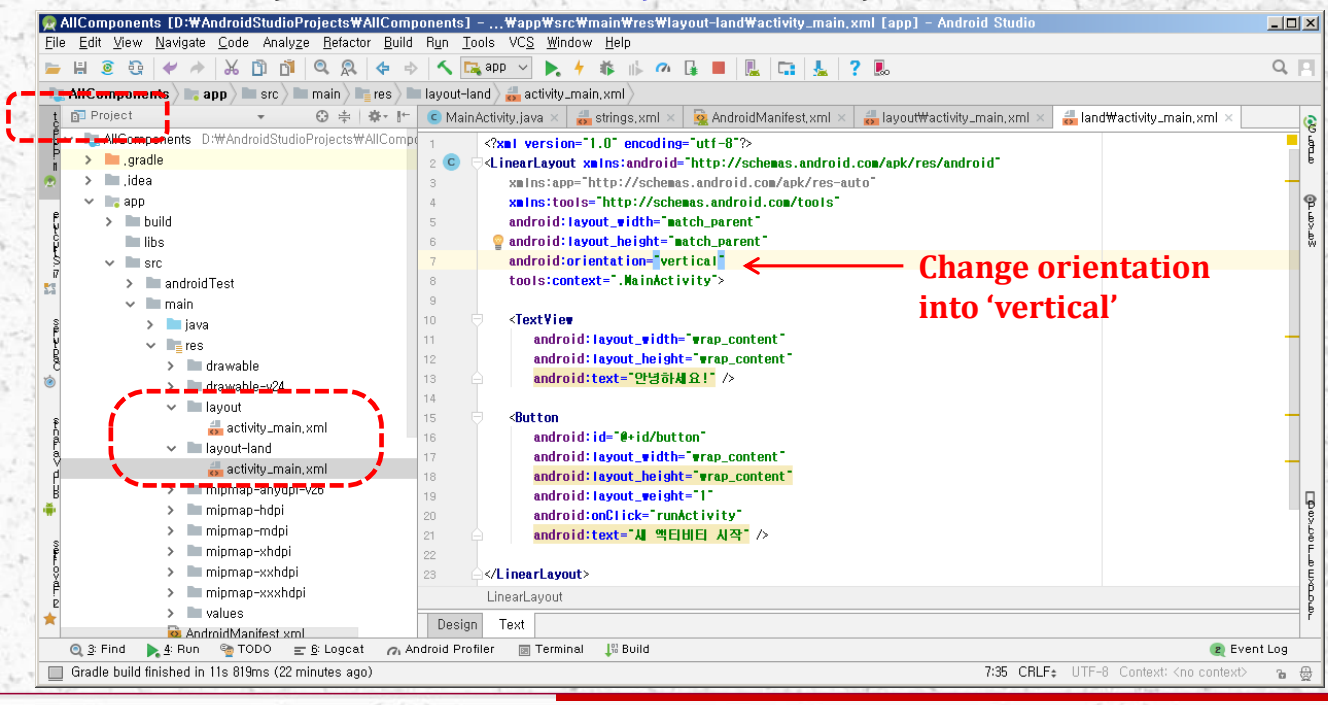
Modify Sample Project: Using Resource

- (Adding layout for landscape mode)
  - In the 'New Resource Directory' dialog
    - Screen orientation: choose 'Landscape' and click 'OK' button
    - Under res folder, layout-land folder is created.



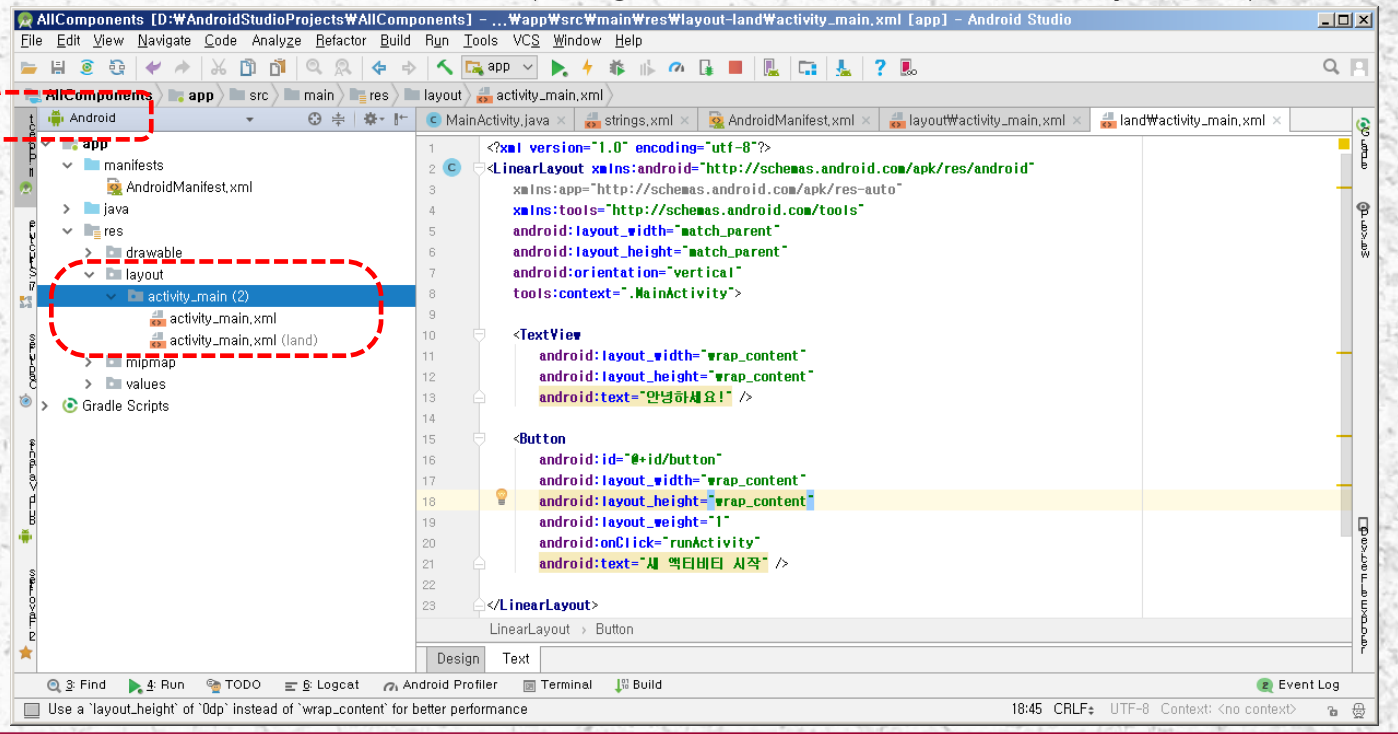
Modify Sample Project: Using Resource

- (Adding layout for landscape mode)
  - Check 'app/src/main/res' folder from the 'Project View'
    - Right-click 'app/src/main/res/layout/activity_main.xml' file & copy, then paste into 'app/src/main/res/layout-land' folder via right-click
    - Edit layout-land foler's activity_main.xml layout file



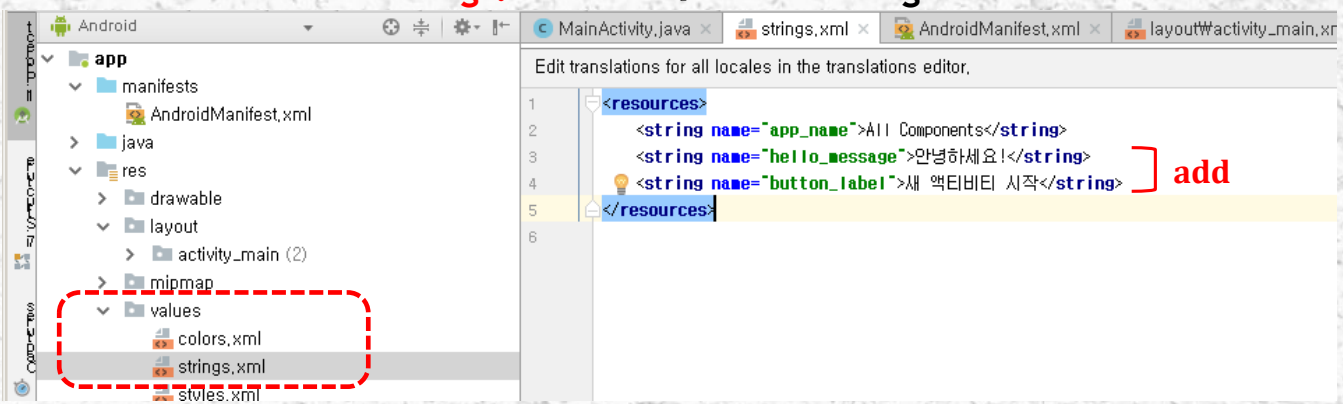
Modify Sample Project: Using Resource

- (Adding layout for landscape mode)
  - Change to 'Android View': check 'app/res/layout' folder
    - See overlapped 'activity_main' items tagged with '(2)'
    - Run & test the result (change Emulator/Device into landscape mode)



Modify Sample Project: Using Resource

- Define resource strings for different locales
  - Edit 'res/values/strings.xml' file → add 2 string elements

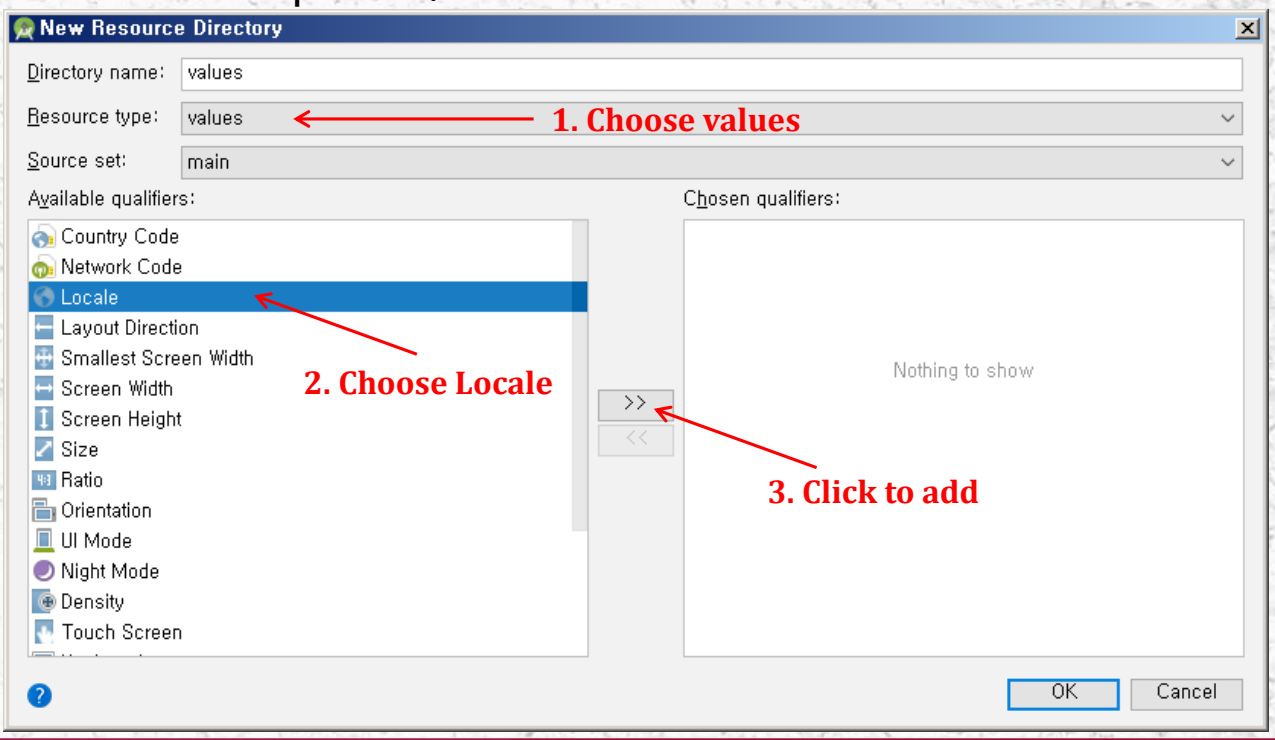


  - Change 'layout/activity_main.xml' & 'layout-land/activity_main.xml'
    - @string/hello_message
    - @string/button_label

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_message" />   ← change

<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:onClick="runActivity"
    android:text="@string/button_label" />   ← change
```

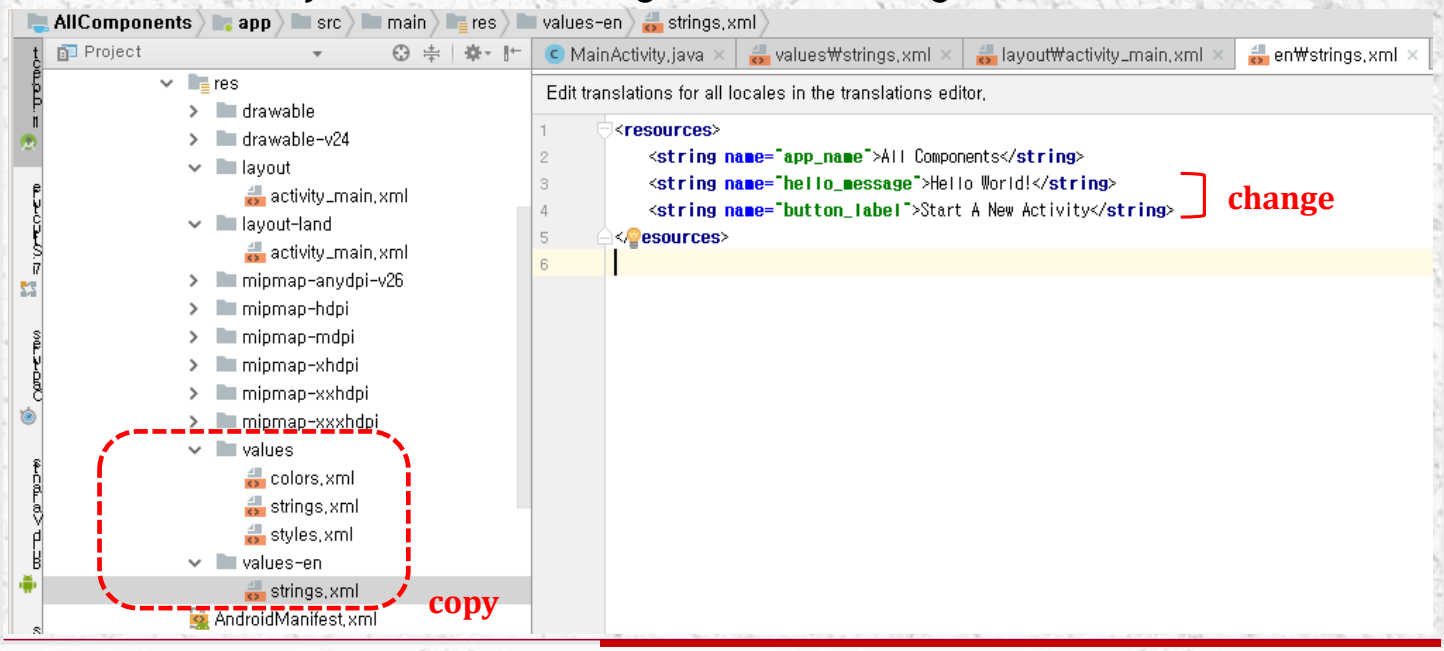
Modify Sample Project: Using Resource

- (Define resource strings for different locales)
  - Right-click 'res' folder → New → Android Resource Directory
    - Resource Type: choose 'values'
    - Available qualifiers: choose 'Locale' and click '>>' button



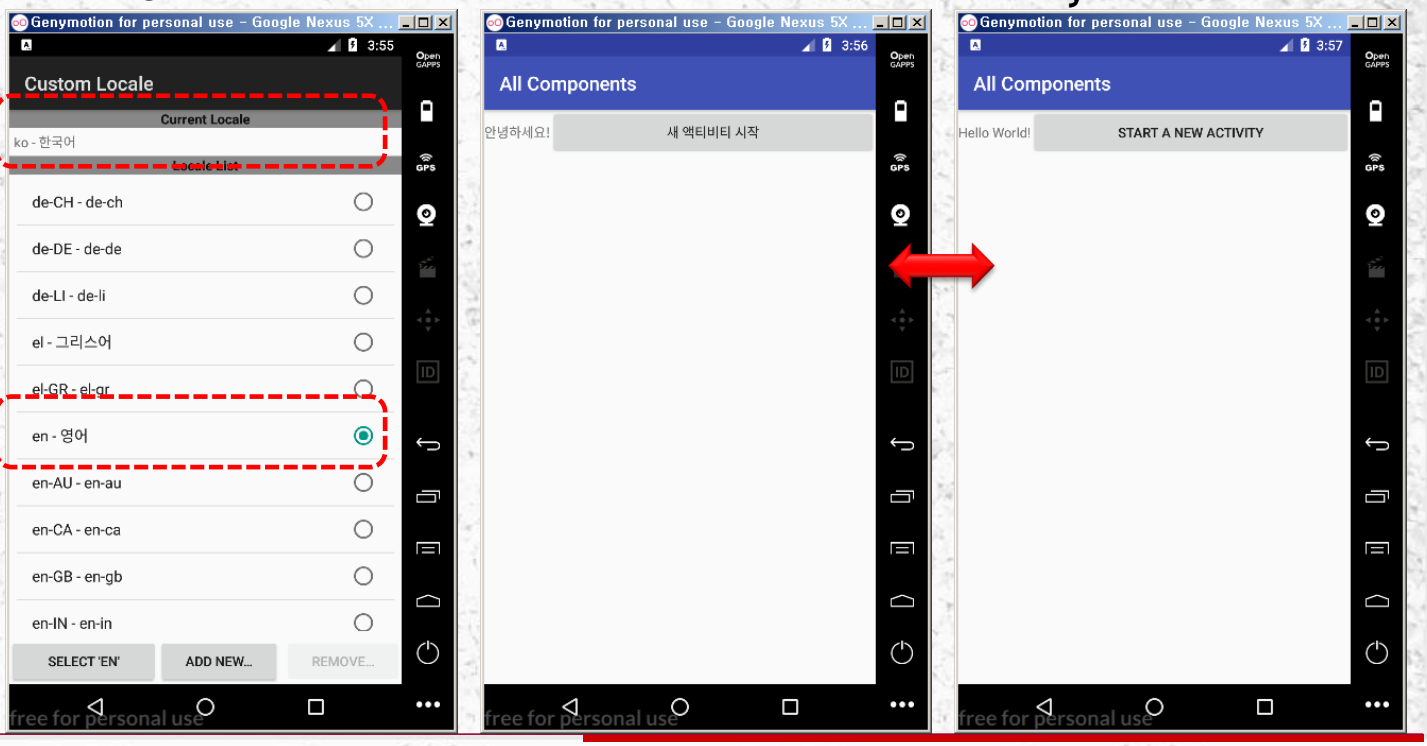
Modify Sample Project: Using Resource

- (Define resource strings for different locales)
  - In the 'New Resource Directory' dialog
    - Choose 'en: English' in Language, 'Any Region' in Specific Region Only.
    - 'OK' click, change to 'Project View', check 'res/values-en' folder
    - Copy 'res/values/strings.xml' file into 'res/values-en' folder.
    - Modify 'res/values-en/strings.xml' file into English locale



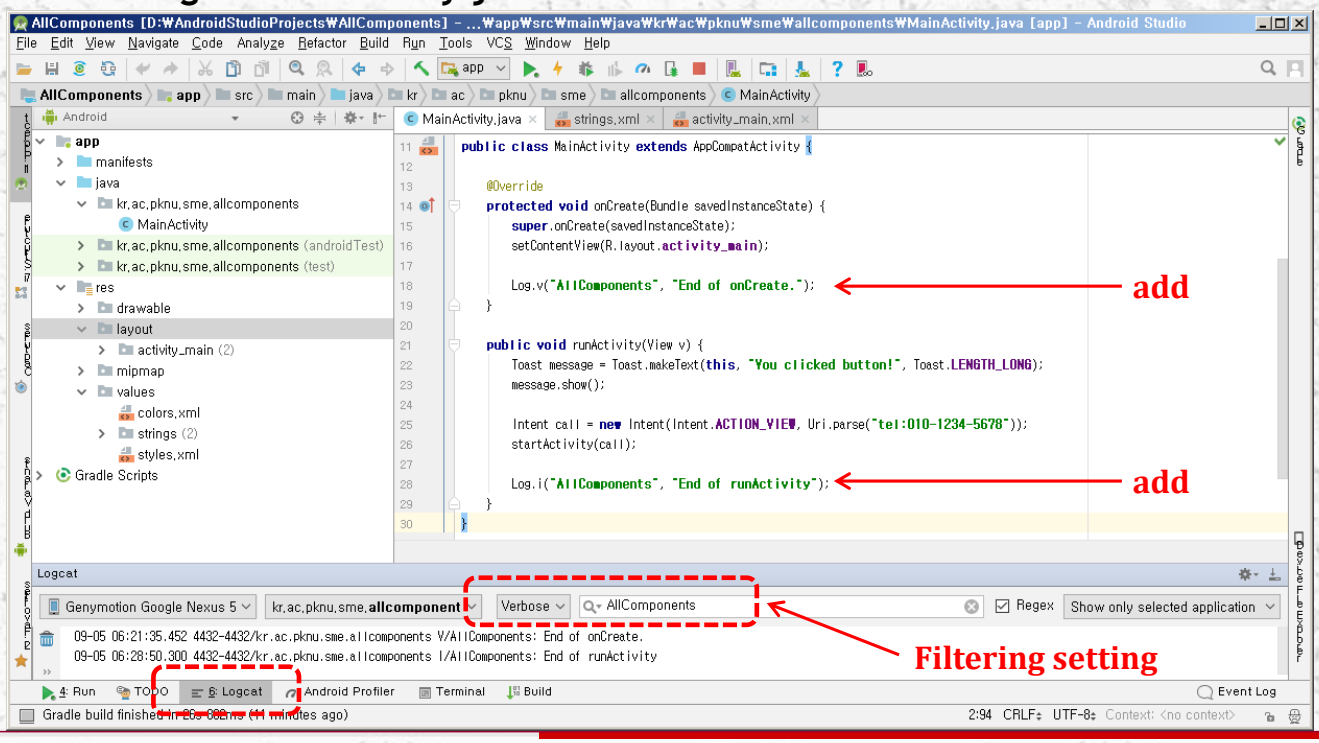
Modify Sample Project: Using Resource

- (Define resource strings for different locales)
  - Run our app in different locales.
    - Emulator/Device → Custom Locale app → flip b/w 'ko-한국어/en-영어'
  - Other locale's resources can also be added in this way!



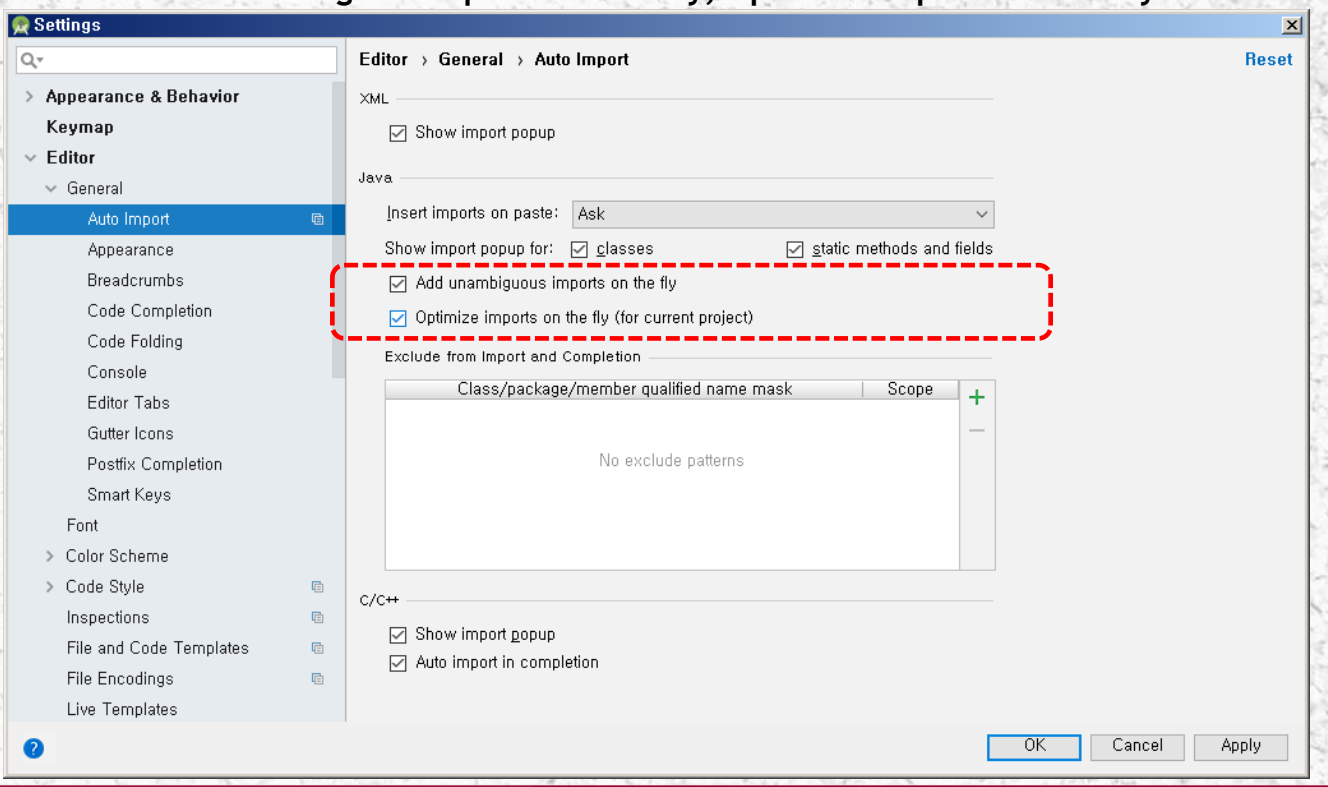
Modify Sample Project: Misc.

- Logcat window: current app's output message
  - Various level of message output: Log.e, Log.w, Log.i, Log.d, Log.v
    - Log.v("MY_TAG_NAME", "Log Message …"); // vervose level
  - Change 'MainActivity.java' file as follows



Modify Sample Project: Misc.

- Auto Import Setting
  - [File] → [Settings] → [Editor] → [General] → [Auto Import]
    - Add unambigious imports on the fly, Optimize imports on the fly



Summary

- Quick review of Android App
  - Design philosophy for component reuse.
  - 4 types of components
    - Activity / Service / Broadcast Receiver / Content Provider
  - Component brokerage using Intent object

- Code & Resource separation
  - Recommend declarative resource definition in XML format.
  - Code and resources are linked via auto-generated R.java file.
    - R.layout.xxx, R.id.xxx, …
    - @id/xxx, @string/xxx, …

- From now on … we need to learn:
  - Process & Threading
  - Networking
  - Location based service using GoogleMaps API
  - …