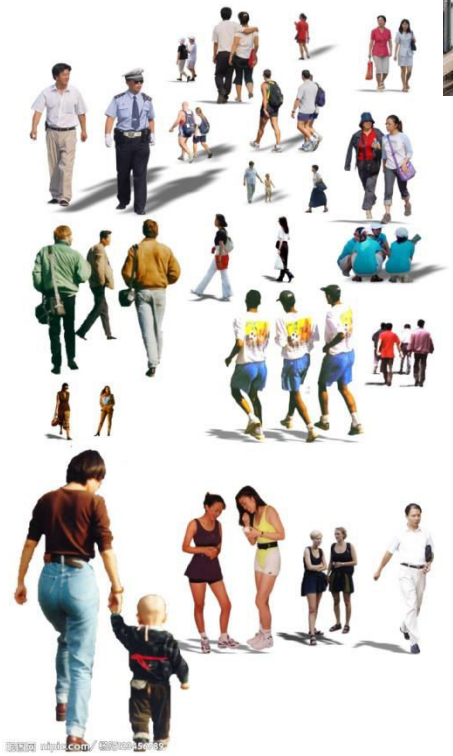
Chapter One
Introduction
Mobile Computing

2.1. What is mobile computing?

- Mobile computing is to describe technologies that
  - enable people to access network services anytime, and anywhere,
  - with portable and wireless communication devices.
- **Why mobile computing?**
  - **User mobility**
    - Between different geographical locations
    - Between different networks
    - Between different communication devices
    - Between different applications
  - **Device portability**
    - Between different geographical locations
    - Between different networks

WHY MOBILE COMPUTING ?

- People are mobile



WHY MOBILE COMPUTING ?

- Devices are mobile

Mobile Computing ...

Among the basic constraints in mobility are:

- Unpredictable variation in network quality
- Lowered trust and robustness of mobile elements
- Limitations on local resources imposed by weight and size constraints
- Concern for battery power consumption

Mobile computing: Vision

- Universal connectivity – anywhere, anytime
- Accommodate heterogeneity of networks and communicators.
- Ubiquitous intelligent environment - embedded computers everywhere
- Easy user interaction
- Context independent access to services + context dependent information

Mobile and Ubiquitous Computing

❑ Mobile Computing

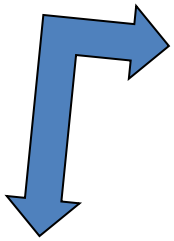
- People traveling with their computers while staying connected to other computers or the Internet.

❑ Ubiquitous Computing

- Integrate computers seamlessly into the world
  - invisible, everywhere computing.
    - Often called **pervasive/invisible computing**.

Ubiquitous Computing: the future-Internet of things

- Internet (past) when a user was allowed only to use content
- Internet (present) a user is allowed to also create content (e.g. face book, twiter, …)
- Internet (future) ?



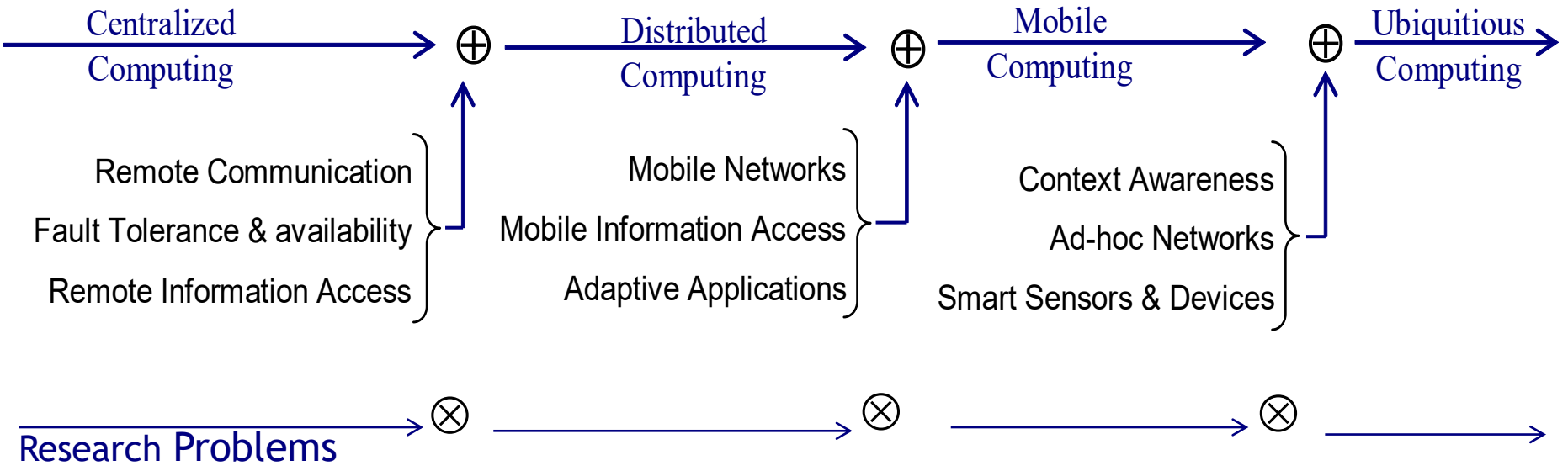
- IP for "Everything" with IPv6? "Things" will be allowed to create and use content

2.2. The Trends in Computing Technology

- Mainframe computing (60's-70's)
  - massive computers to execute big data processing applications
  - very few computers in the world

- Desktop computing (80's-90's)
  - one computer at every desk to help in business related activities
  - computers connected in intranets to a massive global network (internet), all wired
- Mobile computing
- Ubiquitous computing (00's?)
  - tens/hundreds of computing devices in every room/person,
  - becoming "invisible" and part of the environment

Computing: Evolution

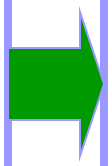


Today, Internet of Things

Tomorrow's ubiquitous world of tags, sensors and smart systems

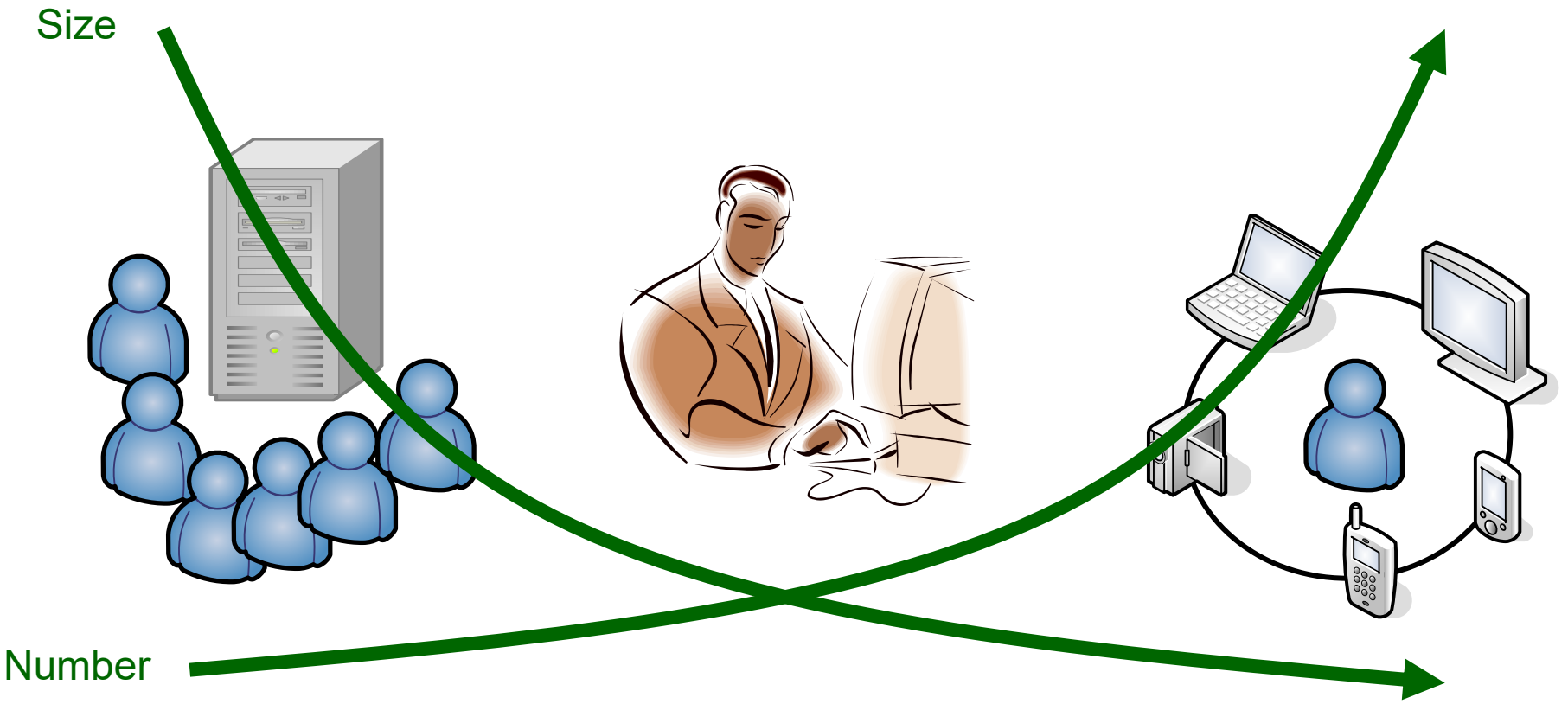
Computing: Evolution
New Forms of Computing

- Distributed Computing (Client/Server)



- Wireless Computing
- Mobile Computing
- Ubiquitous Computing
- Pervasive Computing
- Invisible Computing

Computing: Trend



One Computer for Many People

(Mainframe Computing)

One Computer for One Person

(PC Computing)

Many Computers for One Person

(Ubiquitous/Pervasive Computing)

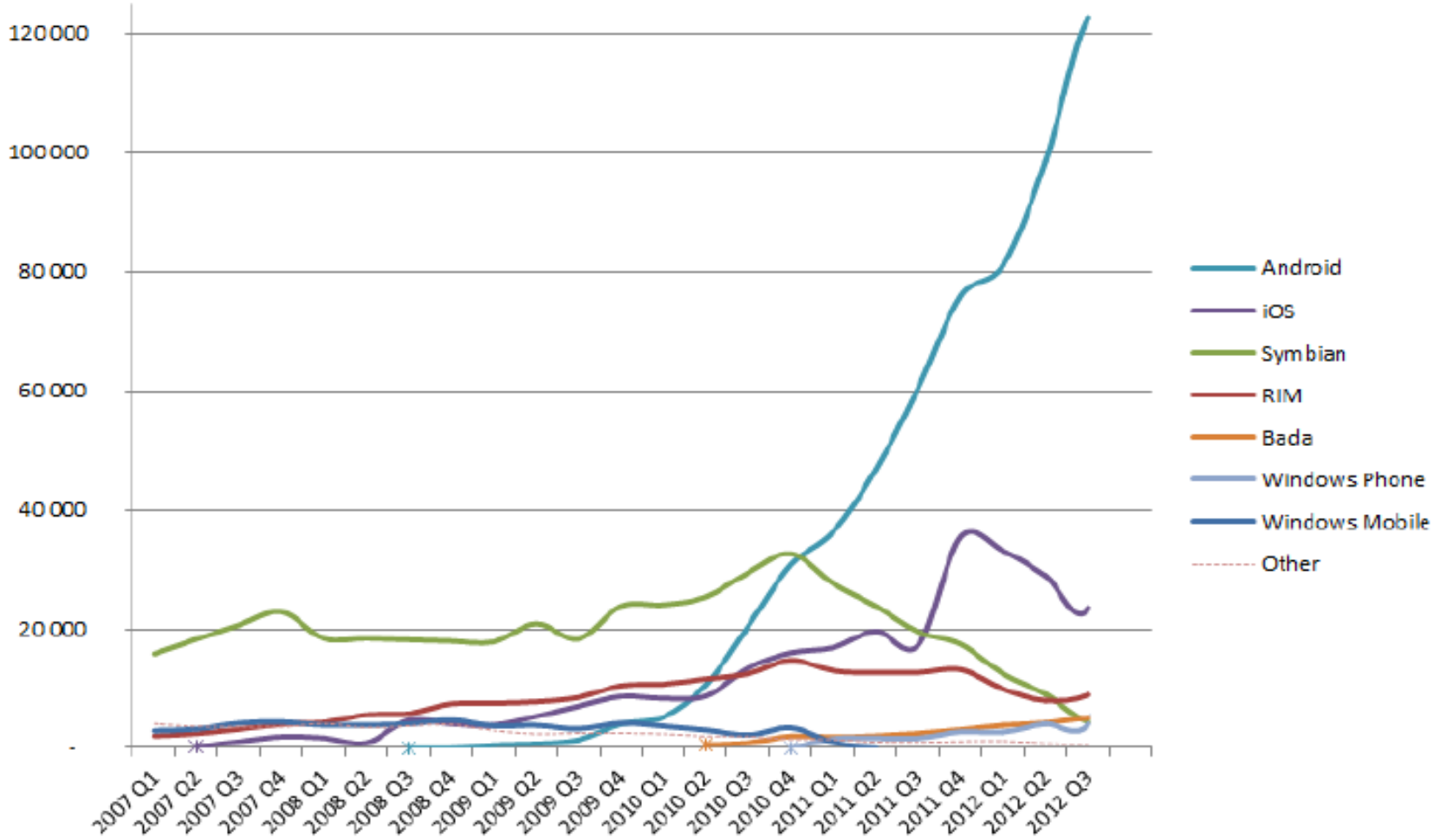
2.3. Mobile communication protocols

Reading Assignment (2%)

2.4 Mobile operating systems

- It deals with the characteristics and requirements of mobile applications.
- It is the essential component that operates the mobile device.
- is the engine of the mobile device. In other terms, it is the operating system of the appliance.

World-Wide Smartphone Sales (Thousands of Units)



Mobile Technologies

- **Technologies:**

  - SMS
    - Communication layer for local apps
    - SMS applications
  - Local Applications:
    - Java 2 Micro Edition (J2ME)
    - Python (Nokia)
    - Android
    - Etc, etc, etc…
  - Mobile Web
    - Internet access over 2G/3G/4G
    - Communication layer for local apps
  - Telephony Apps
    - Phone menus
    - Voice recognition

Local Mobile Applications

- Applications that run on the mobile phone
- Unfortunately, many different platforms:
  - J2ME
  - Android
  - Apple iPhone
  - Microsoft Windows Mobile
  - Blackberry
  - PalmWebOS
  - Nokia (C/C++, Python)
    - Symbian (S60, S80)

symbian OS

J2ME CDC Java

palm

Windows Mobile

android

BlackBerry

NOKIA Connecting People

Local Mobile Applications

- Fast, rich user interfaces
  - Forms, menus, alerts, buttons, pictures, videos, textboxes, touch screen, orientation
- Access to device features
  - Location (GPS, Google maps, compass, …)
  - Voice / speaker
  - Storage
  - Camera
  - Wi-fi (local networking)
  - Bluetooth, IR
  - Mobile network (SMS, data)

Local Mobile Applications

- Unless preinstalled, these applications require installation
  - Bluetooth
  - USB cable
  - SD card

Technology Tradeoffs

	SMS Application	Mobile Web	Local Application
Installed Base	Everyone	Mostly Everyone	Many and growing
Portability	Best	Different Flavors	Phone Specific /Platform specific
Bandwidth Req.	Low	High (nothing local)	Variable (local interaction /cache locally)
User Interface / User Experience	Simple	Adequate	Rich & Responsive
Advanced Features	None	Few	Yes! (GPS, orientation, local networking)

Mobile Application Marketplaces

- Google, Apple, Nokia, Palm, etc. have mobile marketplaces where one can sell applications.
  - Application delivery over the Network!
  - E.g. google play for android apps
- Apple iPhone App store:
  - 35,000 applications
  - 1 Billion downloads!
  - $1 Million USD a day in sales!
- You can target any market!

Application Categories

- Informational
  - Converters, weather, area-guides, finance
  - Location-based
- Data Entry
  - Business records, medical records, etc,
- Multimedia
  - Camera, video, music, photos, ringtones
- Shopping
  - e-commerce, compare prices

Application Categories

- (Social) Networks
  - MySpace, Facebook, Twiter, LinkedIn
- Communication
  - Skype, VoIP, SMS
- Business productivity
  - Legers, spreadsheets, inventory
- Games
  - 2D, 3D

Mobile Web

- Data speeds are increasing and costs are dropping!
- Mobile web is going to explode
  - Look at developed world as an example
  - Serve information from mobile version of website
- Phone browsers are becoming more like desktop browsers

Appropriate Mobile Technology

- Your service might be appropriate as a mobile website
  - You develop the mobile website
    - HTML, Flashlite, PHP, Javascript, etc.
  - Serve information, small amount of text entry
  - Available to users of most phones

- With mobile web, why develop local applications?

Features Available to Local Apps

- More than just serving information!
  - Storage local to phone
  - Local calculations
  - Reduced data requirement
    - Only need access to underlying data
    - Local app UI decides *locally* how to display
    - With mobile web, you get HTML
      - Data and instructions for displaying data (more traffic)
  - Local error checking
    - Check inputted values on phone before sending over network

Features Available to Local Apps

- More than just serving information!
  - Flexible to network outages
    - Cache locally
    - Store information until network becomes available
    - Fallback to SMS (for weak signals)
  - No need for server
    - With mobile web you need a server serving pages
    - Local app can take info from others

Features Available to Local Apps

- More than just serving information!
  - Local networking
    - Bluetooth, IR, Wi-Fi
  - Location Awareness
    - GPS
  - Camera, Voice, Speaker, Video, etc.

- If your service can benefit from any of these, consider a local application

Local Applications

- All these features come at a price
- Have to write applications for each platform
  - J2ME offers some platform independence, but not supported by all phones
    - Mostly Nokia phones here (previously): J2ME
    - Now Android prevails
  - Mobile web has portability issues also

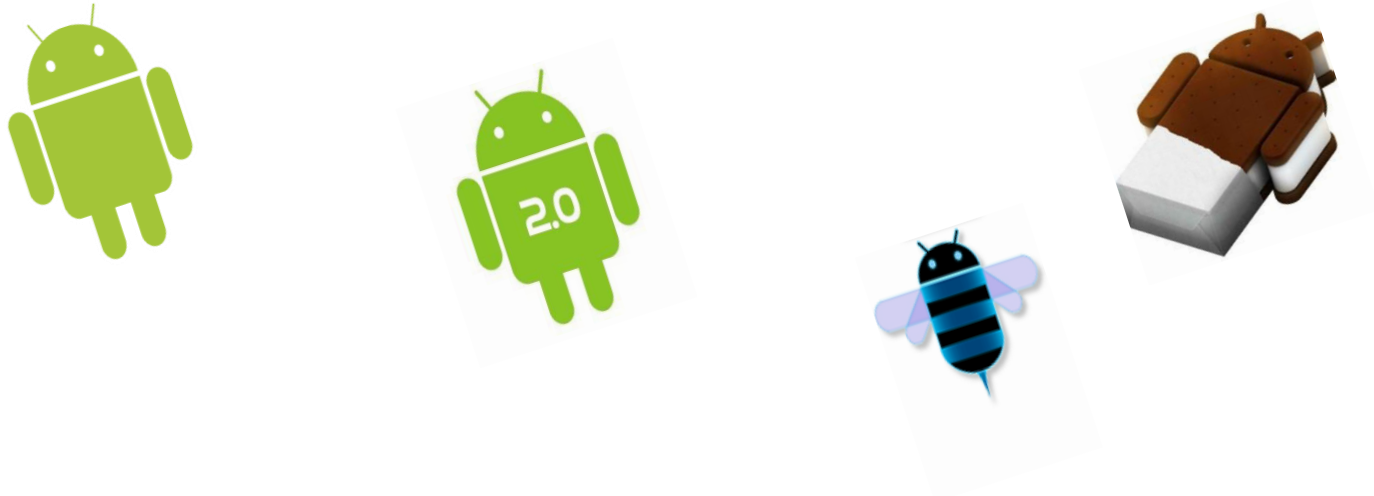
- So make sure you use features not available to mobile web!

Local Application Ideas

- Most Ethiopians don't have computers
  - What can their phone do to help them?
    - Record keeping, business tools, health workers tools, media management, camera tools ……
- Reduce network requirement / cost
  - Caching and optimized protocols for websites / services (future work - web based)

Chapter 2
Mobile Devices - Application development

2.1.1 Setting up an Android Studio Development Environment



Setting ASDE includes

- Installing the Java Development Kit (JDK).
- Installing Android Studio Integrated Development Environment (IDE).
- Installing Android Software Development Kit (SDK).
- Configuring Emulator. (AVD)

System Requirements

- Windows 2003 (32-bit or 64-bit)
- Windows Vista (32-bit or 64-bit)
- Windows 7 (32-bit or 64-bit)
- Windows 8 / Windows 8.1/Windows 10
- Minimum of 2GB of RAM (4GB is preferred)
- 1.5GB of available disk space

Downloading source

http://www.oracle.com/technetwork/java/javase/downloads/index.html

(JDK)

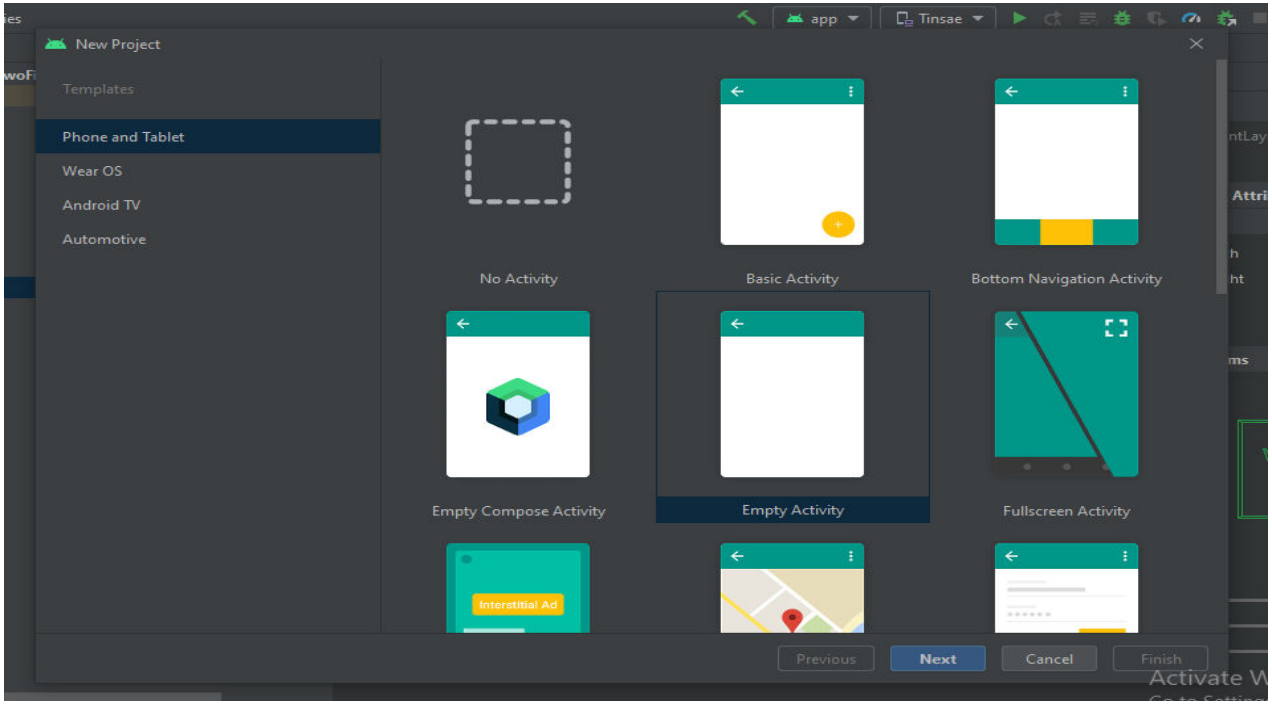
http://developer.android.com/sdk/index.html

(Android Studio)

Note: The Android Studio Bundle (includes the JDK and other development environment inside it)

Setting up Android Studio

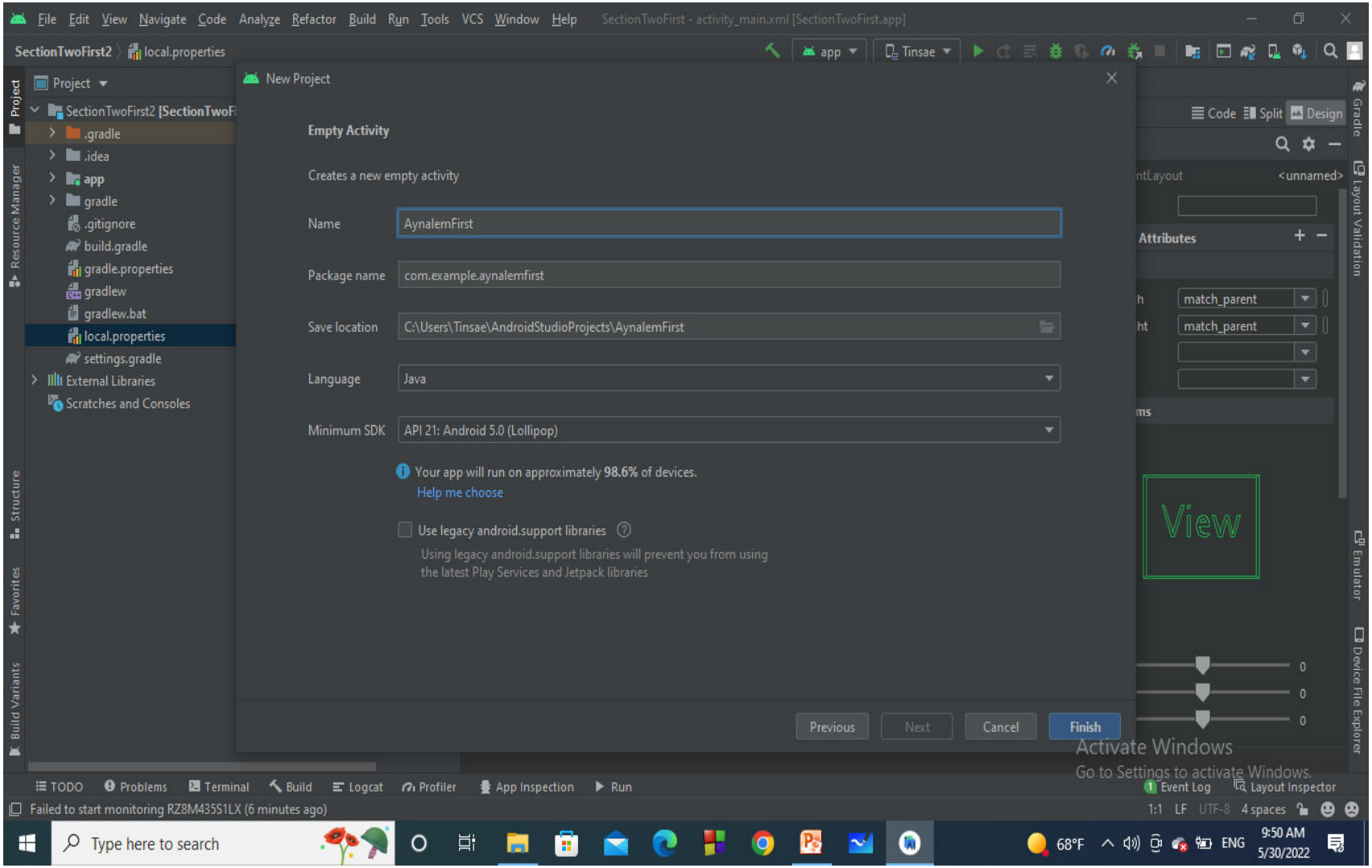
- Open the executable file and run the wizard as usual to install it.
- After Android Studio has finished loading, the setup wizard will appear as shown in Figure below



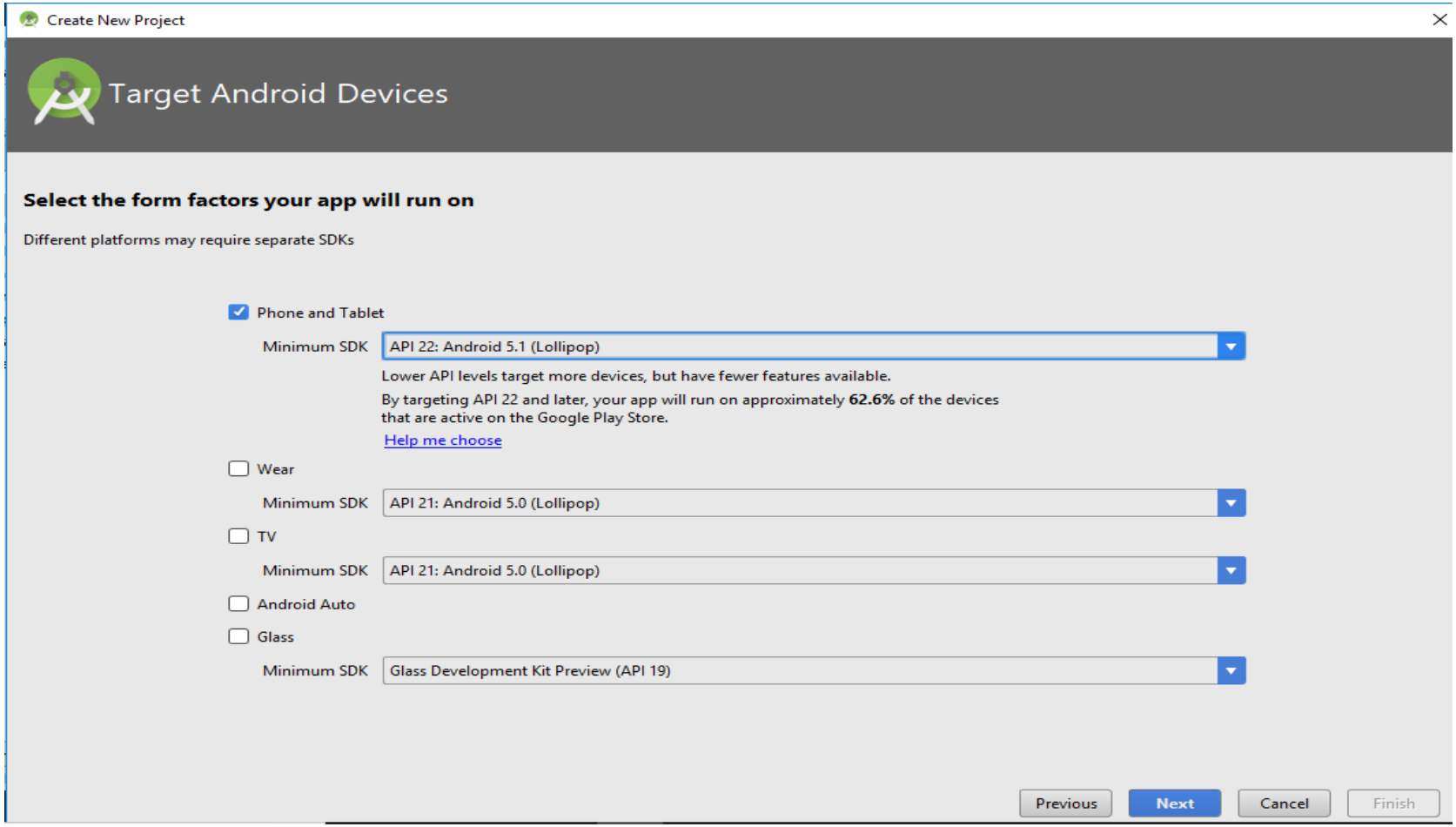
……Installing Android Studio

- When Next button is clicked welcome to Android Studio screen should then appear:

- To create the new project, simply click on the Start a new Android Studio project option from the welcome to android studio window.
- Then the first screen of the New Project wizard will be open as shown in the next Figure .

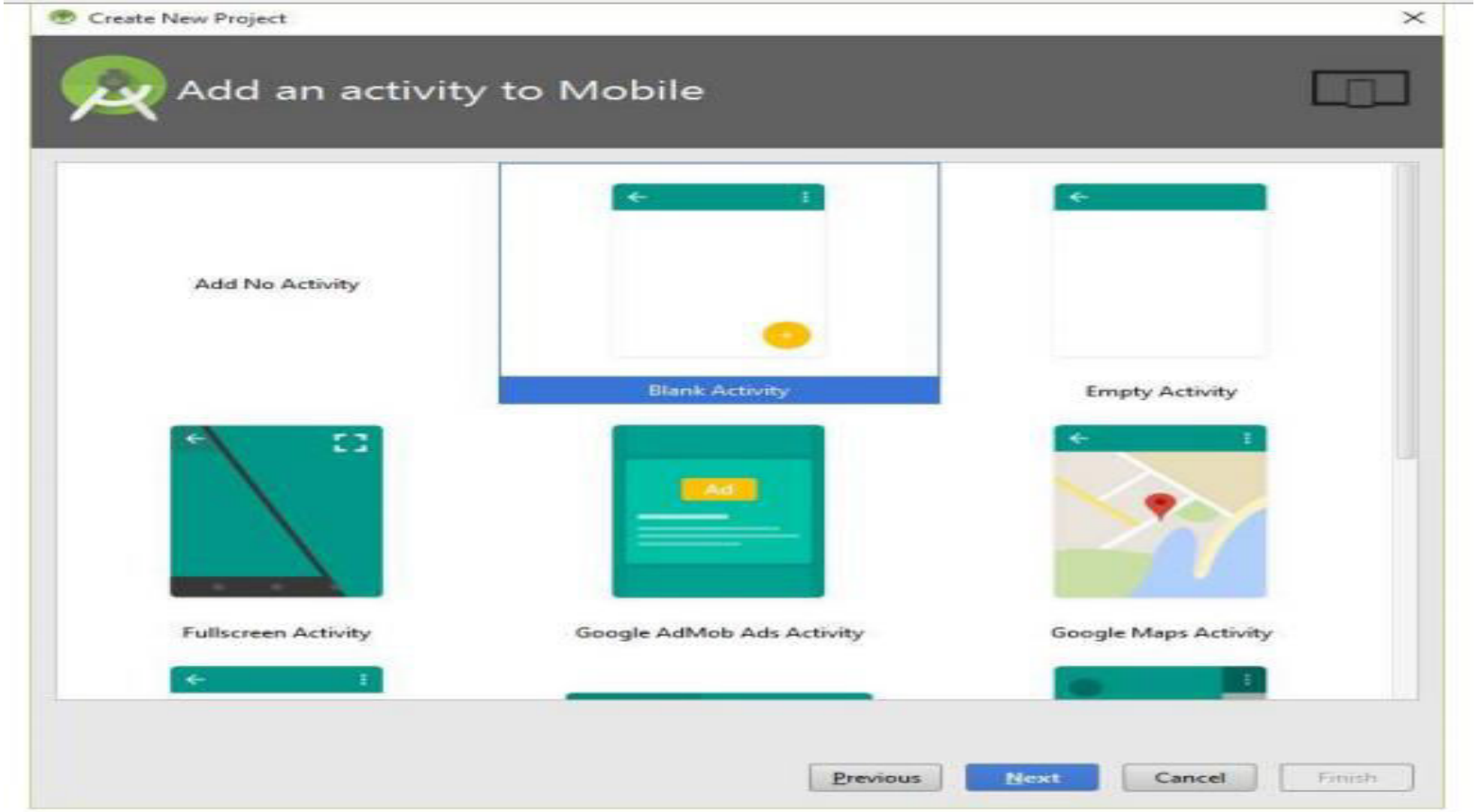

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
SectionTwoFirst - activity_main.xml [SectionTwoFirst.app]
SectionTwoFirst2 local.properties
app
Tinsae
Project
SectionTwoFirst2 [SectionTwoF
.gradle
.idea
app
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
settings.gradle
External Libraries
Scratches and Consoles
New Project
Empty Activity
Creates a new empty activity
Name
AynalemFirst
Package name
com.example.aynalemfirst
Save location
C:\Users\Tinsae\AndroidStudioProjects\AynalemFirst
Language
Java
Minimum SDK
API 21: Android 5.0 (Lollipop)
Your app will run on approximately 98.6% of devices.
Help me choose
Use legacy android.support libraries
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries
Previous
Next
Cancel
Finish
Code
Split
Design
Attributes
match_parent
match_parent
View
Activate Windows
Go to Settings to activate Windows.
TODO
Problems
Terminal
Build
Logcat
Profiler
App Inspection
Run
Event Log
Layout Inspector
Failed to start monitoring RZ8M435S1LX (6 minutes ago)
1:1 LF UTF-8 4 spaces
Type here to search
68°F
ENG
9:50 AM
5/30/2022

- After clicking next adjust target devices for your project this is done by selecting the minimum SDK.



Then Create an Activity

- The next step is to define the type of initial activity that is to be created for the application and follow the wizard.



Installing Android SDK Packages

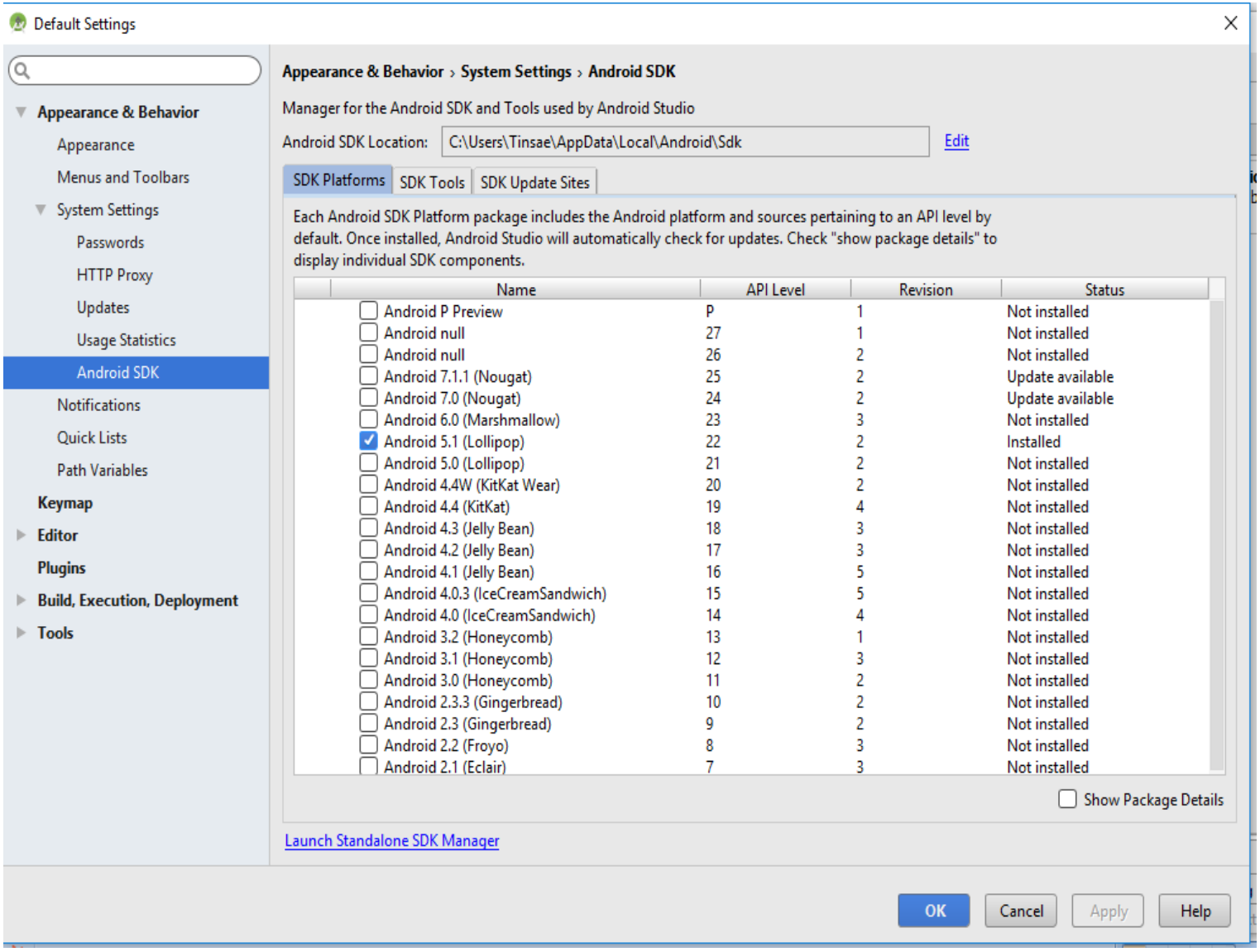
What is android SDK?

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. It includes :

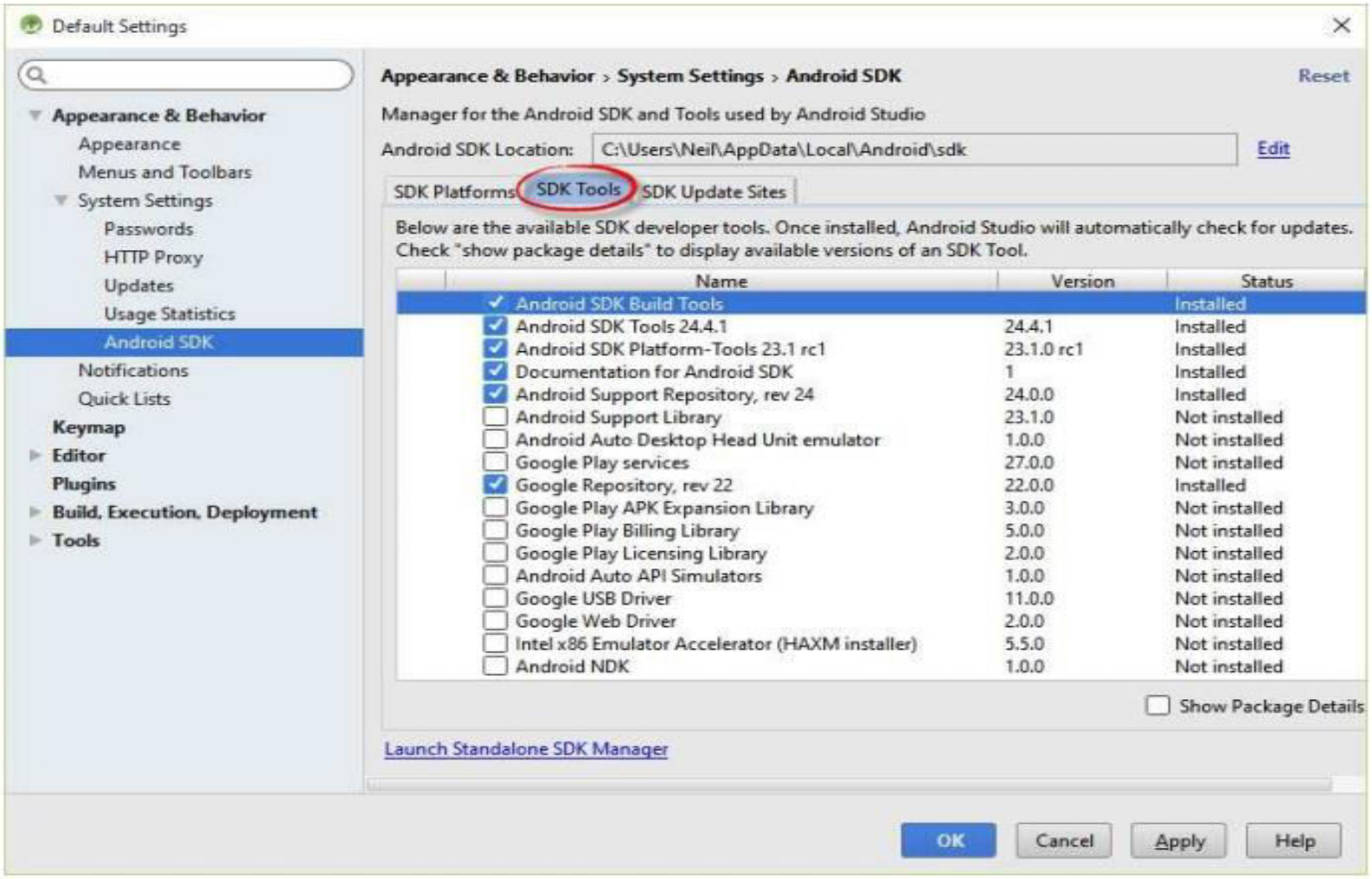
- Required libraries
- Debugger
- An emulator
- Relevant documentation for the Android application program interfaces (APIs)
- Sample source code
- Tutorials for the Android OS

…Installing Android SDK Packages

- This task can be performed using the *Android SDK Manager*.
- which may be launched from within the Android Studio tool by selecting the ***Android - SDK Manager*** option from within the Android Studio welcome dialog.
- Once invoked, the **SDK Manager tool** will appear as illustrated in the below Figure.


Default Settings
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
Passwords
HTTP Proxy
Updates
Usage Statistics
Android SDK
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Build, Execution, Deployment
Tools
Appearance & Behavior › System Settings › Android SDK
Manager for the Android SDK and Tools used by Android Studio
Android SDK Location: C:\Users\Tinsae\AppData\Local\Android\Sdk
Edit
SDK Platforms
SDK Tools
SDK Update Sites
Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.
Name
API Level
Revision
Status
Android P Preview P 1 Not installed
Android null 27 1 Not installed
Android null 26 2 Not installed
Android 7.1.1 (Nougat) 25 2 Update available
Android 7.0 (Nougat) 24 2 Update available
Android 6.0 (Marshmallow) 23 3 Not installed
Android 5.1 (Lollipop) 22 2 Installed
Android 5.0 (Lollipop) 21 2 Not installed
Android 4.4W (KitKat Wear) 20 2 Not installed
Android 4.4 (KitKat) 19 4 Not installed
Android 4.3 (Jelly Bean) 18 3 Not installed
Android 4.2 (Jelly Bean) 17 3 Not installed
Android 4.1 (Jelly Bean) 16 5 Not installed
Android 4.0.3 (IceCreamSandwich) 15 5 Not installed
Android 4.0 (IceCreamSandwich) 14 4 Not installed
Android 3.2 (Honeycomb) 13 1 Not installed
Android 3.1 (Honeycomb) 12 3 Not installed
Android 3.0 (Honeycomb) 11 2 Not installed
Android 2.3.3 (Gingerbread) 10 2 Not installed
Android 2.3 (Gingerbread) 9 2 Not installed
Android 2.2 (Froyo) 8 3 Not installed
Android 2.1 (Eclair) 7 3 Not installed
Show Package Details
Launch Standalone SDK Manager
OK
Cancel
Apply
Help

- In addition to the Android SDK packages, a number of tools are also required



…sdk tools

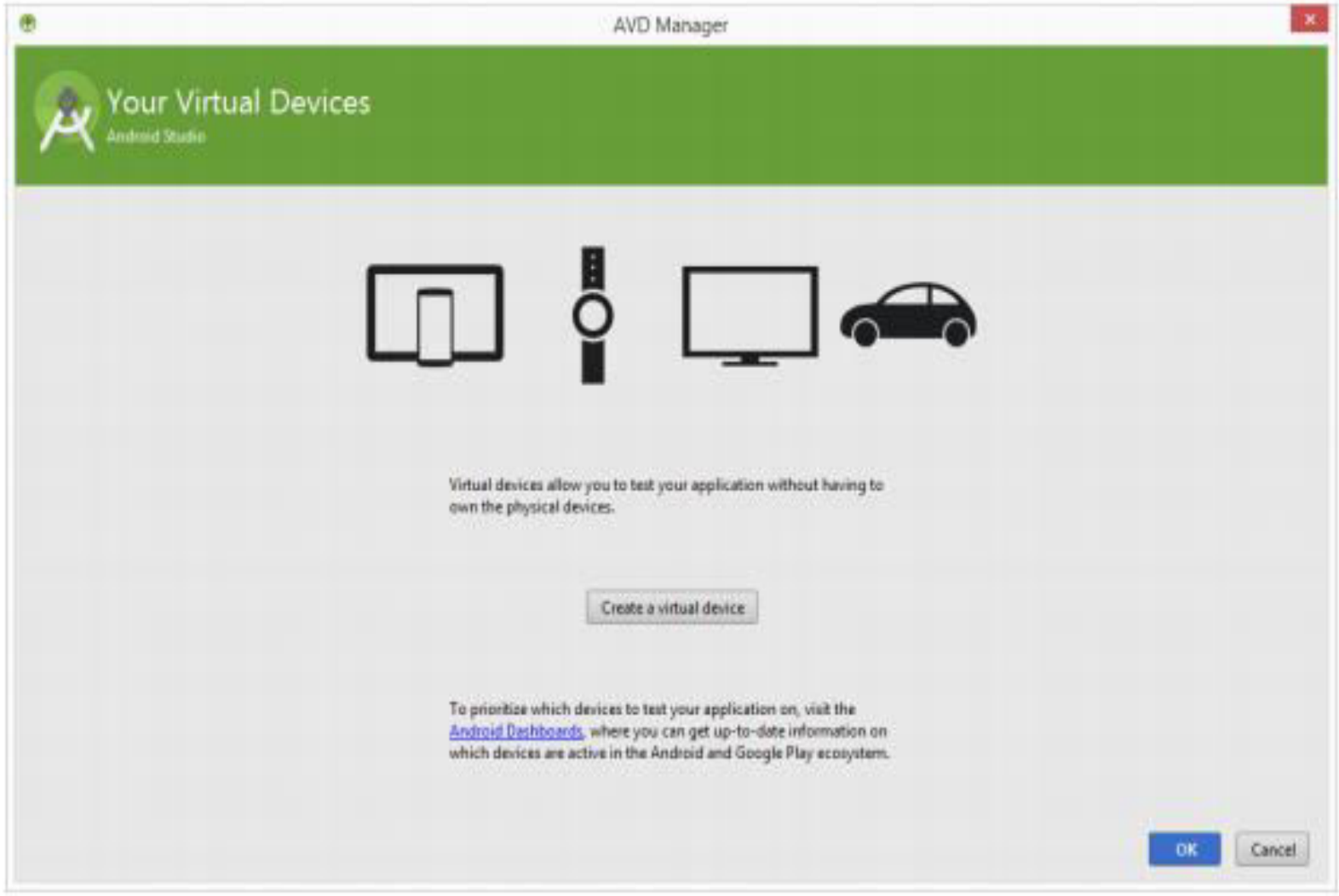
- Within the Android SDK Tools screen, make sure that the following packages are installed.
  - Android SDK Build-tools
  - Android SDK Tools
  - Android SDK Platform-tools
  - Android Support Repository
  - Android Support Library
  - Google Repository
  - Google USB Driver (Windows only)
  - Intel x86 Emulator Accelerator (HAXM installer)

Creating an Android Virtual Device (AVD) /emulators

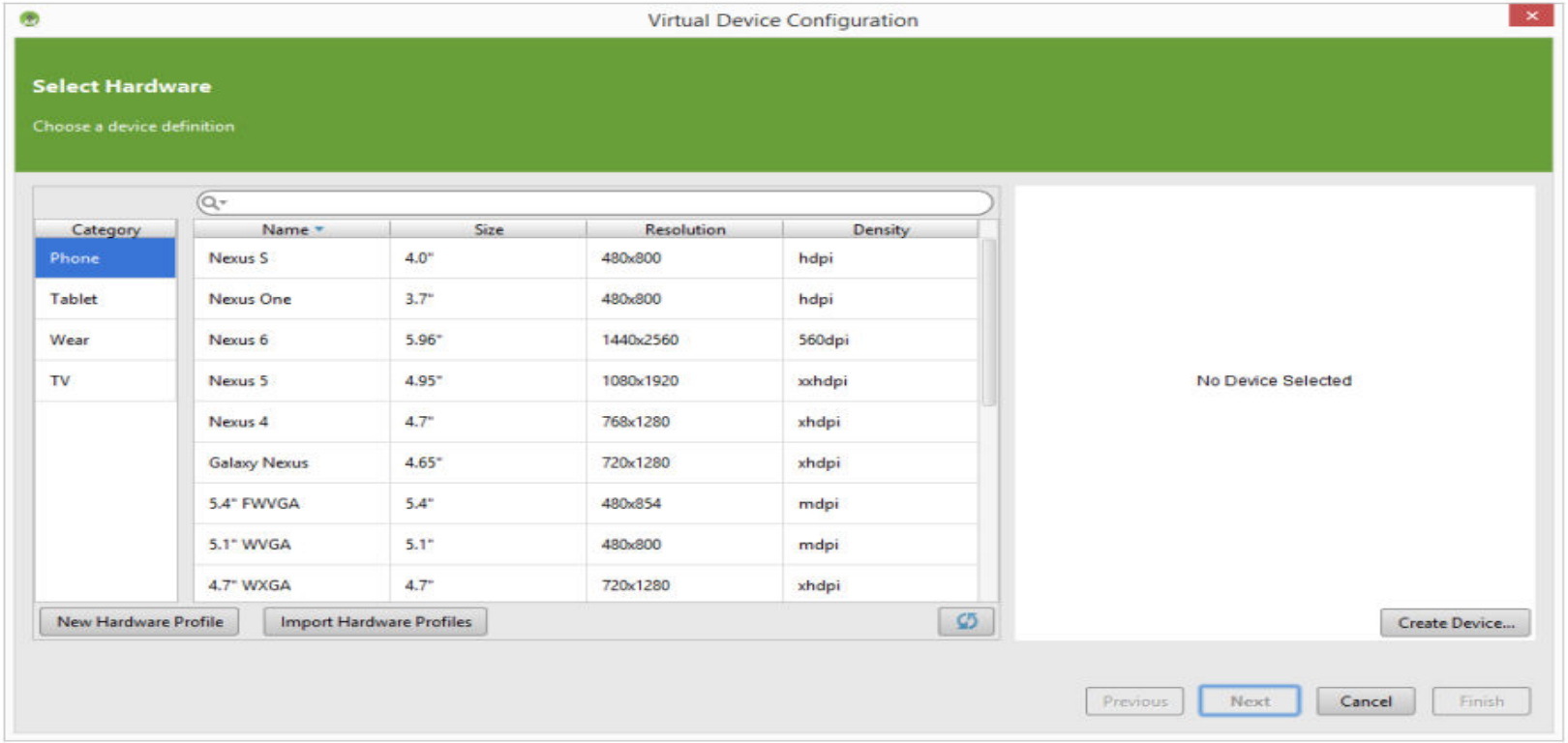
- allow Android applications to be tested without the necessity to install the application on a physical Android based device
- may be configured to emulate a variety of hardware features including options such as screen size, memory capacity and the presence or otherwise of features such as a camera, GPS navigation support or an accelerometer.

Creating a New AVD

- First launch the AVD Manager
- How? by selecting the Tools - > Android -> AVD Manager menu option.
- Once launched, the tool will appear as outlined in Figure below:
- Then click create a virtual device button on the window opened.


AVD Manager
Your Virtual Devices
Android Studio
Virtual devices allow you to test your application without having to own the physical devices.
Create a virtual device
To prioritize which devices to test your application on, visit the Android Dashboards, where you can get up-to-date information on which devices are active in the Android and Google Play ecosystem.
OK
Cancel

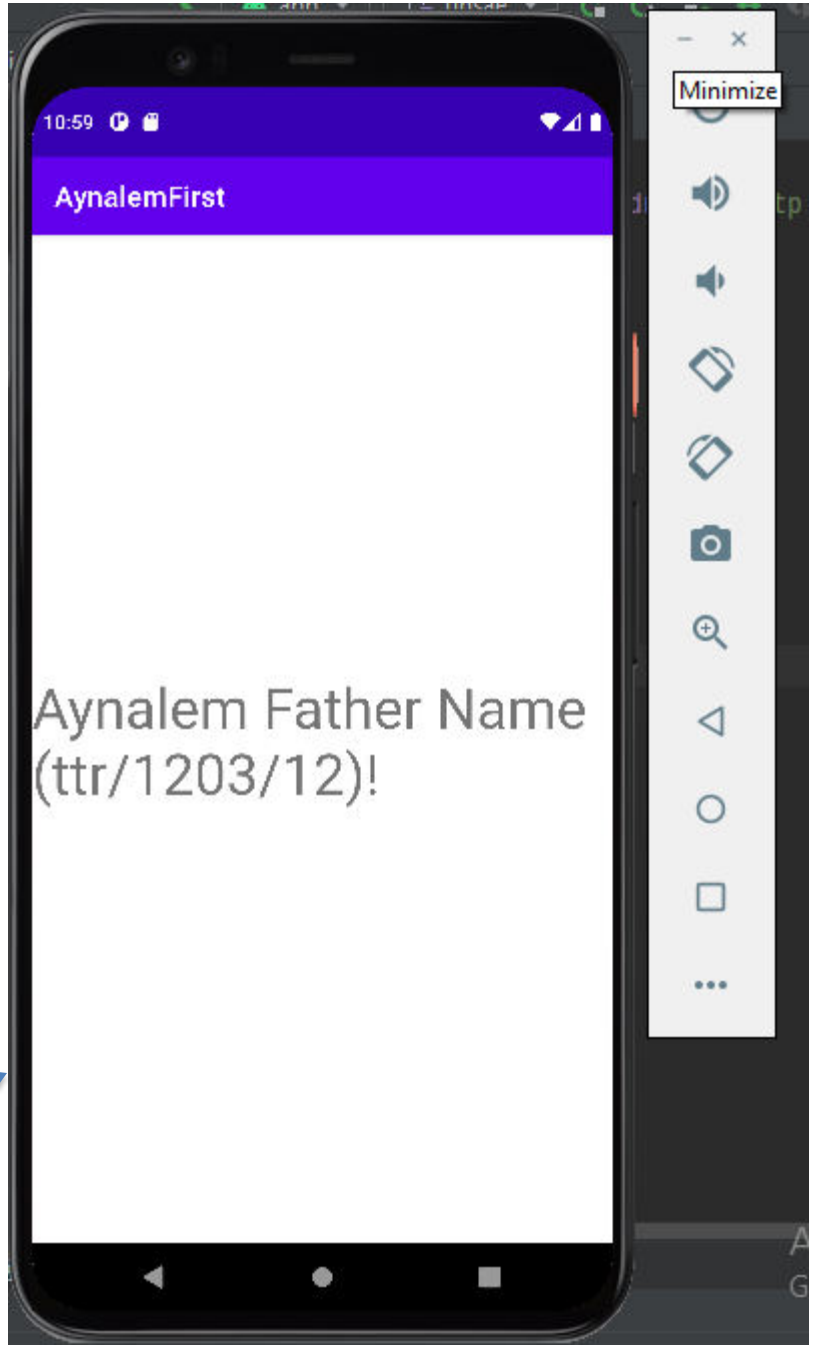
- by clicking on the *Create a virtual device* button the *Virtual Device Configuration* dialog will be invoked as shown below:



Practical Assignment #1
Type - individual

Create a new project by your name then**Configure Android Studio recent version and Display your full name and ID using AVD(Emulator) or Using Your Smart Phone (5%)**

as shown here



C:\Users\Tinsae\AndroidStudioProjects\SummerFirst\app\build\outputs\apk

Chapter 2
Mobile Devices - Application development

2.1. Introduction to Android

Android Overview

- Android was originally created by Andy Rubin as an operating system for mobile phones,
    - around the dawn of this twenty-first century.
- In 2005, Google acquired Android Inc., and made Andy Rubin the Director of Mobile Platforms for Google.
- Over the past decade, Android has matured and evolved into an extremely reliable, embedded operating system platform.
- Having gone from version 1.0 to stable versions at 1.5, 1.6, 2.0, 2.1, 2.2, 2.3, and, recently, 12.0

Android Overview…

- Android has the power of a complete computer operating system.
- It is based on:
  - Linux open source platform and
  - Oracle's (formerly Sun Microsystems's) Java, one of the world's most popular programming languages.
- Android is used as the primary operating system for a rapidly expanding range of consumer electronics, including:
  - Smartphones
  - Netbooks
  - MP4 players
  - Tablets
  - Internet TVs
  - Some desktop systems

Android Phones

HTC G1

Samsung i7500

HTC Hero

Motorola Cliq

Sony X10

HTC Magic

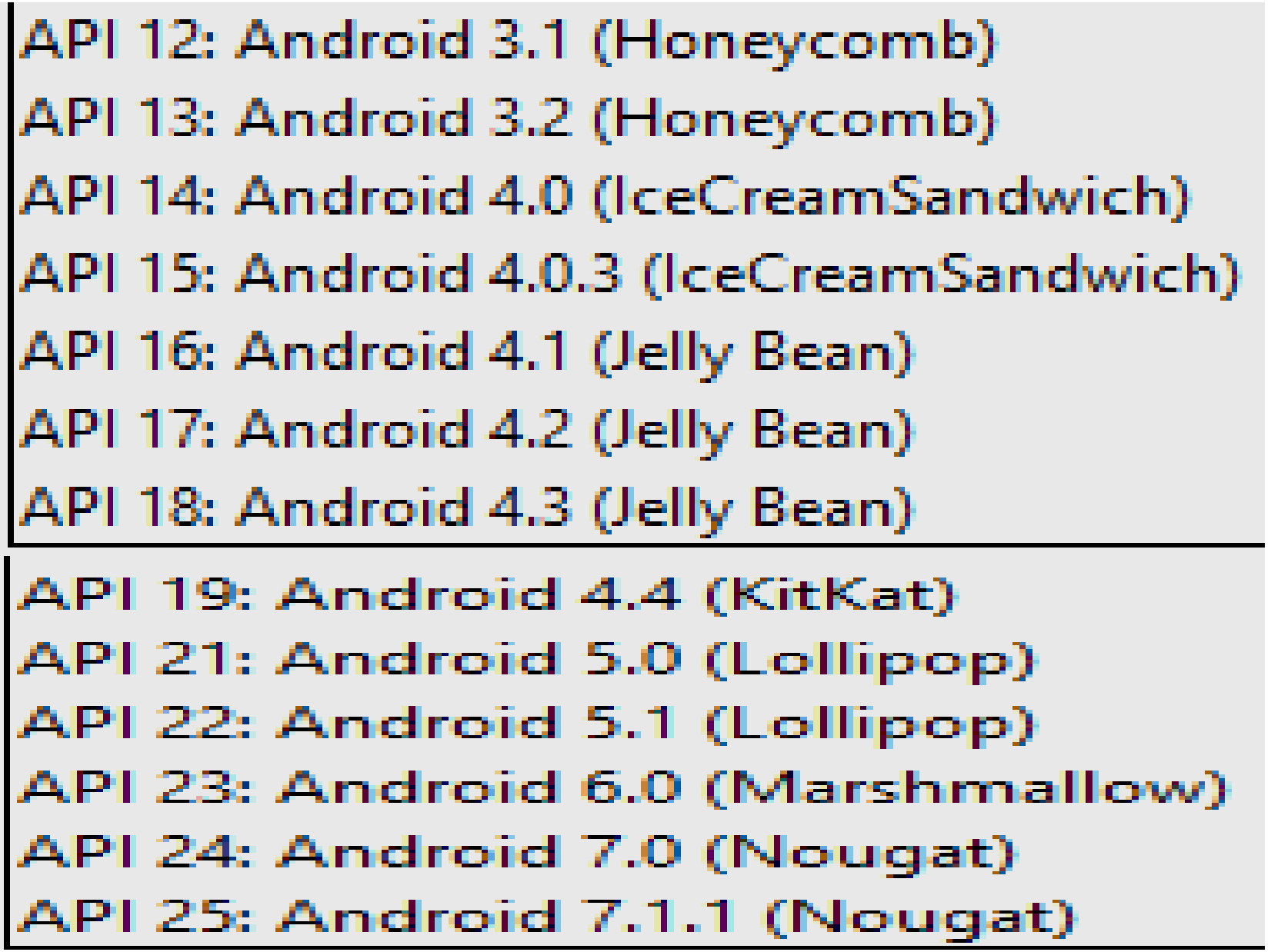
Samsung Moment

Motorola Droid

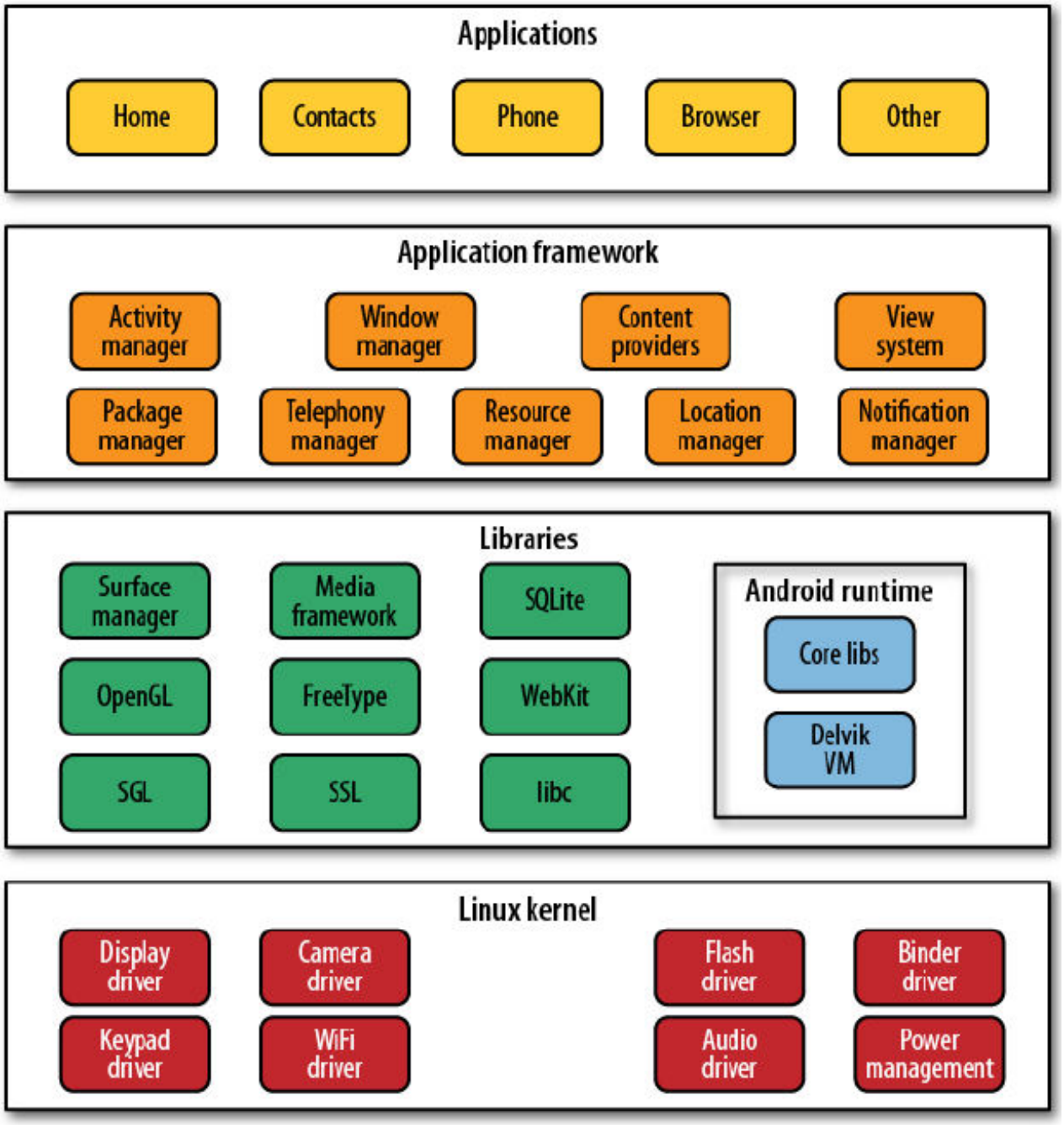
HTC Tattoo

Android Versions

Android version	API level	Nickname
Android 1.0	1	
Android 1.1	2	
Android 1.5	3	Cupcake
Android 1.6	4	Donut
Android 2.0	5	Eclair
Android 2.01	6	Eclair
Android 2.1	7	Eclair
Android 2.2	8	Froyo (frozen yogurt)
Android 2.3	9	Gingerbread
Android 2.3.3	10	Gingerbread
Android 3.0	11	Honeycomb


API 12: Android 3.1 (Honeycomb)
API 13: Android 3.2 (Honeycomb)
API 14: Android 4.0 (IceCreamSandwich)
API 15: Android 4.0.3 (IceCreamSandwich)
API 16: Android 4.1 (Jelly Bean)
API 17: Android 4.2 (Jelly Bean)
API 18: Android 4.3 (Jelly Bean)
API 19: Android 4.4 (KitKat)
API 21: Android 5.0 (Lollipop)
API 22: Android 5.1 (Lollipop)
API 23: Android 6.0 (Marshmallow)
API 24: Android 7.0 (Nougat)
API 25: Android 7.1.1 (Nougat)

Android Platform



Application Framework: it provides an access layer to the framework APIs used by the core applications. It allows components to be used by the developers.

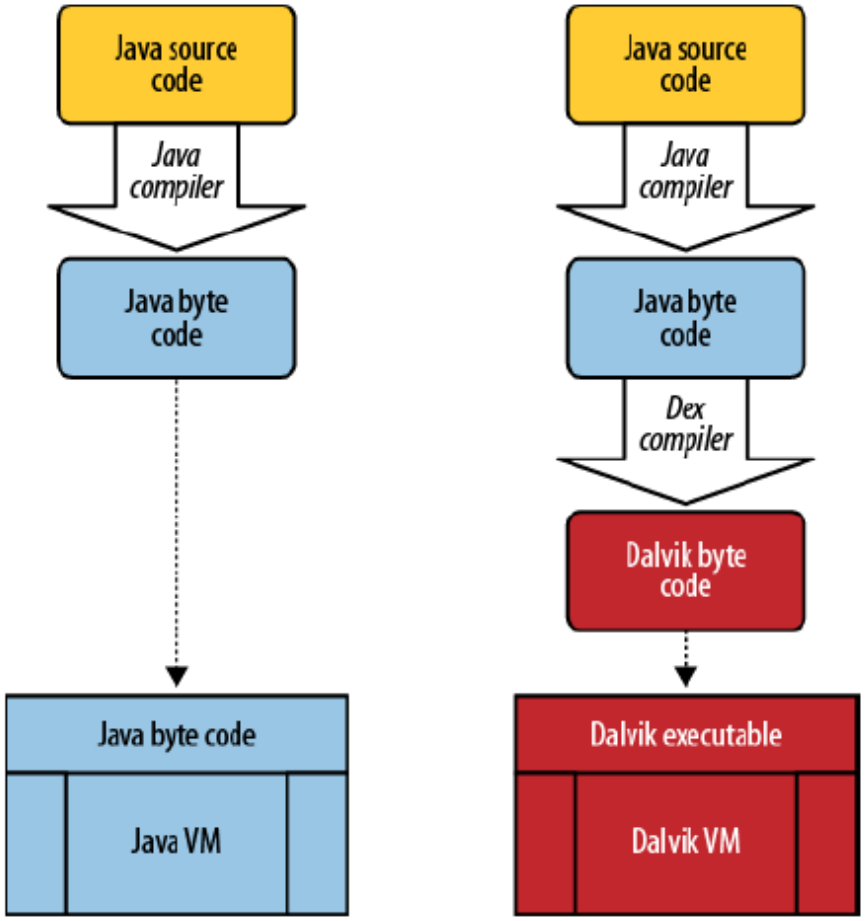
Android Runtime: it provides a set of core libraries which supports most of the functionality in the **core libraries of Java**. The Android Virtual Machine known as Dalvik VM relies on the linux kernel for some underlying functionality such as threading,…

Libraries: Android includes a set of C/C++ libraries. These libraries are exposed to developers through the Android application framework. They includes

- *Webkit* - A fast web-rendering engine used by Safari, Chrome, and other browsers
- *SQLite* - A full-featured SQL database
- *OpenGL* - 3D graphics libraries
- *OpenSSL* - The secure socket layer

Linux Kernel: Android relies on Linux for core system services such as security, memory management, process management and etc.

Android and Java



Android Features

- Linux OS kernel
- Java programming or Kotlin programming
- Open source libraries: SQLite, OpenSSL,WebKit, OpenGL
- A simple and powerful SDK
- No licensing, distribution, or development fees
- Development over many platform
  - Linux, Mac OS, windows
- Excellent documentation

Common Development Environment for Android Apps

1.Android Studio (bundled)

Require

- Android SDK Tools
- Android Platform-tools
- Java Development Kit (JDK)

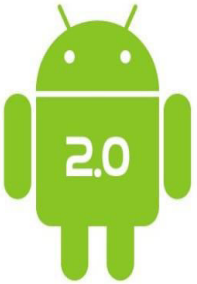
2. Eclipse (oxygen version

Require

- Java Runtime Environment (JRE)
- Java Development Kit (JDK)
- Android Developer Tools
  - Eclipse + ADT plug-in
  - Android SDK Tools
  - Android Platform-tools
  - The latest Android platform
  - The latest Android system image for the emulator

How to set up Android Studio ???

Mobile Devices - Application development


Android apps development

Lecture – 2.2

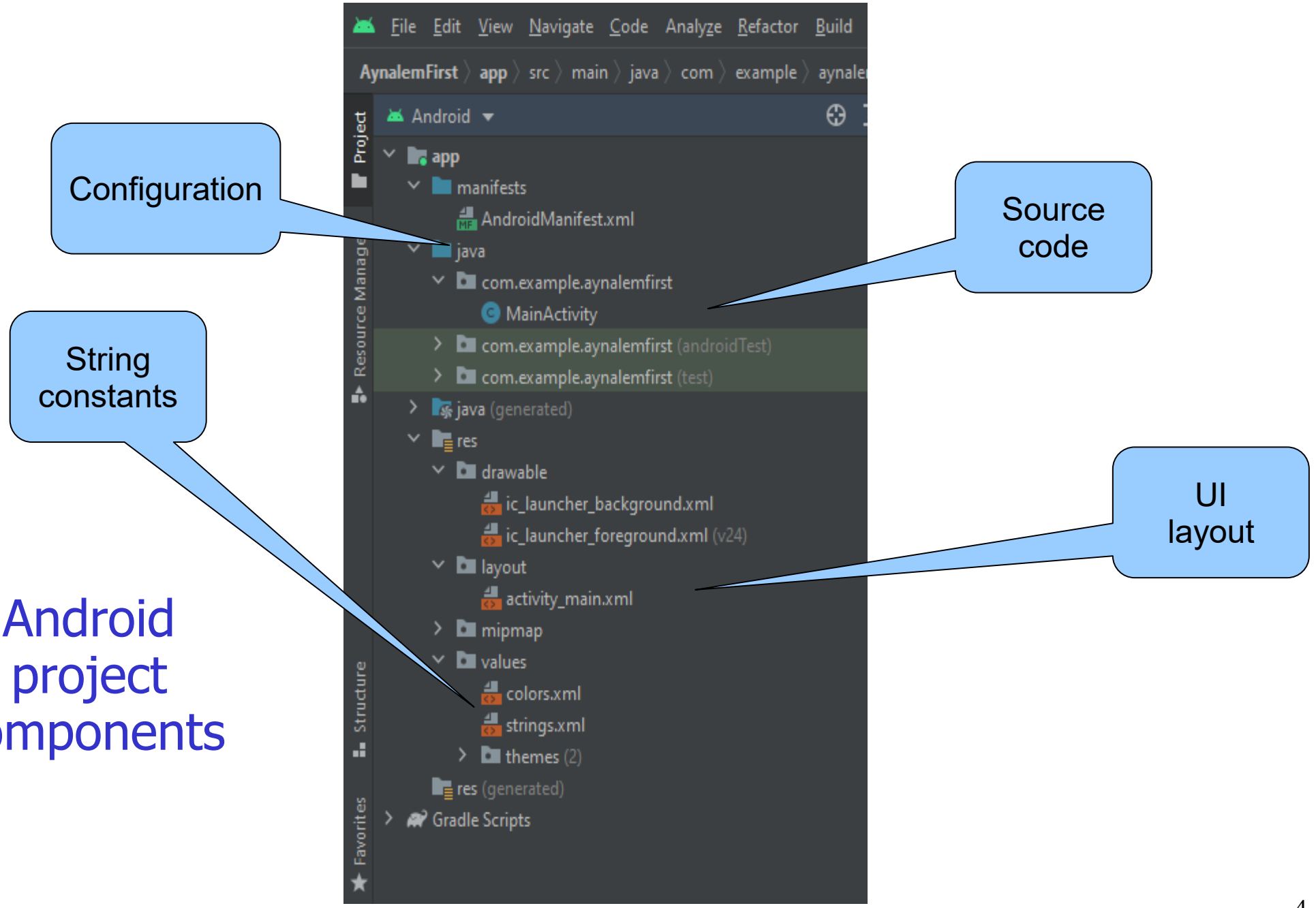
Android Activity

Android Project Components

- Once you created a new project (as 3.1.doc), Android Studio creates the following folders and files in your new project:

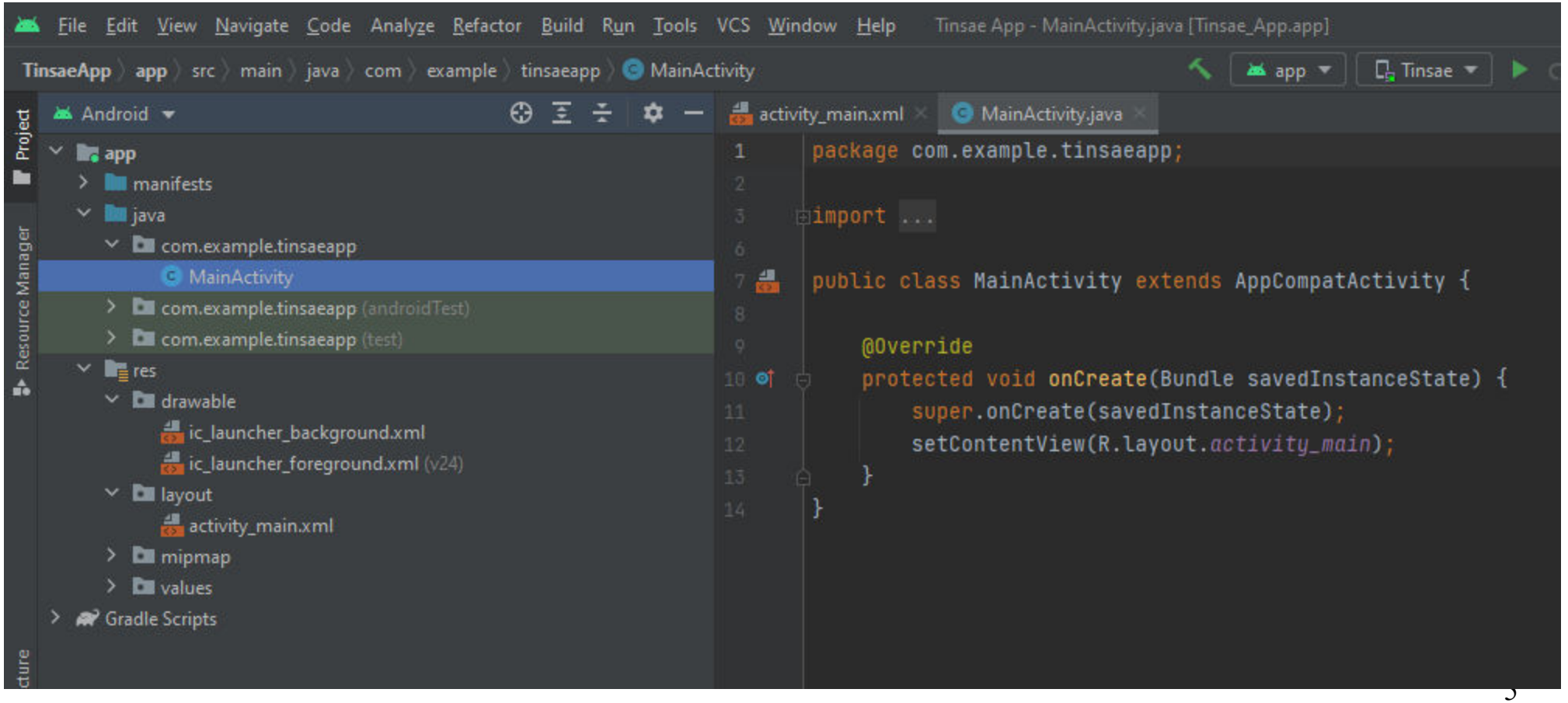
- (java - Source folder) – contains the **java code** of the application. All other Java files for your application go here.
- Res folder (Resource folder) - A folder for your application resources, such as *drawable files*, *layout files*, *string values*, *etc*.
- manifests – (AndroidManifest.xml) the Android Manifest for your project.
- And other Gradle script – contains different scripts such as SDK location, settings … and related.

Android project components



Opening the Home Activity

- The *MainActivity.java* file holds your activity class.
- Android Studio has already written the code to create a UI screen for the application and set its content to the UI defined in the *activity_main.xml*,



Understanding Activities

- An activity is a window that contains the user interface of your applications.
  - applications have one or more activities, and
  - the main aim of an activity is to interact with the user.

- From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's life cycle.

- To create an **`activity`**, you create a Java class that extends the **`Activity base class/ AppCompatActivity:`**

```
public class home extends AppCompatActivity{
```

- Your activity class would then load its UI component defined using the XML file defined in your **res/layout** folder *(activity_home.xml)*.

Activity Java code

```
/* HelloWorld.java */
package com.example.tinsae.dorm;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

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

Activity Java code…

- This line of code is required for the application to run. This calls to the base **Activity** class to perform setup work for the **home** class.

```
super.onCreate(savedInstanceState);
```

- To get the UI to show up on the screen, you have to set the content view for the activity.

```
setContentView(R.layout.activity_home);
```

- **R.layout.activity_home** is the **activity_home.xml** file that is located in the **res/layouts** directory.
- A **bundle** gives you, the developer, a way to pass information back and forth between screens (different activities) in what's known as a bundle.

```
Bundle savedInstanceState
```

Activities and AndroidManifest.xml

- Every activity you have in your application must be declared in your **AndroidManifest.xml** file, like this:

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

 <application
          android:allowBackup="true"
          android:icon="@mipmap/ic_launcher"
          android:label="@string/app_name"
          android:supportsRtl="true"
          android:theme="@style/AppTheme">
    <activity android:name=".home">
        <intent-filter>
         <action android:name="android.intent.action.MAIN" />
          <category
   android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

 </application>

</manifest>
```

Activities and AndroidManifest.xml

- An Android application can be composed of **multiple Activities** …
- Each activity should be declared in the file: **AndroidManifest.xml**
- Add a **child element** to the <application> tag:

```
<application>
   <activity android:name=".MyActivity" />
   <activity android:name=”.SecondActivity" />
</application>
```

Activities and AndroidManifest.xml

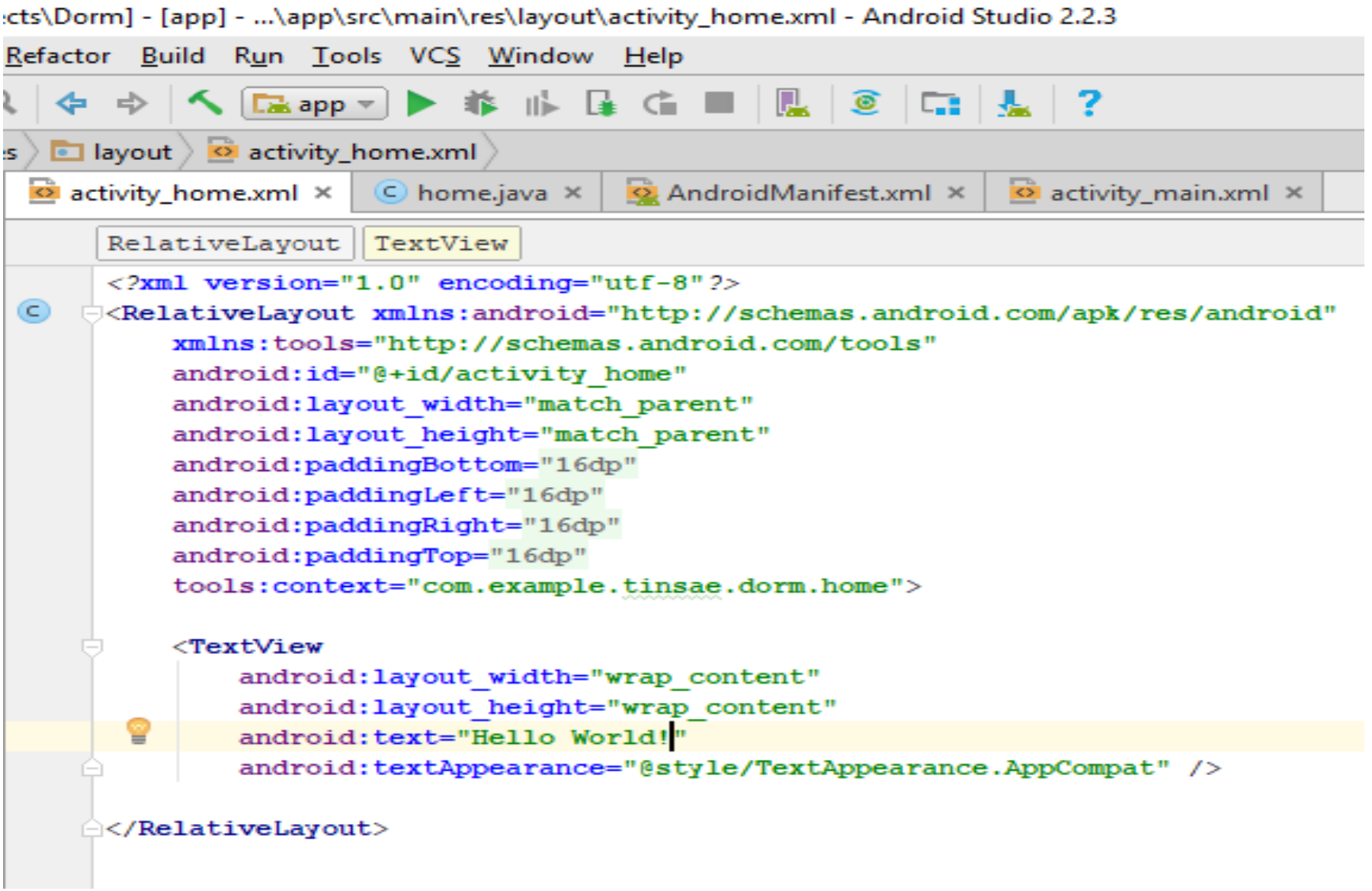
- Each activity has its **Java** class and **layout** file.

```
public class FirstActivity extends Activity {
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_first);
}
```

```
public class SecondActivity extends Activity {
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_two);
}
```

Opening the UI Definition

- Now, let's take a look at our UI interface markup code in the *activity_home.xml* file in the *layout* folder.



Activity XML code

```
/* interface.xml */
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/andro
   id"
   android:orientation= "vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
    >
   <TextView
    android:id="@+id/componentName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Text that will be displayed."
   />
</LinearLayout>
```

Android SDK Layouts

Layout	*Description*
`LinearLayout`	A layout that arranges its children in a single row.
`RelativeLayout`	A layout where the positions of the children can be described in relation to each other or to the parent.
`FrameLayout`	This layout is designed to block out an area on the screen to display a single item. You can add multiple children to a `FrameLayout`, but all children are pegged to the upper left of the screen. Children are drawn in a stack, with the most recently added child at the top of the stack. This layout is commonly used as a way to lay out views in an absolute position.
`TableLayout`	A layout that arranges its children into rows and columns.

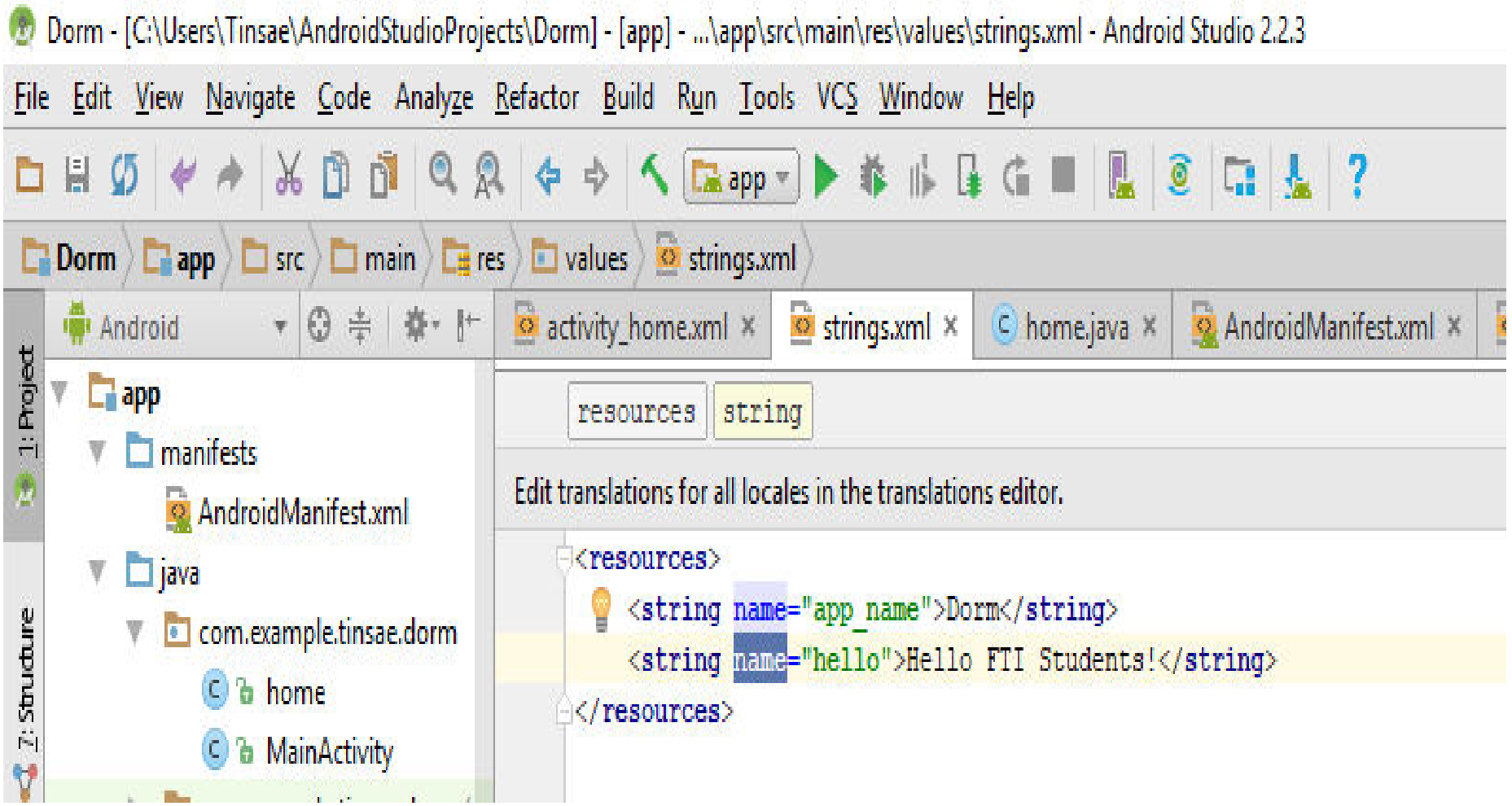
Opening the Strings Resource File

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_home"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.tinsae.dorm.home">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hello"
        android:textAppearance="@style/TextAppearance.AppCompat" />

</RelativeLayout>
```

- So far, we have looked at the Java code, which points to the *activity_home.xml* file, which in turn points to the *strings.xml file*.

Setting a Variable Value in strings.xml

- To set the value of hello defined at android:text="@string/hello", In the string.xml define the string Hello FTI Students! as follows.



Activity String.xml

```
/* interface.xml */
[...]
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="@string/hello”
   android:textAppearance="@style/TextAppearance.AppCompat
   " />
```

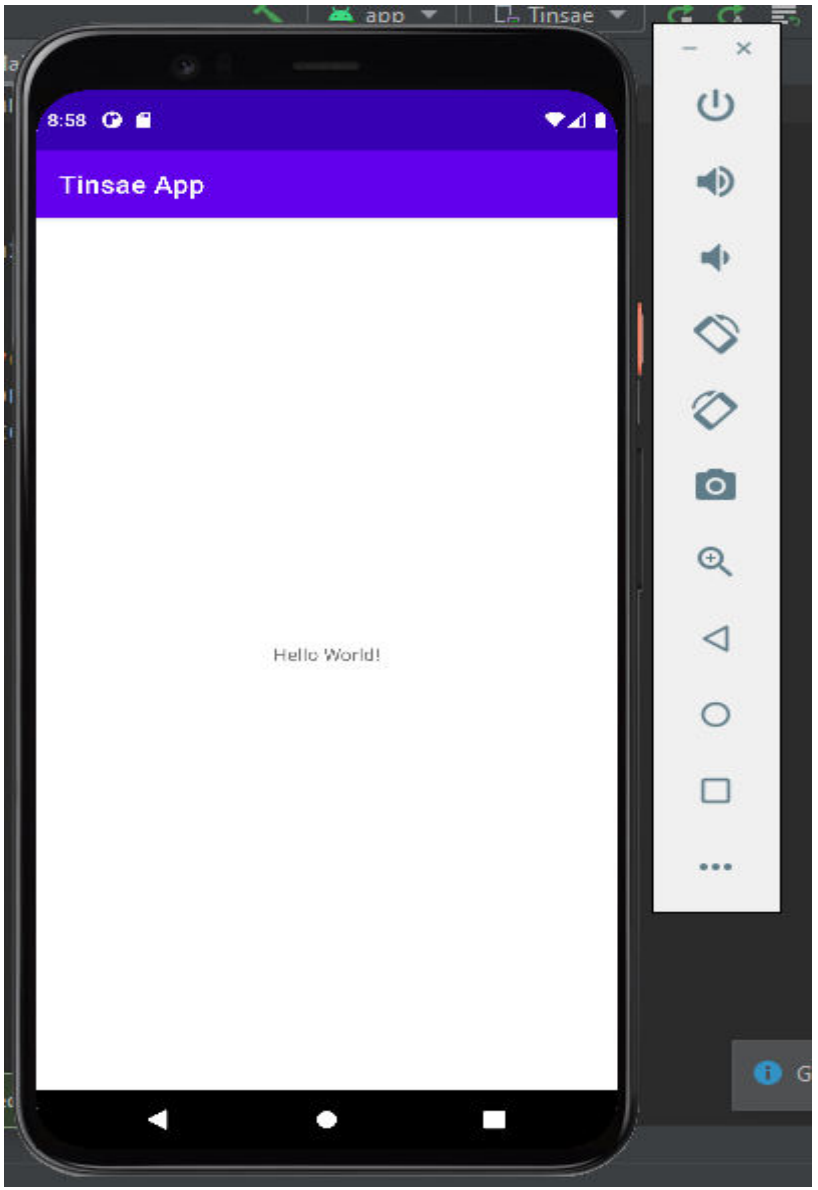
```
/* strings.xml */
<resources>
      <string name="app_name">Dorm</string>
      <string name="hello">Hello FTI Students!</string>
</resources>
```

Activity Home.java

```
/* Home.java */


 package com.example.tinsae.dorm;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class home extends AppCompatActivity {

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

Running the App

- To compile and run the application, click on Run → **Run 'app' and it will be displayed as ….**



Application Components

- Application components are the essential building blocks of an Android application.
- There are following four main components that can be used within an Android application:
- **Activities** - They dictate the UI and handle the user interaction to the Smartphone screen.
- **Services** - They handle background processing associated with an application.
- **Broadcast Receivers** - They handle communication between Android OS and applications.
- **Content Providers** - They handle data and database management issues.

Application Components...

Activities

- An activity represents a single screen with a user interface.
- For example,
  - an email application might have one activity that shows a list of new emails,
  - another activity to compose an email, and
  - another activity for reading emails.
- An activity is implemented as a subclass of Activity class as follows:

```
public class home extends AppCompatActivity {
```

Application Components...

Services

- A service is a component that runs in the background to perform long-running operations.
- 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. /IntentService
- A service is implemented as a subclass of Service class as follows:

```
public class MyService extends Service {
 }
```

Application Components...

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system.
- For example,
  - applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use,
  - so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcasted as an Intent object.

```
public class MyReceiver extends BroadcastReceiver {
  }
```

Application Components...

ContentProvider

- A content provider component supplies data from one application to others on request.
  - Such requests are handled by the methods of the ContentResolver class.
  - The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of ContentProvider class

```
public class MyContentProvider extends ContentProvider {
 }
```

Additional Components

- There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them such as :
  - Widget
  - View (custom view) … etc

Activity's Lifecycle

- Base class mostly for visual components
  - extends **`Activity`**
  - override **`onCreate`**

- Create a class that is a subclass of Activity
- Implement callback methods
  - **`OnCreate()`**:
    - Initialize
    - **`SetContentView()`**

Activity's Lifecycle

- An activity is a window that contains the user interface of your applications.
  - applications have one or more activities, and
  - the main aim of an activity is to interact with the user.

- From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's life cycle.

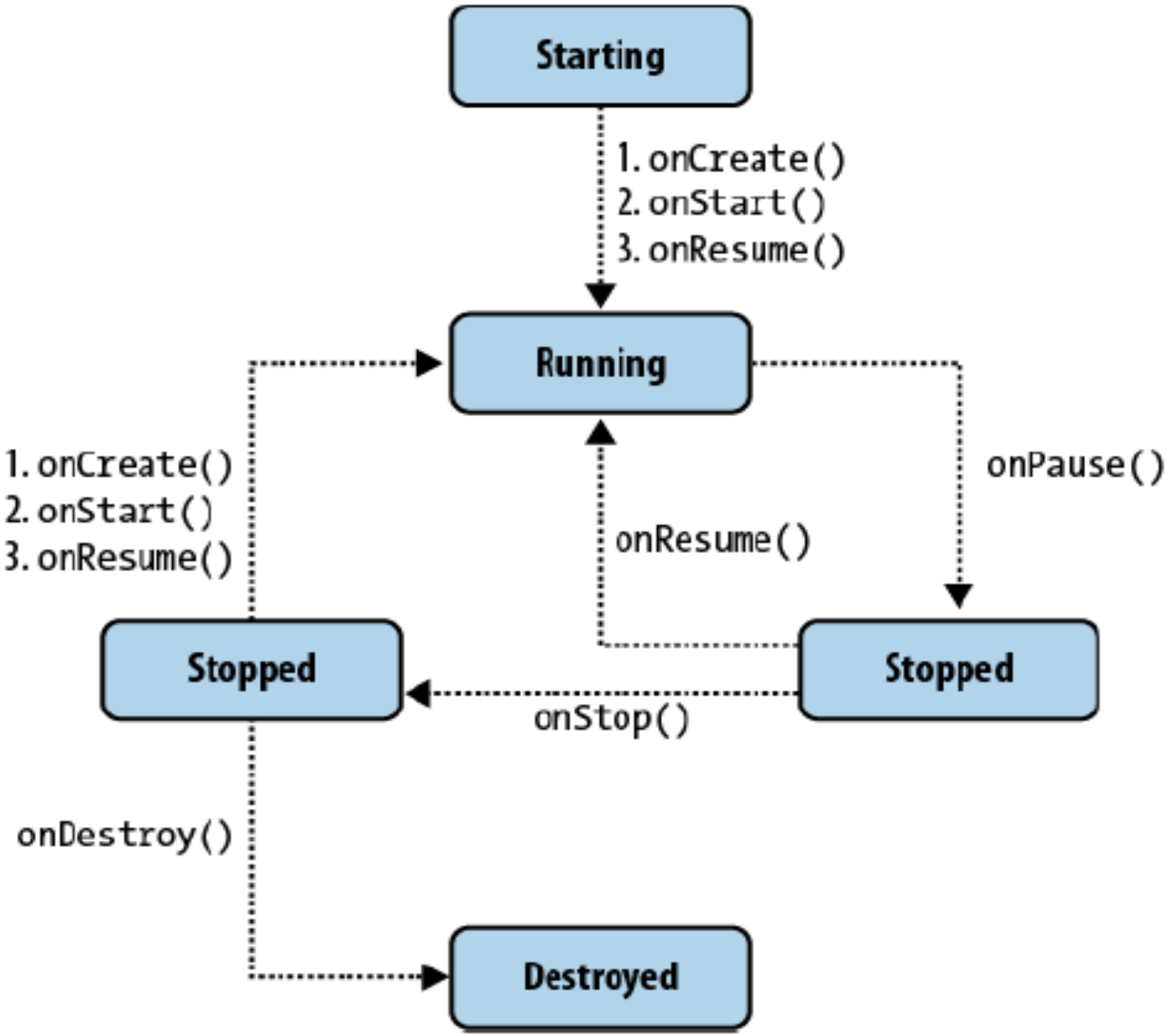
- To create an **`activity`**, you create a Java class that extends the **`Activity base class`**:

```
public class MyHelloWorld extends Activity {
```

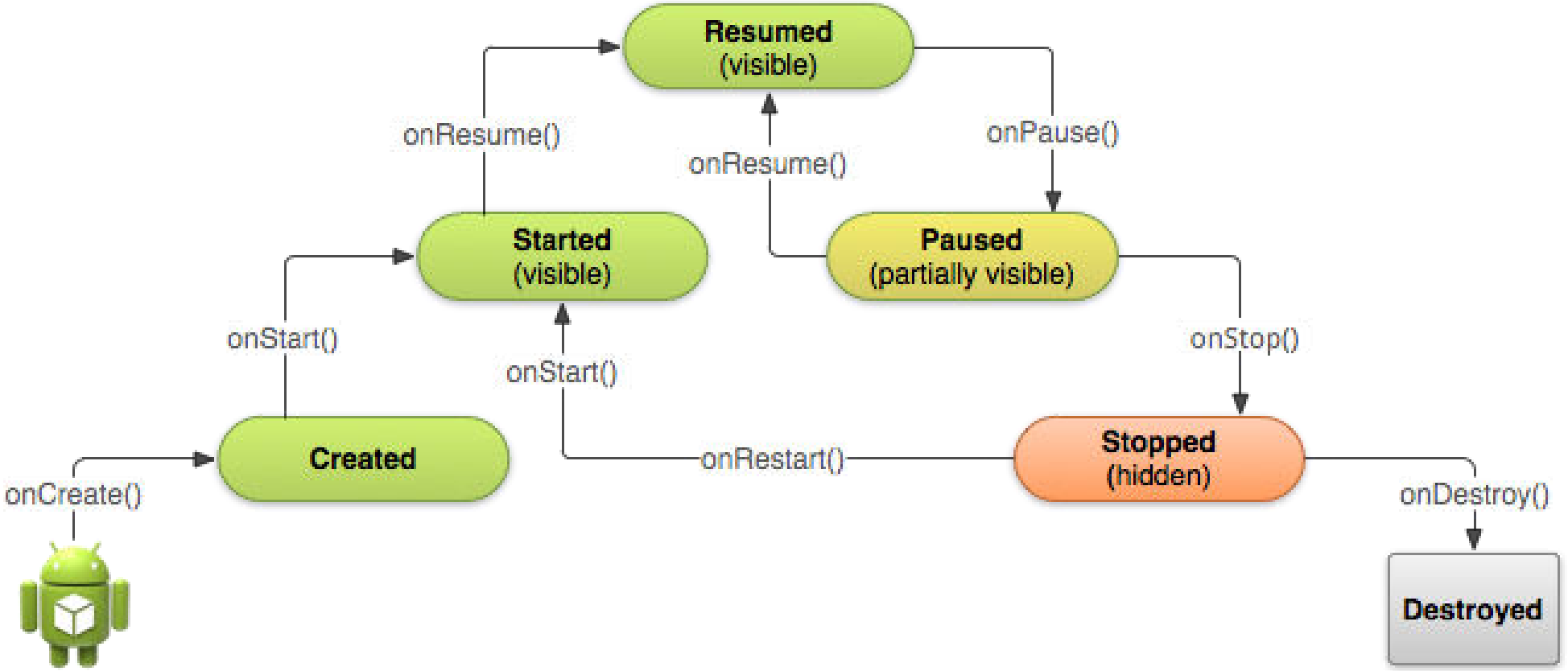
Activities States

- Resumed
  - The activity is in the foreground, and the user can interact.
- Paused
  - The activity is partially overlayed by another activity. Cannot execute any code nor receive inputs.
- Stopped
  - Activity is hidden, in the background. It cannot execute any code.

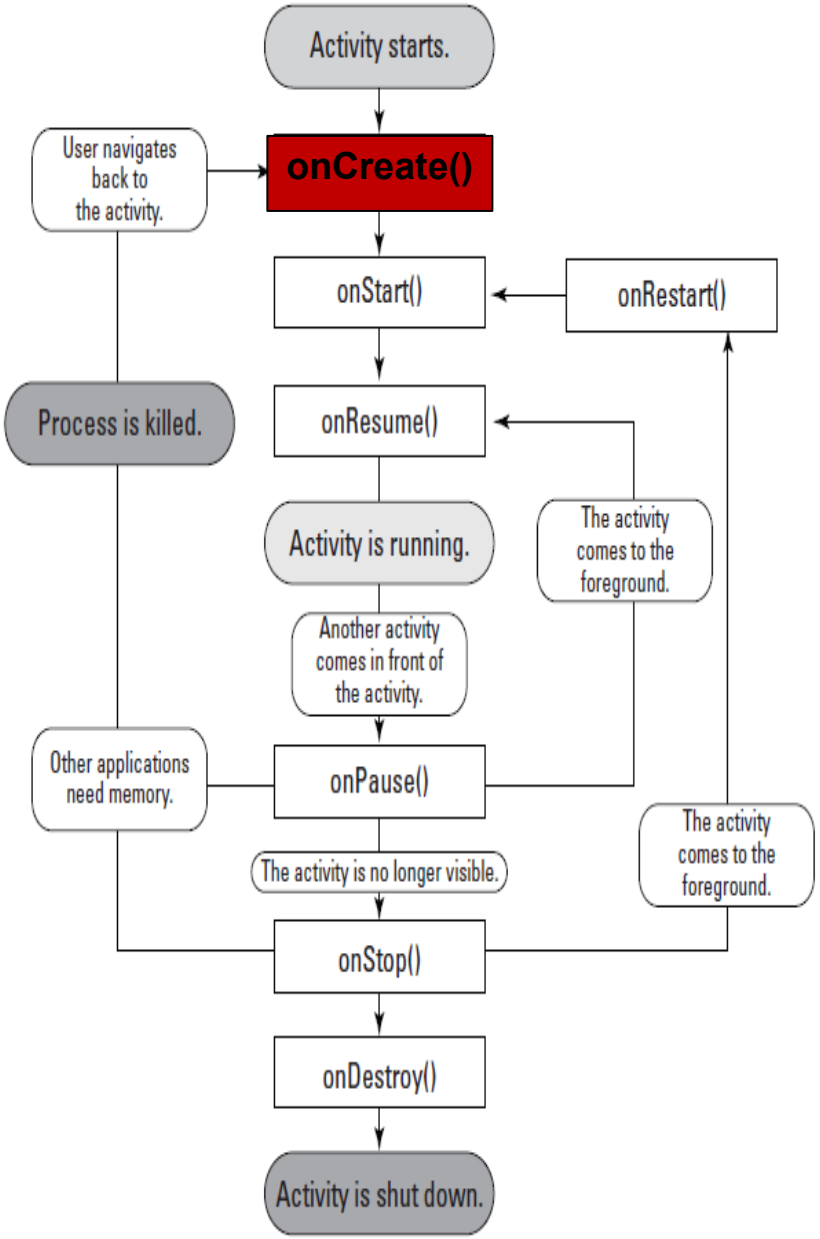
Activity's Lifecycle...



Activity's lifecycle

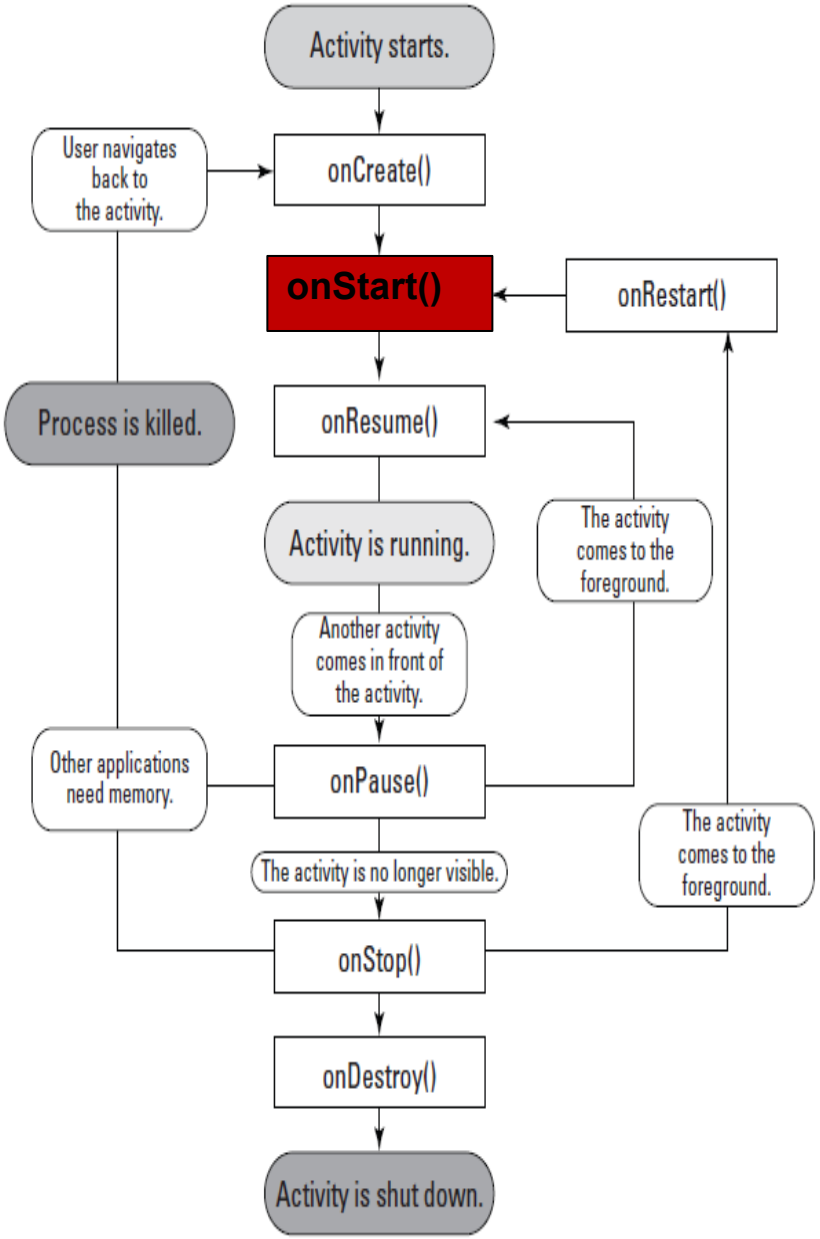


Activity lifecycle



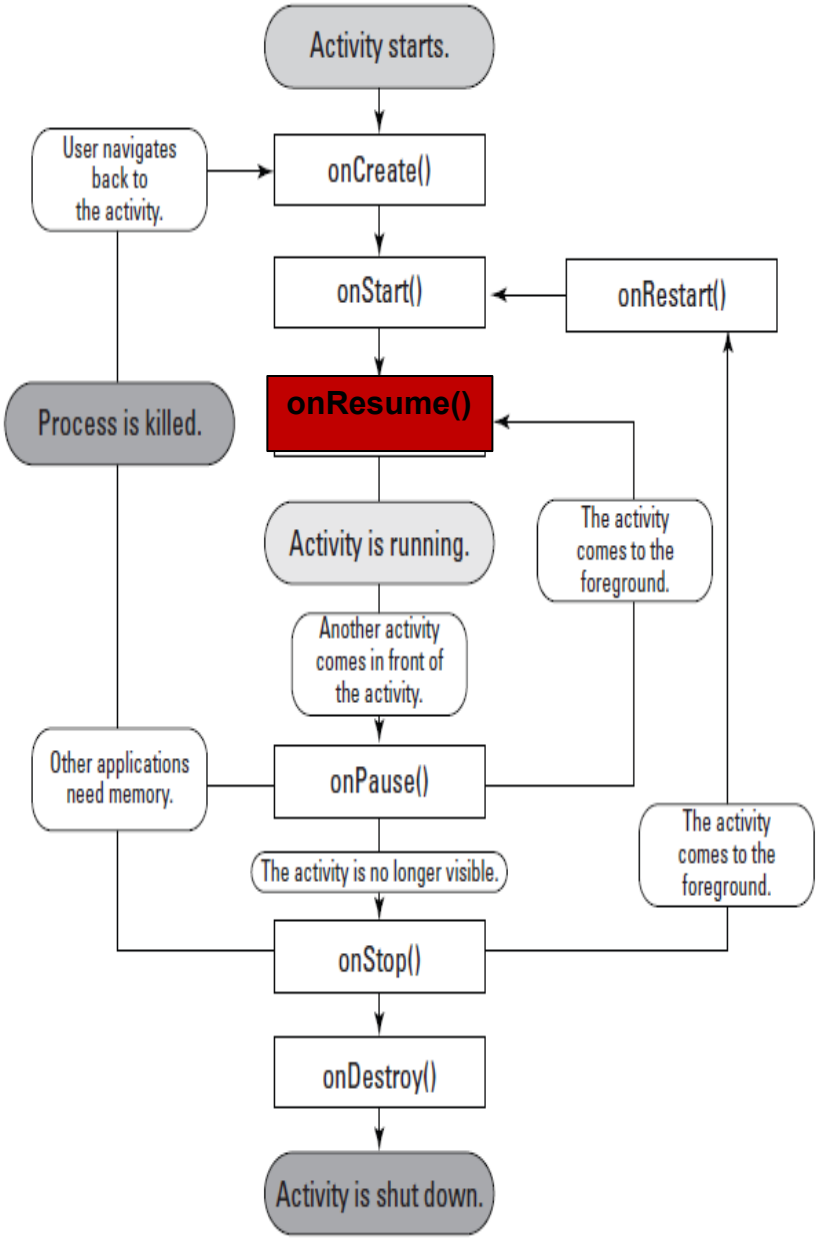
- **`OnCreate()`**
  - Called when the activity is first created
  - Should contain the initialization operations
  - Has a Bundle parameter
  - If onCreate() succesfull terminates, it calls onStart()

Activity lifecycle



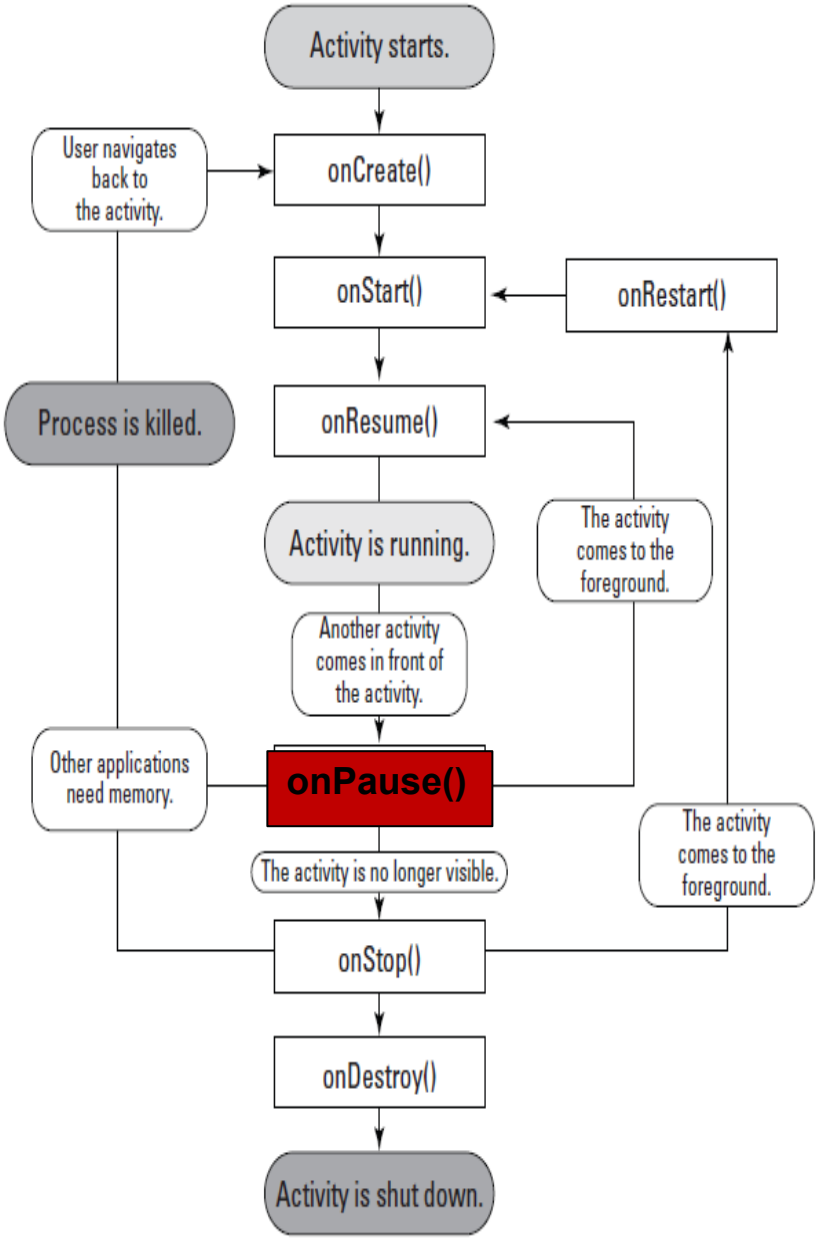
- **`OnStart()`**
  - Called when onCreate() terminates
  - Called right before it is visible to user
  - If it has the focus, then onResume() is called
  - If not, onStop() is called

Activity lifecycle



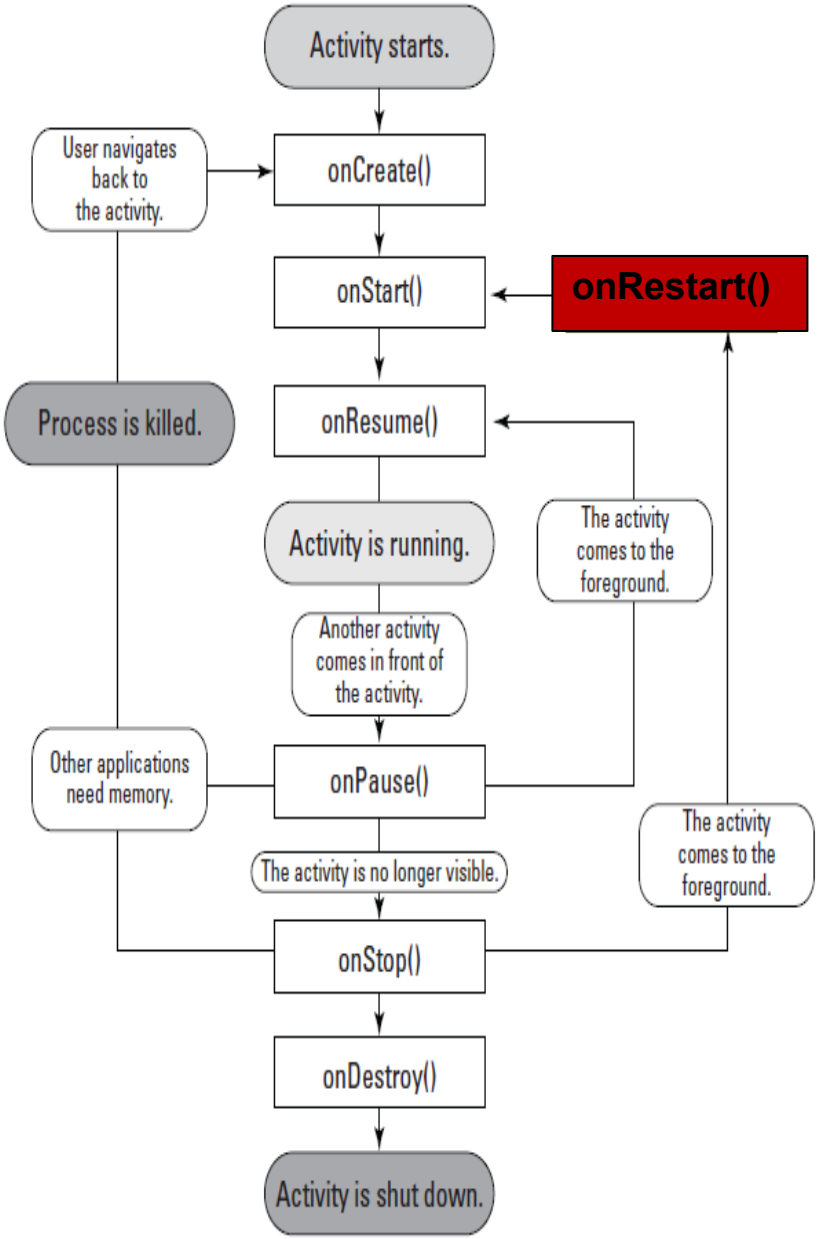
- **`OnResume()`**
  - Called when the activity is ready to get input from users
  - Called when the activity is resumed too
  - If it succesfully terminates, then the Activity is RUNNING

Activity lifecycle



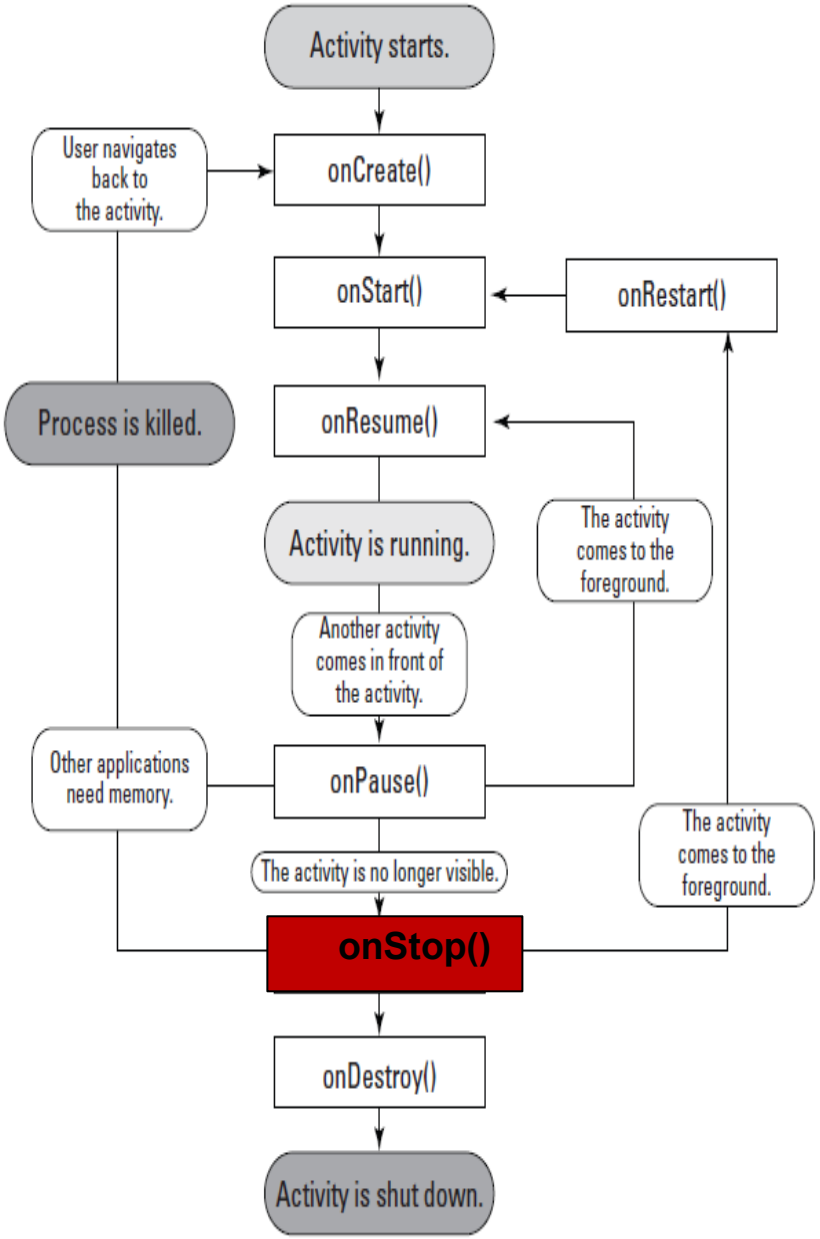
- **`OnPause()`**
  - Called when the current activity is being paused and
  - when another activity comes to the foreground, or
  - when someone presses back
  - Commit unsaved changes to persistent data
  - Stop cpu-consuming processes
  - Make it fast

Activity lifecycle



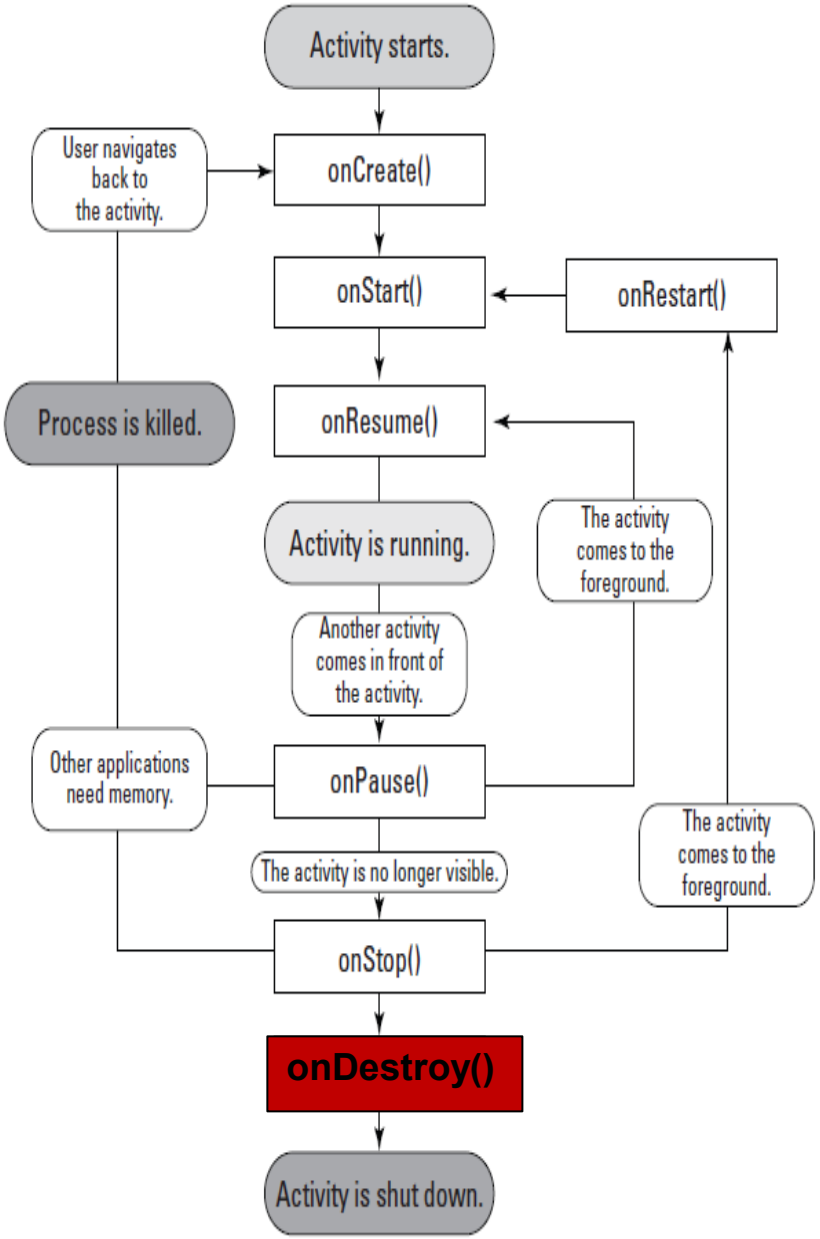
- **`OnRestart()`**
  - Called when the activity has been stopped and is restarting again
  - Similar to **`onCreate()`**
  - We have an activity that was previously stopped

Activity lifecycle



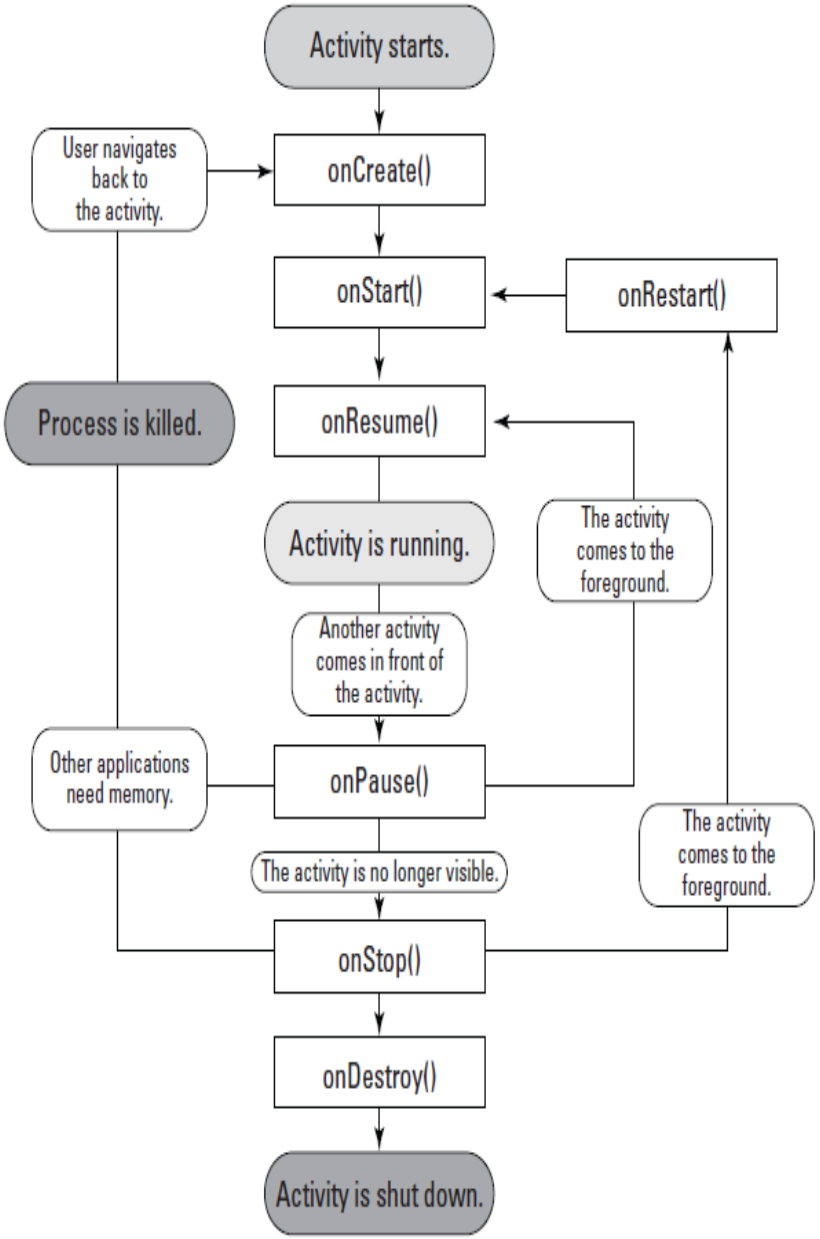
- **`OnStop()`**
  - Activity is no longer visible to the user
  - Could be called because:
    - the activity is about to be destroyed
    - another activity comes to the foreground

Activity lifecycle



- **`OnDestroy()`**
  - The activity is about to be destroyed
  - Could happen because:
  - The systems need some stack space
    - Someone called finish() method on this activity

Activity loops



Mainly 3 different loops

Entire lifetime

- Between onCreate() and onDestroy().
- Setup of global state in onCreate()
- Release remaining resources in onDestroy()

Visible lifetime

- Between onStart() and onStop().
- Maintain resources that has to be shown to the user.

Foreground lifetime

- Between onResume() and onPause().

Activity's Lifecycle (Summary)

- The **Activity base class** defines a series of events that governs the life cycle of an activity. The **Activity class** defines the following events:

- `onCreate()` – Called when the activity is first created
- `onStart()` – Called when the activity becomes visible to the user
- `onResume()` – Called when the activity starts interacting with the user
- `onPause()` – Called when the current activity is being paused and the previous activity is being resumed
- `onStop()` – Called when the activity is no longer visible to the user
- `onDestroy()` – Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- `onRestart()` – Called when the activity has been stopped and is restarting again

Activity states

- **Active** (or running)
  - Foreground of the screen (top of the stack)

- **Paused**
  - Lost focus but still visible
  - Can be killed by the system in extreme situations

- **Stopped**
  - Completely covered by another activity
  - Killed if memory is needed somewhere else

Activity Lifecycle demo: home.java

```
package com.example.tinsae.dorm;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;

public class home extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);
        Toast t= Toast.makeText(home.this,"end of
onCreate",Toast.LENGTH_LONG);
        t.show();
    }
```

mainActivity.java

```
//called when the activity is about to become visible

@Override
public void onStart() {
    super.onStart();
    Toast t= Toast.makeText(home.this,"end of
onStart",Toast.LENGTH_LONG);
    t.show();
}
//called when the activity has become visible
@Override
public void onResume() {
super.onResume();
Toast t= Toast.makeText(home.this,"end of
onResume",Toast.LENGTH_LONG);
t.show();}
```

mainActivity.java

```
//called when another activity is taking focus

@Override
public void onPause() {
super.onPause();
Toast t= Toast.makeText(home.this,"end of
onPause",Toast.LENGTH_LONG);
t.show();
}
//called when the activity is no longer visible
@Override
public void onStop() {
super.onStop();
Toast t= Toast.makeText(home.this,"end of
onStop",Toast.LENGTH_LONG);
t.show();
}
```

mainActivity.java

```
//called just before the activity is destroyed
@Override
public void onDestroy(){
super.onDestroy();
Toast t= Toast.makeText(home.this,"end of
onDestroy",Toast.LENGTH_LONG);
t.show();

}
}
```

Implement the above demo that demonstrates activity life cycle of android and show it to your teacher.



Chapter3-Application development for mobile devices (using Android)

2.3. UI Design

Widgets and Layouts

Android Application

Android Applications' anatomy:

- **Activities** → Application Components (screens)
- **Intents** → Communication between components
- **Views** → occupies a rectangular area on the screen and is responsible for drawing and event handling.
- **Widget** → are subclasses of View. They are used to create interactive UI components such as buttons, checkboxes, labels, text fields, etc.
- **Layouts** → are invisible structured containers used for holding other Views and nested layouts.

Views: Outline

- Could be declared in an XML file
- Could also be declared inside an Activity
- Every view has a unique ID
- Use findViewById(int id) to get it
- View is the base class for **widgets**, which are used to create interactive UI components like buttons, text fields, etc.
- The **ViewGroup** is a subclass of View
  - and provides invisible container that hold other Views or other ViewGroups and
  - define their layout properties.

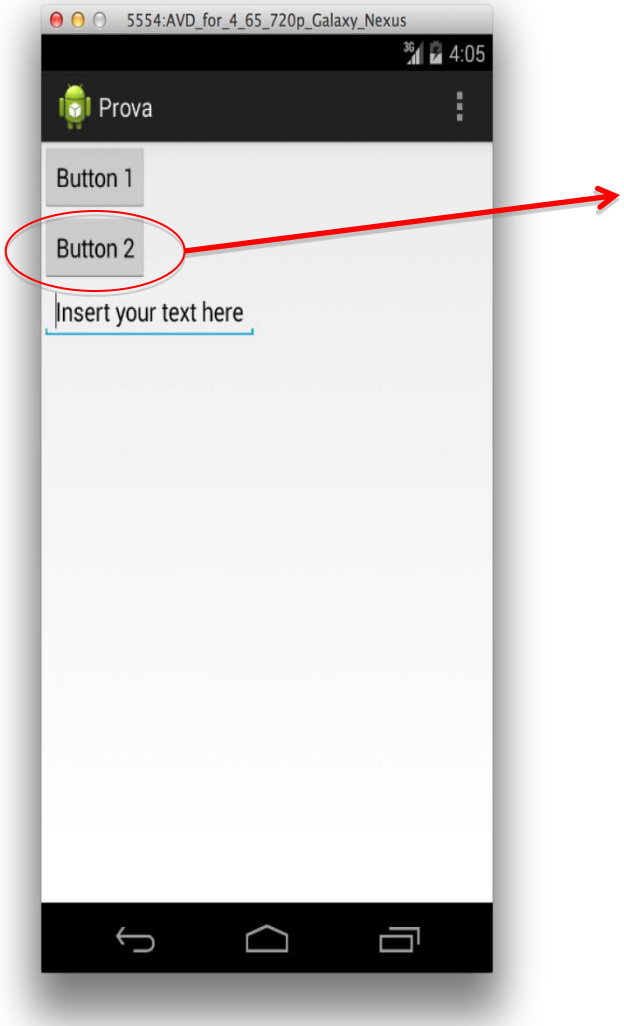
Views: Outline

- Once your layout is defined, you can load the layout resource from your application code, in **yourActivity.onCreate()** callback implementation as shown below:

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

Android: Views objects

Views → basic building blocks for user interface components



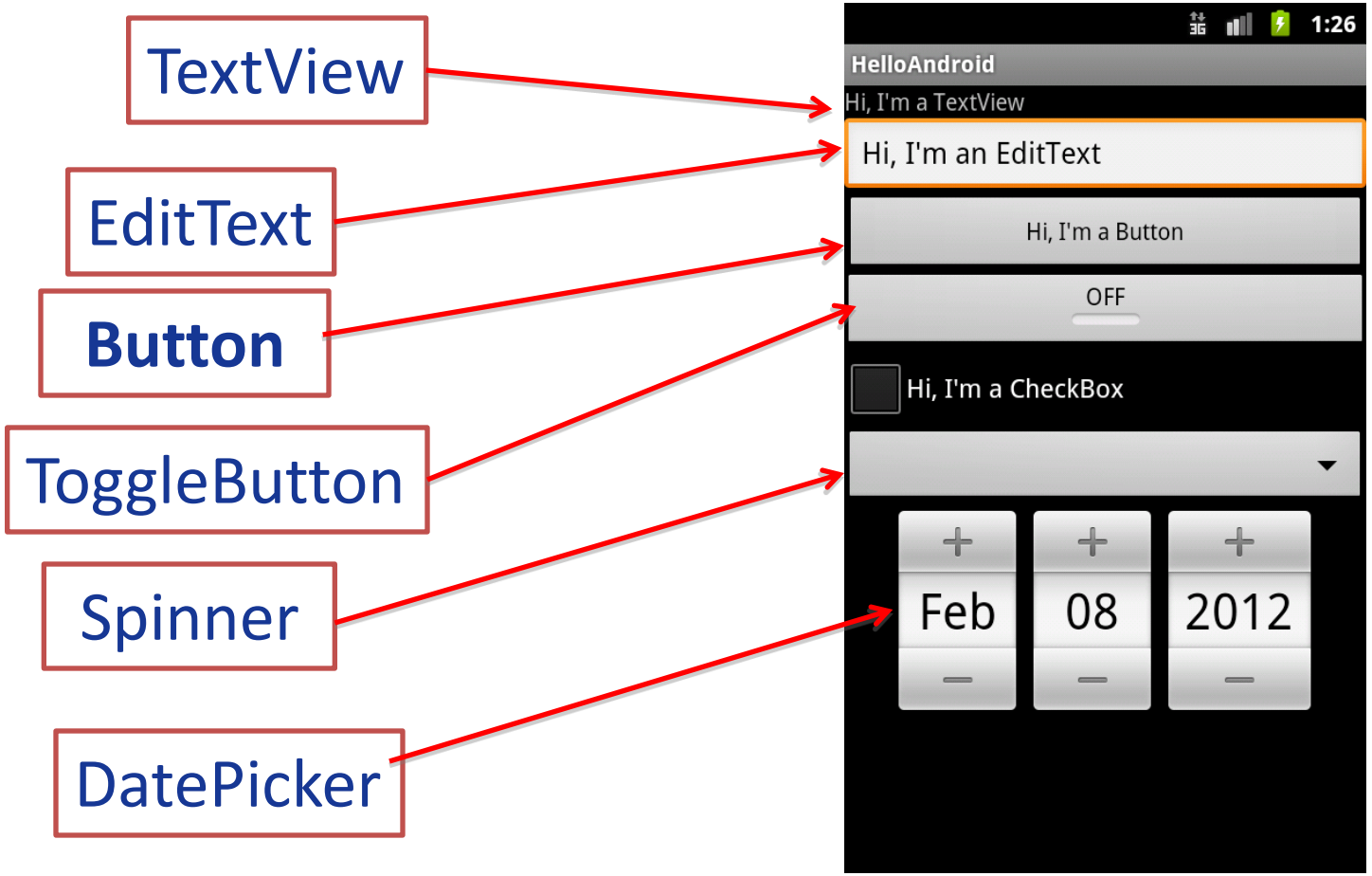
- ✧ Rectangular area of the screen
- ✧ Responsible for **drawing**
- ✧ Responsible for **event handling**

EXAMPLEs of **VIEWS** objects:

- GoogleMap
- WebView
- **Widgets**
- …
- User-defined Views

Android: Views objects

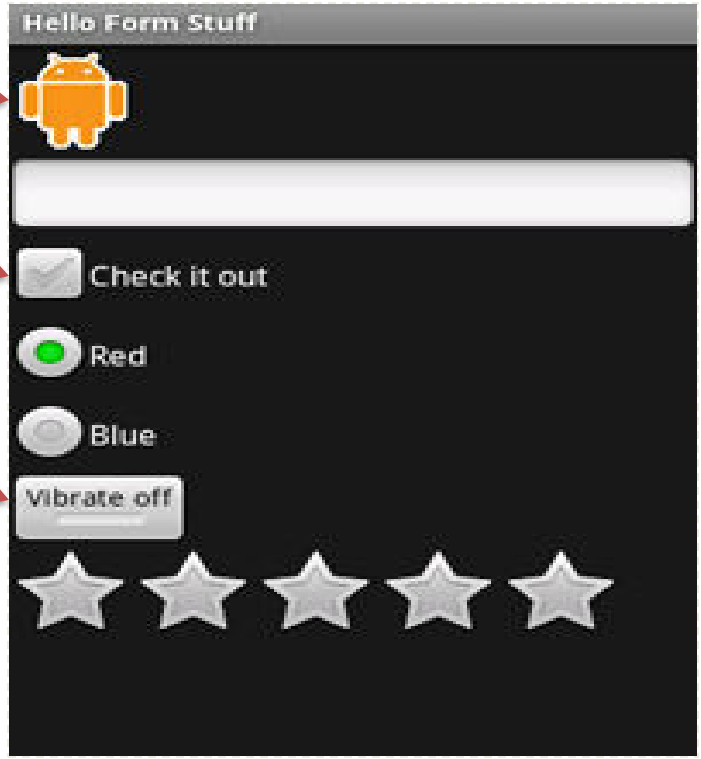
Widget→ Pre-defined interactive UI components (android.view.widgets)



Android: Views objects

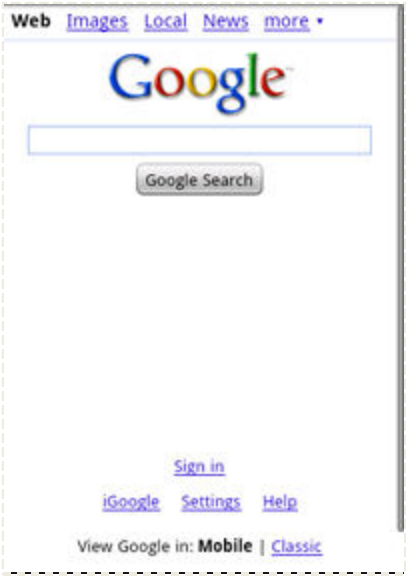
Widget→ Pre-defined interactive UI components (android.view.widgets)

- ImageButton
- CheckBox
- RadioButton
- ToggleButton
- RatingBar



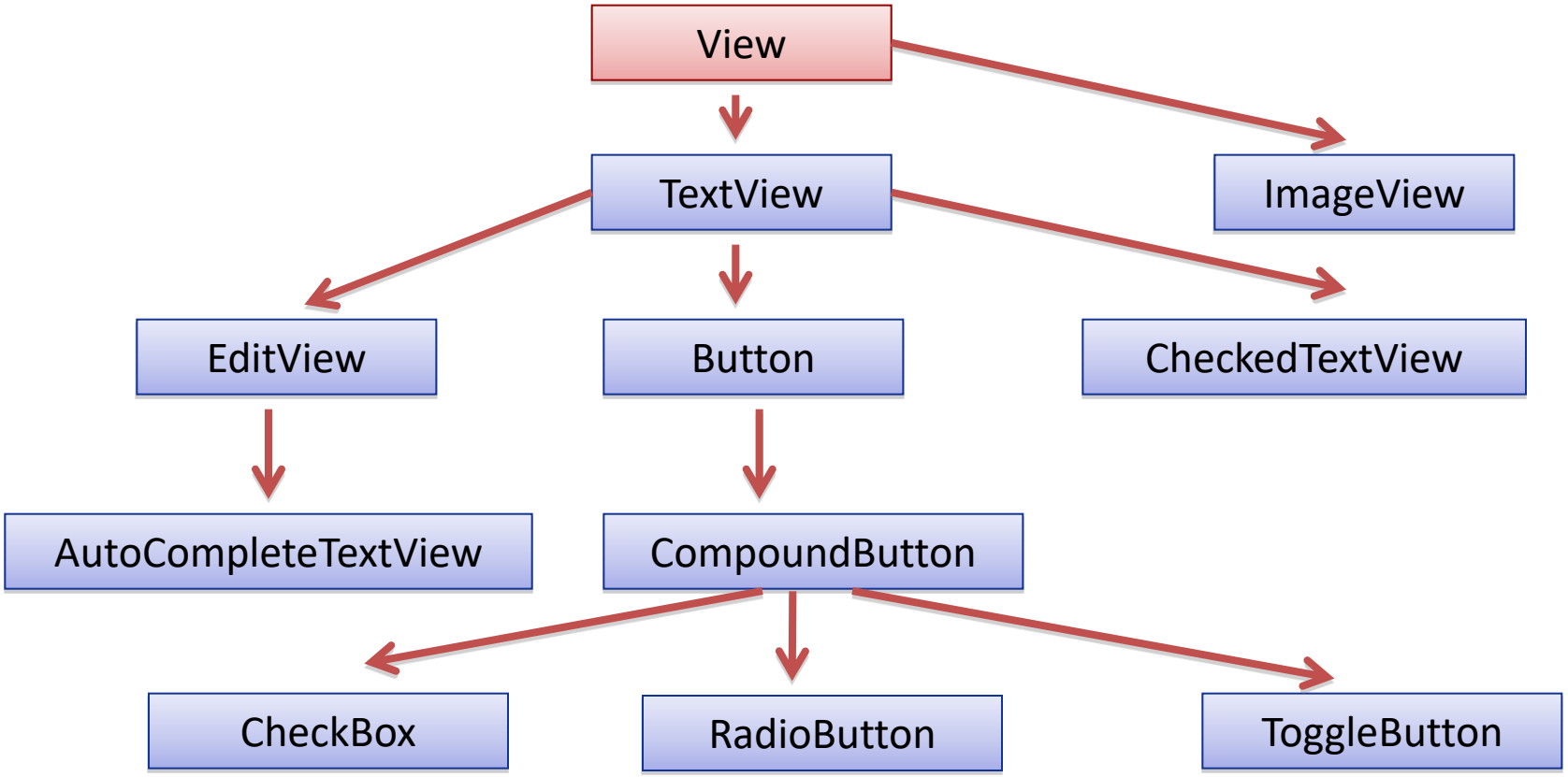
Android: Views objects

- DatePicker
- TimePicker
- Spinner
- AutoComplete
- Gallery
- MapView
- WebView



Widgets: Hierarchy of the classes …

- **Widgets** are organized on a *hierarchy* of classes …



Widgets: Java and XML code

- Widgets can be created in the **XML layout files**

```
< TextView
            android:id="@+id/textLabel”
            android:width=“100dp”
            android:height=“100dp”
            android:layout_width="match_parent"
            android:layout_height="wrap_content”
            android:visibility="visible”
            android:enabled="true”
            android:scrollbars="vertical”
            ….
/>
```

Widgets: TextView

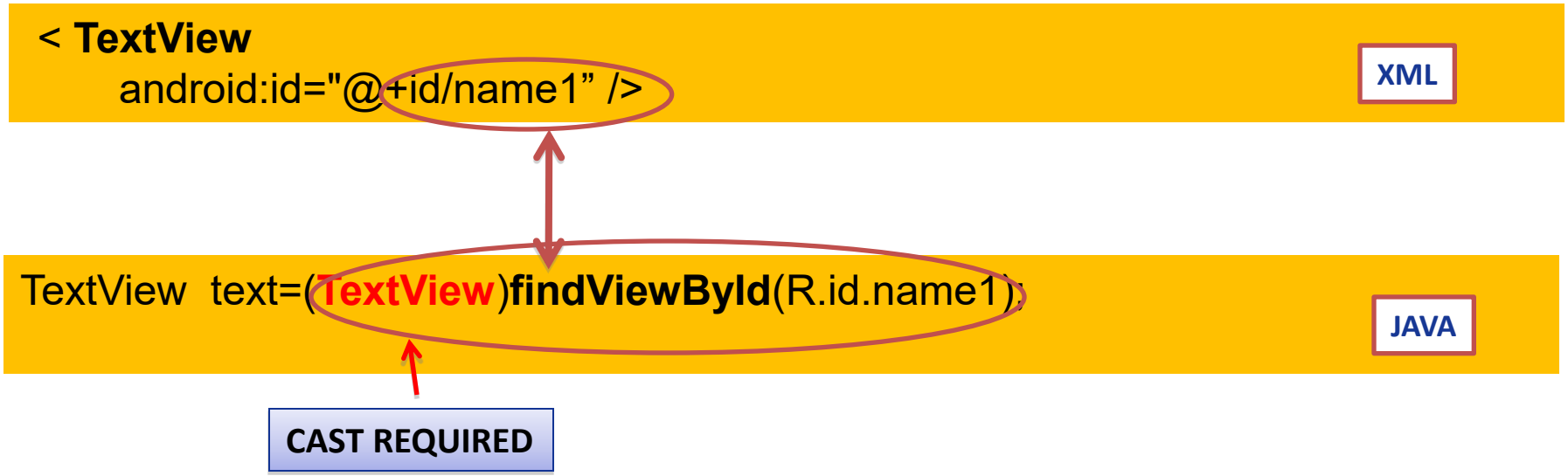
- **XML** tags: **<TextView> </TextView>**

- Could be filled with **strings**
- Not directly editable by users
- Usually used to display **static** informations

```
 <TextView
        android:text= "Welcome to android world"
        android:id="@+id/textLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content”
/>
```

Widgets: Java and XML code

- **Widgets** can be created in XML and accessed in Java



Example : (out put – Hello FTI Students

Activity_id_access.xml layout text view

```
<TextView
    android:text="TextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_alignParentStart="true"
    android:layout_marginStart="63dp"
    android:layout_marginTop="52dp"
    android:id="@+id/txtIA" />
```

When text view txtIA accessed in IdAccess.Java

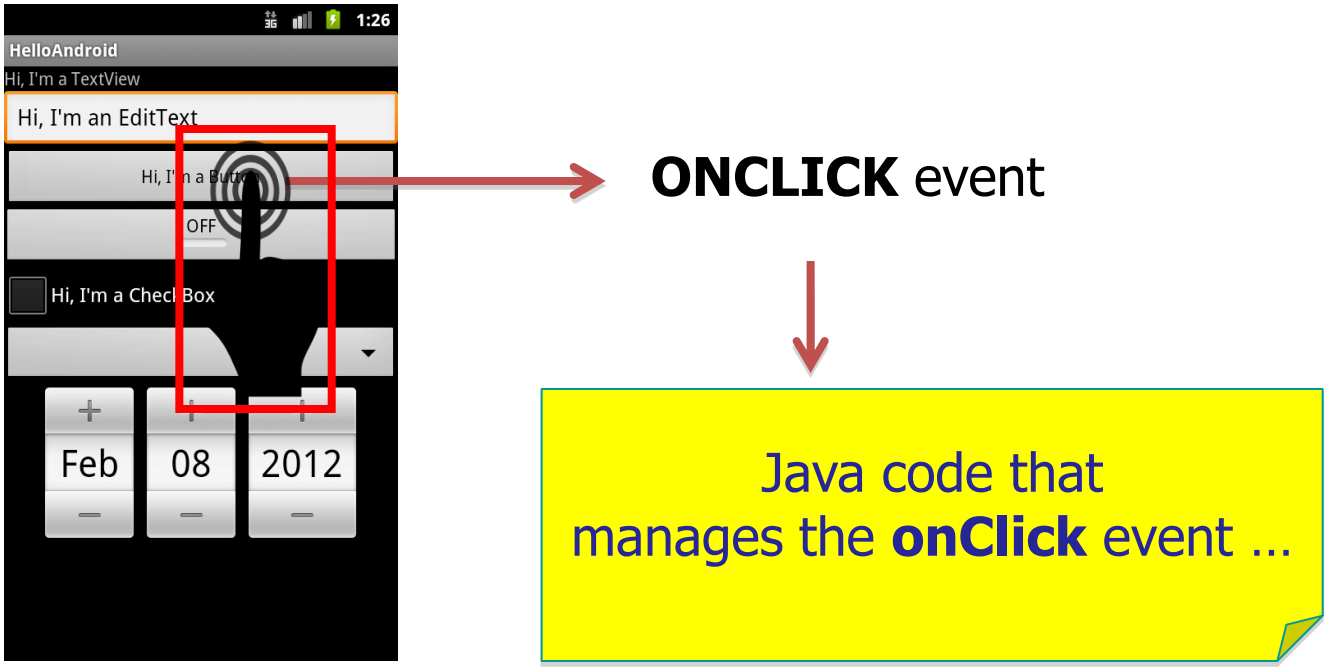
```
package com.example.tinsae.dorm;

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

public class IdAccess extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_id_access);
        TextView ia= (TextView) findViewById(R.id.txtIA);
        ia.setText("Hello FTI Students");
    }
}
```

TextView inside .java

```
onCreate(Bundle savedInstanceState) {//inside on create
setContentView(R.layout.activity_id_access);
TextView ia= (TextView) findViewById(R.id.txtIA);
ia.setText("Hello FTI Students");
ia.setTextColor(Color.RED); //set red color for text view
ia.setTextSize(20); //set 20dp size of text
ia.setBackgroundColor(Color.BLACK);//set
background color
```

Widgets: Java and XML code

- Each Widget can generate events, that can be captured by **Listeners** that define the appropriate actions to be performed in response to each event.



Widgets: EditText

- **XML** tags: **<EditText> </EditText>**
  - Similar to a TextView, but **editable** by the users
  - An appropriate **keyboard** will be displayed

```
<EditText
    android:id="@+id/edtext"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:inputType="choices"
/>
```

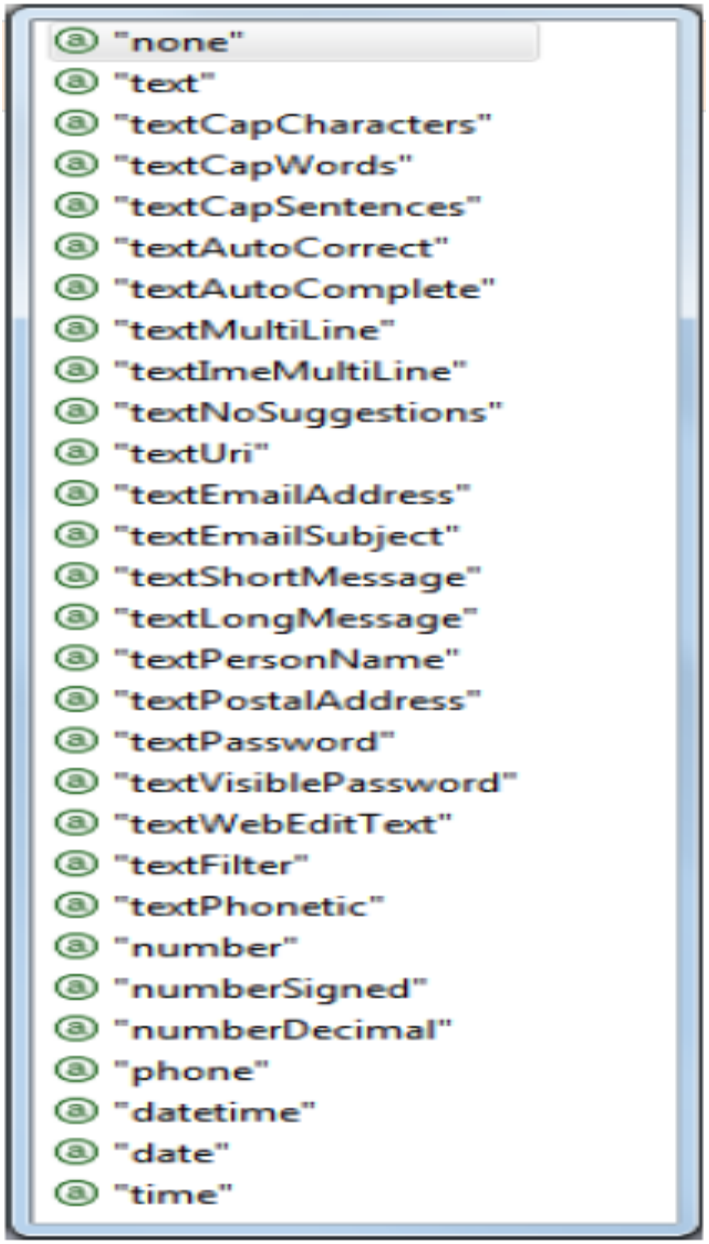
Widgets: EditText

Input Type Formats

- Setting the EditText box to accept a particular choice of data-type, is done through the XML clause

  `android:inputType="choices"`

- where **`choices`** include any of the single values shown in the figure.
- You may combine types, for instance: **`textCapWords|textAutoCorrect`**

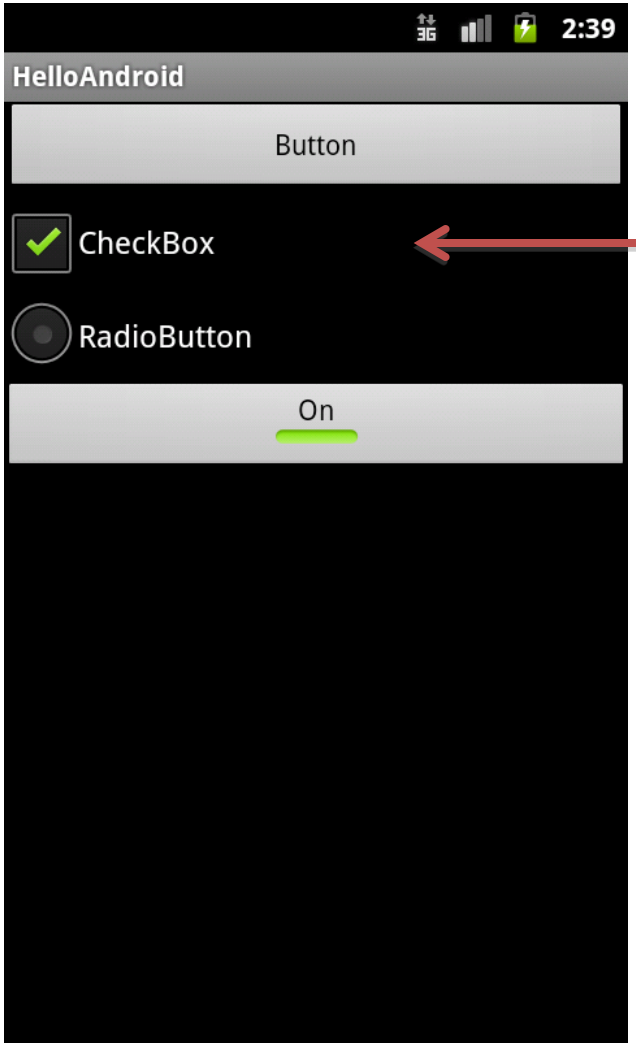


Widgets: Button

- **XML** tags: **<Button> </Button>**
- Superclass of a TextView, but not directly **editable** by users
- Can generate events related to click, long click, drag, etc

```
<Button
      android:text= "Save"
      android:id="@+id/idButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:background="#F00"
      android:textColor="#0F0"
          />
```

Widgets: CheckBox



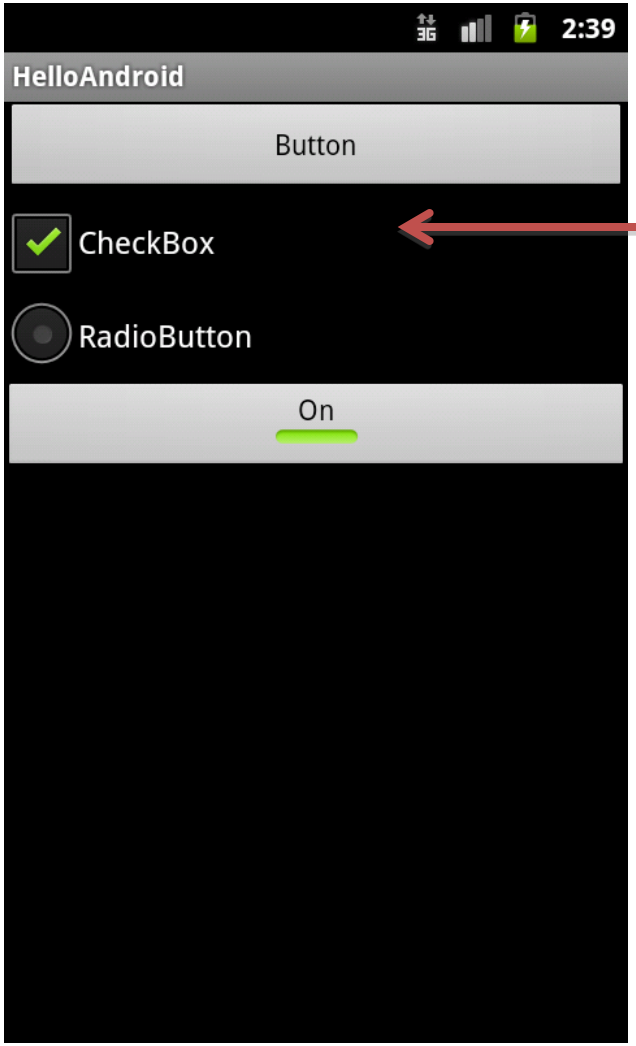
checkBox CompoundButton

XML tags: <CheckBox>

</CheckBox>

```
<CheckBox
android:layout_width="wrap_con
tent"
android:layout_height="wrap_co
ntent"
android:id="@+id/buttonCheck"
android:text="CheckBox"
android:checked="true"
/>
```

Widgets: CheckBox…

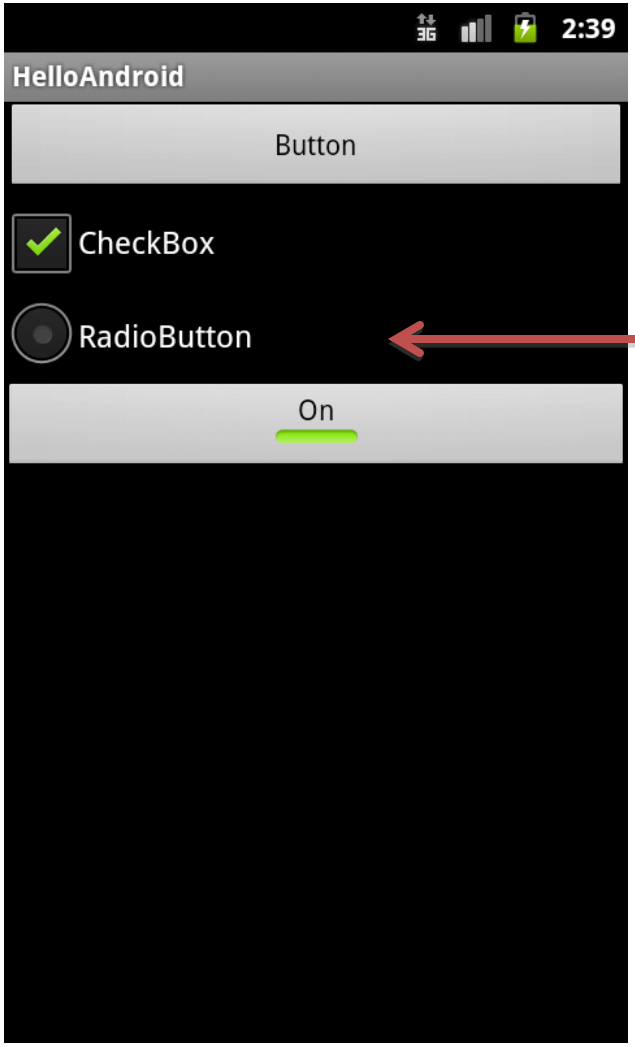


checkBox CompoundButton

- public boolean **isChecked**(): Returns true if the button is checked, false otherwise.
- public boolean **setChecked**(bool)

Listener:
onCheckedChangeListener

Widgets: Radio Button



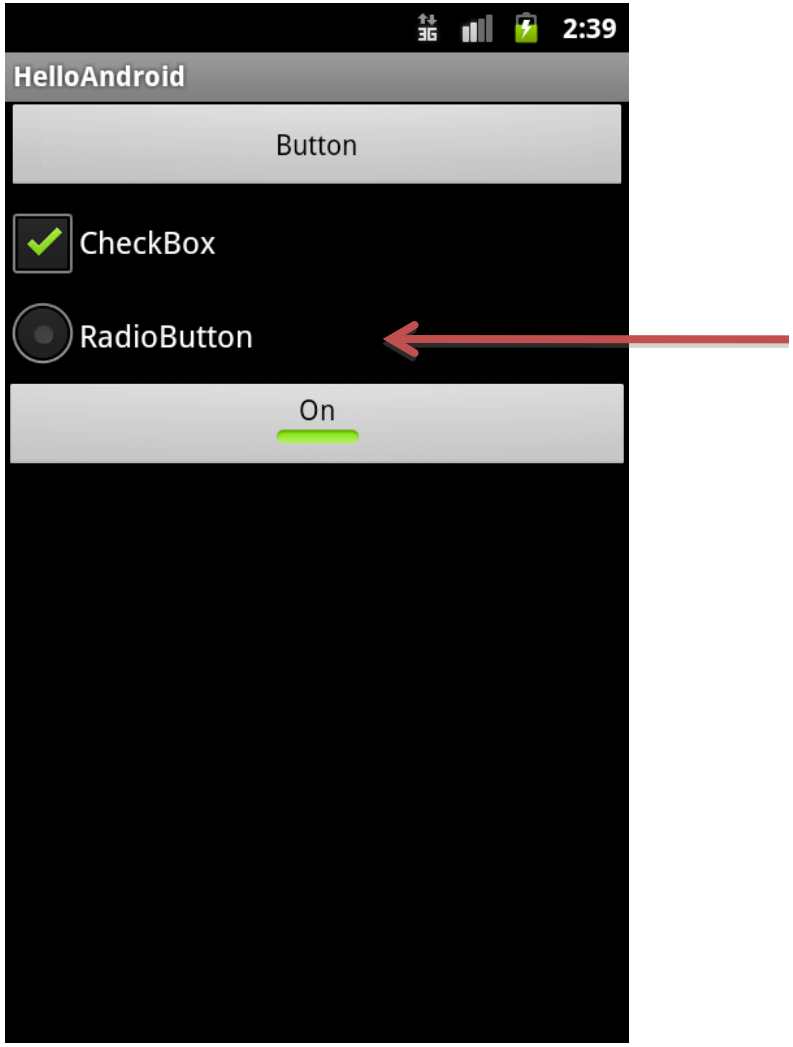
radioButton CompoundButton

XML tags: **<RadioButton>**

</RadioButton>

```
<RadioButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/buttonRadio"
android:text="RadioButton"
android:checked="true"
                />
```

Widgets: Radio Group

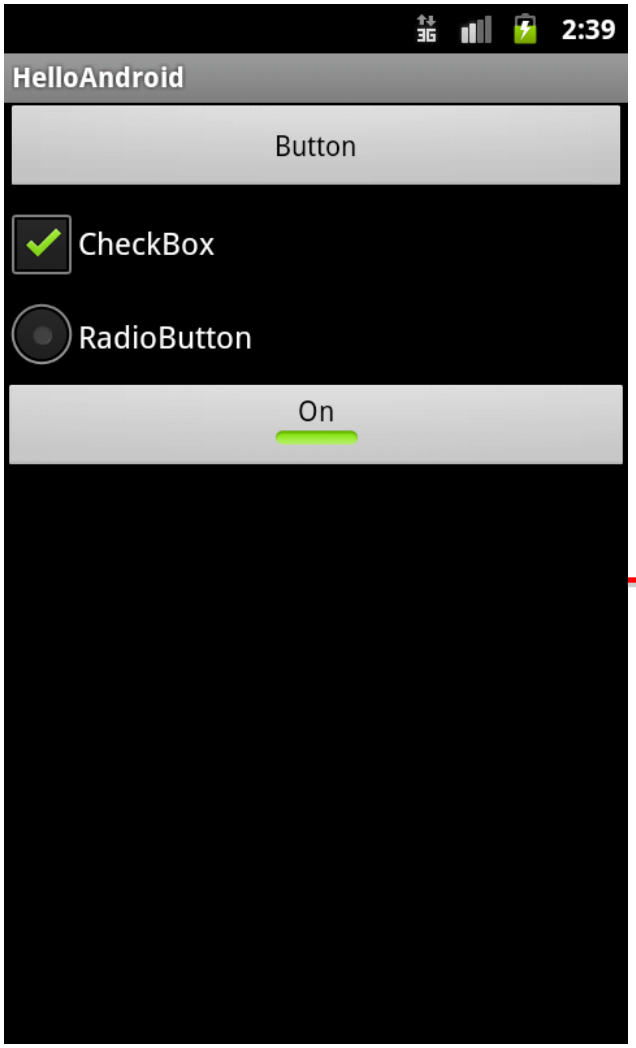


RadioGroup CompoundButton

- Define multiple (mutual-exclusive) options through a <**RadioGroup**> tag.
- Only one button can be checked within the same **RadioGroup**.

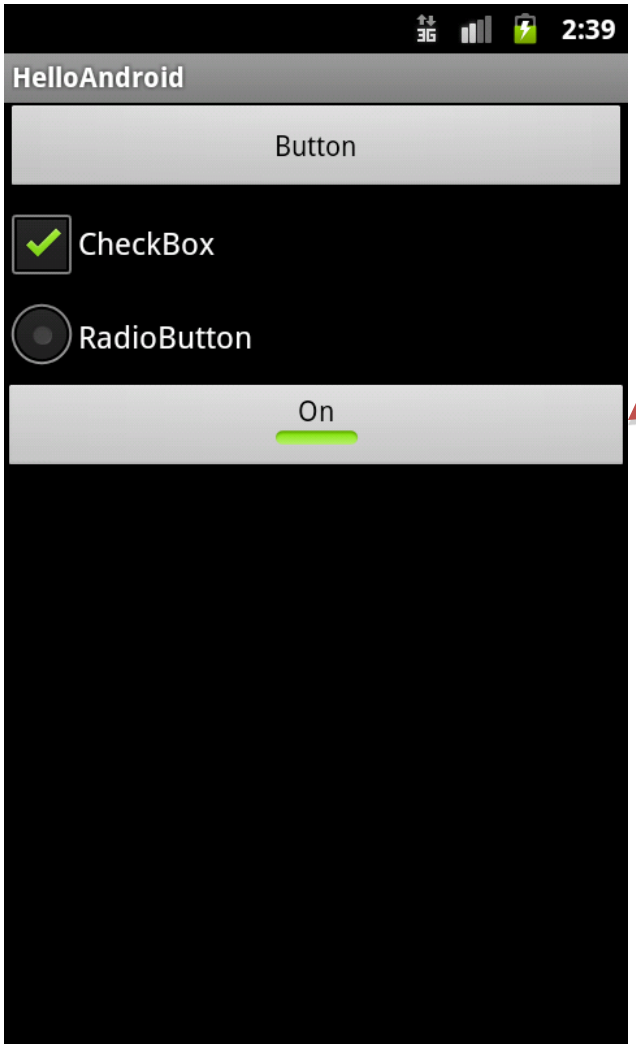
Listener:
OnCheckedChangeListener

Widgets: Widgets: Radio Group




```
<RadioGroup
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:orientation="vertical"
android:text= "Option type"
>
<RadioButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/buttonRadio1"
android:text="Option 1"
android:checked="true"/>
<RadioButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/buttonRadio2"
android:text="Option 2" />
</RadioGroup>
```

Widgets: Toggle Button



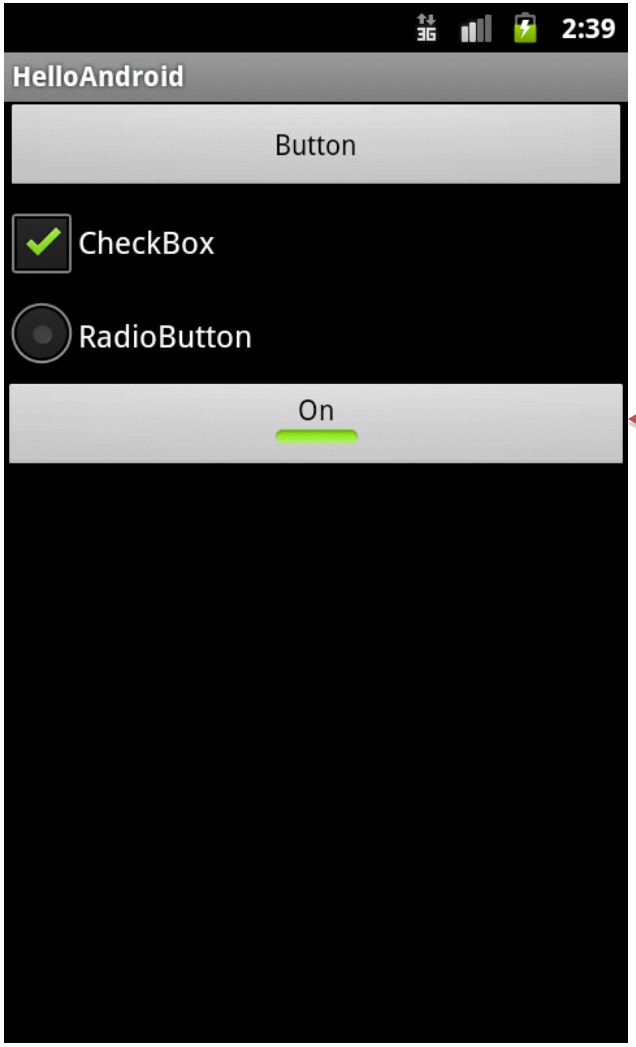
toggleButton CompoundButton

XML tags: **<ToggleButton>**

</ToggleButton>

```
<ToggleButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/toggleButtonId"
android:textOn="Button ON"
android:textOff="Button OFF"
android:checked="true"
/>
```

Widgets: Toggle Button…



toggleButton CompoundButton

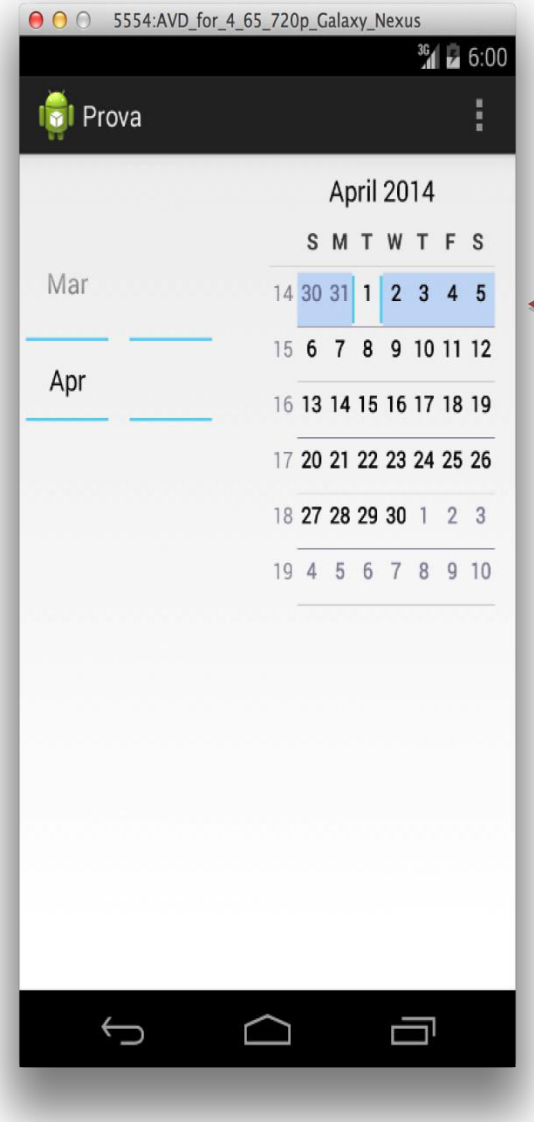
- It can assume only 2 states: *checked* / *unchecked*
- Different labels for the states with: **android:textOn** and **android:textOff** XML attributes.

Listener:
OnCheckedChangeListener

Analog and Digital Clocks

```
<AnalogClock
   android:id="@+id/analogClock1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="AnalogClock" />

<DigitalClock
   android:id="@+id/digitalClock1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="DigitalClock" />
```

Widgets: Date Picker



DatePicker component

XML tags: **<DatePicker>**

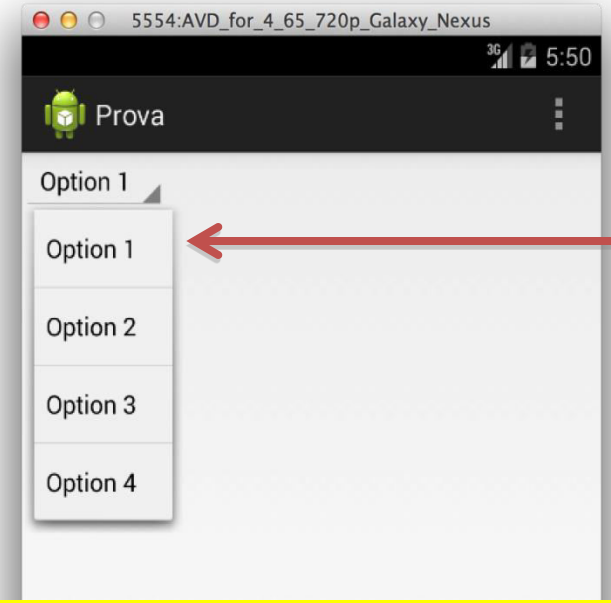
</DatePicker>

```
<DatePicker
android:layout_width="wrap_content"
android:layout_height="wrap_content”
android:id=“@+id/datePickerId”
/>
```

Time and Date Picker

```
<TimePicker
    android:id="@+id/time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />

<DatePicker
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/time"
        />
```

1.Widgets: Spinners - Using xml



Spinner component

XML tags: **<Spinner>**

</Spinner>

```
<Spinner
android:layout_width="wrap_c
ontent"
android:layout_height="wrap_
content"
android:id="@+id/spinnerId"
android:entries="@array/sexO
ptions">

</Spinner>
```

```
<resources>
  <string-array
name="sexOptions">
    <item>Option 1</item>
    <item>Option 2</item>
    <item>Option 3</item>
    <item>Option 4</item>
</string-array>
</resources>
```

res/values/string.xml

Widgets: Spinners

Spinner component

XML tags: **<Spinner>**

</Spinner>

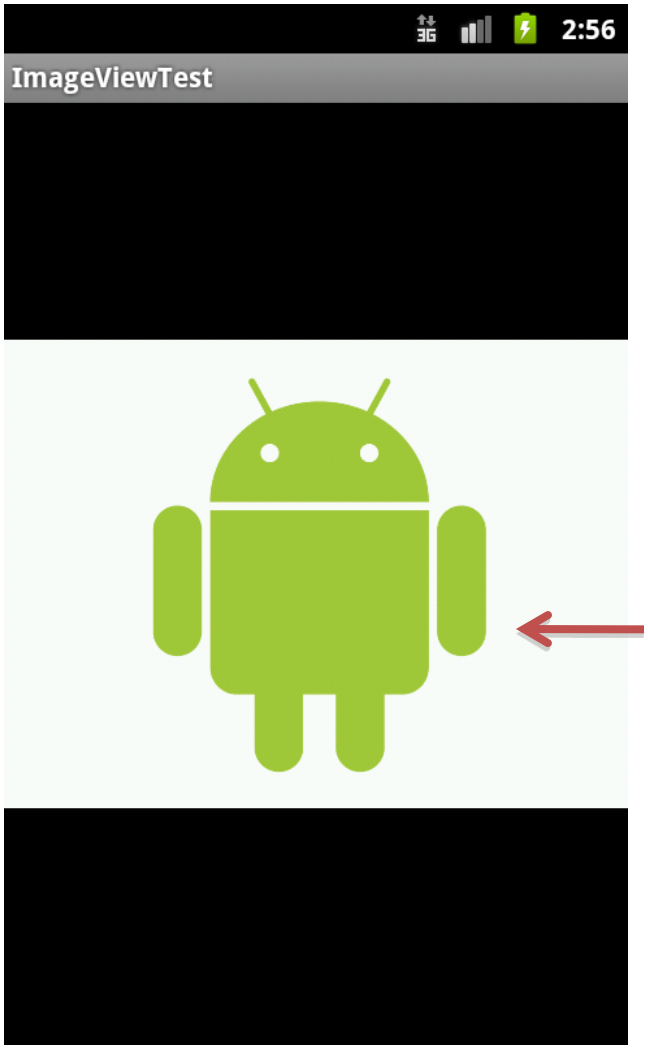
- Provides a quick way to select values from a specific set.
- The spinner value-set can be defined in XML (through the **entries** tag) or through the Array/*SpinnerAdapter* in Java

Listener:
OnItemSelectedListener

2.Spinners – Using ArrayAdapter

```
Spinner spsex= (Spinner) findViewById(R.id.spinnerId);
ArrayList<String> sex = new ArrayList<String>();//array
        sex.add("Male");
        sex.add("Female");
        sex.add("-- Gender --");
        String[] arra = sex.toArray(new String[0]);//converting to sting
        ArrayAdapter aAsex= new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, sex);// array adapter

asex.setDropDownViewResource(android.R.layout.simple_spi
nner_dropdown_item);// setting format of array adapter
        spsex.setAdapter(aAsex);//Setting AA to the spinner
        spsex.setSelection(2);//selection default "-- Gender --“
```

Widgets: ImageView



ImageView component

XML tags: **<ImageView>**

</ImageView>

```
<ImageView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/imageId"
android:src="@drawable/mar">
```

Source: mar.png in drawable/

Widgets: ImageView

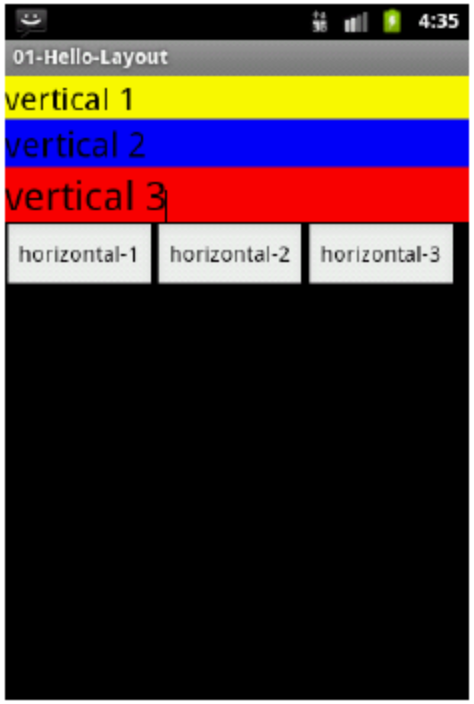
- **ImageView**: subclass of View object.
- Some methods to manipulate an image:
  - void **setScaleType**(enum scaleType)
  - void **setAlpha**(double alpha)
  - void **setColorFilter**(ColorFilter color)

CENTER, CENTER_CROP, CENTER_INSIDE, FIT_CENTER, FIT_END, FIT_START, FIT_XY, MATRIX

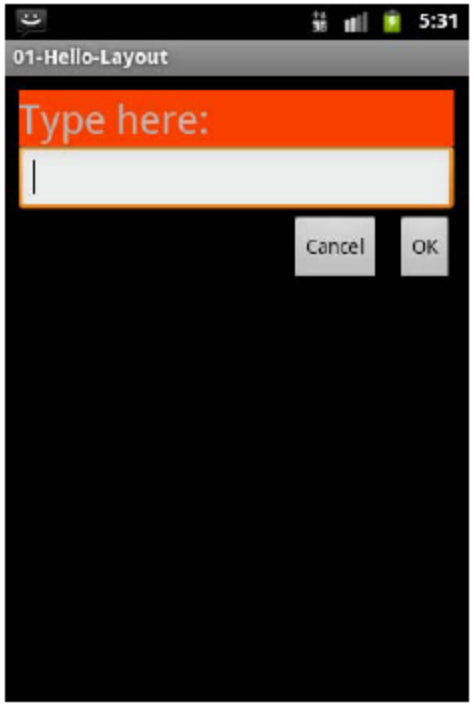
Layouts

- Some layouts are pre-defined by Android
- Some of these are
  - LinearLayout
  - RelativeLayout
  - TableLayout
  - FrameLayout
  - AbsoluteLayout
- A layout could be declared inside another layout

Layouts



Linear Layout
A LinearLayout places its inner views either in horizontal or vertical disposition.



Relative Layout
A RelativeLayout is a ViewGroup that allows you to position elements relative to each other.

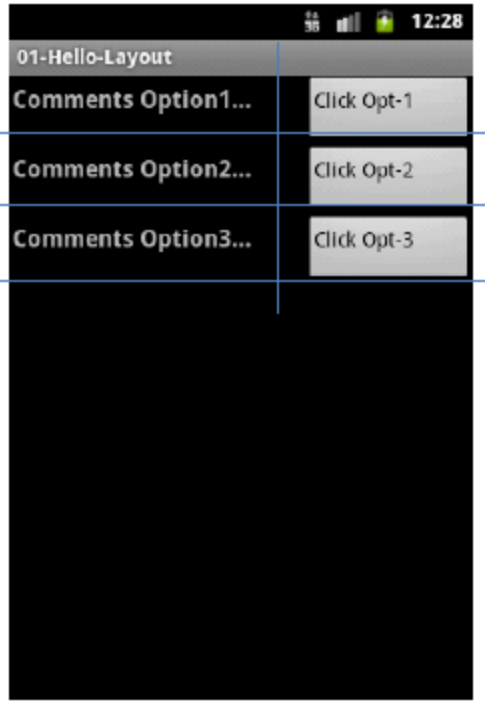


Table Layout
A TableLayout is a ViewGroup that places elements using a row & column disposition.

LinearLayout

- Arrange views on a single row or column, depending on android:layout_orientation
- orientation is one of HORIZONTAL or VERTICAL
- Has two other attributes:
  - gravity
  - weight

LinearLayout

Configuring a **LinearLayout** usually requires you to set the following attributes:

- **orientation** *(vertical, horizontal)*
- **fill model** *(match_parent, wrap_contents)*
- **weight** *(0, 1, 2, ...n)*
- **gravity** *(top, bottom, center,...)*
- **padding** *(dp – dev. independent pixels)*
- **margin** *(dp – dev. independent pixels)*

LinearLayout: Fill Model

- All widgets inside a LinearLayout must include 'width' and 'height' attributes.

```
android:layout_width
android:layout_height
```

- Values used in defining height and width can be:
  - A specific dimension such as 125dp (device independent pixels dip)
  - wrap_content indicates the widget should just fill up its natural space (Auto size).
  - match_parent (previously called 'fill_parent') indicates the widget wants to be as big as the enclosing parent.

LinearLayout : Fill Model

Medium resolution is: 320 x 480 dpi.
Shown on a Gingerbread device

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLinearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff0033cc"
    android:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/labelUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="#ffff0066"
        android:text="User Name"
        android:textColor="#ff000000"
        android:textSize="16sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/ediName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnGo"
        android:layout_width="125dp"
        android:layout_height="wrap_content"
        android:text="Go"
        android:textStyle="bold" />
</LinearLayout>
```

Row-wise

Use all the row

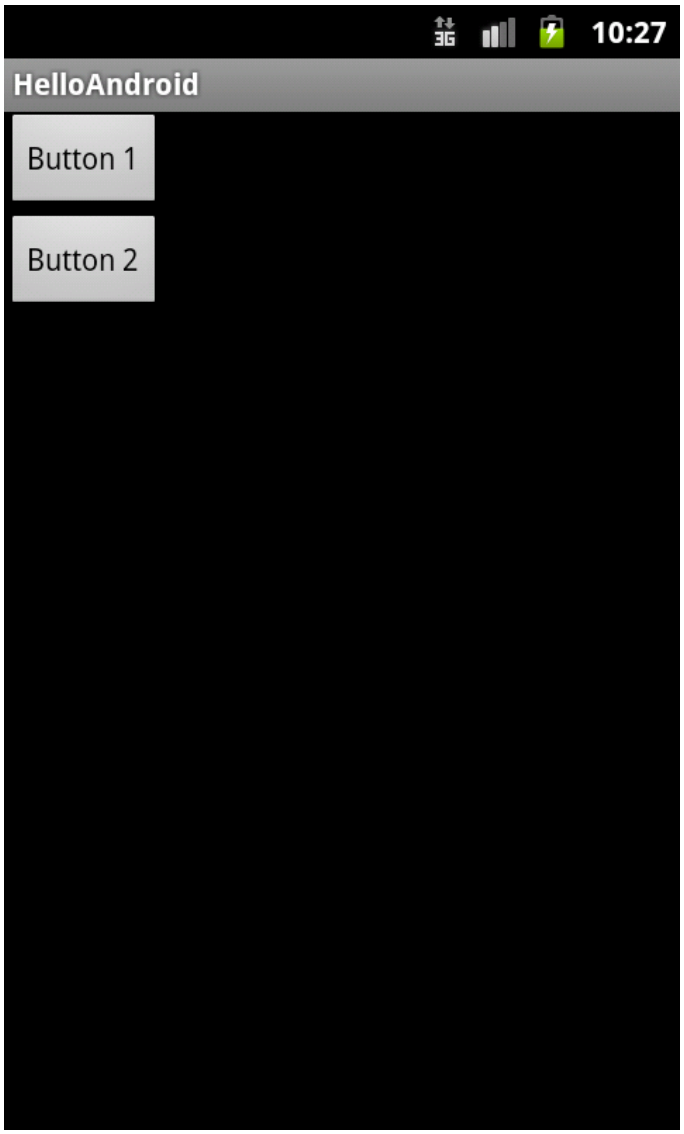
Specific size: 125dp

LinearLayout

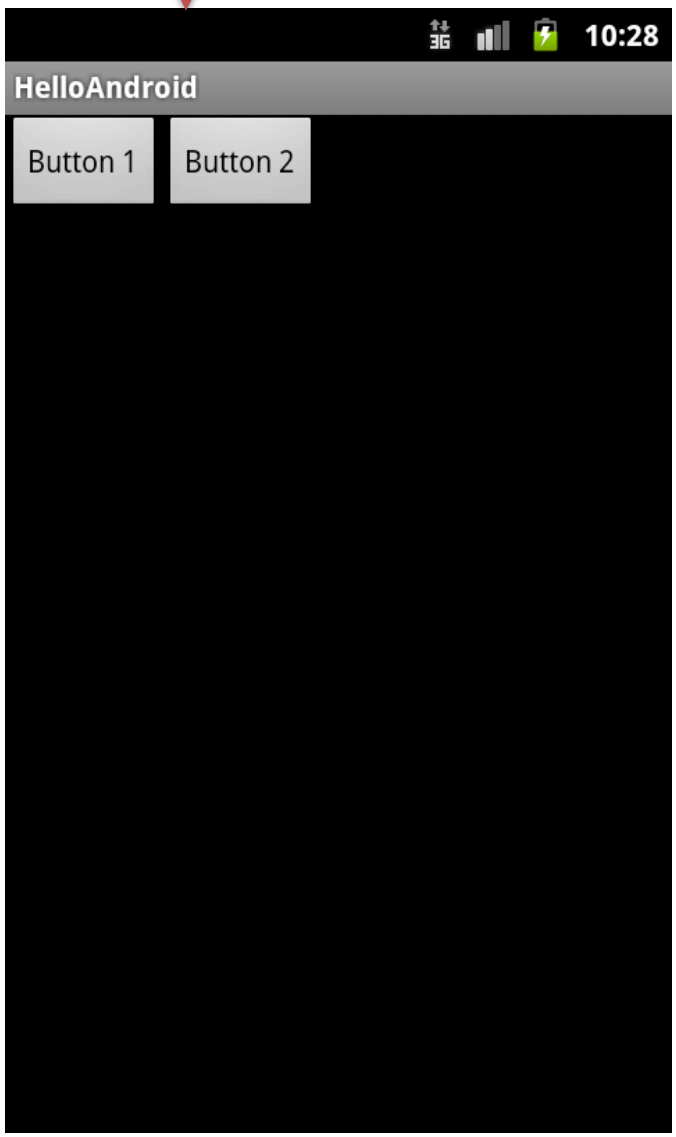
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" > <!-- Also horizontal -->

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="button1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="button2" />
</LinearLayout>
```

LinearLayout

Vertical for row



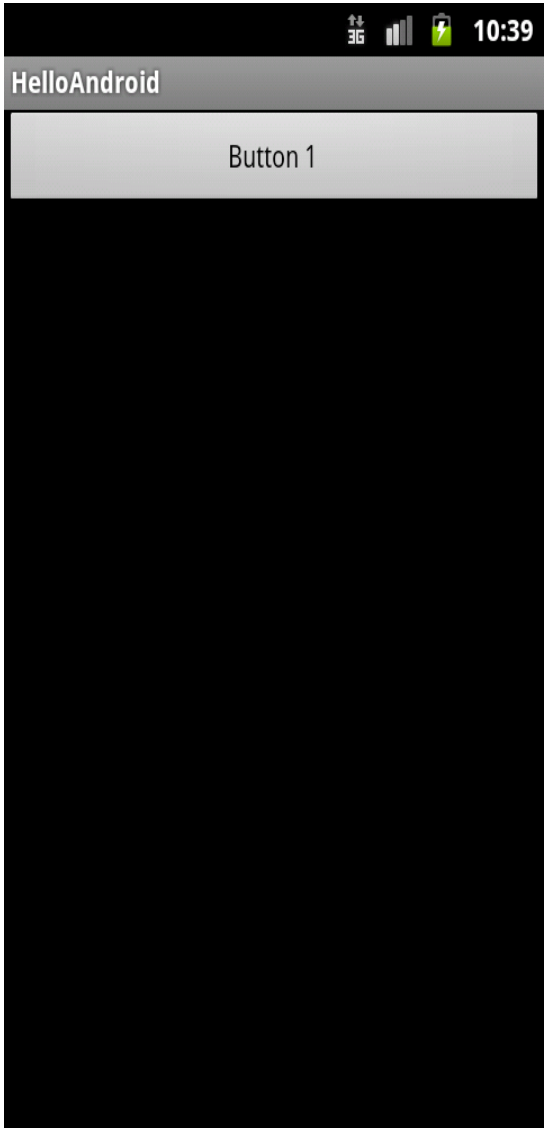
Horizontal for column

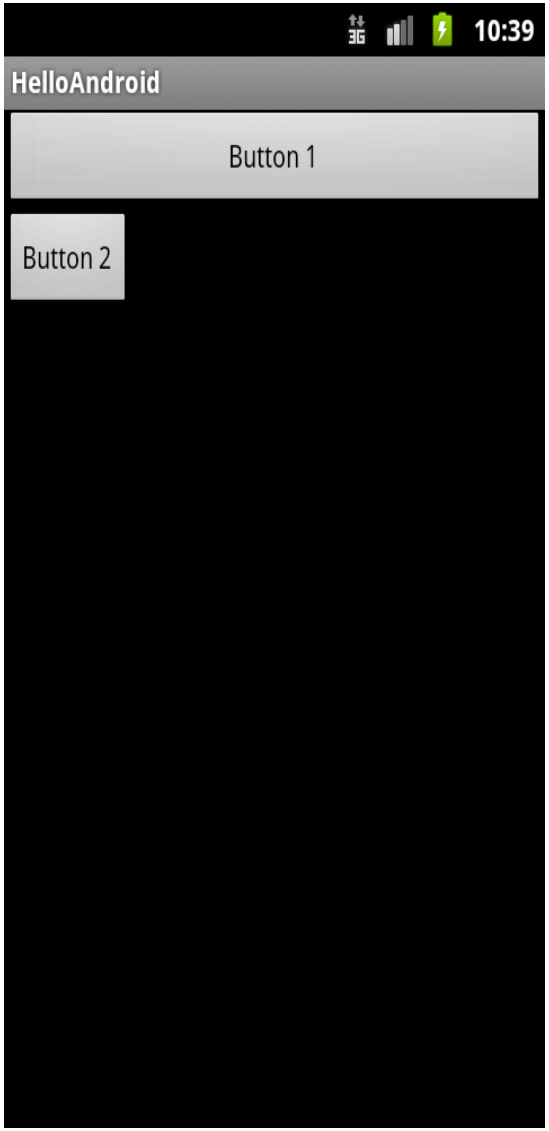


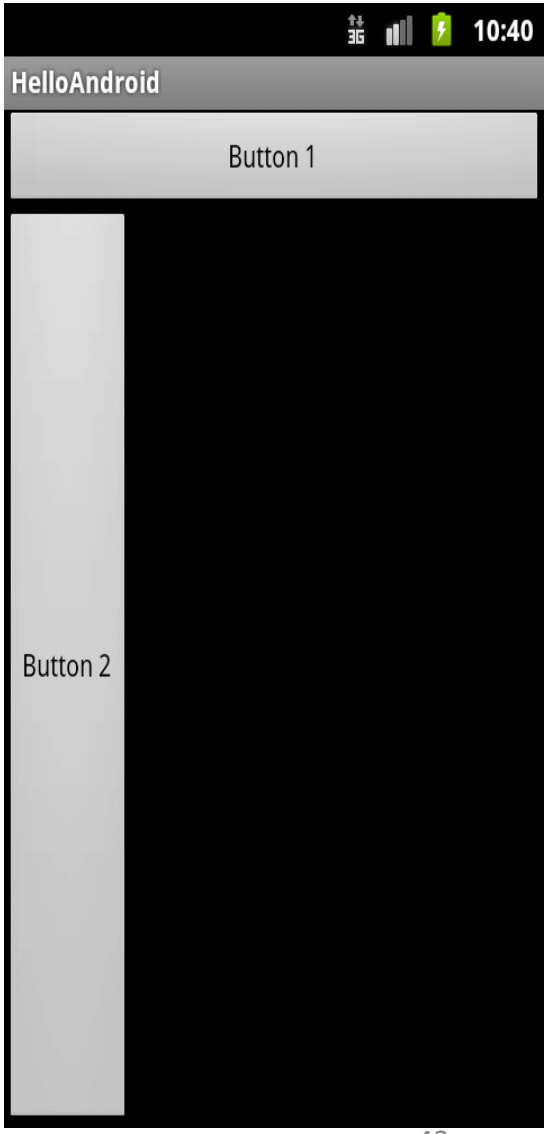
LinearLayout

```
<?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" >

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="button1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:text="button2" />
</LinearLayout>
```

LinearLayout







LinearLayout weight

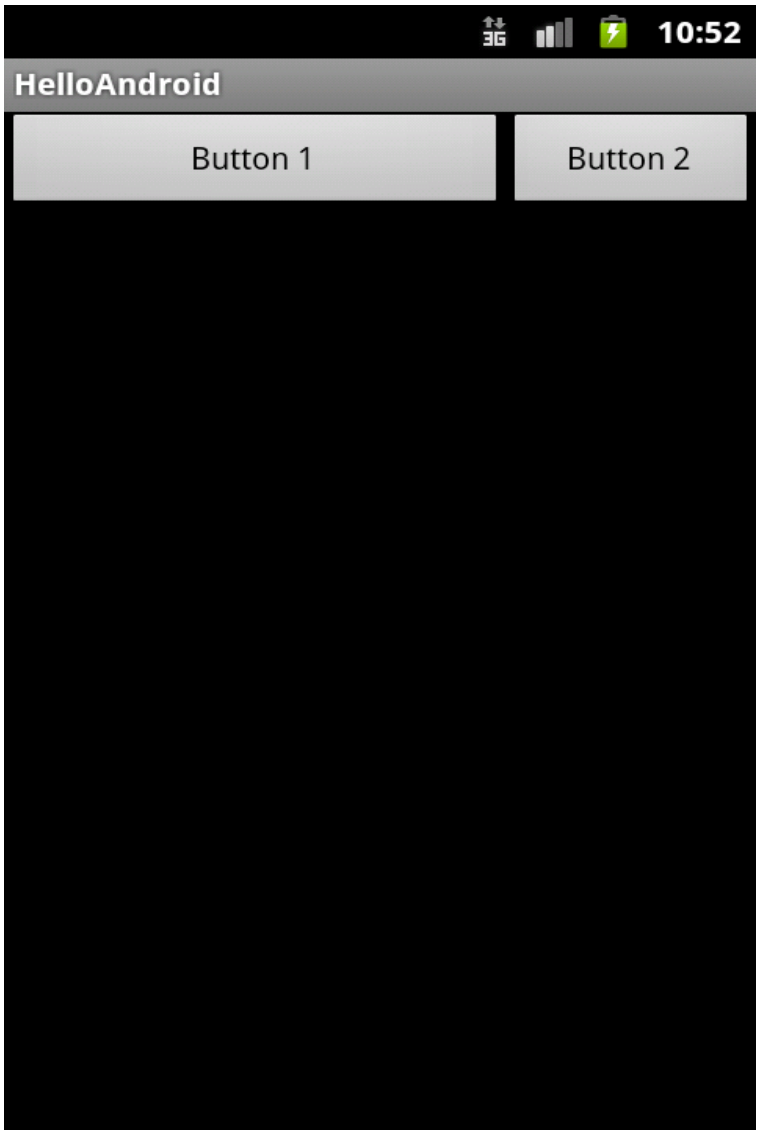
- The extra space left unclaimed in a layout could be assigned to any of its inner components by setting its Weight attribute.
- Use **0** if the view should not be stretched.
- The bigger the weight the larger the extra space given to that widget.

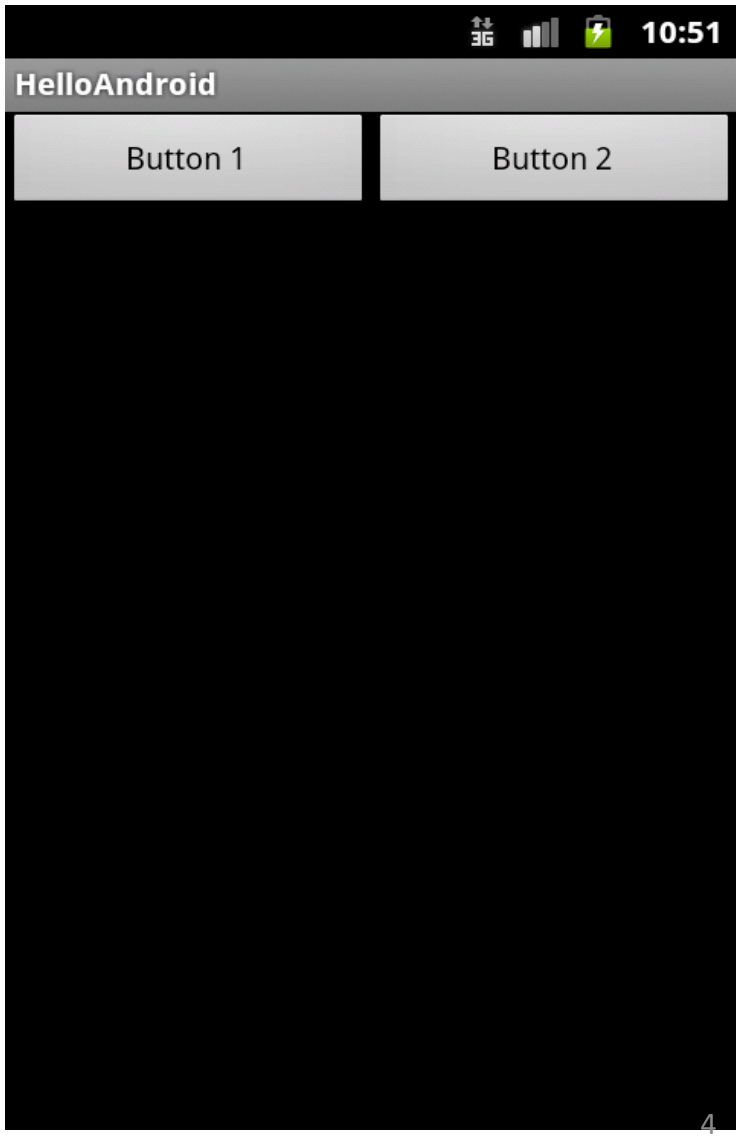
```
android:layout_weight="1"
```

LinearLayout weight

```
<?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="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="button1"
        android:layout_weight="1" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="button2"
        android:layout_weight="2"         />
</LinearLayout>
```

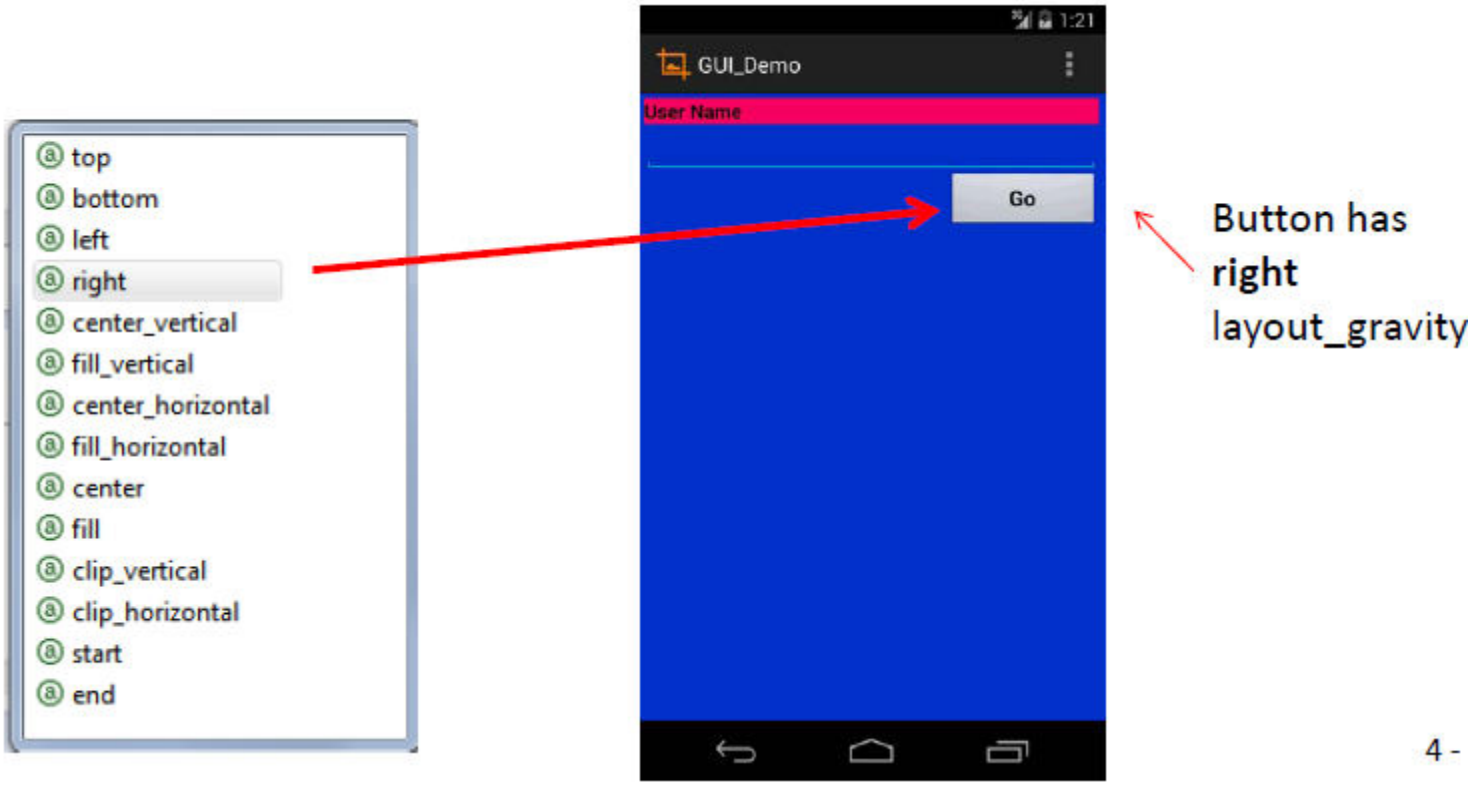
LinearLayout weight





LinearLayout gravity

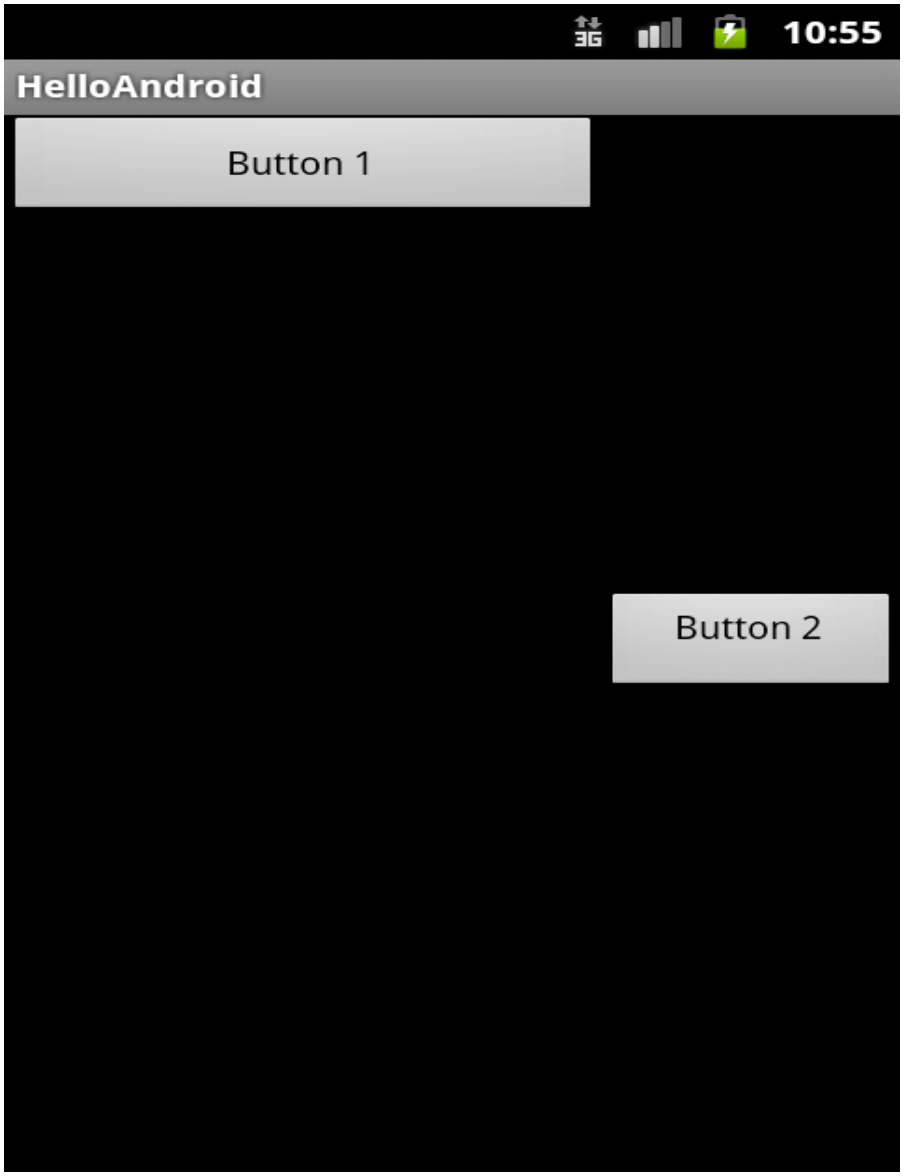
- **Gravity** is used to indicate how a control will align on the screen.
- By default, widgets are *left- and top-aligned.*
- You may use the XML property **`android:layout_gravity="…"`**
- *to set other possible arrangements: left, center, right, top, bottom, etc*



LinearLayout gravity

```
<?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="horizontal" >

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="button1"
        android:layout_weight="1" />
    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="button2"
        android:layout_weight="2"
        android:layout_gravity="center_vertical"
         </LinearLayout>
```

LinearLayout gravity



RelativeLayout

- Disposes views according to the container or according to other views
- Useful to align views

android:layout_above	Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form

android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource,
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form

RelativeLayout

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

<TextView
        android:id="@+id/usernameLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/username"
        android:text="Username" />

<EditText
        android:id="@+id/username"
        android:text="username"
        android:inputType="text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@+id/usernameLabel" >
    </EditText>
```

RelativeLayout

```
<TextView
        android:id="@+id/passwordLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBaseline="@+id/password"
        android:text="Password" />

 <EditText
        android:id="@+id/password"
        android:text="password"
        android:inputType="textPassword"
        android:layout_below="@+id/username"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/username"
        android:layout_alignParentRight="true"
        android:layout_toRightOf="@+id/passwordLabel" >
    </EditText>

</RelativeLayout>
```

RelativeLayout

TableLayout

- As the name say, similar to a Table
- Has some attributes to customize the layout:
  - android:layout_column
  - android:layout_span
  - android:stretchColumns
  - android:shrinkColumns
  - android:collapseColumns
  - android:layout_weight="1"
- Each row is inside a <TableRow> element

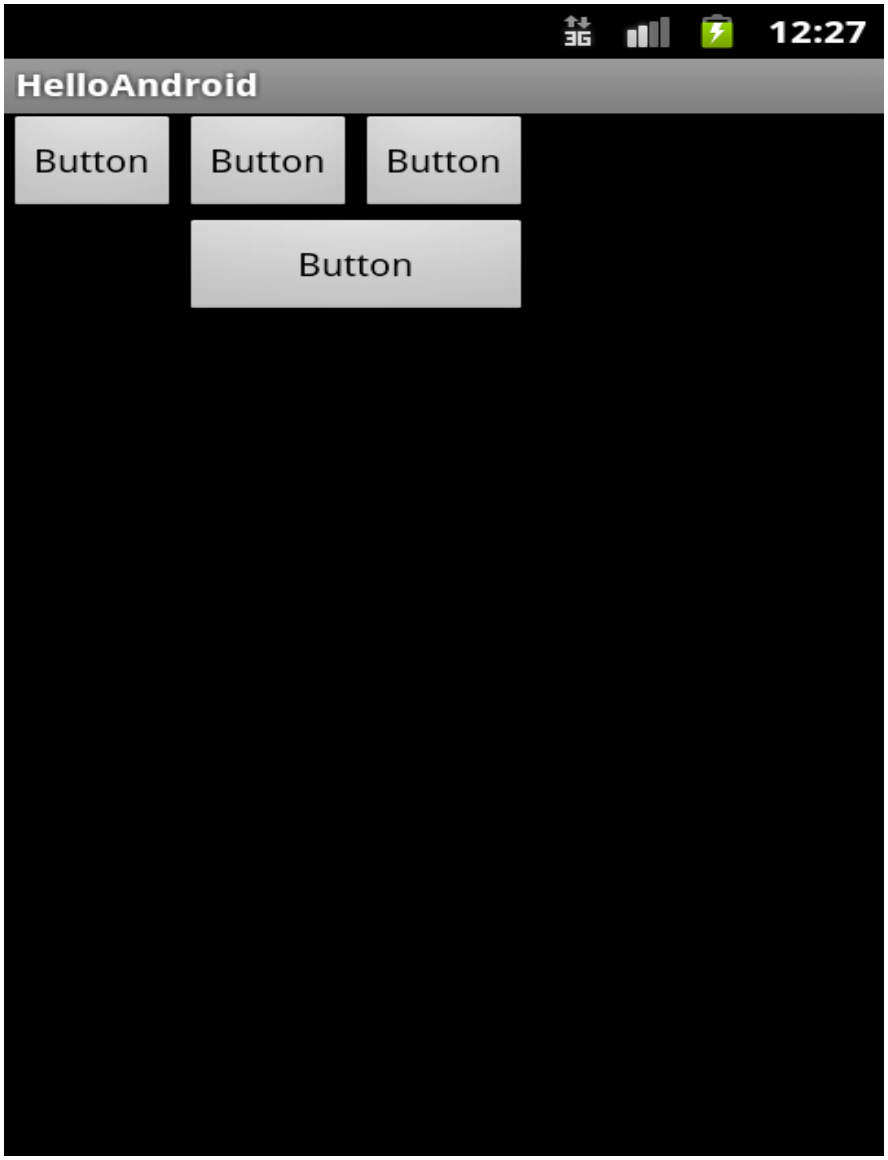
TableLayout

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

  <TableRow
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/firstRow">
       <Button        android:id="@+id/button1"
                      android:layout_width="wrap_content"
                      android:layout_height="wrap_content"
                      android:text="Button" />
       <Button
                      android:id="@+id/button2"
                      android:layout_width="match_parent"
                      android:layout_height="match_parent"
                      android:text="Button" />
       <Button        android:id="@+id/button3"
                      android:layout_width="match_parent"
                      android:layout_height="match_parent"
                      android:text="Button" />
    </TableRow>
```

TableLayout

```
 <TableRow
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:id="@+id/secondRow">

    <Button          android:layout_column="1"
                    android:layout_span="2"
                    android:id="@+id/button4"

                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="Button">
    </Button>
</TableRow>

</TableLayout>
```

TableLayout



59

FrameLayout and AbsoluteLayout

- FrameLayout
  - Adds an attribute, android:visibility
  - Makes the user able to define layouts managing the visibility of views
- AbsoluteLayout
  - Deprecated
  - Specify position with x and y
  - Pay attention to different resolutions

60

Layout Lab 2

Exercise 2

Step #1: Modify the Layout

- Change that layout to look like this:

```
<?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"
>
<LinearLayout
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Name:"
/>
<EditText android:id="@+id/name"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
</LinearLayout>
```

Exercise 2

Step #1: Modify the Layout…

```
<LinearLayout
android:orientation="horizontal"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
>
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Address:"
/>
<EditText android:id="@+id/addr"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
</LinearLayout>
<Button android:id="@+id/save"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Save"
/>
</LinearLayout>
```

Step #2: Support All Screen Sizes

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/and
roid"
package="apt.tutorial"
android:versionCode="1"
android:versionName="1.0">
<supports-screens
android:xlargeScreens="true"
android:largeScreens="true"
android:normalScreens="true"
android:smallScreens="false "/>
<application android:label="@string/app_name">
<activity android:name=".LunchList"
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>
```

Exercise 1

Step #4: Switch to a TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:stretchColumns="1"
>
<TableRow>
<TextView android:text="Name:" />
<EditText android:id="@+id/name" />
</TableRow>
<TableRow>
<TextView android:text="Address:" />
<EditText android:id="@+id/addr" />
</TableRow>
<Button
android:id="@+id/save"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="Save"
/>
</TableLayout>
```

Step #4: Switch to a TableLayout…

- Notice that we replaced the three LinearLayout containers with a
- TableLayout and two TableRow containers. We also set up the EditText column to be stretchable.
- Recompile and reinstall the application, then run it in the emulator.
- You should see something like this:

Step #5: Add a RadioGroup…

```
<TableRow>
<TextView android:text="Type:" />
<RadioGroup android:id="@+id/types">
<RadioButton android:id="@+id/take_out"
             android:text="Take-Out"
/>
<RadioButton android:id="@+id/sit_down"
             android:text="Sit-Down"
/>
<RadioButton android:id="@+id/delivery"
             android:text="Delivery"
/>
</RadioGroup>
</TableRow>
```

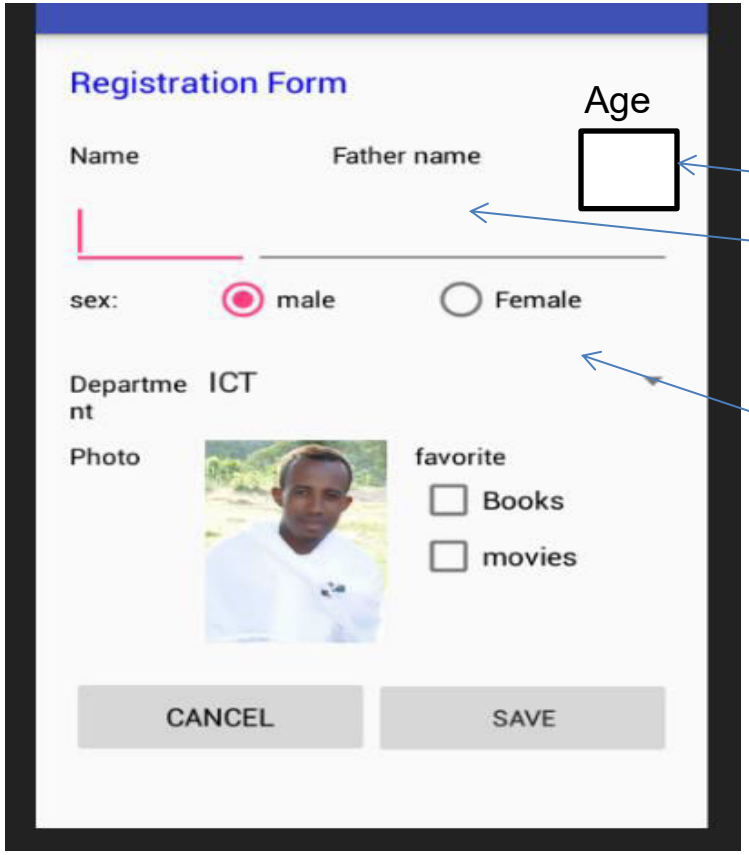
Step #5: Add a RadioGroup…

- Our RadioGroup and RadioButton widgets go inside the TableLayout, so they will line up with the rest of table.

- you can see this once you recompile, reinstall, and run the application:

Assignment II (5%)

Using android studio create the following Views as shown below and display it on emulator and your phone.



EditView

Spinner with options ICT, Manufacturing, Automotive ….



Chapter3-Application development for mobile devices (using Android)

2.4 . Event Handling

Event Handling

- Events are a useful way to collect data about a user's interaction with interactive components of your apps like,
  - button presses or
  - screen touch etc.
- The Android framework maintains an event queue into which events are placed as they occur,
  - and then each event is removed from the queue on a first-in, first-out (FIFO) basis.
- You can **capture** these events in your program and take appropriate action as per requirements.

Event Handling…

- **There are three concepts related to Android Event Management:**
- Event Listeners: The Event Listener is the object that receives notification when an event happens.

  e.g. OnClickListener(), OnFocusChangeListener() , OnItemSelectedListener() //for spinner

- Event Listeners Registration: Event Registration is the **process** by which an Event Handler gets registered with an Event Listener
  - so that the handler is called when the Event Listener fires the even.
- Event Handlers: is **the method that actually handles** the event which is registered by the event listener.

  e.g. onClick() , onTouch() , onItemSelected

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.

Event Listeners & Event Handlers…

Event Handler	Event Listener & Description
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.

Event Listeners Registration

- **Event Registration** is the process by which an Event Handler gets registered with an Event Listener,
  - so that the handler is called when the Event Listener fires the event.
- **Ways to register your event listener for any event:**
  - Using Layout file activity_main.xml to specify event handler directly.
  - Activity class implements the Listener interface.
  - Using an Anonymous Inner Class

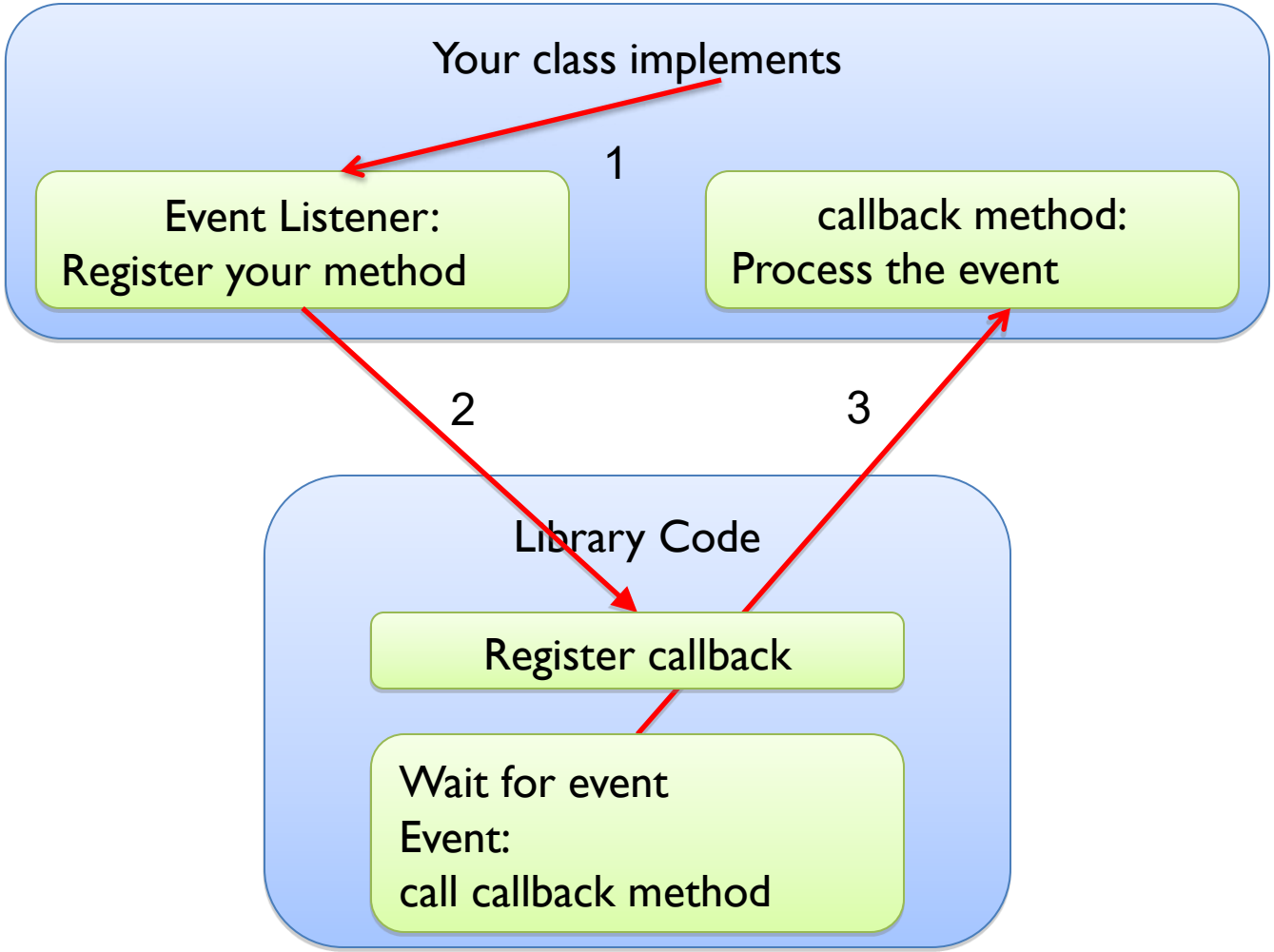
Interfaces

- An interface in Java is special type
- A class with only method signatures
  - Methods have no body
  - Can never create an instance of an interface
- Classes can *implement* the interface
  - the class will implement all the methods of an interface definition
- All methods of an interface are `public`
- An interface type can be used just like any other type
  - return type of method
  - argument type of method

Callbacks

- Executable code passed as argument to another class
- The class calls the code when an event happens
- Examples:
  - Call a method when a user presses a button on a phone (J2ME, Android)
  - Call a method when an SMS message is received

Callbacks


Your class implements
1
Event Listener:
Register your method
callback method:
Process the event
2
3
Library Code
Register callback
Wait for event
Event:
call callback method

Interfaces and Callback

- Interface defines the callback method that will be called when the event happens
  - Defines the arguments you are passed
- You create a class that implements the interface (listeners), which will call callback method
  - The code you want to execute when the event happens
- Must register the callback first

Views and Events

Views/Widgets are interactive components …

- … Upon certain action, an appropriate **event** will be fired
- Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc

PROBLEM: How to **handle** these events?

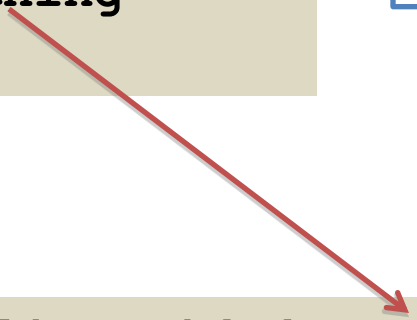
1. Directly from **XML**
2. Activity class implements the **Listener interface**
3. Using an **Anonymous Inner Class**

Views and Events

- For a limited set of components, it is possible to manage the events through **callbacks**, directly indicated in the XML.

```
<Button
android:text="Button"
android:id="@+id/idButton"
android:onClick= "doSomething"
/>
```

XML Layout File



Java class

```
public void doSomething(View w) {
// Code to manage the click event
}
```

Views and Events

Views/Widgets are interactive components …

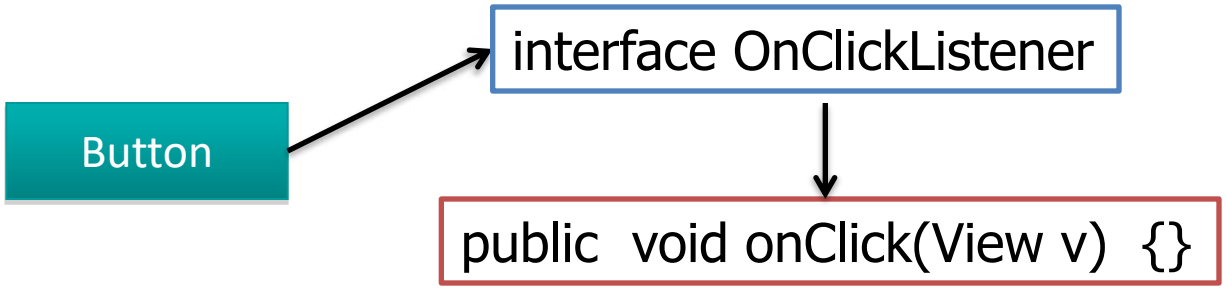
- … Upon certain action, an appropriate **event** will be fired
- Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc

PROBLEM: How to **handle** these events?

1. Directly from **XML**
2. Activity class implements the **Listener interface**
3. Using an **Anonymous Inner Class**

Views and Events

- Each View contains a collection of **interfaces** (**listeners**).
  - Each listener handles a single **type of events.**
  - Each listener contains a single **callback** method.
  - The callback is invoked in occurrence of the event.



Views and Events

To handle OnClick events through Listener Interface:

1. Implement the **interface** in the current Activity

2. Implement the **callback** method (onClick)

3.Register the Listener to the Button through the View.**setOnClickListener**() method

```
public class ExampleActivity extends Activity implements
OnClickListener {
 …
   Button btn=(Button)findViewById(R.id.buttonNext);
   btn.setOnClickListener(this);
   …
  public void onClick(View v) { } }
```

Views and Events

- Views/Widgets are interactive components …
  - ✧ … Upon certain action, an appropriate **event** will be fired
  - ✧ Events generated by the user's interaction: click, long click, focus, items selected, items checked,drag, etc
- **PROBLEM**: How to **handle** these events?

1. Directly from **XML**
2. Activity class implements the **Listener interface**
3. Using an **Anonymous Inner Class**

Views and Events

To handle OnClick events through anonymous inner class:

1. Create an **anonymous** OnClickListener object
2. Implement the **callback** method (onClick) for the anonymous object
3. Register the Listener to the Button through the View.**setOnClickListener**() method

```
Button btn = (Button)findViewById(R.id.btn);
btn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
                // Event management
        }
});
```

Views and Events
(Summary)

Some ActionListeners:

- **interface OnClickListener**
  abstract method: *onClick()*
- **interface OnLongClickListener**
  abstract method: *onLongClick()*
- **interface OnFocusChangeListener**
  abstract method: *onFocusChange()*
- **interface OnKeyListener**
  abstract method: *onKey()*

Views and Events
(Summary)

Some ActionListeners:

- **interface OnCheckedChangeListener**
  abstract method: *onCheckedChanged()*
- **interface OnItemSelectedListener**
  abstract method: *onItemSelected()*
- **interface OnTouchListener**
  abstract method: *onTouch()*
- **interface OnCreateContextMenuListener**
  abstract method: *onCreateContextMenu()*

Event listeners registration using an anonymous inner class

- Following are the simple steps to show how we will make use of separate Listener class to register and capture click event.

- Similar way you can implement your listener for any other required event type.

Event listeners registration using an anonymous inner class

- General steps

1. You will use Eclipse IDE to create an Android application and name it as *EventDemo* under a package *com.example.eventdemo .*
2. Define required constants in *res/values/strings.xml* file.
3. Modify the detault content of *res/layout/activity_main.xml* file to include Android UI controls.
4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

2. Define required constants in *res/values/strings.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">EventDemo</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="button_small">Small Font</string>
<string name="button_large">Large Font</string>
</resources>
```

3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.

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

<Button
android:id="@+id/button_s"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_small"
/>
```

3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.

```
<Button
android:id="@+id/button_1"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_large"
/>

<TextView
android:id="@+id/text_id"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_world"
/>
</LinearLayout>
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends Activity {
@Override
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
//--- find both the buttons---
Button sButton = (Button)findViewById(R.id.button_s);
Button lButton = (Button)findViewById(R.id.button_l);
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
// -- register click event with first button ---
sButton.setOnClickListener(new View.OnClickListener(){
   public void onClick(View v) {
// --- find the text view -
TextView txtView = (TextView)findViewById(R.id.text_id);
// -- change text size -
txtView.setTextSize(14);
     }
   });
// -- register click event with second button ---
lButton.setOnClickListener(new View.OnClickListener() {
public void onClick(View v) {
// --- find the text view -
TextView txtView = (TextView) findViewById(R.id.text_id);
// -- change text size -
txtView.setTextSize(24);
       }
    });
}
```

Registration using the activity implements listener interface

- Here your Activity class implements the **Listener interface** and you put the handler method in the main Activity and then you call setOnClickListener(this).
- Following are the simple steps to show how we will implement Listener class to register and capture click event.
- Similar way you can implement your listener for any other required event type.

Registration using the activity implements listener interface

- **Steps**

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. We are not making any change in *res/values/strings.xml* file, it will also remain as shown above.
3. We are also not making any change in *res/layout/activity_main.xml*, it will remain as shown below.

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

2. Define required constants in *res/values/strings.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">EventDemo1</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="button_small">Small Font</string>
<string name="button_large">Large Font</string>
</resources>
```

3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.

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

<Button
android:id="@+id/button_s"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_small"
/>
```

3. Modify the default content of *res/layout/activity_main.xml* file to include Android UI controls.

```
<Button
android:id="@+id/button_1"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_large"
/>

<TextView
android:id="@+id/text_id"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_world"
/>
</LinearLayout>
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends Activity implements
OnClickListener {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
//--- find both the buttons---
Button sButton = (Button)findViewById(R.id.button_s);
Button lButton = (Button)findViewById(R.id.button_l);
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
// -- register click event with first button ---
sButton.setOnClickListener(this);
// -- register click event with second button ---
lButton.setOnClickListener(this);
    }
//Implement the OnClickListener callback onClick()
public void onClick(View v) {
  if(v.getId() == R.id.button_s) {

// --- find the text view --
TextView txtView = (TextView) findViewById(R.id.text_id);
// -- change text size --
txtView.setTextSize(14);
return;
      }
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
else if(v.getId() == R.id.button_1)
   {
// --- find the text view -
TextView txtView = (TextView)
findViewById(R.id.text_id);

// -- change text size -
txtView.setTextSize(24);
return;
    }
}
```

Registration using layout file activity_main.xml

- Here you put your event handlers in Activity class without implementing a Listener interface or call to any listener method.
- Rather you will use the layout file (activity_main.xml) to specify the handler method via the android:onClick attribute for click event.
- You can control click events differently for different control by passing different event handler methods.
- The event handler method must have a void return type and take a View as an argument.
- However, the method name is arbitrary, and the main class need not implement any particular interface.

REGISTRATION USING LAYOUT FILE ACTIVITY_MAIN.XML

1. We do not need to create this application from scratch, so let's make use of above created Android application *EventDemo*.
2. We are not making any change in *res/values/strings.xml* file, it will also remain as shown above.
3. Modify layout file *res/layout/activity_main.xml*, to specify event handlers for the two buttons.
4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.
5. Run the application to launch Android emulator and verify the result of the changes done in the application.

2. Define required constants in *res/values/strings.xml* file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">EventDemo2</string>
<string name="action_settings">Settings</string>
<string name="hello_world">Hello world!</string>
<string name="button_small">Small Font</string>
<string name="button_large">Large Font</string>
</resources>
```

3. Modify layout file *res/layout/activity_main.xml*, to specify event handlers for the two buttons.

- Here we have to add android:onClick="methodName" for both the buttons, which will register given method names as click event handlers.

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

<Button
android:id="@+id/button_s"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_small"
android:onClick="doSmall"
/>
```

3. Modify layout file *res/layout/activity_main.xml*, to specify event handlers for the two buttons.

```
<Button
android:id="@+id/button_l"
android:layout_height="wrap_content"
android:layout_width="match_parent"
android:text="@string/button_large"
android:onClick="doLarge"
/>

<TextView
android:id="@+id/text_id"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_world"
/>
</LinearLayout>
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends Activity{
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
```

4. Modify *src/MainActivity.java* file to add click event listeners and handlers for the two buttons defined.

```
//--- Implement the event handler for the first button.
public void doSmall(View v) {
// --- find the text view -
TextView txtView = (TextView)findViewById(R.id.text_id);
// -- change text size -
txtView.setTextSize(14);
return;
}
//--Implement the event handler for the second button.
public void doLarge(View v) {
// --- find the text view -
TextView txtView = (TextView)findViewById(R.id.text_id);
// -- change text size -
txtView.setTextSize(24);
return;
    }
```

Example/All in one

In this example all the event handling methods are included.

Xml fille

```
<?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout
    xmlns:android="http://schemas.android.com
    /apk/res/android"

    xmlns:tools="http://schemas.android.com/t
    ools"
```

```
android:id="@+id/activity_xml_event"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

  android:paddingBottom="@dimen/activity_v
  ertical_margin"

  android:paddingLeft="@dimen/activity_horiz
  ontal_margin"

  android:paddingRight="@dimen/activity_hor
  izontal_margin"
```

```
android:paddingTop="@dimen/activity_vertica
    l_margin"

    tools:context="com.example.tinsae.eventhan
    dling.xmlEvent">

        <Button
            android:text="Button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="42dp"
            android:id="@+id/button2"
```

```
android:layout_below="@+id/button" />

    <Button
        android:text="Button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button"
        android:onClick="handle"


  android:layout_alignStart="@+id/button2"
  />
```

```
<Button
        android:text="Button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="42dp"
        android:id="@+id/button3"
        android:layout_below="@+id/button2"

    android:layout_alignStart="@+id/button2"
    />
    </RelativeLayout>
```

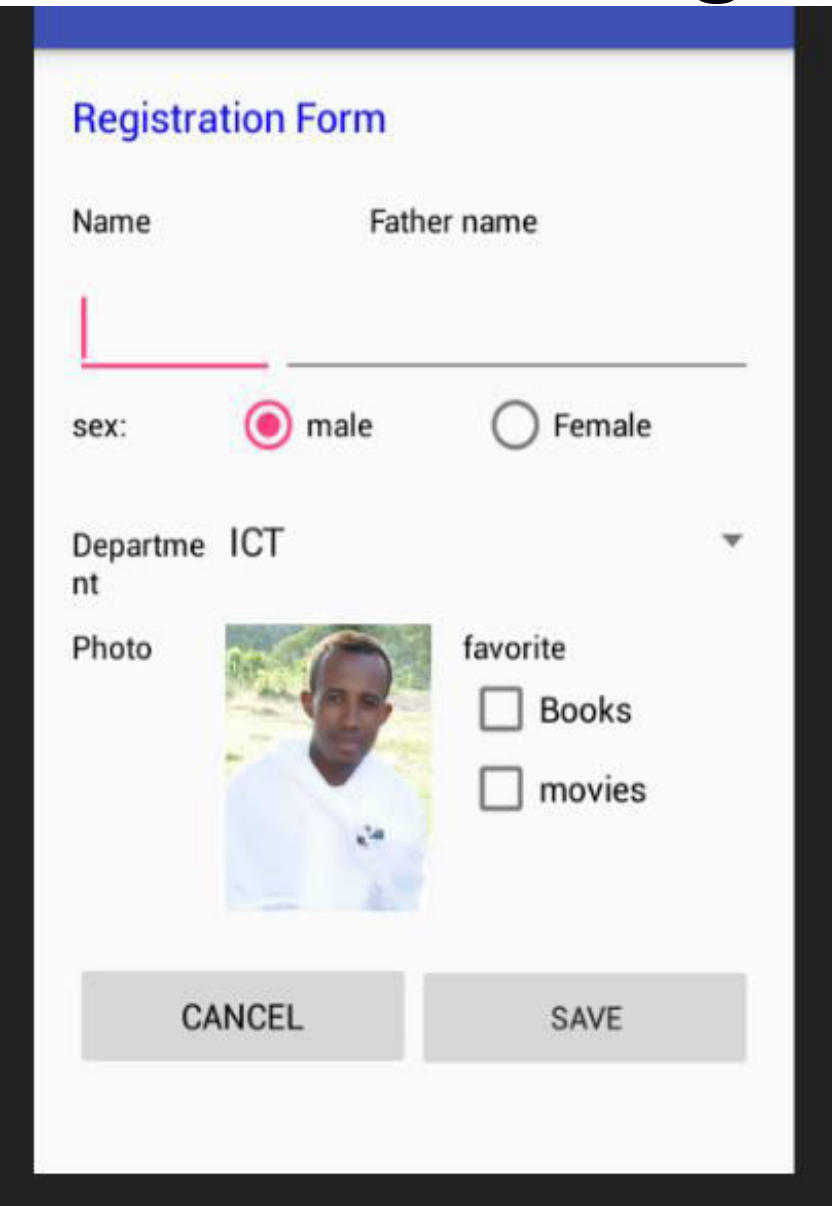
xmlEvent.Java

```
package com.example.tinsae.eventhandling;
    import android.support.v7.app.AppCompatActivity;
    import android.os.Bundle;
    import android.view.View;
    import android.view.View.OnClickListener;
    import android.widget.Button;
    import android.widget.EditText;
    import android.widget.Toast;
    public class xmlEvent extends AppCompatActivity
    implements OnClickListener {
    Button b2,b1,b3;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_xml_event);
            b2=(Button)findViewById(R.id.button2);
```

```
b1=(Button)findViewById(R.id.button);
        b3=(Button)findViewById(R.id.button3);

        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Toast t=Toast.makeText(xmlEvent.this,"button 2 is
    clicked using anonymous Inner class",Toast.LENGTH_LONG);
                t.show();
            }
        });

        b3.setOnClickListener(this);

    }
```

```
 // using xml
        public void handle(View v)
        {
      Toast t=Toast.makeText(this,"button 1 is clicked using
    directly from xml",Toast.LENGTH_LONG);
           t.show();
        }
// by implementing interface
        @Override
        public void onClick(View v) {
           if(v.getId() == R.id.button3) {
              Toast t=Toast.makeText(this,"button 3 is clicked by
    implementing Listner Interface",Toast.LENGTH_LONG);
              t.show();     }//end of if
        }end of onclick
    }//end of class
```

Assignment III (7%)



1. Create an event handler for the save button that you created in Assignment II, to toast all the user inputs when the button is clicked.

   e.g: "Your name is : Asefa Boka Sex: Male
   Department: ICT and Your favorite is: Books"

2. And create an event for the button Cancel, which handles it appropriately (i.e. clear all the inputs)



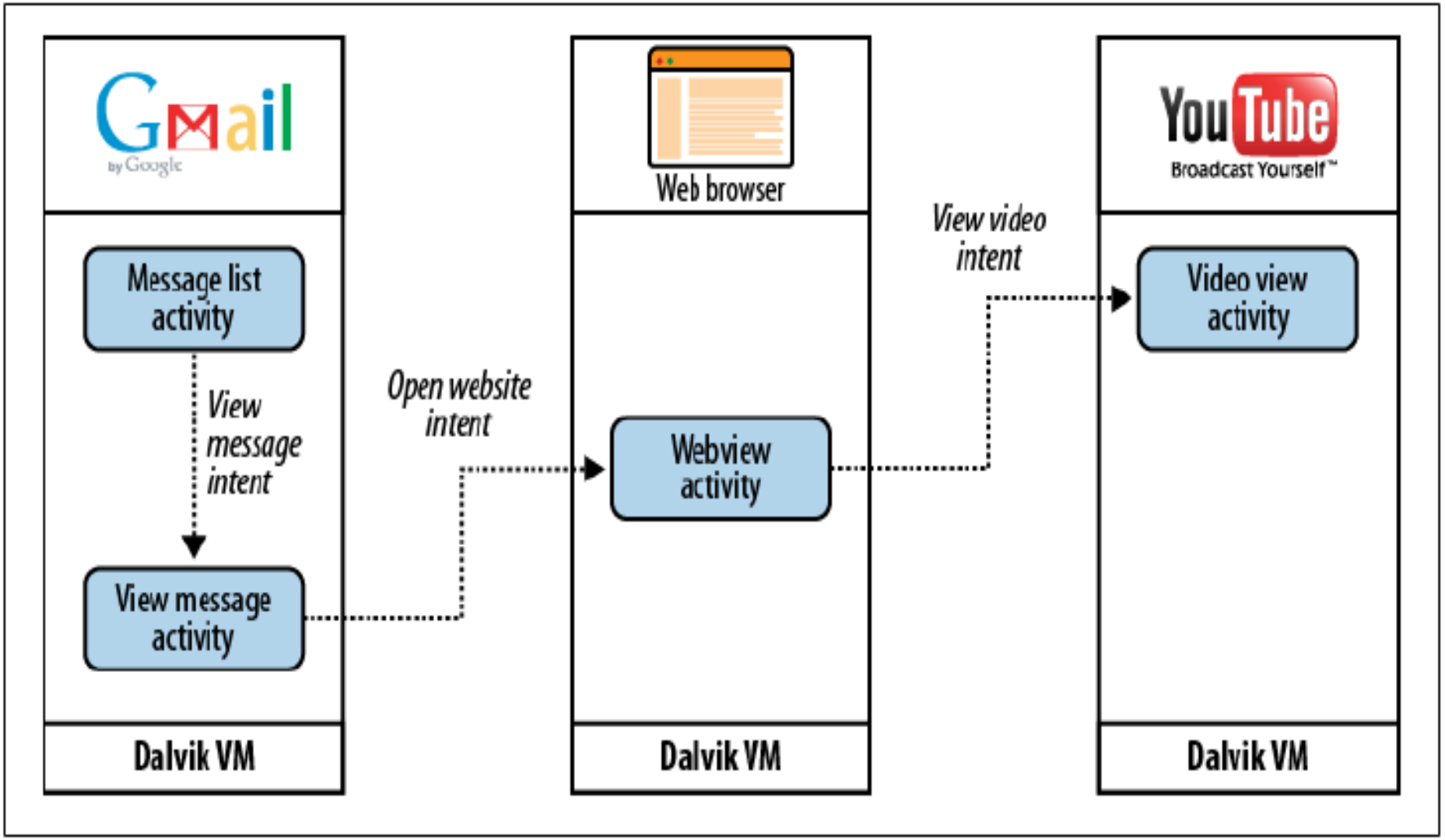
Chapter3-Application development for mobile devices (using Android)

2.5 - Android Intents

Android Intents

- An Intent is Android's method for relaying certain information from one Activity to another.
- One of the key aspects of Android programming is using the intent to call activities from other applications.
- The intents can communicate messages among any of the three core components of an application - *activities, services,* and *broadcast receivers.*
- For example:
  - your activity could send an intent saying it simply wants someone to open up a web page.
  - if your application needs to enable a user to call a particular person saved in the Contacts application, you can use an Intent object to bring up the Contacts application, from which the user can select the person to call.

Android Intents…



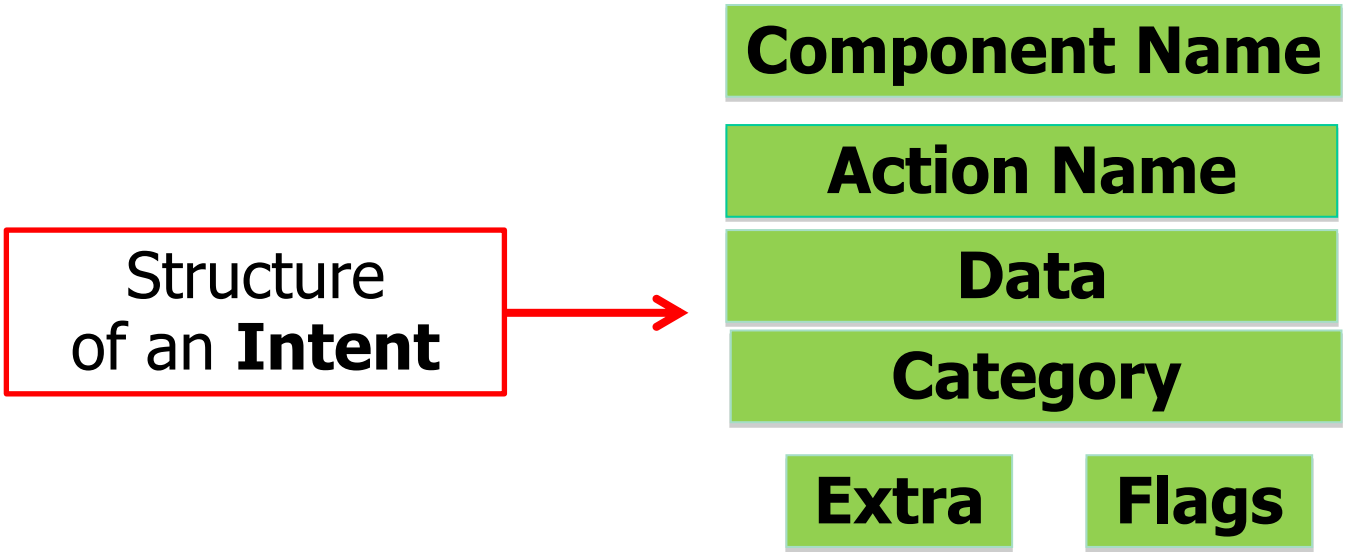
Intent Definition

Intent: facility for late run-time binding between components in the same or different applications.

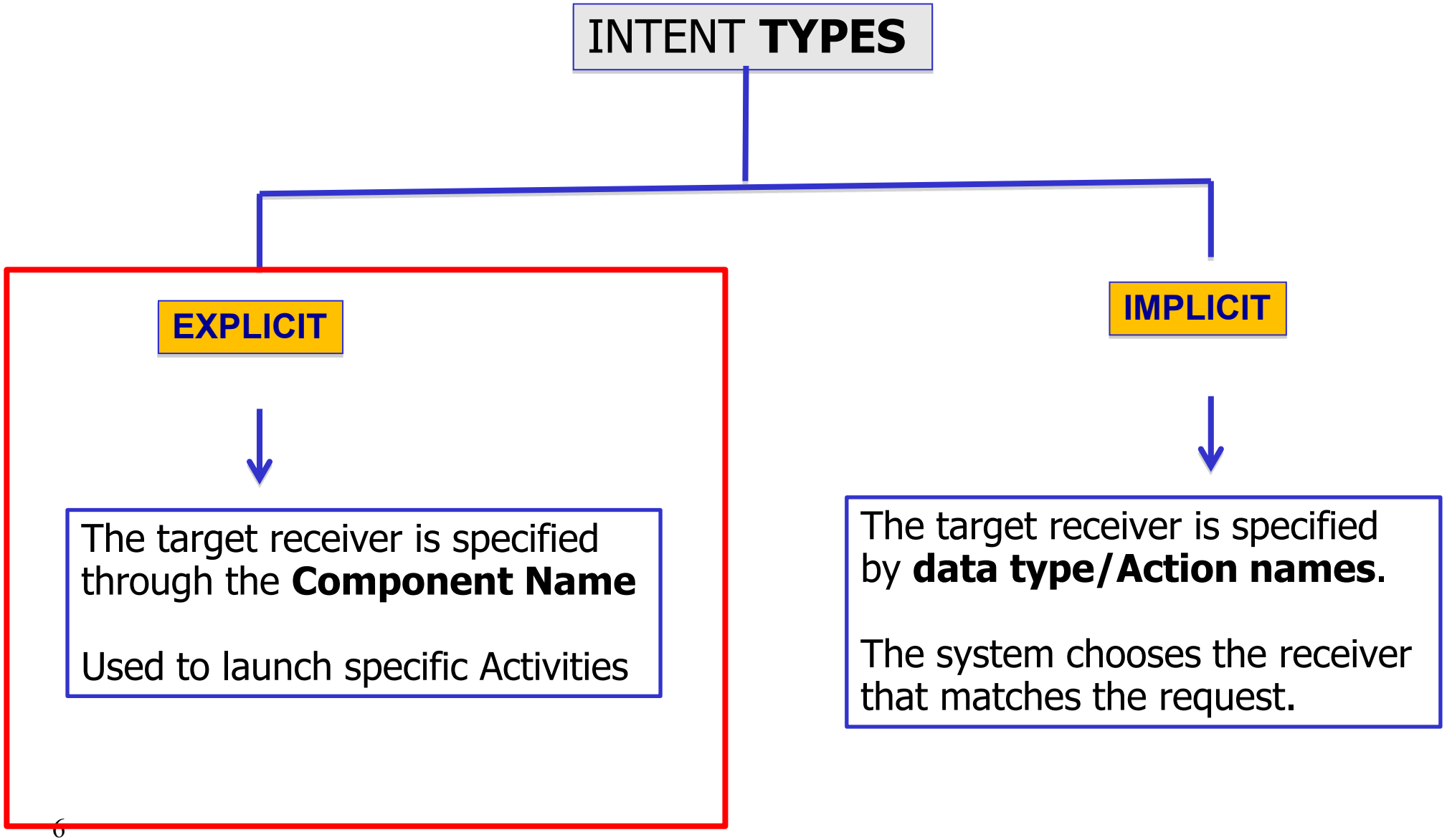
- **Call** a component from other or same application
- Possible to **pass data** between components
- Components: *Activities, Services, Broadcast receivers* …
- Something like:
  - "Android, please do **that** with **these** data"
- **Reuse** already installed applications and components

Intent Definition...

- We can think of an "**Intent**" object as a **message** containing a bundle of information.



Intent types



Android Intent: Components

Component that should handle the intent (i.e. the **receiver**).

It is **optional** (in implicit intent)

void setComponent(Component Name)

Android Intent: COMPONENT NAME

- It is an android Component Name object representing either *Activity*, *Service* or *Broadcast Receiver* class.
- If it is set, the Intent object is delivered to an instance of the designated class,
  - otherwise Android uses other information in the Intent object to locate a suitable target (implicit).
- The component name is set by:
  - setComponent(), setClass(), or setClassName() and
  - read by getComponent().

.....cont

- **Explicit** Intent: Specify the name of the Activity that will handle the intent. Different ways :

```
Intent intent=new Intent(this,
SecondActivity.class);
startActivity(intent);
```

```
Intent intent=new Intent();
ComponentName component=new
ComponentName(this,SecondActivity.class);
intent.setComponent(component);
startActivity(intent);
```

```
startActivity(new
Intent("com.example.intent1.ACTIVITY2"));
```

Example1 (Explicit)

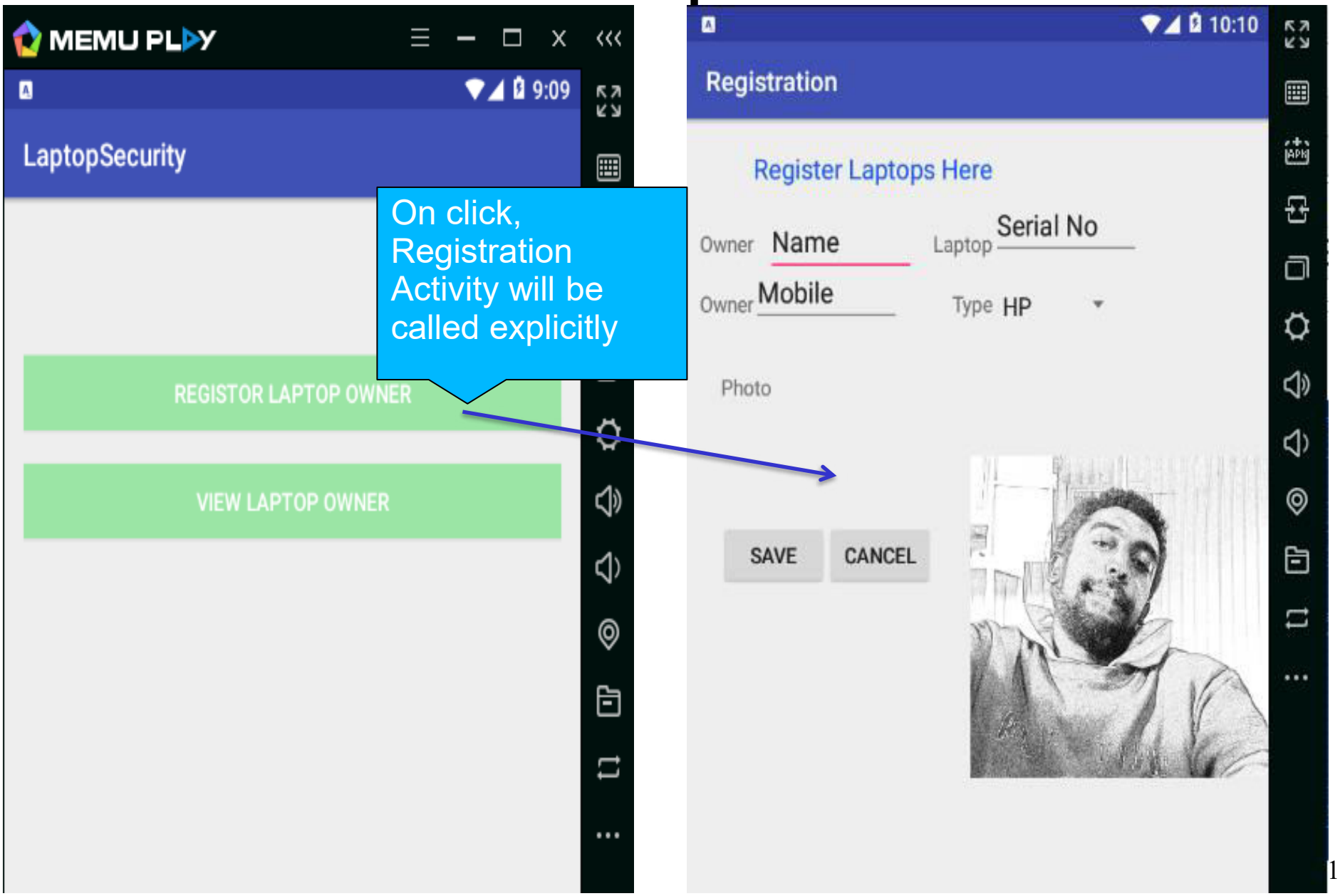
```
package com.example.tinsae.laptopsecurity;
    import android.content.Intent;
    import android.support.v7.app.AppCompatActivity;
    import android.os.Bundle;
    import android.view.*;
    import android.view.View;
    import android.widget.Button;
    public class MainActivity extends AppCompatActivity {
        Button bView;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main);
            Button bRegistration= (Button) findViewById(R.id.btnRegistration);
            bView= (Button) findViewById(R.id.btnView);

            //on click event on button registration
            bRegistration.setOnClickListener(new View.OnClickListener() {
                @Override
```

```
public void onClick(View view) {
                //Explicit intent to call an activity with in application
                Intent toRegistration = new Intent(MainActivity.this,
   Registration.class);
                startActivity(toRegistration);
            }
        });
        //on click event on button view laptop owners
        bView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                //Explicit intent to call an activity with in application
                Intent toView = new Intent(MainActivity.this, ViewOwner.class);
                startActivity(toView);

            }
        });
    }
}
```

activity_main.xml

```
<Button
        android:text="Registor Laptop Owner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_marginTop="84dp"
        android:id="@+id/btnRegistration"
        android:layout_alignParentStart="true"
        android:textColor="#ffffff"
        android:background="#5515f433"/>
    <Button
        android:text="View Laptop Owner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/btnView"
        android:textColor="#ffffff"
        android:background="#5515f433"
        android:layout_below="@+id/btnRegistration"
        android:layout_alignParentStart="true"
        android:layout_marginTop="21dp" />

</RelativeLayout>
```

Output


MEMU PLAY
9:09
LaptopSecurity
On click, Registration Activity will be called explicitly
REGISTOR LAPTOP OWNER
VIEW LAPTOP OWNER
10:10
Registration
Register Laptops Here
Owner Name
Laptop Serial No
Owner Mobile
Type HP
Photo
SAVE
CANCEL

Example2 (Explicit)

Registration.java

```
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
public class Registration extends AppCompatActivity {
  //widgets
   EditText Oname,SN,Mobile;
   Button save;
   Spinner Ltype;
   //variables
   String OwnerName,serialNumber,mobile,laptop_type;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      //View loaded from xml
      setContentView(R.layout.activity_registration);
```

```
//casting        save=(Button)findViewById(R.id.btnSave);
          Oname= (EditText) findViewById(R.id.edtName);
          SN= (EditText) findViewById(R.id.edtSN);
          Mobile=(EditText) findViewById(R.id.edtMobile);
          Ltype =(Spinner) findViewById(R.id.spLtype);
         //Anonymous Inner Class Event Listener for the spinner
          Ltype.setOnItemSelectedListener(new
   AdapterView.OnItemSelectedListener() {
              @Override
              public void onItemSelected(AdapterView<?> adapterView, View
   view, int i, long l) {
                 //Assigning selected item of the listener
                 laptop_type=Ltype.getSelectedItem().toString();
              }
              @Override
              public void onNothingSelected(AdapterView<?> adapterView) {
              }
          });
         //Anonymous Inner Class Event Listener for the save button
         save.setOnClickListener(new View.OnClickListener() {
```

```
@Override
            public void onClick(View view) {
                //collecting inputs
                OwnerName=Oname.getText().toString();
                serialNumber=SN.getText().toString();
                mobile=Mobile.getText().toString();
                //explicit Intent
               Intent toconfirm = new Intent(Registration.this,
   SavedResult.class);
                    toconfirm.putExtra("ON", OwnerName);
                    toconfirm.putExtra("SN",serialNumber);
                    toconfirm.putExtra("MOB",mobile);
                    toconfirm.putExtra("LT",laptop_type);
                  startActivity(toconfirm);
              }
          });
      }
   }
```

activity_registration.xml

```
<?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:orientation="vertical" android:layout_width="match_parent"
       android:layout_height="match_parent">
       <TextView
       android:text="Register Laptops Here"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_alignParentTop="true"
       android:layout_alignParentStart="true"
       android:layout_marginStart="55dp"
       android:layout_marginTop="78dp"
       android:textSize="20dp"
       android:textColor="#1859f1"
       android:id="@+id/textView" />

       <TextView
          android:text="Owner"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:layout_below="@+id/textView"
          android:layout_alignParentStart="true"
          android:layout_marginStart="11dp"
          android:layout_marginTop="29dp"
          android:id="@+id/textView3" />
```

```
<EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:hint="Name"
        android:ems="5"
        android:id="@+id/edtName"
        android:layout_alignBaseline="@+id/textView3"
        android:layout_alignBottom="@+id/textView3"
        android:layout_alignStart="@+id/textView"
        android:layout_marginStart="12dp" />

    <TextView
        android:text="Laptop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignBottom="@+id/textView3"
        android:id="@+id/textView5"
        android:layout_alignEnd="@+id/textView7" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:hint="Serial No"
        android:ems="5"
        android:layout_alignBottom="@+id/edtName"
        android:layout_toEndOf="@+id/textView"
        android:id="@+id/edtSN" />
```

```
<TextView
        android:text="Owner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/edtName"
        android:layout_alignStart="@+id/textView3"
        android:layout_marginTop="21dp"
        android:id="@+id/textView6" />

    <EditText
        android:layout_width="150dp"
        android:layout_height="wrap_content"
        android:inputType="textPersonName"
        android:hint="Mobile"
        android:ems="5"
        android:id="@+id/edtMobile"
        android:layout_below="@+id/edtName"
        android:layout_toEndOf="@+id/textView3" />

    <Spinner
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/spLtype"
        android:entries="@array/brand"
        android:layout_alignTop="@+id/textView7"
        android:layout_alignStart="@+id/edtSN" />
```

```
<TextView
    android:text="Type"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView7"
    android:layout_alignBaseline="@+id/textView6"
    android:layout_alignBottom="@+id/textView6"
    android:layout_toStartOf="@+id/edtSN" />

<TextView
    android:text="Photo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/spLtype"
    android:layout_alignStart="@+id/textView6"
    android:layout_marginStart="18dp"
    android:layout_marginTop="35dp"
    android:id="@+id/textView8" />

<Button
    android:text="Cancel"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@+id/btnSave"
    android:id="@+id/button2"
    android:layout_alignBottom="@+id/btnSave"
    android:layout_toEndOf="@+id/btnSave" />
```

```
<Button
        android:text="Save"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/btnSave"
        android:layout_below="@+id/textView8"
        android:layout_alignStart="@+id/textView8"
        android:layout_marginTop="86dp" />

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/c"
        android:id="@+id/imageView4"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/textView7"
        android:layout_marginStart="15dp" />

  </RelativeLayout>
```

SavedResult.java

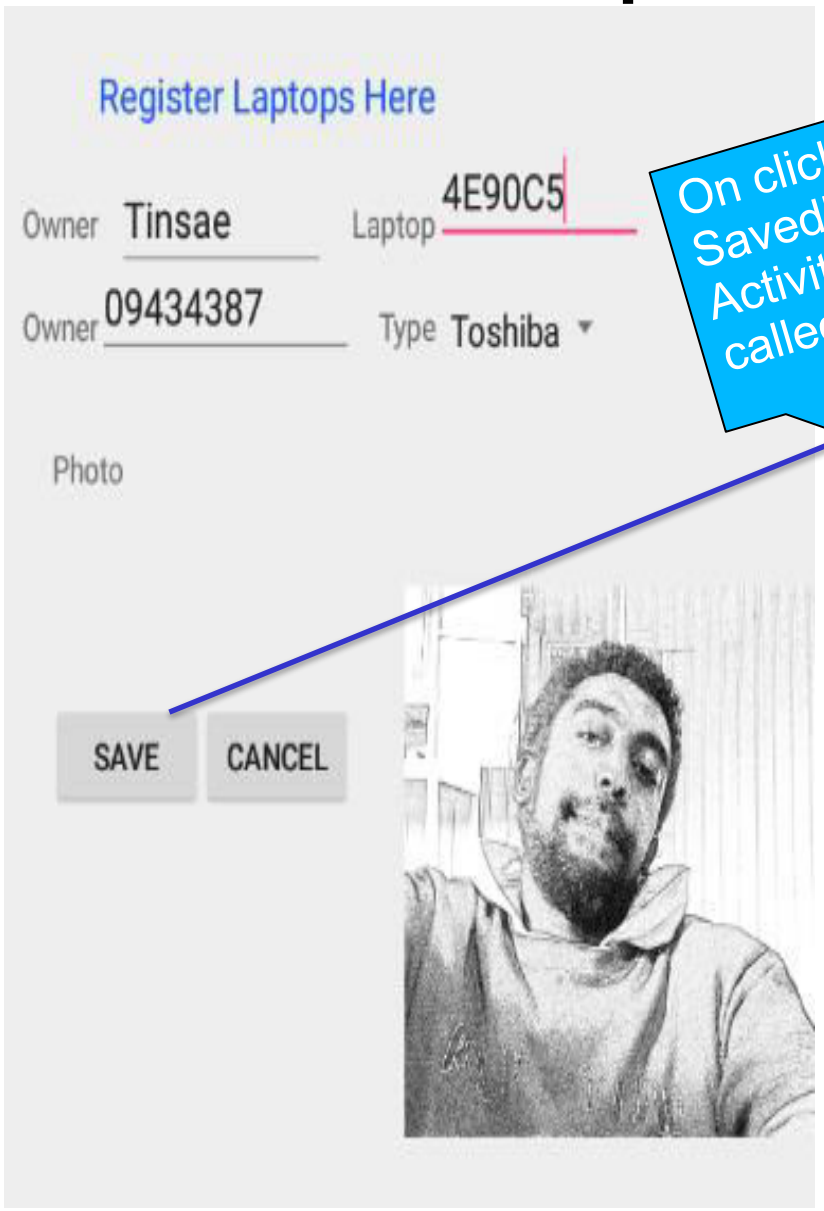
```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
public class SavedResult extends AppCompatActivity {
        String name,sn,mobile,laptopType;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_saved_result);
        EditText result= (EditText) findViewById(R.id.edtResult);
        // getting extras sent with the explicit intent
        name =getIntent().getExtras().getString("ON");
        sn =getIntent().getExtras().getString("SN");
        mobile =getIntent().getExtras().getString("MOB");
        laptopType =getIntent().getExtras().getString("LT");
        //setting what is sent by the intent to edit text
        result.setText("The Owner is :"+name+"          Laptop
Serial:"+sn+"" +
        "       " +            "Mobile :"+mobile+"
LaptoType:"+laptopType);
    }
}
```

activity_saved_result.xml

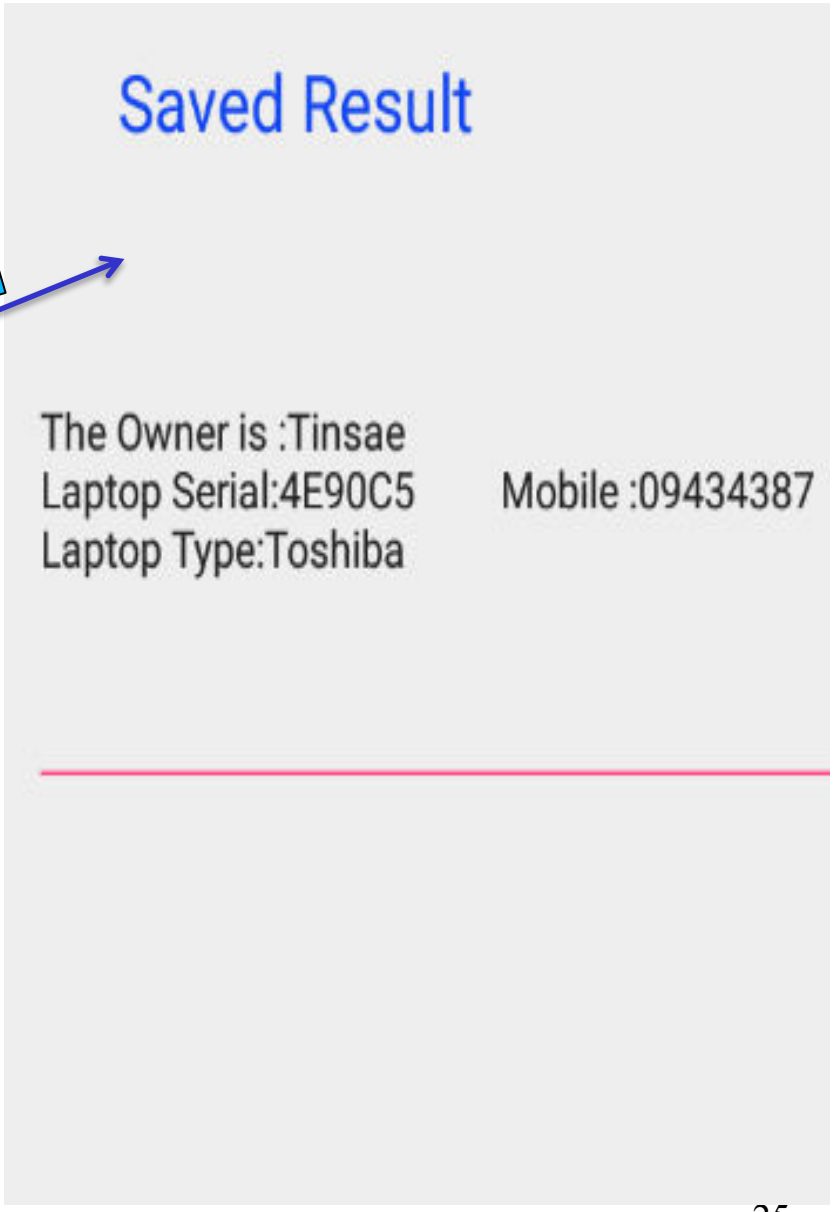
```
?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/activity_saved_result"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="@dimen/activity_vertical_margin"
        android:paddingLeft="@dimen/activity_horizontal_margin"
        android:paddingRight="@dimen/activity_horizontal_margin"
        android:paddingTop="@dimen/activity_vertical_margin"
        tools:context="com.example.tinsae.laptopsecurity.SavedResult“>


        <TextView
            android:text="Saved Result"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginStart="44dp"
            android:layout_marginTop="12dp"
            android:textColor="#1447ed"
            android:textSize="33dp"
            android:id="@+id/textView9"
            android:layout_alignParentTop="true"
            android:layout_alignParentStart="true" />
```

```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="250dp”
  android:inputType="textPersonName|textMultiLine"
      android:ems="10"
      android:layout_marginTop="14dp"
      android:id="@+id/edtResult"
      android:layout_below="@+id/textView9"
      android:layout_alignParentStart="true" />
  </RelativeLayout>
```

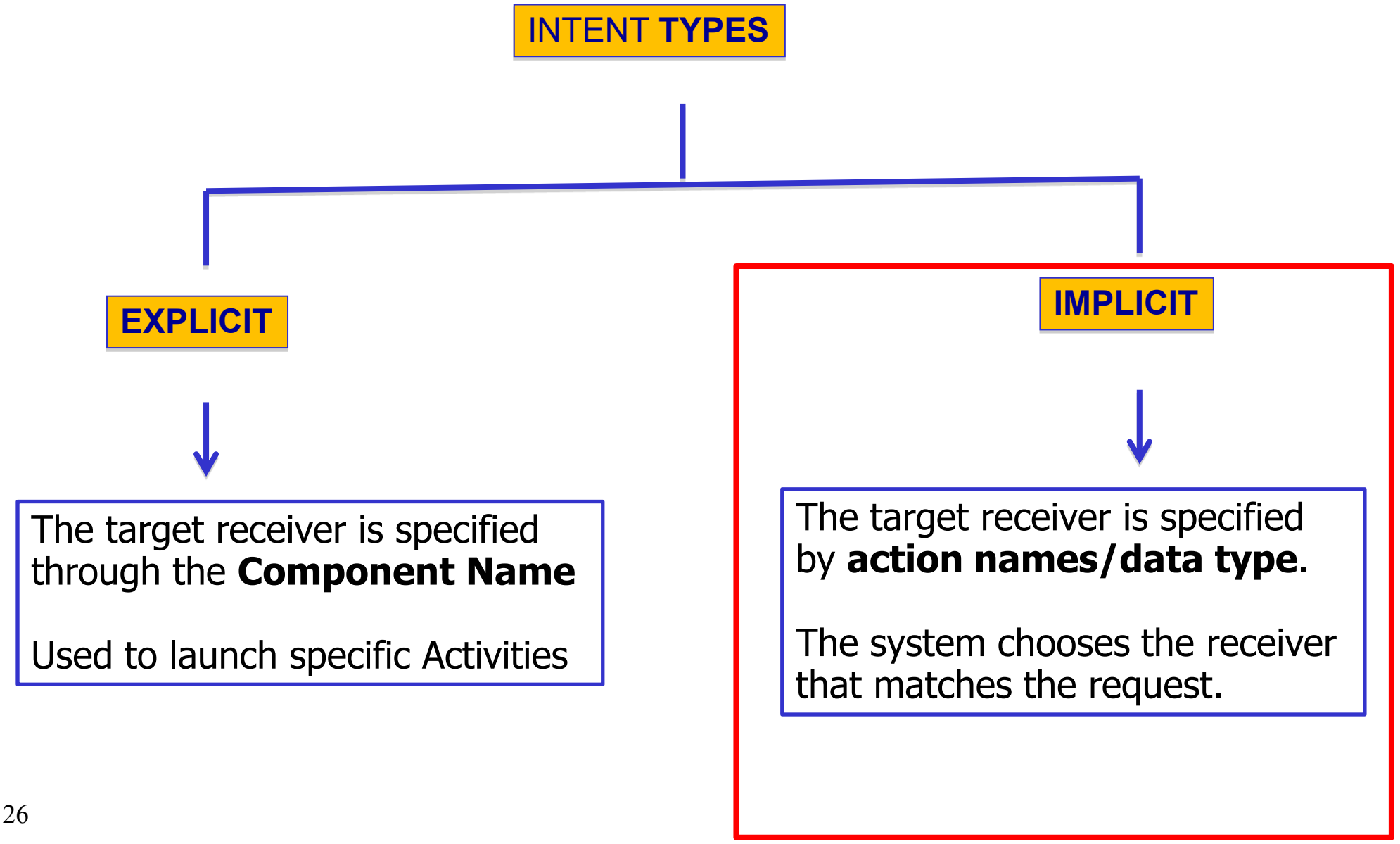
Out put (Example 2)



On click, SavedResult Activity will be called explicitly



Intent types

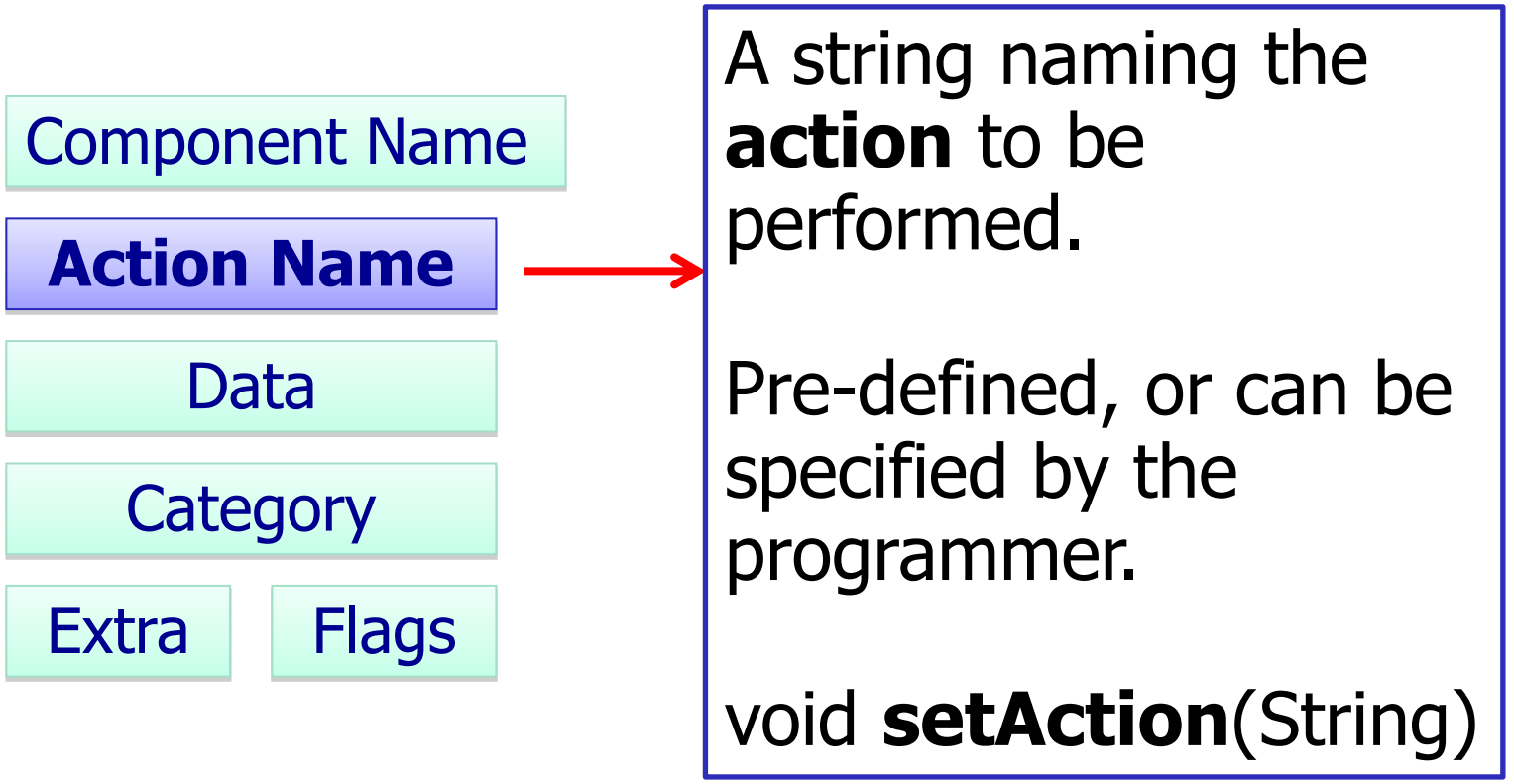


Intent types: Implicit Intents

- **Implicit** Intents: do not name a target (*component name is left blank*) …
- When an Intent is launched, Android checks out which activies might answer to the Intent …
- If at least one is found, then that activity is started!
- Binding does not occur at compile time, nor at install time, but at run-time …(***late run-time binding***)

Intent Components

- We can think to an "**Intent**" object as a **message** containing a bundle of information.



Android Intent: Actions ...

```
// Implicit Intent by specifying ACTION and URI

Intent i = new Intent(Intent.ACTION_VIEW,
Uri.parse("http://www.google.com"));

// Starts Implicit Activity

startActivity(i);
```

```
Intent i = new
       Intent(android.content.Intent.ACTION_VIEW,
        Uri.parse("geo:37.827500,-122.481670"));
         startActivity(i);
```

Android Intent: Standard Actions

Action Name	Description
ACTION_ALL_APPS	List all the applications available on the device.
ACTION_ANSWER	Handle an incoming phone call.
ACTION_CALL	Perform a call to someone specified by the data.
ACTION_DIAL	Dial a number as specified by the data.
ACTION_BATTERY_LOW	This broadcast corresponds to the "Low battery warning" system dialog.
ACTION_CALL_BUTTON	The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
ACTION_CAMERA_BUTTON	The "Camera Button" was pressed.
ACTION_DEVICE_STORAGE_LOW	A sticky broadcast that indicates low memory condition on the device.
ACTION_GET_CONTENT	Allow the user to select a particular kind of data and return it.
ACTION_INSTALL_PACKAGE	Launch application installer.
ACTION_MEDIA_BUTTON	The "Media Button" was pressed.

Android Intent: Standard Actions

Action Name	Description
ACTION_NEW_OUTGOING_CALL	An outgoing call is about to be placed.
ACTION_POWER_CONNECTED	External power has been connected to the device.
ACTION_SEARCH	Perform a search.
ACTION_SEND	Deliver some data to someone else.
ACTION_SET_WALLPAPER	Show settings for choosing wallpaper.
ACTION_SHUTDOWN	Device is shutting down.
ACTION_VIEW	Display the data to the user.
ACTION_WEB_SEARCH	Perform a web search.
ACTION_TIME_CHANGED	he time was set.
ACTION_MAIN	Start as a main entry point, does not expect to receive data.

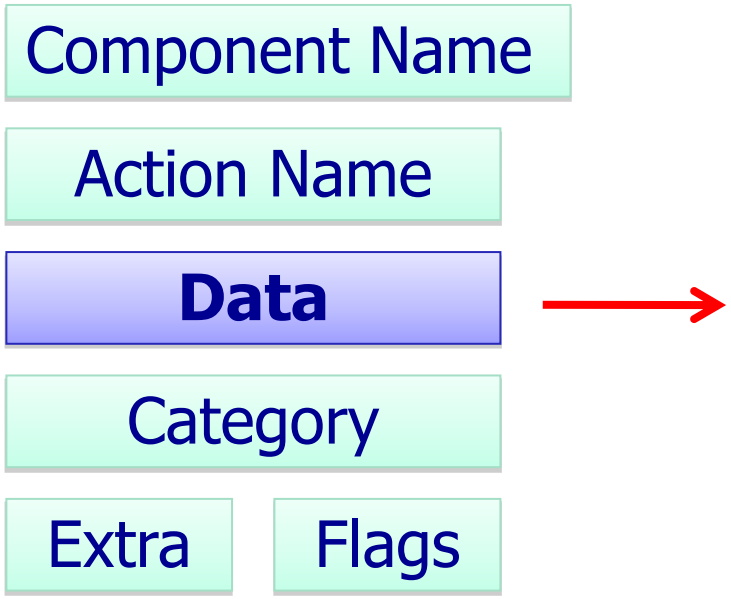
Android Intent: Special Actions

Action Name	Description
ACTION_IMAGE_CAPTION	Open the camera and receive a photo
ACTION_VIDEO_CAPTION	Open the camera and receive a video
ACTION_DIAL	Open the phone app and dial a phone number
ACTION_SENDTO	Send an email (email data contained in the extra)
ACTION_SETTINGS	Open the system setting
ACTION_WIRELESS_SETTINGS	Open the system setting of the wireless interfaces
ACTION_DISPLAY_SETTINGS	Open the system setting of the display

The action in an Intent object can be set by the **setAction()** method and read by **getAction()**.

Intent (IMPLICIT)

- We can think to an "**Intent**" object as a **message** containing a bundle of information.



Data passed from the caller to the called Component.

Def. of the data (**URI**) and Type of the data (**MIME** type)

void **setData**(Uri)
void **setType**(String)

Android Intent: DATA…

- Each data is specified by a **name** and/or **type**.
- **name**: Uniform Resource Identifier (**URI**)
- **scheme://host:port/path**

EXAMPLEs

tel://003-232-234-678

content://contacts/people

http://www.google.com/

Android Intent: DATA…

- Each data is specified by a **name** and/or **type**.
- **type**: **MIME** (Multipurpose Internet Mail Extensions)-type
- Composed by two parts: a type and a subtype

EXAMPLEs

Image/gif image/jpeg image/png image/tiff
text/html text/plain text/javascript text/css
video/mp4 video/mpeg4 video/quicktime video/ogg
application/vnd.google-earth.kml+xml

Android Intent: DATA…

- The URI of the data to be acted on and the MIME type of that data.
- The setData() method specifies data only as a URI,
- setType() specifies it only as a MIME type, and
- setDataAndType() specifies it as both a URI and a MIME type.
- The URI is read by getData() and the type by getType().

Some examples of action/data pairs

Action/Data pair	Description
ACTION_VIEW content://contacts/people/1	Display information about the person whose identifier is "1".
ACTION_DIAL content://contacts/people/1	Display the phone dialer with the person filled in.
ACTION_VIEW tel:123	Display the phone dialer with the given number filled in.
ACTION_CALL tel:123	Display the phone dialer and call with the given number filled in.
ACTION_EDIT content://contacts/people/1	Edit information about the person whose identifier is "1".
ACTION_VIEW content://contacts/people/	Display a list of people, which the user can browse through.

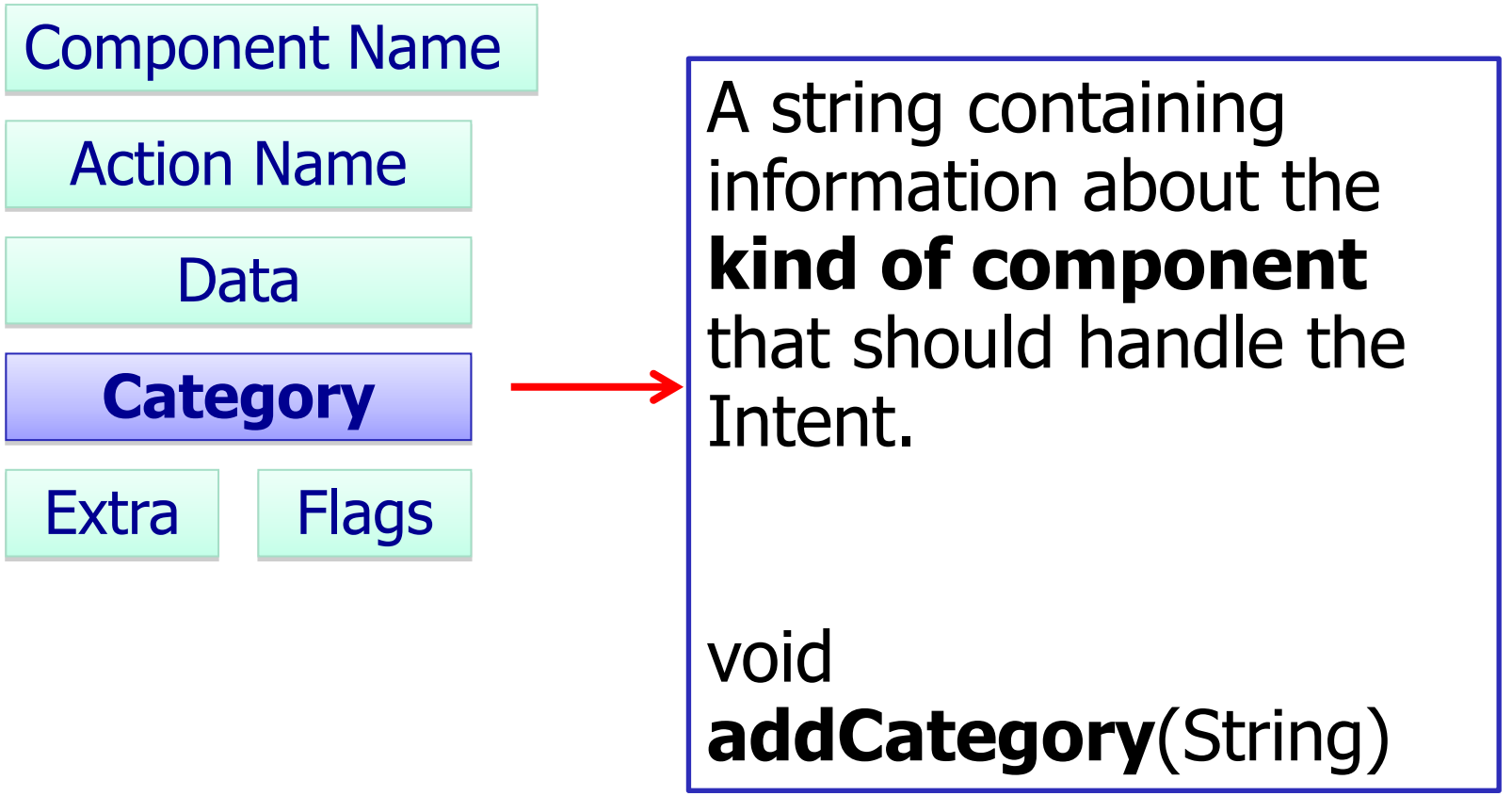
Intent types: Implicit Intents

```
Intent i = new
Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("http://www.facebook.com"));
startActivity(i);
```

Action to perform **Data to perform the action on**

- Implicit intents are very useful to **re-use code** and to **launch external applications …**
- *More than a component can match the Intent request …*

Android Intent: Components



Android Intent: CATEGORY

- **Category**: string describing the **kind of component** that should handle the intent.
- The category is an optional part of Intent object and
- The addCategory() method places a category in an Intent object,
- removeCategory() deletes a category previously added, and
- getCategories() gets the set of all categories currently in the object.

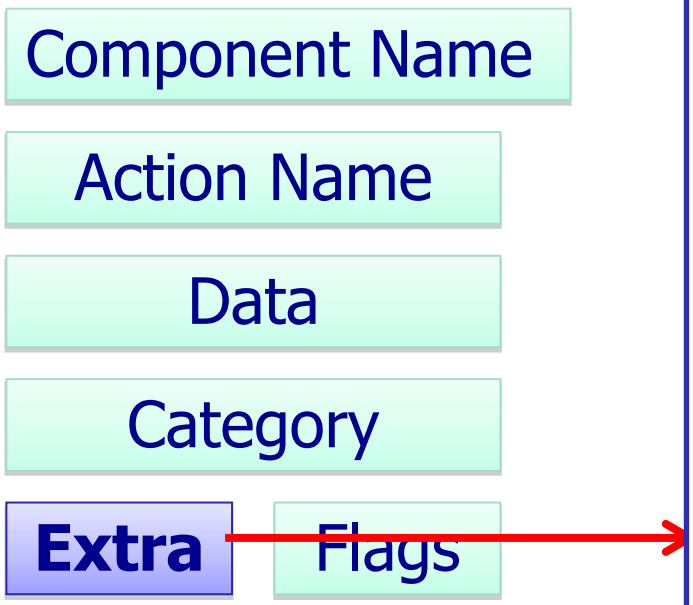
Android Intent: CATEGORY

Category Name	Description
CATEGORY_APP_BROWSER	Used with ACTION_MAIN to launch the browser application.
CATEGORY_APP_CALCULATOR	Used with ACTION_MAIN to launch the calculator application.
CATEGORY_APP_CALENDAR	Used with ACTION_MAIN to launch the calendar application.
CATEGORY_APP_CONTACTS	Used with ACTION_MAIN to launch the contacts application.
CATEGORY_APP_EMAIL	Used with ACTION_MAIN to launch the email application.
CATEGORY_APP_GALLERY	Used with ACTION_MAIN to launch the gallery application.
CATEGORY_APP_MAPS	Used with ACTION_MAIN to launch the maps application.
CATEGORY_APP_MUSIC	Used with ACTION_MAIN to launch the music application.

Android Intent: CATEGORY

Category Name	Description
CATEGORY_HOME	The activity displays the HOME screen.
CATEGORY_LAUNCHER	The activity is listed in the top-level application launcher, and can be displayed.
CATEGORY_PREFERENCE	The activity is a preference panel.
CATEGORY_BROWSABLE	The activity can be invoked by the browser to display data referenced by a link.

Android Intent: Components



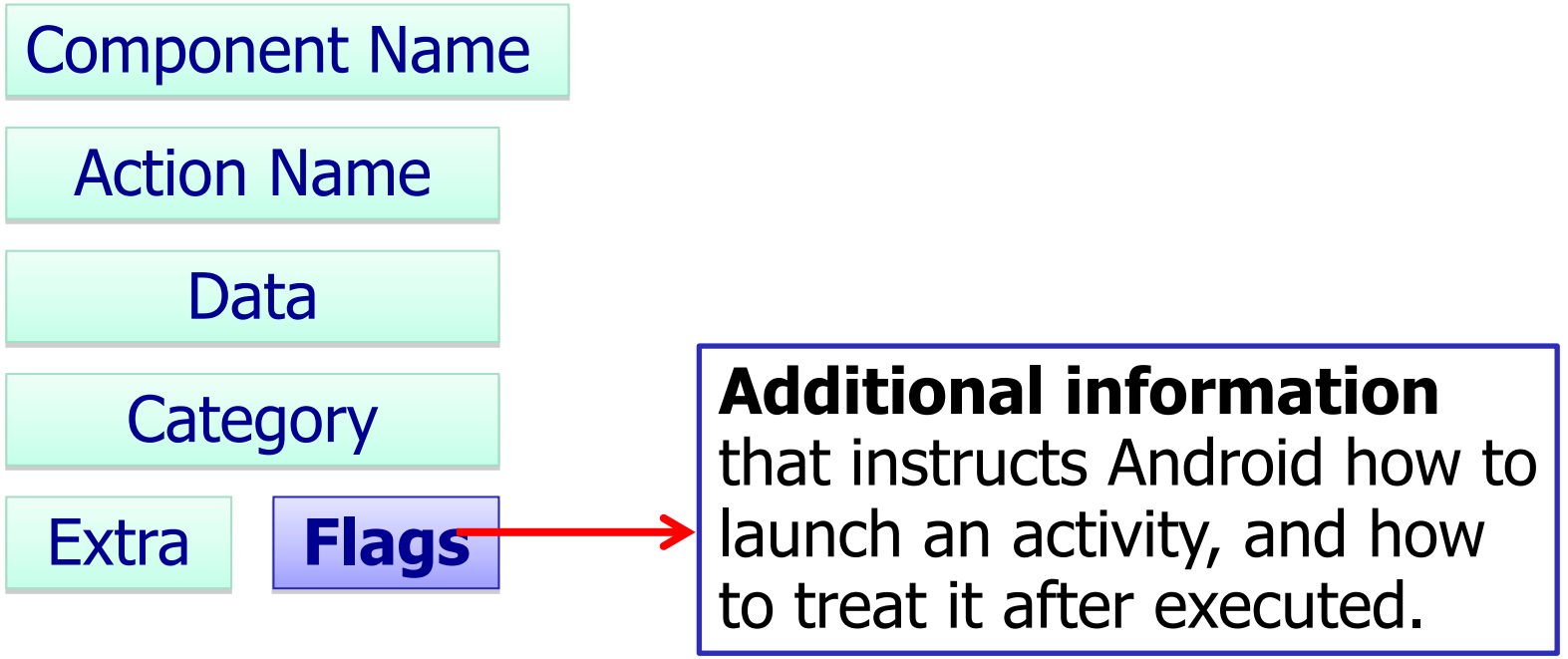
Additional information that should be delivered to the handler(e.g. parameters).

void **putExtras**()
getExtras()

Android Intent:standard Extra Data

Extra data	Description
EXTRA_ALARM_COUNT	Used as an int extra field in AlarmManager intents to tell the application being invoked how many pending alarms are being delivered with the intent.
EXTRA_EMAIL	A String[] holding e-mail addresses that should be delivered to.
EXTRA_PHONE_NUMBER	A String holding the phone number originally entered in ACTION_NEW_OUTGOING_CALL, or the actual number to call in a ACTION_CALL.
EXTRA_SUBJECT	A constant string holding the desired subject line of a message.

Android Intent: Components



Example (implicit)
Implicit.java

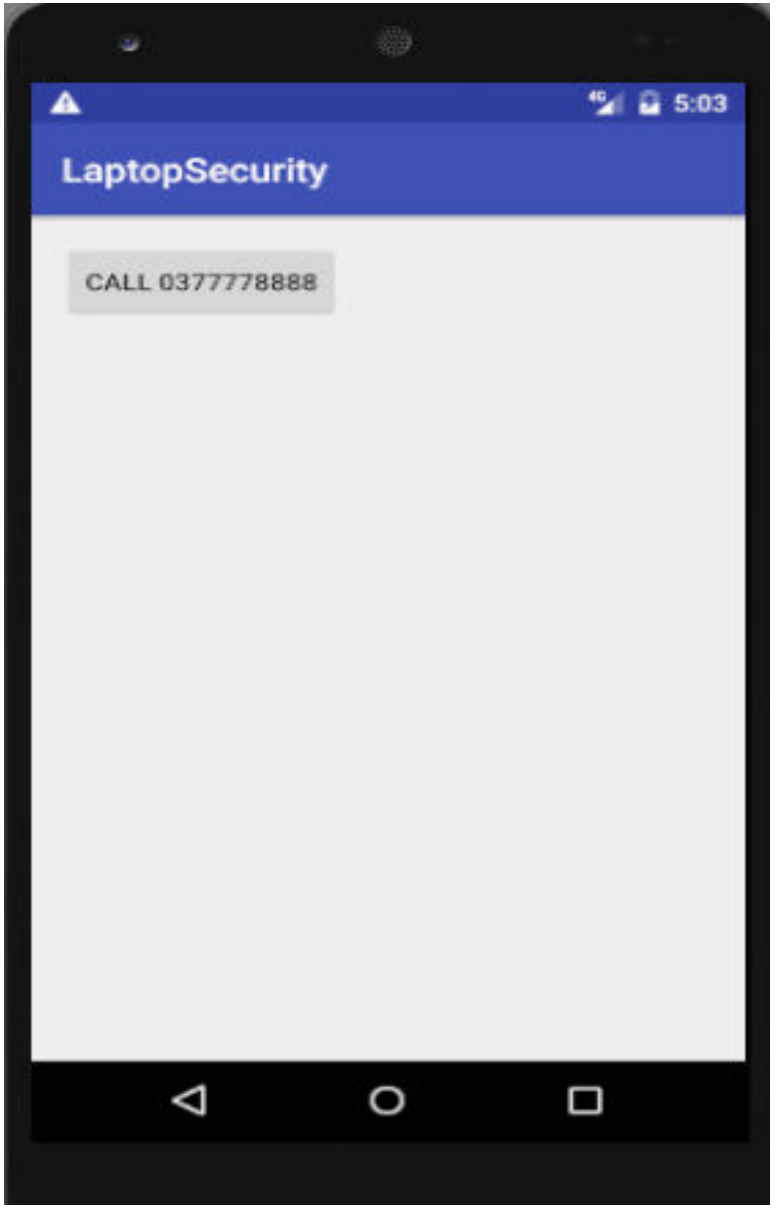
```
import ….;
 public class Implicit extends AppCompatActivity {
 Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_implicit);
        button = (Button) findViewById(R.id.buttonCall);

        button.setOnClickListener(new
 android.view.View.OnClickListener() {
            @Override
            public void onClick(android.view.View view) {
```

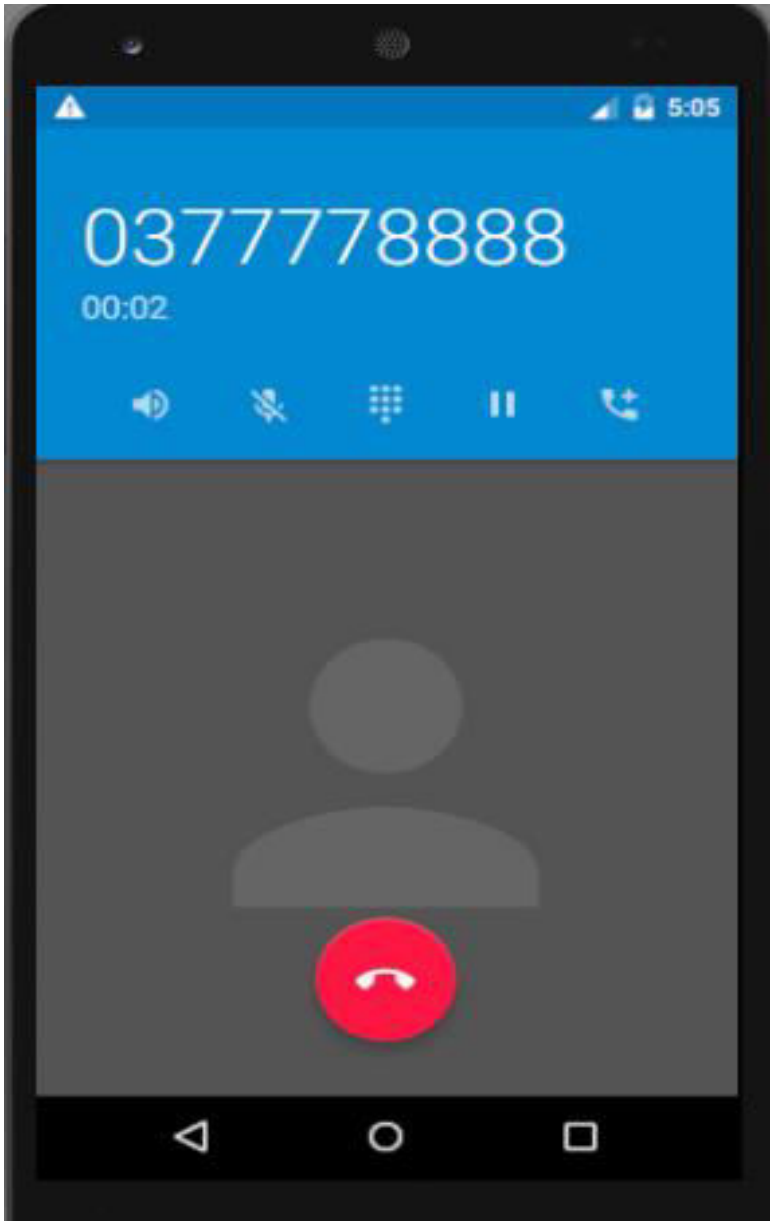
```
Intent callIntent = new Intent(Intent.ACTION_CALL);

   callIntent.setData(Uri.parse("tel:0377778888"));

                  if
   (ActivityCompat.checkSelfPermission(Implicit.this,
                        Manifest.permission.CALL_PHONE) !=
   PackageManager.PERMISSION_GRANTED) {
                     return;
                  }
                  startActivity(callIntent);

               }
            });
      }
   }
```

activity_implicit.xml

```
<?xml version="1.0" encoding="utf-8"?>
    <RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
       xmlns:tools="http://schemas.android.com/tools"
       android:id="@+id/activity_implicit"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:paddingBottom="@dimen/activity_vertical_margin"
       android:paddingLeft="@dimen/activity_horizontal_margin"
       android:paddingRight="@dimen/activity_horizontal_margin"
       android:paddingTop="@dimen/activity_vertical_margin"
       tools:context="com.example.tinsae.laptopsecurity.Implicit">
       <Button
          android:id="@+id/buttonCall"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="call 0377778888" />
    </RelativeLayout>

//use this security inside manifest file

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

Out put





Android Intents: How It Works

- In this example, you saw how you can use the Intent class to invoke some of the built-in applications in Android (such as Maps, Phone, Contacts, and Browser).
- In Android, intents usually come in pairs: action and data.
  - The action describes what is to be performed, such as editing an item, viewing the content of an item, and so on.
  - The data specifies what is affected, such as a person in the Contacts database. The data is specified as an **`Uri object`**.

Some examples of action are as follows:

- **`ACTION_VIEW`**
- **`ACTION_DIAL`**
- **`ACTION_PICK`**

Some examples of data include the following:

- **`http://www.google.com`**
- **`tel:+651234567`**
- **`geo:37.827500,-122.481670`**
- **`content://contacts`**

Android Intent: How It Works…

- Collectively, the action and data pair describes the operation to be performed. For example:
  - To dial a phone number, you would use the pair **`ACTION_DIAL/tel:+651234567`**.
  - To display a list of contacts stored in your phone, you use the pair **`ACTION_VIEW/content://contacts`**.
  - To pick a contact from the list of contacts, you use the pair **`ACTION_PICK/content://contacts`**.

Intent with Results

- Activities can return results (e.g. data)
- Sender side: invoke the

startActivityForResult()

- **onActivityResult**(int requestCode, int resultCode, Intent data)
- **startActivityForResult**(Intent intent, int requestCode);

```
Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, request_code);
…
public void onActivityResult(int requestCode, int
resultCode, Intent data) {
   // Invoked when SecondActivity completes its
operations …
}
```

Intent with Results

- Receiver side: invoke the **setResult()**
  - void **setResult**(int resultCode, Intent data)

```
Intent data=new Intent();
data.setData();
setResult(RESULT_OK, data);
finish();
```

- The result is delivered to the caller component only after invoking the **finish()** method!

Intent: How It Works…

- Since you are expecting a result from the Contacts application, you invoke it using the **`startActivityForResult()`** method, passing in the **`Intent`** object and a **`request code`**.
- You need to implement the **`onActivityResult()`** method in order to obtain a result from the activity:

```
public void onActivityResult(int requestCode,
  int resultCode, Intent data)  {
      if (requestCode == request_Code)  {
       if (resultCode == RESULT_OK) {
      Intent i = new Intent(
android.content.Intent.ACTION_VIEW,
           Uri.parse(data.getData().toString()));
          startActivity(i);
              }
            }
          }
```

Intent types: Intent Resolution

- The **intent resolution** process resolves the Intent-Filter that can handle a given Intent.
- Three tests to be passed:
  - **Action field** test
  - **Category field** test
  - **Data field** test
- If the Intent-filter passes all the three test, then it is selected to handle the Intent.

Intent types: Intent Resolution

- (**ACTION** Test): The action specified in the Intent must match one of the actions listed in the filter.
  - If the filter does not specify any action → **FAIL**
  - An intent that does not specify an action → **SUCCESS** as as long as the filter contains at least one action.

```
<intent-filer … >
<action
android:name="com.example.intent1.SECONDACTIV
    ITY"/>
</intent-filter>
```

Intent types: Intent Resolution

- (**CATEGORY** Test): Every category in the Intent must match a category of the filter.
  - If the category is not specified in the Intent → Android assumes it is CATEGORY_DEFAULT, thus the filter must include this category to handle the intent

```
<intent-filer … >
   <category
    android:name="android.intent.category.DEF
    AULT"/>
</intent-filter>
```

Intent types: Intent Resolution

- (**DATA** Test): The URI of the intent is compared with the parts of the URI mentioned in the filter.

```
<intent-filer … >
   <data android:mimeType="audio/*
    android:scheme="http"/>
   <data android:mimeType="video/mpeg
    android:scheme="http"/>
</intent-filter>
```

- Both URI and MIME-types are compared.

Assignment 4 - Group(5%)

Include explicit as well as implicit intent to your project.